UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Fakultät für Elektrotechnik, Informatik und Mathematik
Heinz Nixdorf Institut und Institut für Informatik
Fachgebiet Softwaretechnik
Zukunftsmeile 1
33102 Paderborn

Fuzzy Matching of Comprehensive Service Specifications

PhD Thesis
to obtain the degree of
"Doktor der Naturwissenschaften (Dr. rer. nat.)"

by

MARIE CHRISTIN PLATENIUS
ROEINGHSTRASSE 5
33102 PADERBORN

Supervisor:
Prof. Dr. Wilhelm Schäfer

Paderborn, July 19, 2016

ABSTRACT

In the last decades, software development turned from monolithic software products towards flexibly combinable software services. Correspondingly, a growing number of service providers started offering their services in world-wide service markets. As a consequence, customers, so-called service requesters, can buy software services from such markets. In general, service requesters buy services that match their functional and non-functional requirements best. Thus, among all services provided in the market, the services that match these requirements best need to be discovered by considering these requirements. For the purpose of this so-called *service matching*, the requester's requirements specification and the providers' service specifications are compared. As a result, the extent to which the provided service specification satisfies the requirements specification is returned.

The described scenario leads to two main challenges. (1) Existing matching approaches deliver inaccurate results because each of them only considers a specific kind of requirements. The combination of multiple matching approaches, however, is a tedious and error-prone task and this task had to be done manually up to now. Challenges include design decisions about control flow and data flow as well as the aggregation of matching results. (2) The more complex the specifications we are dealing with are, the more the probability for (partially) imperfect specifications increases. For example, requesters often have vague requirements or deliver imprecise requirements specifications. Providers often provide incomplete specifications for several reasons including the high specification effort or a lack of technical knowledge. Furthermore, matching approaches themselves can be imprecise on purpose to keep complex matching problems efficiently solvable. In these cases, traditional matching approaches deliver adulterated and deceptive matching results that do not notify the users about the induced uncertainty.

In order to deal with complex service specifications, in this thesis, we propose the idea of *matching processes* that combine multiple existing matching approaches and aggregate their matching results. Thereby, a variety of different requirements can be covered. For this purpose, a model-driven development framework for comprehensive and configurable service matching, called MatchBox, is introduced. MatchBox simplifies, validates, and partially automatizes complex integration tasks.

In order to cope with uncertainty induced into the matching procedure, we propose concepts for *Fuzzy Matching*. On the basis of well-defined fuzziness sources and types, the amount of induced fuzziness is quantified and returned as part of an informative matching result that reflects the induced uncertainty to the user. Thereby, fuzzy matching provides valuable information about the quality of the matching result, which improves the decision-making of both service requesters and service providers.

We realized our concepts in a prototype that was also used for the evaluation including four case studies. Both researchers and practitioners benefit from the contributions presented by this thesis. By combining multiple research areas in a novel way, this thesis describes and evaluates concepts that go significantly beyond the state of the art in service matching. Thereby, it constitutes an important step to bring service matching into practice.

ZUSAMMENFASSUNG

Softwareentwicklung hat sich in den letzten Jahren von dem Entwurf und der Erstellung monolithischer Softwareprodukte zu flexibel kombinierbaren Software-Diensten (Services) gewandelt. Entsprechend begannen immer mehr Software-Anbieter, solche Dienste auf weltweiten Märkten bereitzustellen, um es Kunden zu ermöglichen, Software-Dienste von dort zu kaufen. Kunden kaufen die Dienste, die ihre funktionalen sowie nicht-funktionalen Anforderungen bestmöglich erfüllen. Daher müssen solche Dienste unter allen angebotenen Diensten unter Berücksichtigung der Kunden-Anforderungen gesucht werden. Der zu diesem Zweck erfolgende Abgleich der Anforderungsbeschreibungen mit den Beschreibungen der angebotenen Dienste nennt sich *Service Matching*. Das Ergebnis des Service Matchings stellt dar, in welchem Maße eine angebotene Servicebeschreibung auf die Anforderungsbeschreibung passt.

Das beschriebene Szenario führt zu zwei wesentlichen Herausforderungen. (1) Existierende Ansätze zum Service Matching liefern inakkurate Ergebnisse, weil sie jeweils nur einen Teil der Anforderungen berücksichtigen. Die Kombination mehrerer solcher Matching-Ansätze ist allerdings eine mühsame und fehleranfällige Aufgabe, die bisher manuell erledigt werden muss. Herausfordernd hierbei sind unter anderem die Entwurfsentscheidungen bezüglich Kontrollfluss und Datenfluss oder auch die Aggregation von Matching-Ergebnissen. (2) Je komplexer die abzugleichenden Beschreibungen sind, desto größer ist die Wahrscheinlichkeit, dass es sich um (teilweise) unvollkommene Beschreibungen handelt. Zum Beispiel haben Kunden oft nur vage Anforderungen oder sie beschreiben ihre Anforderungen ungenau. Anbieter beschreiben zudem oft nur unvollständig ihre Services, beispielsweise wegen eines zu hohen Beschreibungsaufwandes oder fehlender Expertise. Weiterhin arbeiten einige Matching-Ansätze approximativ, damit sie das Matching-Problem schneller lösen können. In all diesen Fällen liefern traditionelle Matching-Ansätze verfälschte und irreführende Ergebnisse, die den Benutzer nicht über die vorhandene Ungewissheit informieren.

Um mit komplexen Servicebeschreibungen umzugehen, wurde im Rahmen dieser Dissertation das Konzept von *Matching-Prozessen* erarbeitet, die mehrere Matching-Ansätze kombinieren und ihre Ergebnisse aggregieren können. Durch diese Kombination kann eine Vielfalt von unterschiedlichen Anforderungen berücksichtigt werden. Ermöglicht wird dies durch das modellgetriebene Framework MatchBox für umfangreiches und konfigurierbares Service Matching. Insbesondere vereinfacht, validiert und teilautomatisiert MatchBox die komplexe Aufgabe der Integration mehrerer Ansätze zum Service Matching.

Um mit Ungewissheiten während des Matchings umgehen zu können, schlagen wir das Konzept des unscharfen Matchings, das sogenannte *Fuzzy Matching* vor.

Basierend auf wohldefinierten *Fuzziness*-Quellen und -Typen wird dabei das Ausmaß der auftretenden Ungewissheit quantifiziert und als Teil eines informativen Matching-Ergebnisses ausgegeben. Dieses Matching-Ergebnis reflektiert die auftretende Ungewissheit und informiert den Benutzer, der somit bessere Entscheidungen treffen kann.

Die genannten Lösungen wurden im Rahmen eines Prototypen umgesetzt und anhand von vier Fallstudien evaluiert. Der Nutzen der Ergebnisse liegt sowohl in der Forschung als auch in der Praxis. Durch die neuartige Kombination mehrerer unterschiedlicher Forschungsgebiete stellt diese Dissertation Ergebnisse vor, die deutlich über den Stand der Forschung im Bereich des Service Matchings hinausgehen. Aufgrund dessen stellt sie einen wichtigen Schritt dar, um Service Matching in der Praxis sinnvoll einzusetzen.

DANKSAGUNG

Es gibt viele Menschen die auf unterschiedlichste Weise zu dieser Arbeit beigetragen haben. An diese Stelle möchte ich all diesen Menschen danken.

Zuallererst danke ich meinem Doktorvater Wilhelm Schäfer. Er stand mir immer beratend zur Seite und ermutigte mich stets, eine wissenschaftliche Karriere einzuschlagen. Von ihm habe ich viel gelernt und ich war immer stolz darauf, ein Teil der „AG Schäfer" zu sein.

Ich danke auch den Mitgliedern meiner Prüfungskommission, Steffen Becker, Gregor Engels, Eyke Hüllermeier, Paola Inverardi und Heike Wehrheim für ihre Bereitschaft, sich mit der Arbeit auseinanderzusetzen und an meiner Verteidigung teilzunehmen. Paola Inverardi, Heike Wehrheim und Steffen Becker danke ich außerdem für die Anfertigung ihrer Gutachten.

Während meiner Promotionszeit wurde ich besonders unterstützt von Markus von Detten, Matthias Becker, Steffen Becker und Svetlana Arifulina. Aus der Arbeit an unseren gemeinsamen Papieren, den zahlreichen Diskussionen und insbesonderen den kritischeren Nachfragen habe ich viel gelernt. Ein großer Dank geht auch an all meine weiteren AG-Kollegen. Ihre Unterstützung in Form von Diskussionen, Feedback zu Papieren, Feedback zu Vorträgen und gemeinsamen Kaffeepausen war mir immer viel Wert. Ihr wart mir nicht nur Kollegen, sondern auch Freunde. Dazu zählen: Anas Anis, Matthias Becker, Steffen Becker, Christian Brenner, Eric Bodden, Christopher Brink, Nicola Danielzik, Stefan Dziwok, Markus Fockel, Jens Frieben, Johannes Geismann, Christopher Gerking, Faezeh Ghassemi, Jutta Haupt, Christian Heinzemann, Jörg Holtmann, Thorsten Koch, Stefan Krüger, Sebastian Lehrig, Jürgen Maniera, Ahmet Mehic, Sven Merschjohann, Matthias Meyer, Lisa Nguyen, Faruk Pasic, Goran Piskachev, Uwe Pohlmann, Claudia Priesterjahn, Jan Rieke, David Schmelter, David Schubert, Lars Stockmann, Christian Stritzke, Julian Suck, Oliver Sudmann, Dietrich Travkin, Markus von Detten, Benedikt Wohlers und Jinying Yu. Ein herzlicher Dank geht auch an Matthias Becker, Stefan Dziwok, Christopher Gerking, Faezeh Ghassemi, Markus Fockel, Claudia Priesterjahn, David Schubert und Markus von Detten für das Korrekturlesen diverser Abschnitte meiner Arbeit.

Bei technischen und administrativen Fragen standen mir Jutta Haupt und Jürgen Maniera immer freundlich und hilfreich zur Seite. Vielen Dank dafür!

Ich hatte außerdem das Privileg im Sonderforschungsbereich 901 mitarbeiten zu dürfen. Ich danke all meinen Kollegen die ich dort kennenlernen durfte für die tolle Zusammenarbeit. Besonders danke ich hierfür Svetlana Arifulina, Frederik Bäumer, Matthias Becker, Galina Besova, Sonja Brangewitz, Alexander Jungmann, Felix Mohr, Ronald Petrlic, Lorijn van Rooijen und Sven Walther. Im Rahmen von gemeinsamen Papieren hatte ich außerdem das Glück mit Steffen Becker, Gregor Engels, Eyke Hüllermeier und Wilhelm Schäfer

zusammenarbeiten zu dürfen. Ich danke auch dem Vorstand und dem Geschäftsführer des SFB 901 für ihre Unterstützung.

Weiterhin danke ich allen Studierenden die mich im Rahmen von SHK-Stellen, Abschlussarbeiten, und einer Projektgruppe unterstützt haben und ohne die meine Arbeit nicht möglich gewesen wäre. Insbesondere danke ich Dorina Bano, Paul Börding, Melanie Bruns, Faezeh Ghassemi, Aljoscha Hark und Sven Merschjohann.

Sowohl aus meiner Zeit in Paderborn, als auch aus meiner Heimatstadt Löhne habe ich viele liebe Freunde, die mich immer wieder ermutigt haben, auch wenn mein Terminkalender während meiner Promotionszeit teilweise recht voll war. Danke für euer Verständnis.

Nicht zuletzt danke ich meiner Familie, die mich immer unterstützte, an mich glaubte und stolz auf mich war. Vor allem danke ich meinem Freund und Partner Mario dafür, auch dann für mich dagewesen zu sein, wenn es mal etwas stressig wurde.

Contents

Titlepage II

Abstract IV

Zusammenfassung VI

Danksagung VIII

Table of Contents XII

1 Introduction 1
- 1.1 Service Matching . . . 1
- 1.2 Running Example . . . 2
- 1.3 Problem Statement . . . 4
- 1.4 Overview of the Solution . . . 6
- 1.5 Application Scenarios . . . 8
  - 1.5.1 On-The-Fly Computing . . . 8
  - 1.5.2 Mobile Apps . . . 8
  - 1.5.3 Business-to-Business Applications . . . 9
  - 1.5.4 Industry 4.0 . . . 9
- 1.6 Structure of this Thesis . . . 10

2 Foundations 11
- 2.1 Software Components and Services . . . 11
- 2.2 Service Markets . . . 11
- 2.3 Service Specification . . . 13
  - 2.3.1 Signatures . . . 13
  - 2.3.2 Behavioral Specifications . . . 14
  - 2.3.3 Privacy . . . 14
  - 2.3.4 Reputation . . . 16
- 2.4 Service Matching . . . 18
  - 2.4.1 Ontological Signature Matching . . . 19
  - 2.4.2 Behavioral Matching Approaches . . . 21
  - 2.4.3 Privacy Matching . . . 22
  - 2.4.4 Reputation Matching . . . 24
- 2.5 Foundations in Aggregation . . . 24

2.6 Measures for Validation . . . 26

3 Comprehensive Service Matching 29
3.1 Scientific Contributions . . . 31
3.2 Example Scenario . . . 31
3.3 Requirements . . . 34
3.4 Matching Process Models . . . 35
3.5 Overview of the MatchBox Workflow . . . 38
3.6 Phase 1: Setup . . . 39
3.6.1 Integration of Matchers . . . 41
3.6.2 Integration of Aggregation Strategies . . . 44
3.7 Phase 2: Process Configuration . . . 47
3.7.1 a) Manual Process Configuration . . . 47
3.7.2 b) Process Generation . . . 49
3.8 Phase 3: Execution . . . 52
3.8.1 Input Assignment . . . 53
3.8.2 Execution . . . 54
3.8.3 Matching Results Inspection . . . 55
3.8.4 Matching Results Validation . . . 57
3.9 Prototype Implementation . . . 57
3.9.1 Architecture and Technologies . . . 57
3.9.2 Main Features . . . 59
3.10 Validation . . . 61
3.10.1 Validation Questions . . . 61
3.10.2 Case Study 1.1: Matching Process Variability . . . 62
3.10.3 Case Study 1.2: Matcher Integration Effort . . . 65
3.10.4 Discussion . . . 69
3.10.5 Satisfaction of the Requirements . . . 70
3.10.6 Threats to Validity . . . 71
3.11 Limitations . . . 72
3.12 Related Work . . . 73
3.12.1 Survey Procedure . . . 73
3.12.2 Comparison of Processes in Service Matching Approaches . . . 74
3.13 Conclusion . . . 80

4 Fuzzy Service Matching 81
4.1 Scientific Contributions . . . 83
4.2 Foundations of Fuzzy Modeling . . . 84
4.2.1 Fuzzy Sets and Fuzzy Logic . . . 84
4.2.2 Possibility Theory . . . 85
4.3 Requirements . . . 86
4.4 Fuzziness Classification . . . 87
4.4.1 Fuzziness Types . . . 87
4.4.2 Fuzziness Sources . . . 90
4.4.3 Fuzziness Occurrences . . . 94
4.5 Fuzzy Matching Procedure . . . 97

4.6 Fuzzy Matching Approaches . . . 99
4.6.1 Fuzzy Matching Based on Fuzzy Logic . . . 100
4.6.2 Fuzziness Quantification as an Extension . . . 116
4.7 Prototype Implementation . . . 121
4.7.1 Architecture and Technologies . . . 122
4.7.2 Main Features . . . 122
4.8 Validation . . . 123
4.8.1 Validation Questions . . . 124
4.8.2 Case Study 2.1: Evaluation of Fuzzy Reputation Matching based on Fuzzy Logic . . . 124
4.8.3 Case Study 2.2: Evaluation of Fuzziness Quantification . . . 131
4.8.4 Discussion Summary . . . 136
4.8.5 Satisfaction of the Requirements . . . 136
4.8.6 Threats to Validity . . . 137
4.9 Limitations . . . 138
4.10 Related Work . . . 139
4.10.1 Survey Procedure . . . 139
4.10.2 Comparison of Fuzzy Service Matching Approaches . . . 139
4.10.3 Fuzziness in Related Software Engineering Disciplines . . . 146
4.11 Conclusions . . . 148

5 Integrated and Comprehensive Fuzzy Matching Processes 149
5.1 Scientific Contributions . . . 149
5.2 Combining Comprehensive and Fuzzy Matching . . . 150
5.2.1 Requirements . . . 150
5.2.2 Framework Extensions for Fuzzy Matching . . . 150
5.2.3 Aggregating Fuzzy Matching Results . . . 152
5.2.4 Coping with Imperfect Information by Self-Adaption . . . 153
5.2.5 Prototype Implementation . . . 154
5.3 Back-Transformation of Matching Results . . . 155
5.3.1 Requirements . . . 157
5.3.2 Step 1: Transformation into a Matcher's Input . . . 158
5.3.3 Step 2: Matching Process Generation and Enhanced Matching . . . 162
5.3.4 Step 3: Back-Transformation and Representation . . . 163
5.3.5 Prototype Implementation . . . 164
5.3.6 Limitations . . . 164
5.3.7 Related Work . . . 165
5.4 Integrating a Matcher into a Service Market Architecture . . . 167
5.4.1 Requirements . . . 168
5.4.2 Integrating a Matcher based on Architectural Tactics . . . 173
5.4.3 Exemplary Application: Service Matchers in OTF Markets . . . 178
5.4.4 Related Work . . . 182
5.5 Conclusions . . . 183

6 Conclusion 185
6.1 Results and Conclusions . . . 185

6.2 Future Research Challenges 186

Bibliography **189**
Own Publications 189
Supervised Theses 193
Literature 194
Referenced Websites 224

List of Figures **227**

List of Tables **231**

A Metamodels **233**
A.1 MatchBox Main Metamodels 233
A.2 Matching Process Generator Metamodels 235
A.3 Service Specification Metamodels 236
A.3.1 Privacy Specification 237
A.3.2 Reputation Specification 238

B Integrated Matchers and Aggregation Strategies **241**
B.1 Matcher Definitions 241
B.2 Aggregation Strategy Definitions 241

C Literature Surveys **245**
C.1 Keywords 245
C.2 Exclusion Criteria 247

D Evaluation Data **249**
D.1 Validation of Matching Processes 249
D.2 Validation of Fuzzy Matching 249
D.3 Validation of Market Integration 249

1

INTRODUCTION

In the last decades, software development has turned from monolithic software products towards more flexible, reusable solutions. Especially paradigms like component-based software engineering (CBSE) or service-oriented computing (SOC) reflect this development. CBSE deals with independently composable software components accessed via well-defined interfaces [SGM02]. SOC builds on such software components and addresses their provision as readily deployed and flexibly composable Software-as-a-Service (SaaS) [PTDL08]. Examples for such software services are map services (e.g., Google Maps [Gooa]), hotel reservation services (e.g., HRS [Ser]), image processing services (e.g., Instagram [Ins]), or services related to university management tasks (e.g., PAUL [UPBa]).

In the last years, *service providers* started offering their software components and services in world-wide software (service) markets. One example is plug-in marketplaces for software platforms (e.g., Eclipse Marketplace [Ecl]). Especially the number and size of mobile app markets (e.g., Google Play [Goob], Windows Phone Store [Mica]) as well as markets for web services and cloud services (e.g., Amazon Web Services [Amab], SalesForce [Sal], IBM Cloud Marketplace [IBMCM]) is rapidly increasing [SO11].

As a consequence, various kinds of customers can buy software services from such markets. A customer querying such a market takes the role of a *service requester*. Service requesters usually have very specific requirements regarding the software they are interested in. These requirements can be functional (e.g., "the service should be able to manage exam registrations") or non-functional (e.g., "the service should be fast" or "the service should adhere to my privacy preferences"). Thus, among all services provided in a market, the service that satisfies these requirements best needs to be discovered [NK13, RKLB09] or to be composed from several constituent services [DS05].

1.1 Service Matching

Both, service discovery and service composition are based on *service matching* approaches. Figure 1.1 illustrates the roles and artifacts relevant for service matching approaches: The requester provides a Requirements Specification, while the provider offers a Service Specification of the service(s) she provides. The matching approach takes both specifications as inputs and returns a Matching Result as an output. The matching result indicates the extent of in how far the specification of the provided service satisfies the given requirements specification [PvdH03]. This procedure is repeated for all services and all providers in the market, such that we get one matching result per service. These results are comparable and can be used in order to determine which service is the most relevant one for the requester.

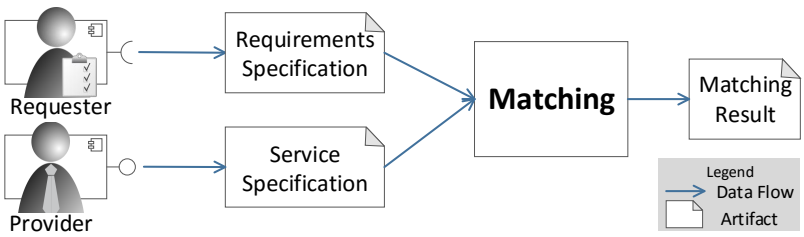


Figure 1.1: The Most Important Roles and Artifacts Involved in Service Matching

Note that a requester does not need to be human; it can also be the required interface of a software component that is to be composed to the discovered component.

While most established software markets today are still limited to a relatively simple, keyword-based search, there exists a variety of advanced matching approaches in academia. These approaches consider more complex specifications describing various requirements and service properties [DHC12, PvDB+13]. For example, there are approaches for signature matching (e.g., [ZW97, SW05]), protocol matching (e.g., [CIW99, MXB10]), or for matching quality-of-service (QoS) properties (e.g., [BBM10]). All these approaches have in common that they try to detect (and quantify) *mismatches*, i.e., situations where the provided service specification does not fully satisfy the given requirements specification.

Service matching approaches trace back to approaches for component search (component retrieval) [LPDA04] from the research area of component-based software engineering. The main difference between matching specifications for software components and matching service specifications is that service matching approaches are able to incorporate more concrete quality properties because services can be viewed as deployed components [BPB15]. For example, properties like performance depend on the concrete hardware a component is deployed on. In contrast, there is no difference in component and service matching when considering structural properties (e.g., signatures) or behavioral properties (e.g., protocols). As a consequence, the concepts described in this thesis are targeted for service matching but most of them can also be applied in the context of (undeployed) software components or similar software entities (e.g., plug-ins, apps), as long as they are described by (semi-)formal specifications.

1.2 Running Example

In the following, we describe the domain of university management services that is used as a running example throughout this thesis. A university management service is a software service used to simplify tasks related to managing a university, including its teaching activities. Figure 1.2 shows an example service. This example service is composed from three services: a Course Manager, an Exam Manager, and a Room Manager. The Room Manager provides the functionality to reserve lecture halls and seminar rooms. Both the Course Manager and the Exam Manager depend on the Room Manager, in order to book rooms for course meetings or for exams.

Figure 1.3 shows a requester's requirements specification for such a room management service, as well as an exemplary specification of the Room Manager service from Figure 1.2. Please note that these specifications are simplified for illustration purposes.

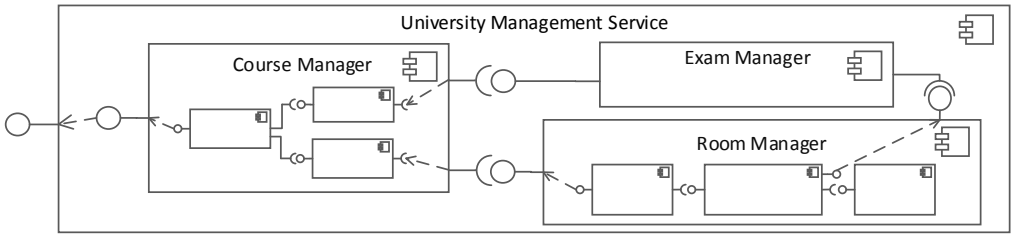


Figure 1.2: University Management Example Services

As explained above, requirements can be complex and heterogeneous. The requirements depicted in Figure 1.3 (a) are related to several *specification aspects*: signatures, pre- and post-conditions, protocols, privacy properties, pricing, reputation, and performance. We explain the depicted requirements specification briefly in the following.

At the top of Figure 1.3 (a), there are two operations: bookRoom and confirm. According to the signatures, bookRoom takes four input parameters (bookerName, address, time, and capacity) of the data types String, MailAddress, Date, and Integer. Furthermore, bookRoom returns an object room of type Room. All data types refer to concepts from an underlying ontology depicted in Figure 1.3 (c). This ontology captures knowledge from a specific domain; in this example, knowledge from the university management domain.

Apart from signatures, further functional requirements are specified with pre- and post-conditions and protocols. The predicates used in the depicted pre- and post-conditions refer to properties from the underlying ontology in order to specify the semantics of the described service. The precondition shows that the requested service operates only if there is at least one Room with the given capacity that is not booked at the given time. In contrast, as specified in the post-condition, the returned room is to be booked at the given time. As specified in the protocol, the two operations bookRoom and confirm are to be called one after another.

Furthermore, there are several non-functional requirements. For example, the requester wants a service that costs at most 0.01 Euros per invocation. In addition, the service should be rated with an average reputation value of at least about four stars (following a five star range as known from today's app stores). The privacy requirements define that the given input data is not allowed to be delegated to other services and that this data may only be stored for 12 months by the current service at most. Referring to the performance of the required service, the requester specifies the required response time as fast.

In this example, the specification of the provided service looks fairly similar to the requirements specification (see Figure 1.3 (b)). In general, the provided service Room Manager provides the requested room reservation functionality. In addition, it provides an operation editBookingTime for editing a reservation. We assume that the requesters and the providers follow the same ontology for the specification of requirements and the provided services. For this, we refer to ontology mapping approaches available in the literature (see [CSH06] for an overview) that can be applied to take care of an appropriate transformation. The non-functional properties vary slightly from the non-functional requirements. For example, the reputation of the service is 3.88 stars, which is an aggregate of the ratings previous users assigned to this service regarding their own satisfaction with the service.

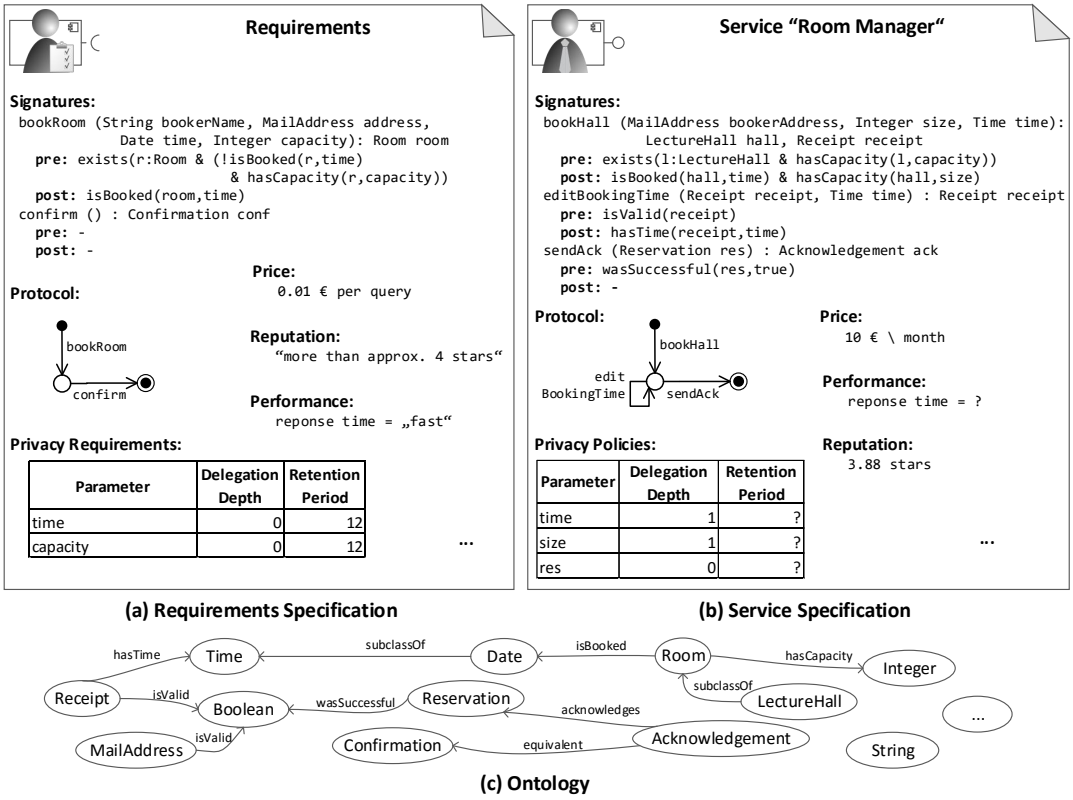


Figure 1.3: Example Requirements, Service Specification, and Ontology

In order to find out whether the provided service Room Manager satisfies the given requirements, matching approaches are applied to the two specifications. For example, standard ontological signature matching approaches (e.g., [KK12b, SW05]) would determine that the signature of the provided operation bookHall matches the signature of the required operation bookRoom quite well, while the signature of the provided operation sendAck partially matches the signature of the required operation confirm. These matching results are based on the input and output parameters and the relations between the ontological concepts their types refer to. For example, the ontological concept LectureHall is a subtype of Room. Other matching approaches apply different calculations to determine a matching result. For example, pre- and post-conditions are often evaluated using SMT-solving [WHdM09]. On the contrary, quality properties are often transformed into fuzzy sets [Zad65] in order to determine a degree of match.

1.3 Problem Statement

Taking a more detailed look at the example described in Section 1.2, two main challenges can be discovered: The complexity of the given input specifications and the presence of imperfect information, as discussed in the following.

C1: Dealing with Complex Input Specifications

Since requesters have diverse requirements related to various service properties, the matching needs to cover all these requirements and properties. Service providers can be assumed to be interested in providing comprehensive specifications of their services in order to improve their sales opportunities. Furthermore, there are methods to derive further parts of service specifications, for example, behavior protocols [BIPT09]. Consequently, matching approaches need to handle complex input specifications in order to compute accurate matching results.

The input specifications from Figure 1.3 are complex in several ways. For example, there are many different specification aspects that need to be considered. However, current matching approaches cover only one or few aspects each [PvDB+13, DHC12]. Thus, the matching results those approaches deliver are inaccurate in a sense that multiple requirements are ignored, which can lead to *false positives* (mismatching services incorrectly determined as well-matching services).

One naive solution is to use several matching approaches one after another. However, this solution raises several problems, too. For example, we do not get one matching result but several results leading to a more complicated decision-making for the users (requester or provider). Here, the question is how to aggregate matching results computed by single matching approaches to one final matching result. Also, some of these matching approaches may depend on each other, such that the correct data flow and control flow matters. All these design decisions have an impact on the overall performance and accuracy of matching. Accordingly, reconfigurations of the combined matching implementations are needed frequently. Thus, combining matching approaches is a complex task and doing this manually is tedious and error-prone. We need a flexible solution that simplifies this task.

C2: Dealing with Imperfect Information

In current service markets, specifications are very often lacking, incomplete, or unclear [HS07, SO11]. The more complex the specifications we are dealing with, the more the probability for such imperfect specifications increases. In addition, requirements specifications as well as service specifications are often imperfect because great parts of them are typically created by humans. For example, requesters often have vague requirements or deliver imprecise requirements specifications. This is the case in the requirements specification in Figure 1.3 (a) for reputation ("approx. 4") and for response time ("fast").

In addition, providers often provide incomplete service specifications. For example, in the depicted specification of the Room Manager service, the provider specified neither information about the response time nor about retention time. One reason could be that the provider does not willingly publish all details about her service in order to protect business interests. Alternatively, she may not know all details of her service. For example, if her service depends on an imprecisely specified third-party server, it is difficult to provide reliable information about quality properties such as performance.

Another reason for imperfect information in general is that requesters and providers often use different specification languages. Thus, transformations into a common language are needed in order to make the specifications comparable. However, depending on the expressiveness of this common language, the transformations can be lossy such that the target

specification becomes incomplete or imprecise, even though the source specification might be perfect.

In all these cases, the matching result becomes uncertain because information needed to determine a precise matching result is missing. However, current matching approaches cannot cope with such fuzziness or they ignore it. As a consequence, they deliver adulterated matching results that do not notify the users about the induced fuzziness, or they even produce false positives or false negatives.

1.4 Overview of the Solution

We propose an approach for *Fuzzy Matching of Comprehensive Service Specifications* in order to tackle the two challenges discussed above. Figure 1.4 shows different working packages this approach consists of. In this thesis, we focus on three working packages: Comprehensive Matching, Fuzzy Matching, and Integrated and Comprehensive Fuzzy Matching Processes. These working packages are based on Comprehensive Service Specifications in order to be able to consider many different requirements and service properties, which is essential to appropriate service matching as explained above. The working package Comprehensive Service Specifications including concepts for model transformations for such specifications has been covered in the thesis by Arifulina [Ari].

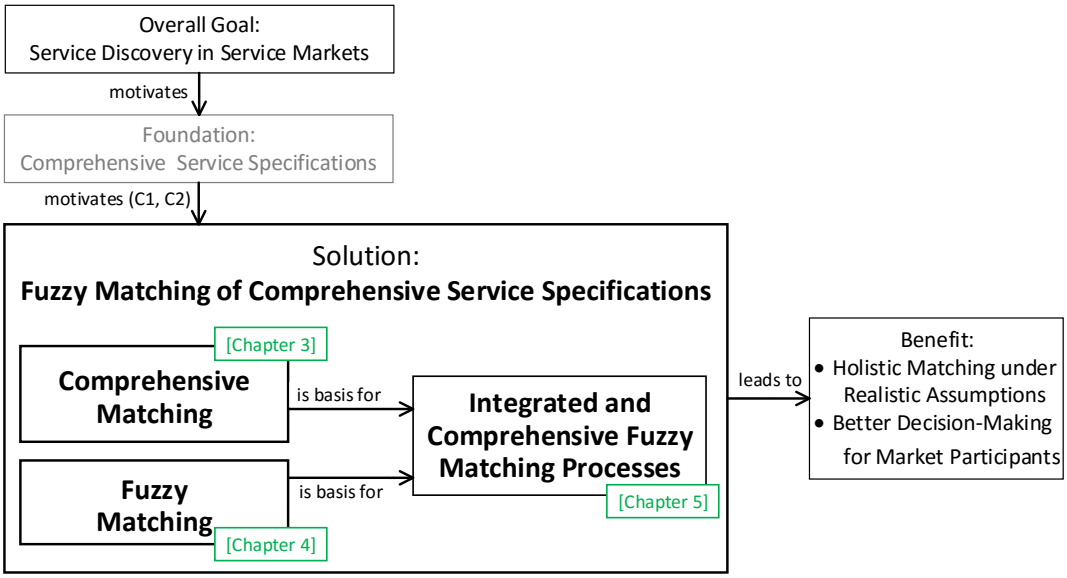


Figure 1.4: Overview of the Solution

In order to deal with comprehensive and, therefore, complex service specifications (C1), we propose the idea of Comprehensive Matching Processes in Chapter 3. Such processes allow for both the integration and combination of multiple existing matching approaches in order to compute aggregated matching results that consider many different kinds of requirements. For this purpose, a model-driven development approach and framework for comprehensive and configurable service matching, called *MatchBox*, is introduced. This approach creates and processes matching process models that consist of several *matching*

steps and work on a well-defined description of matching components. Using MatchBox, not only the configuration of single matching components, but also control flow and data flow can be created, adapted, and validated on model level. After creating the matching process models, they can be processed on given input specifications fully automatically, while still maintaining modularization to handle the complexity of huge matching problems. MatchBox, thereby, simplifies, validates, and partially automatizes complex integration tasks that were required to be done completely manually before.

However, as discussed above, the more complex service specifications become, the more the risk of imperfect information within these specifications and, as a consequence, the uncertainty during matching increases (C2). In order to cope with this uncertainty, we propose concepts for Fuzzy Matching in Chapter 4. Fuzzy Matching first detects (potential) sources for uncertainty within the input specifications and classifies them on the basis of well-defined *fuzziness sources* and *fuzziness types*. Furthermore, it quantifies the amount of induced fuzziness. For this purpose, we developed concepts based on so-called fuzziness scores and on fuzzy logic and possibility theory. Finally, the classified and quantified fuzziness is returned as part of a comprehensive matching result that reflects the induced uncertainty to the user. Thereby, fuzzy matching provides valuable information about the quality of the matching result, which improves the decision-making of both service requesters and service providers. In this thesis, we explain our fuzzy matching approaches focusing on matching non-functional service properties like reputation, privacy, and performance.

In order to enable fuzzy matching of multiple aspects and in order to deal with imperfect information within matching processes, the concepts of matching processes and fuzzy matching are brought together and developed further into Integrated and Comprehensive Fuzzy Matching Processes in Chapter 5. Here, the advantages of both fuzzy matching and comprehensive matching processes are combined and their integration into service market infrastructures is discussed.

All in all, most existing matching approaches never made it to practice. One of the reasons for this fact is that they are based on unrealistic assumptions. In contrast to these approaches, this thesis provides concepts to integrate comprehensive matching approaches that can cope with imperfect information. As a benefit, service markets are provided with better matching approaches that are applicable under realistic assumptions and support the decision-making of market participants (e.g., service requesters and providers) by providing extra knowledge. Thereby, the concepts developed in this thesis go significantly beyond the state of the art in service matching.

The thesis contains multiple interdisciplinary works where several research areas are combined on a methodological level: In MatchBox (Chapter 3), service matching meets software architecture as well as concepts from component-based and model-driven software engineering. For fuzzy matching (Chapter 4), we married service matching concepts with foundations from theoretical areas like fuzzy logic as well as concepts from psychological areas like decision theory and uncertainty theory. In Chapter 5, also elements from economical research are added for investigating transactions in service markets.

1.5 Application Scenarios

The fact that the concepts presented in this thesis have been developed with a focus on flexibility and low effort makes them broadly applicable. In the following, we discuss four example application scenarios.

1.5.1 On-The-Fly Computing

On-The-Fly (OTF) Computing [SFB901] addresses the provision of software and infrastructure services in heterogeneous, world-wide service markets (so-called OTF markets). In OTF Computing, we assume that a customer's requirements are so complex that only well-evaluated service compositions constructed by intermediaries with special domain knowledge can satisfy them.

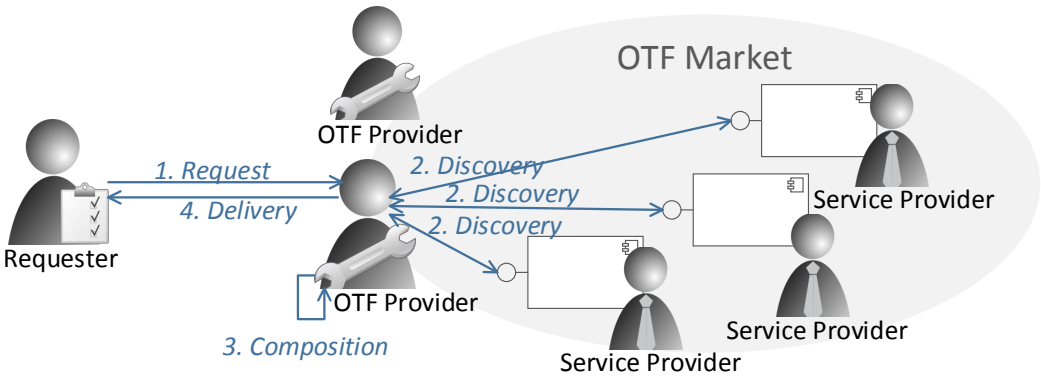


Figure 1.5: OTF Service Market and Interaction Between its Roles

Figure 1.5 shows how the most important roles interact in an OTF market: The Requester starts by sending a request. The request contains all requirements the requester has for the service she wants to buy. As the requester is not able to identify and contact all relevant service providers that provide a service that satisfies these requirements, she sends the request to an OTF Provider. An OTF Provider serves as a broker and intermediary between the requester and the Service Providers. The OTF Provider is a domain expert and in charge of discovering all services she needs in order to compose and deploy them in a way that they collectively satisfy the requester's request.

The special need for matching processes and fuzzy matching in OTF Computing emerges due to the heterogeneity of the OTF markets. First of all, matching processes are more flexible and adaptable than state-of-the-art solutions. Thus, they can be optimized to different instances (e.g., different domains) of such a market. In order to provide appropriate matching for all these different markets, fuzzy matching is important as we also need to handle markets with special properties, e.g., a high number of provided services, non-standardized domain terminology, and inexperienced users.

1.5.2 Mobile Apps

As already mentioned, today's mobile app stores have constantly been growing over the years [BK12]. As an example, Google's Play Store was placed in 2015 at 1.5 million

apps [Sta, AF]. Thus, proper discovery mechanisms for apps are becoming more and more important. Furthermore, Android apps also provide means to support composition. As a consequence, matching concepts for app specifications are required.

At the moment, mobile apps are specified in two ways: Firstly, they are described by the specification published in the app market, consisting of structured text about general and technical features (e.g., provider, publishing date, rating, operating system requirements, privacy permissions) as well as some free text describing the functionality. Secondly, a machine-readable specification (e.g., in case of Android apps, the Manifest file) is needed for deployment.

Both specifications can be used in a limited way for the service matching approaches described in this thesis. For example, a matching process including matching components for reputation, privacy, and technical deployment information could be applied. Fuzzy matching makes sense where incomplete or imprecise information is given. This is often the case for service descriptions based on natural language, for example. Also, the requirements are given by various kinds of end users with varying background knowledge, which means that the probability for fuzzy requirements increases.

1.5.3 Business-to-Business Applications

While service matching in On-The-Fly Computing and mobile apps addresses end users to a great extent, an application to the collaboration between two companies (Business-to-Business, B2B) may be even more realistic. In such scenarios, services are especially useful in general if continuously service-oriented application landscapes are applied [EHH$^+$]. Specifications of B2B services as well as corresponding requirements specifications can be expected to be more complex as they are created by experts. In this case, fuzzy matching is essential, as there is even more space for incompleteness and imprecision if more service aspects are described and more expressive specification languages are used. Furthermore, the need for well-defined and flexible matching processes increases with every added aspect.

1.5.4 Industry 4.0

Industry 4.0 is the movement towards the intelligent interconnectivity between humans, machines, and industrial processes [BMWI]. Within this scope, production engineering meets information and communication technologies with the vision that intelligent machines cooperate with each other autonomously.

As a consequence, the specification and matching of interfaces of these machines is an essential task. In contrast to the examples focused in this thesis, not only the software on application level needs to be matched, but also on the platform and infrastructure level [DMES14]. Thus, a matching process applied to this scenario needs to focus on matching steps related to such features with a close relation to hardware. But still, matching steps related to quality properties such as performance and privacy or security properties are useful as especially real-time properties as well as privacy and security issues need to be considered in industrial processes, too. Also, fuzzy matching becomes important because it can often be processed more efficiently than accurate matching approaches. This is important because matching within industrial processes often has to be performed at runtime in order to enable self-adaption.

1.6 Structure of this Thesis

This thesis is structured into six chapters. Chapter 2 describes foundations for this thesis, e.g., the state of the art within service specification and service matching. Chapter 3 details on comprehensive service matching processes. Our work on fuzzy matching is described in Chapter 4. Chapter 5 focuses on integrated and comprehensive fuzzy matching processes. Last, Chapter 6 concludes the thesis and gives an outlook on possible future work. The implementation, validation, and scientific contributions are discussed as part of the main chapters of this thesis.

The appendices contain additional information. Appendix A shows meta models used within MatchBox. Appendix B lists definitions of matching components integrated into MatchBox. Appendix C provides additional information on literature surveys conducted within the scope of this thesis. Finally, Appendix D gives additional information used for the validation of our concepts.

2

FOUNDATIONS

In this chapter, the foundations required for the concepts presented in the following chapters are illustrated. This includes a basic introduction to services and software components in general. Furthermore, we discuss on the basis of our running example introduced in Section 1.2 how services are specified and matched.

2.1 Software Components and Services

A *service* is a software component that is deployed and running on a service provider's platform. One example for a service is Google Maps. Google Maps is provided by Google and offers the functionality of querying and showing a map of a certain location to a user.

Figure 2.1 shows a simplified view of the constituent parts of a Service as a UML class diagram: An undeployed service is a (Software) Component that has an arbitrary number of Interfaces. An interface contains at least one Operation. Quality information (Quality-of-Service, QoS) are most often specified with respect to the deployed service and not to the component because special context information (e.g., usage profiles, hardware requirements) have to be taken into account [BPB15]. A service can be deployed on various kinds of infrastructures, for example, "in the cloud".

Another important difference between service discovery and the selection of (commercial-off-the-shelf) components is how they are handled afterward: Commercial-off-the-shelf component are usually adapted to a system's needs in the sense of code and interface modifications, after they have been selected [MRE07]. This possibility is not provided when dealing with services because, in this case, usually the service's implementation is not revealed. Instead, adapters in the sense of additional components or connectors delegating to the adapted service [GHJV95] can be generated (e.g., [IT13, IMTA05, GMS12]).

2.2 Service Markets

A *service market* allows trading, i.e., buying and selling services. There are not many established markets for services in the sense of readily deployed software components till date although paradigms like Service-Oriented Architectures (SOA) and Service-Oriented Computing (SOC) have been investigated for several years now. The service registry standard UDDI that was popular in the area of web services has been officially discontinued in 2006 [Jos07]. Nevertheless, along with the emergence of a number of cloud providers, multiple platforms to obtain web services for usage in the cloud appeared, e.g., Amazon Web Services [Amab]. Furthermore, there are markets for software products similar to

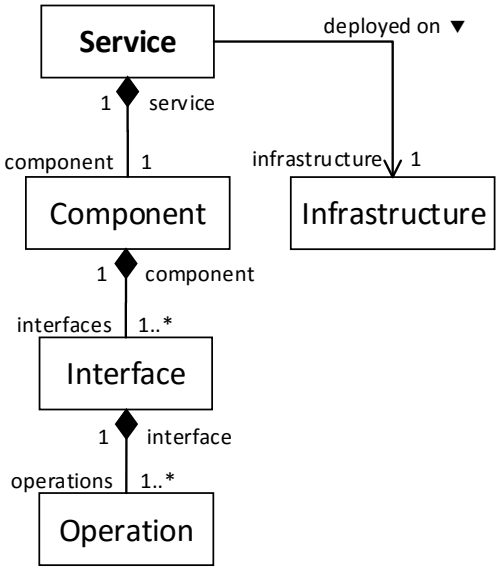


Figure 2.1: Parts of a Service

services, like software components in the form of plug-ins or apps. For example, according to Schlauderer and Overhage [SO11], StrikeIron [SI], Salesforce's AppExchange [Sal], and Google's Apps Marketplace (now Google Play [Goob]) represented the leading markets in 2010. A well-established market for software plug-ins is the Eclipse Marketplace [Ecl]. Some years later, we can now also add many more app markets to the list, e.g., Apple's AppStore [App], Amazon's App Shop [Amaa], and Microsoft's Windows Store [Micb]. In the remainder of this section, we summarize the common traits among these markets and describe what can be expected to become the future trend for service markets. [PBS14]

Chapter 1 already introduced service requesters and service providers as main roles in a service market. Furthermore, there can be third parties, e.g., the broker (see Section 1.5.1), or the market operator, who provides and manages the market [SO11]. Each of these roles can be responsible for the service discovery [W3C]. One actor can play more than one role within a market. For example, there can also be intermediaries that act as both requesters and providers. This is usually the case when a requester buys a service to compose it with other (third-party of self-developed) services using service composition techniques and then provides the resulting, more complex service to the market again. In general, service providers make their service offers available to requesters by publishing service specifications that enable discovering their services. Such service specifications can be stored in different locations, e.g., directly on a server provided by the market operator or distributively by the service providers. [PBS14]

For us, the most important component in a service market is the *matcher*, which is a matching component implementing service matching approaches required for service discovery. In addition, there can also be further components. Examples are: a service composition engine, a service certification component, service analysis components, a reputation system, escrow services, or monitoring systems [SO11, SFB901, PBS14]

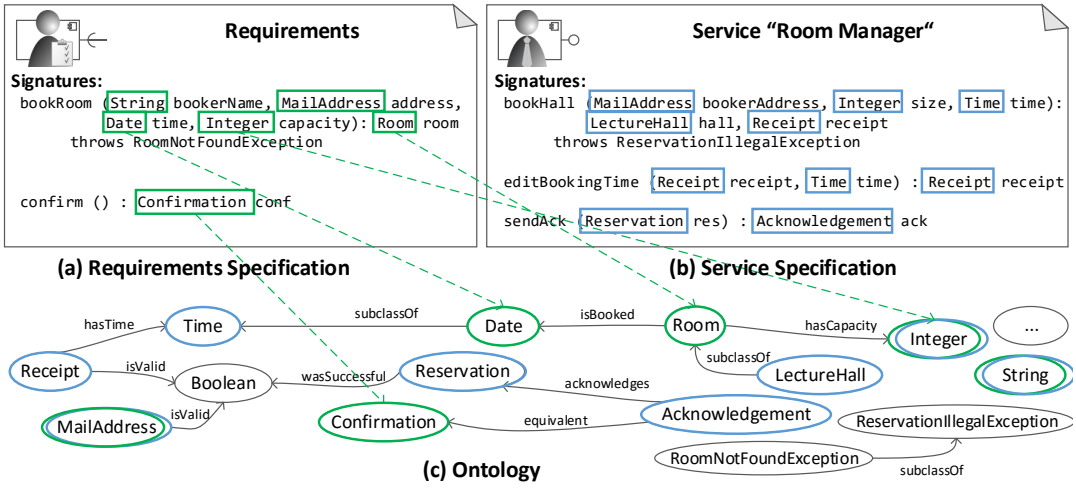


Figure 2.2: Signature Specification Example

2.3 Service Specification

In this thesis, we build on comprehensive service specifications comprising a collection of service specification parts that we call *service specification aspects* [Ari]. Each aspect describes a service property or a set of related service properties, e.g., signatures, protocols, or privacy-related properties. This section illustrates some exemplary service specification parts that will reappear in the later sections as a basis for the descriptions of different matching approaches.

2.3.1 Signatures

The *Signatures* aspect consists of the following language constructs: operation name, input and output parameters including their types and names, and exception types. Figure 2.2 shows the signatures and the ontology from our running example presented in Figure 1.3. The signatures follow the principles of signatures known from object-oriented programming languages like Java. The metamodel we use to specify these signatures and further examples can be found in our technical report [PJvR+16].

Figure 2.2 highlights the references from the utilized parameter types to an underlying domain ontology using colored rectangles in green (requirements specification) and in blue (specification of the provided service) and correspondingly colored ovals for the concepts that are part of the ontology. In addition, also a signature's exception types can refer to ontological concepts (e.g., ReservationIllegalException and RoomNotFoundException. An ontology defines concepts and relations in a specific application domain. Thereby, it captures domain knowledge that can be leveraged during specification and matching of services in this domain. We can assume that requesters and providers follow the same ontology because ontology mapping approaches available in literature (e.g., [CSH06]) can be applied to translate between different ontologies.

The references to the ontology assign more meaning to a data type than just its name is able to provide. This extra information is especially important for matching, as described in Section 2.4.1.

Ontologies can be specified in various languages. The specifications presented in this thesis refer to ontologies defined using the Web Ontology Language, OWL2 [GHM+08].

2.3.2 Behavioral Specifications

In order to specify a service's required or offered behavior, usually, pre- and post-conditions are defined (e.g., [HGEJ12a]). Our example language for the specification of pre- and post-conditions is based on first-order logic and extended with ontological references [PJvR+16]. As an example consider the precondition of the required bookRoom operation introduced in Figure 1.3:

$$exists(r : Room \quad \& \quad (!isBooked(r, time) \quad \& \quad hasCapacity(r, capacity)) \tag{2.1}$$

Here, the variables time and capacity refer to the parameters Date time and Integer capacity defined as input parameters within the Signature of bookRoom. Based on these references, this precondition specifies that, before the service is executed, there has to exist an object of type Room that is not booked at the given time and that has the given capacity. The predicates isBooked and hasCapacity are defined in the ontology as object properties referring to the concepts that represent the data types. As we can learn from this example, conditions allow us to specify the semantics of an operation much better than the signature alone.

Protocols define allowed operation call sequences for the interaction with a service. They are often specified using automata. For example, the protocol depicted in Figure 1.3 a) defines that first bookRoom has to be invoked and then confirm can be invoked.

As we can see from these examples, both pre- and post-conditions as well as protocols refer to one or more operation signatures. In this way, different aspects are connected with each other [Ari].

More information about our example languages for conditions and protocols can be found in [PJvR+16].

2.3.3 Privacy

These days, privacy preservation becomes more and more important, especially in global software markets [PJP+14]. Thus, we need approaches to describe a user's preferences as well as a service's properties related to privacy handling.

The privacy specification language used within this thesis has been constructed by combining several existing specification languages: Most parts are based on the privacy policy model by Costante et al. [CPZ13a, CPZ13b], while further elements stem from the approaches presented by Kapitsaki [Kap13] and Tbahriti et al. [TMG+13]. We adapted, united, and extended these approaches for our purposes. In particular, we connected the privacy specification to the above described signature specification (see Section 2.3.1) [PAPS15].

Figure 2.3 a) depicts the privacy preferences, i.e., requirements related to a services's privacy policies, with the requester's requirements specification. The privacy preferences are

Requirements

Signatures:

```
  bookRoom (String bookerName, MailAddress address, Date time, Integer capacity): Room room
...
```

Privacy Requirements:

Parameter	Purpose	Delegation Depth	Retention Period	Visibility	Location Limit	Sensitivity
bookerName	RoomBooking	1	12	-	Europe	High
address	Contact	0	12	-	Europe	Very High
time	RoomBooking	1	48	Notification Services	Europe	Medium
capacity	RoomBooking	1	∞	-	Europe	Low
room	Confirmation	1	48	Notification Services	Europe	Medium

(a) Requirements Specification

Service "Room Manager"

Signatures:

```
  bookHall (String bookerAddress, Time time): LectureHall hall, Receipt receipt
...
```

Privacy Policies:

Parameter	Purpose	Delegation Depth	Retention Period	Visibility	Location Limit
bookerAddress	Contact	1	24	-	Germany
time	RoomBooking	0	24	-	Germany
hall	Confirmation	1	24	-	Germany
receipt	Confirmation	1	24	-	Germany

(b) Service Specification

Figure 2.3: Example Specifications of Privacy Preferences and Properties

visualized in a tabular notation. Each row represents one privacy condition. Each privacy condition refers to a parameter from a requested operation's signature and specifies what kind of usage regarding the data corresponding to this parameter the requester accepts. Thus, the depicted table contains five rows with privacy conditions, one for each parameter from the operation bookRoom: bookerName, address, time, capacity, and room. The columns represent the different restrictions that can be specified regarding each parameter [PAPS15]:

Purpose defines the reason for the collection of the corresponding data and its usage [CPZ13a]. For example, the requirements for the depicted parameters allow to use the name (bookerName) of the service's user, i.e., of the reserving person, to be used only as part of the RoomBooking process. In contrast, the email address (address) of the reserving person may only be used in order to contact her. The requested time of the reservation and the capacity of the room to be reserved may be used within the scope of RoomBooking. The output room may be used for Confirmation purposes. All terms used as purpose have to be defined in an ontology. A parameter may also appear in several rows with different purposes.

Delegation Depth refers to the amount of levels (i.e., services) a parameter may be forwarded to [PAPS15]. This becomes relevant if a service is used as part of a composition. The privacy requirements for the parameters bookerName, time, capacity, and room are

rather strict with respect to delegation as they allow only one level delegation of the given data. The address is not even allowed to be delegated at all. The default range for delegation depth is [0,10] [CPZ13a]. Additionally, the values undefined and infinity are supported.

Retention Period defines for how long (here: how many months) the service will store the given data [PAPS15]. The more privacy-critical a parameter is, the more critical it is to store. However, storage may be necessary in order to perform certain purposes. In our example, the RoomManager (see Figure 2.3 b)) stores the address for 24 months. This is due to the provided functionality for the room management staff to contact people that booked rooms within the last two years. For retention period, we support values in [0,100], as suggested in [CPZ13a], as well as undefined and infinity.

Visibility restricts a policy to a set of service providers, service categories, or even specific services. For example, one can allow certain data to be visible only to services provided by Google, only to notification services, or only to the Google Alert Service. In our example, the requester allows the parameters time and room to be visible for all services of kind Notification Service.

Location Limit restricts privacy policies with respect to a location. For example, the depicted requirements specification allows provided and delegated data only to be processed in Europe. The location limit field can contain one or more terms defined in an underlying ontology of locations [PAPS15]. Thus, the granularity of selectable values (e.g., cities vs. countries vs. continents) depends on the ontology used.

Sensitivity defines the importance the requester assigns to each requirement. For example, she could choose between "very low", "low", "medium", "high", "very high", and "mandatory". In our example specification, the privacy requirements for the user's email address (address) are defined as "very high" because it is very sensitive to the user. Similar, the user's name bookerName got a sensitivity of "high". The other data is no personal data and correspondingly less sensitive.

Figure 2.3 b) depicts the privacy policies of the provided service RoomManager as specified by the service provider. The provider specifies the privacy policies with the same concepts as the requester, except for Sensitivity.

Typically, the values specified for different columns correlate with each other. For example, if a parameter is very sensitive, then, most often, its delegation depth and retention period are also rather strict, i.e., close to 0.

Privacy specifications are typically used for personal data, e.g., name, or address. However, in combination with such personal data, also less sensitive data may become critical as they might be used for development of profiles. For example, location and time in combination with a person's name could allow third persons to track down a person's private time schedule. Thus, our approach also covers non-personally-identifiable information.

The corresponding metamodel is depicted and described in Appendix A.3.1.

2.3.4 Reputation

The *reputation* of a service is measured based the experiences of previous users expressed in the form of *ratings*. Such ratings represent the users' satisfaction with a certain service. Thereby, reputation indicates not only a service's popularity but also its trustworthiness

c1: Rep(Service) ≥ 4
c2: Rep_{RT}(Service) ≥ 4
based on 40 ratings
(w = 2)
c3: Rep_{Sec}(Service) ≥ 3
based on 20 ratings
of the last 3 months
c4: Rep(Provider) ≥ 4

a) Requirements Specification

Entity	Context	# Ratings	
Service: RoomManager	Overall	200	Rep(200 ratings) = 4.5
Service: RoomManager	ResponseTime	80	Rep (80 ratings) = 3
Service: RoomManager	Security	40	Rep (40 ratings, 3 months) = 2.75
Service: BookARoomPro	Overall	80	Rep (80 ratings) = 3.35
Service: BookARoomPro	ResponseTime	20	Rep (20 ratings) = 1.5
Service: BookARoomPro	Security	10	Rep (10 ratings, 3 months) = 4.5
...	...	...	...
Provider: UBServices Inc.	Overall	130	Rep (130 ratings) = 3.5

b) Available Ratings & Exemplary Reputation Values

Figure 2.4: Examples for Reputation Specifications

based on human judgment [Sha83]. Technically, we model the reputation of a service as an *aggregation* of a set of ratings. A rating has several properties, including its value (e.g., 4 stars within a 5-star scale), its age (e.g., two months), and its context [WV07] (e.g., a rating about the performance or about the availability of a service). [PSB+]

Usually, ratings are stored within a secure and privacy-preserving reputation system. For example, in our earlier work [JBPP14b], we presented a privacy-preserving reputation system that allows market components (e.g., a matcher) the flexible access of reputation values aggregated using replaceable aggregation functions.

In the following, we explain the specification language we use to describe reputation requirements. This language is based on the master's thesis of Bano [Ban14] and collaborative work with scientists from the areas of economics, security, and machine learning [JBPP14b, JBPP14a]

As an example, consider the requirements specification in Figure 2.4 a). As depicted in this figure, reputation requirements can be modeled as a list of conditions. This example specification consists of four conditions, c1 – c4.

Each of the conditions in the requirements specification refers to properties related to service reputation. For example, c1 requires that the overall reputation of a service is greater or equal to 4 (based on a 5-star scale as it is common in today's app stores). The conditions c2 and c3 refer to context-specific reputation values, i.e., the perceived response time of the service (c2) and the perceived security of a service (c3). As a further restriction to the reputation values requested by c2 and c3, the reputation value must have been aggregated based on at least a specific number of ratings (here: 40 and 20, respectively). Such restrictions are useful as a reputation value's credibility increases with the number of ratings it has been calculated from. In c3, there is also a restriction with respect to time. Here, the reputation value should have been created based on ratings from the last three months. These kinds of restrictions are based on the idea that recent ratings are more relevant than old ones. This especially happens if the rated service has been updated or if the environment of the raters has changed (e.g., the global sensitivity to security increased due to some incident). C4 is about the reputation of the service's provider and requires this reputation value to be at least 4 stars. Furthermore, it is possible to introduce weights per condition (e.g., "(w = 2)" in c2). Using weights, a user can specify that the satisfaction of one condition is more important (here: twice as important) than the satisfaction of the other conditions.

Figure 2.4 b) shows an exemplary extract of the contents of a reputation system in a tabular notation. These contents are used to evaluate the conditions of the requirements specification

explained above. For example, general and context-specific reputation values for the services RoomManager and BookARoomPro are depicted. Furthermore, the reputation system contains the reputation of the service provider UBServices Inc., which is the provider of both services. The third column depicts the total number of ratings that are available for a service. The rightmost column depicts some exemplary reputation values calculated based on these ratings. Note that these are dynamic values which are usually not stored in the reputation system but derived during the matching procedure from the ratings stored in the system based on a selected aggregation operator. There are several possibilities to aggregate ratings to reputation values. The example values depicted in the figure are those that are needed to evaluate the given requirements specification.

The metamodel for the reputation requirements specification and the ratings is described in Appendix A.3.2.

2.4 Service Matching

In the following, we give some basic information on service matching. Note that we distinguish between the conceptual *matching approach* and the implemented *matcher*.

Figure 2.5 shows a typical matching scenario in the form of a sequence diagram. As can be seen there, the foundation is that providers publish service specifications (serviceSpecs). At a later point in time, a requester publishes a requirements specification (reqSpec). The publication of a requirements specification by the requester is often called a *request* and it triggers the following actions. Both specifications are sent to a matcher. Before being able to match the specifications, they are transformed into an appropriate input for the matcher. For example, a more formal description could be derived at this point. As a result, the matcher returns a matchingResult which can be used to select between the most fitting offered services. Before being able to use a selected service, the different parties need to negotiate the usage and its costs. The topic of negotiation is out of the scope of this thesis; however, it is ongoing research by economic scientists [SFB901].

Note that, depending on the concrete market and the concrete application scenario, the interactions shown in the sequence diagram could vary. For example, the interaction with the broker could be dropped or the broker could interact in a different way. For example, the broker could first gather a set of matching results for different services, compare them, and only send a service recommendation to the requester instead of the concrete matching result. In the depicted diagram, the requester compares the matching results herself. Matching results could also be forwarded to the providers in order to deliver early feedback about how well their offers match on an average and reasons for common mismatches. Furthermore, the broker could also take part in the negotiations. All in all, the depicted sequence diagram abstracts from a larger number of services and service providers, from possible repeated requests, as well as from the requester's and provider's negotiations. Also, additional components could be part of the scenario, e.g., a reputation system or a composition engine (see Section 2.2).

A variety of existing matchers and matching approaches is already described and compared in published surveys. For example, Dong et al. [DHC12] enhances an earlier classification of semantic web service matchers presented by Klusch [Klu08]. Their comparison focuses on ontological signature matchers. Similarly, Bellur et al. [BVG08] present a classification

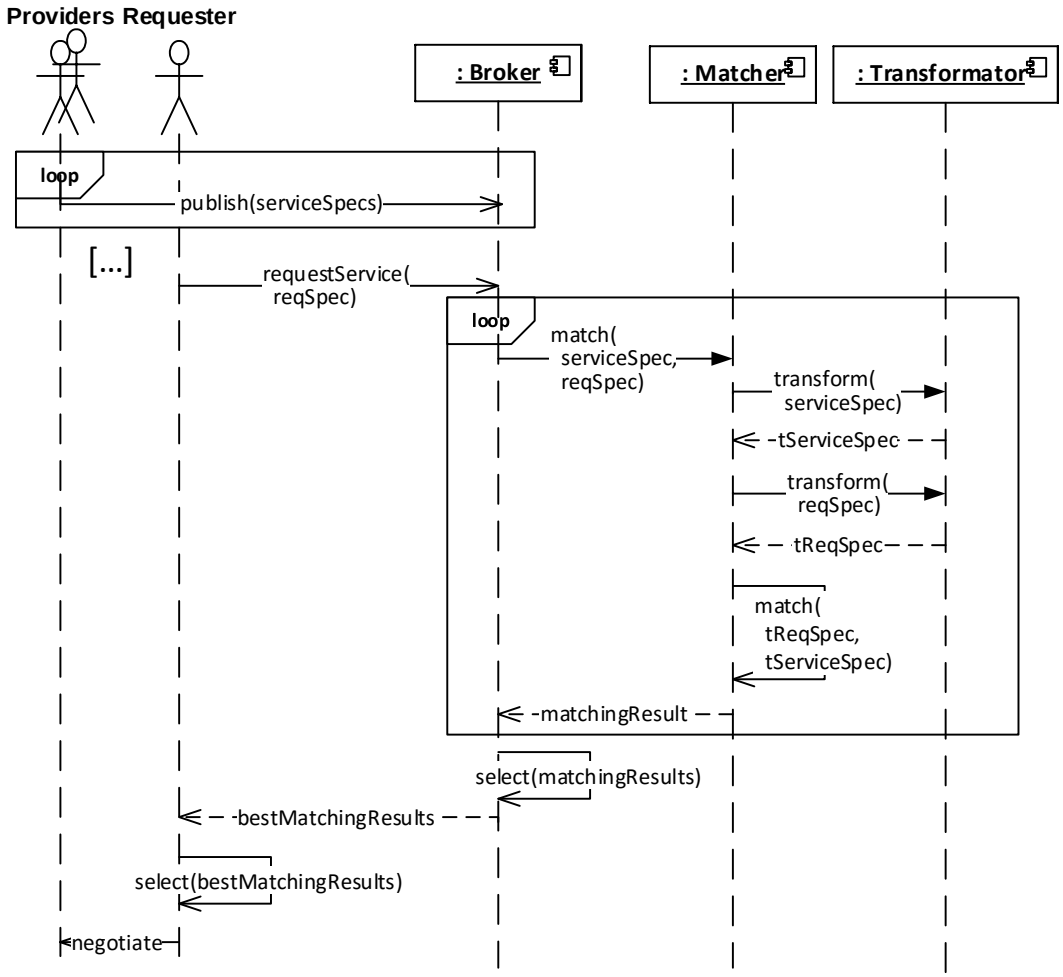


Figure 2.5: Example Matching Scenario as a Sequence Diagram

of algorithms used within semantic web service matchers, not regarding specific matching approaches but the underlying algorithms, e.g., greedy or description logic. The related topic of component retrieval is surveyed in [LPDA04]. Furthermore, service matching is partly related to well-researched information retrieval methods. However, information retrieval does not consider executability [MPMM98].

Next, we describe some exemplary matching approaches that will appear again in later chapters of this thesis as example matchers. Appendix B shows matcher definitions according to the MatchBox framework (see Chapter 3) for the matchers described here. The matching approaches described in the following build on the specification languages that are explained in Section 2.3. From here on, put a higher focus on privacy and reputation matching than on other approaches, in correspondence with the main chapters of the thesis.

2.4.1 Ontological Signature Matching

Operation signatures like the ones discussed in Section 2.3.1 can be matched using various signature matching approaches. The signature matching approach used as an example in this

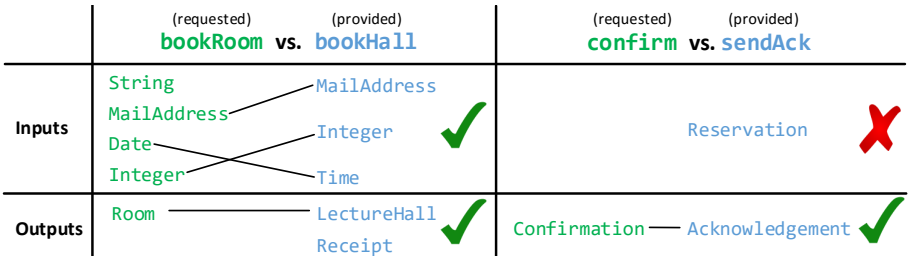


Figure 2.6: Signature Matching Example

thesis is based on an ontological matching of data types and exceptions and a string similarity matching for signature and parameter names.

The matching of input and output parameters serves as a basis for our signature matching. On the one hand, there are usually mandatory input parameters which the service requester has to provide, so that a service provider's service is able to perform the functionality desired by the service requester. On the other hand, the service provider's service has to deliver all outputs requested by the service requester.

Accordingly, the main matching principle for signatures is that two signature specifications match if the following statements hold [APB+15]:

1. The input parameters of the provided service's signature specification are a subset of the input parameters of the requested service's signature specification.
2. The output parameters of the requested service's signature specification are a subset of the output parameters of the provided service's signature specification.

Thus the matcher compares all data types that are part of the signatures pairwise, using a bipartite graph matching; the bipartite graph with the highest score delivers the final matching result. In this process, each data type pair is matched based on the given ontology(-ies) and follows the standard rules of co- and contravariance:

1. An input parameter type in the requested service's signature specification has to be either equal or more specific (e.g., a subclass) than a corresponding input parameter type in the provided service's signature specification.
2. An output parameter type of the provided service's signature specification has to be either equal or more specific (e.g., a subclass) than an output parameter type in the requested service's signature specification.

As an example, have a look at Figure 2.6 that shows how we match the parameter types from the signatures from Figure 2.2. Although each signature from the requirements specification is matched to each signature from the specification of the provided service, here, we focus on the two most promising matching pairs: The requested signature bookRoom is matched to the provided signature bookHall in the left part of the table and the requested signature confirm is matched to the provided signature sendAck in the right part of the table. The upper part of the table shows the matching of input parameter types and the lower part shows the matching of the output parameter types. In particular, the figure shows the graphs that are built up for the bipartite graph matching in the four inner fields: Types from the requested signatures bookRoom and confirm on the left side of each field and the types from the provided signatures bookHall and sendAck on the right side of each field.

The signature of bookHall matches completely the signature of bookRoom because all input types of the provided bookHall operation find a corresponding type within the requested bookRoom operation: MailAddress and MailAddress as well as Integer and Integer match trivially, while Time and Date match because Time is a supertype of Date, as shown in the ontology in Figure 2.2. The outputs match because the requested Room finds a match at the provided side: Here, LectureHall matches to Room because, according to the ontology, LectureHall is a subtype of Room.

The signatures depicted on the right side of the table do not match perfectly: The provided signature sendAck requires an input parameter of type Reservation, however, the requested signature confirm does not expect any input parameter at all. Thus, a mismatch occurs.

The output parameters of sendAck and confirm, however, match. The reason is that, in contrast to pure co- and contravariance as known from object-oriented programming, ontological relations also include the notion of *equivalence*. The reason is that we apply subsumption reasoning as known from description logics [BHS09]. In this way, synonyms can be used. The utilization of synonyms is highly common in service specification because different market participants have a different terminology. For example, in Figure 2.2, Acknowledgement and Confirmation are specified to be equivalent. Since the equivalence relation is symmetric, these two types always match each other.

These type matching concepts also correspond to the *flexible match* introduced by Zaremski and Wing [ZW95]. In addition to the parameters' types, our signature matching approach is also able to compare the parameters' names using cosine similarity as a string similarity metric.

Matching exception types follows the principles of matching parameter types:

1. The requested signature's exceptions have to be either equal or more general (i.e., superclasses) as the provided signature's exceptions.
2. The provided signature's exceptions have to be a subset of the requested signature's exceptions.

Our ontological signature matcher is highly configurable. In particular, the user can define which parts of a signature should be matched. For example, in many domains, parameter names and exception types are less conclusive than parameter types. In such cases, a matcher configured in a way such that only parameter types are matched can lead to better results than other configurations. Configuration possibilities also include to determine that the matcher matches only inputs or only outputs in order to support service composition strategies like forward chaining or backward chaining [DS05, RS05].

2.4.2 Behavioral Matching Approaches

In order to consider a service's behavior during service matching, matching approaches that go beyond structural matching approaches are needed [GCB08, BJPW99]. Such approaches include matching approaches for pre- and post-conditions as well as protocol matching approaches. As an example for specifications based on pre- and post-conditions and protocols, refer to Section 2.3.2.

In condition matching, usually, a full match is defined as follows: The required precondition must be equally strict or stricter than its provided counterpart, whereas

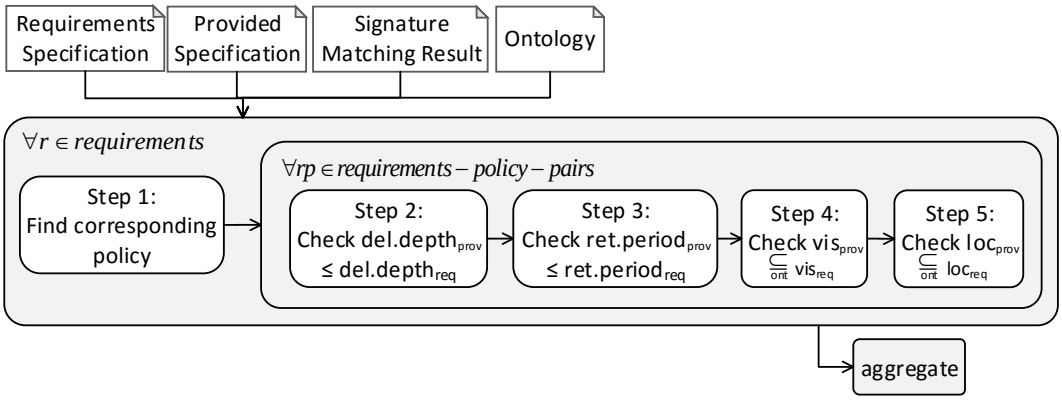


Figure 2.7: Privacy Matching Algorithm (based on [PAPS15])

the provided post-condition must be equally strict or stricter than its required counterpart [Mey88].

Protocol matching often involves model-checking approaches. One simple approach is based on trace-inclusion: Here, the provided protocol matches the required protocol if the provided protocol accepts at least those sequences that are accepted by the the required protocol.

As already noted in Section 2.3.2, pre- and post-conditions as well as protocols are often defined based on signatures. Correspondingly, these matching approaches also required a preceding signature matching approach to deliver initial mappings: Condition matching requires a mapping between the required and the provided parameters, protocol matching requires a mapping between the required and the provided operations.

[PJvR+16] describes more details about behavioral matching approaches defined for our example specification languages.

2.4.3 Privacy Matching

Next, we present a matching approach for specifications of privacy-related service properties. This approach is based on the approaches by Costante et al. [CPZ13a, CPZ13b] and Kapitsaki [Kap13]. It processes specifications defined in the specification language introduced in Section 2.3.3. Our descriptions refer to the example specifications depicted in Figure 2.3.

The privacy policies of a provided service specification match the requester's requirements specification if the provided policies are more strict than the requested ones [PAPS15]. For this purpose, the matching algorithm iterates over the privacy requirements (i.e., the rows of the table) and checks all the columns one after another as depicted in Figure 2.7:

- Step 1: First, the algorithm tries to find a policy in the provider's specification that corresponds to the current requirement. This is done based on the parameter mapping created during a preceding signature matching (see Section 2.4.1). For our running example, we get (amongst others) the following mapping: address–bookerAddress, time–time, capacity–size, room–hall. As the parameter `bookerName` has not been assigned in the given signature matching result, there has to be no corresponding policy

for the requester's `bookerName` requirement. As a consequence, the requirement can never be violated by the provided service because `bookerName` is not taken as an input; thus, this requirement can be ignored. Furthermore, the specified purposes are checked with ontological subsumption reasoning, like the types in signature matching (see Section 2.4.1). The next steps are only performed for the requirements that can be assigned with a corresponding provided policy.

- Step 2 and Step 3: These very similar steps match delegation depth and retention period: Both checks succeed with a full match if the provider's value is lower than or equal to the requester's value. In our example, the delegation depth specified for the policy of bookerAddress does not match the corresponding requirement as the requester does not allow any delegation of the user's address at all. In all other cases, the delegation depths match because the provided policies are more restrictive than the requirements in this aspect. However, there are mismatches with regard to the specified retention periods because the provided service would store all data for 24 months, while the requester only allows a storage of 12 months for the highly sensitive parameters bookerName and address.
- Step 4 and Step 5: These steps match visibility and location limits: Both succeed with a full match if the provider's values are a subset of the requester's values, using ontological subsumption reasoning, again. In our example, this is the case for all depicted policies. In particular, the example policies also match regarding the location limit because the locations specified in the provider's policies are part of the locations specified by the requester (like Germany is contained in Europe).
- Aggregate: In order to get a result for one whole row, the results for the single fields are aggregated. At the end, the final matching result for the whole privacy aspect is aggregated taking into account the requester's sensitivity specifications; the final result is a value between 0 (does not match at all) and 1 (matches perfectly).

Due to the mismatches within the bookerAddress policy, the final matching result for our example must be less than 1. The concrete value is calculated as depicted in Figure 2.7. In the following, we demonstrate Step 3, the retention period matching, on the basis of the example requirement corresponding to the parameter address. This partial matching result is calculated as follows:

$$rtResult_r = min(1, rPAddend + \frac{retPeriod_{provider} - retPeriod_{requester}}{maxRetPeriod}) \quad (2.2)$$

$$\begin{aligned} rtResult_{address} &= min(1, 0.5 + \frac{retPeriod_{bookerAddress} - retPeriod_{address}}{100}) \\ &= min(1, 0.5 + ((24 - 12)/100)) \\ &= 0.62. \end{aligned}$$

The value parameter $rPAddend$ is part of the matcher's configuration and denotes the minimal impact of a retention period mismatch. By configuring this value, the user can customize how much each of the properties contributes to the privacy matching result. In this example, we set $rPAddend = 0.5$. The difference is divided by 100 (default maximum value for retention period) for normalization. [PAPS15]

The results for the other properties (delegation depth, visibility, and location limits) are calculated similarly and aggregated to one result per requirement. The results per

requirement are aggregated to a final result for the whole privacy aspect. To this end, we use a weighted average about the results of each requirement multiplied with a multiplier corresponding to its sensitivity level. The higher the sensitivity level, the larger is the multiplier ("very low" = 0.2, "low" = 0.4, "medium" = 0.6, "high" = 0.8, "very high" = 1). For policies with the sensitivity "mandatory", any kind of mismatch immediately results in a matching result of 0. According to these values, for our example, we have the following aggregation function:

$$m_{RoomManager} = \frac{m_{address} \cdot 1 + m_{time} \cdot 0.6 + m_{capacity} \cdot 0.2 + m_{room} \cdot 0.6}{1 + 0.6 + 0.2 + 0.6} \tag{2.3}$$

In addition to the final numerical matching result, the requester can be provided with a detailed log about the matching process including a list of all mismatches per row, such that the requester is provided with the possibility to selectively relax her privacy preferences, if no provided service matches. [PAPS15]

More information about this privacy matching approach can be found in our earlier publications [PAPS15] and [APB+15].

2.4.4 Reputation Matching

In the following, we describe a reputation matching approach based on the specification language and example described in Section 2.3.4.

First of all, the available ratings need to be aggregated into reputation values [JBPP14b, JBPP14a]. Several aggregation operators are possible. The selection of the aggregation operators that are applied to the ratings within the matching procedure depends on requirements specification. As soon as the required reputation values considering all requested restrictions have been determined, a simple exact matching approach based on the specifications presented in Section 2.3.4 comes down to simple numerical comparisons.

As an example, consider the RoomManager service with the ratings depicted in Figure 2.4. For this service, c1 evaluates to `true` because the overall reputation value can be calculated based on 200 ratings and turns out to be 4.5 (cf. the first row of the reputation system depicted in Figure 2.4). In contrast, BookARoomPro already got a mismatch for c1 because its overall reputation is only 3.35.

Some of the conditions cannot be evaluated because the restrictions regarding the requested number of ratings are not satisfied. For example, BookARoomPro has only been rated 20 times with respect to response time and only 10 times with respect to security. This means that c2 and c3 do not match regardless of the reputation values for response time and security.

Mismatching conditions get a result of 0, while matching conditions get a result of 1. The results per condition are aggregated into an overall result using the weighted average from all the conditions' matching results. Hence, the results in our example from Figure 2.4 are as follows:

RoomManager: $(1 + 0 * 2 + 0 + 0)/4 = 0.25$
BookARoomPro: $(0 + 0 * 2 + 0 + 0)/4 = 0.0$

The described matching approach is based on preliminary work from a Master's Thesis [Ban14], a Bachelor's Thesis [Neu15], and collaborative work [JBPP14b, JBPP14a]

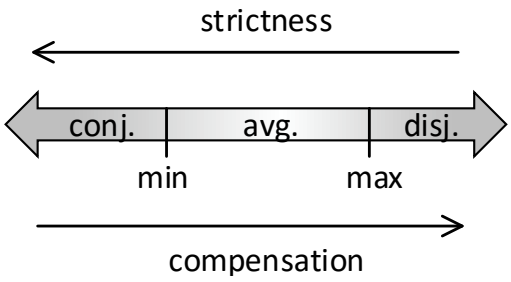


Figure 2.8: Overview of Aggregation Operators (based on [TN07])

within the collaborative research centre 901 [SFB901]. In Chapter 4, this reputation matching approach is redeveloped into a more advanced fuzzy matching approach.

2.5 Foundations in Aggregation

Aggregation is a well-known task within decision-making problems and in multi-criteria decision-making problems in particular. Definitions in the literature about aggregation operators vary, however, they all have in common, that they require an aggregation operator to be a function that yields a value between the minimum and the maximum of the input values [TN07]. In our case, we need aggregation in order to derive a final matching result from several intermediate matching results returned by the different matchers for the different specification aspects. This situation appears on multiple levels: For example, intermediate results could stem from multiple matching approaches, or they could be the result of multiple processing steps within one matching approach. The selection of an appropriate aggregation operator depends on many characteristics as explained in the following.

As shown in Figure 2.8, the space of aggregation operators can be clustered in different regions: conjunctive operators (referring to the "and" connective, e.g., the minimum), disjunctive operators (referring to the "or" connective, e.g., the maximum), and generalized averaging operators in between [GMMP09]. These operators vary in their strictness, respectively in their degree of compensation. Using a conjunctive operator, a high overall value requires all the individual value to be high on all conditions, too. Conjunctive operators include the class t-norms. In contrast, disjunctive operators are fully compensating such that a high overall value is already achieved if a single requirement has a high value. Averaging operators are in-between conjunctive and disjunctive operators. This includes, e.g., the (weighted) arithmetic mean. Accordingly, we could also call the minimum function "strict" and the maximum function "tolerant". Another example for a conjunctive operator is the product. The product is less strict than the minimum.

Another reason in addition to strictness and compensation to choose not the minimum or the maximum but something in between is the distinguishability: the minimum and the maximum can be completely determined by only one value. For matching results, this means, that we may have many equal matching results even though the services differ in very different properties. In contrast, when the product or another averaging operator is chosen, all values contribute to the result.

The choice of aggregation operators also depends on the level of measurement. As an example, take the arithmetic and the weighted mean, which are the most widely used aggregation operators [TN07]. Which mean can be applied on the level of measurement the underlying data has: For example, for the interval scale, we can use the arithmetic mean, while, for the ordinal scale, a proper selection would be the median. When applying the weighted mean within the scope of multi-criteria decision-making, the weighting vector corresponds to the importance of the criteria, while the single weights correspond to the degree of compensation allowed among the criteria. In this thesis, for matching results, we assume an interval scale. The criteria to be weighted when using the weighted mean are the various matching results for the various service properties. For fuzziness scores (see Chapter 4), we have an ordinal scale. This restricts the number of available aggregation operators but allows us to work with other data.

Further alternative and more complex aggregation operators are OWA (ordered weighted average) operators [TN07]. Here, one is able to assign weights according to a value's rank (e.g., weight the best result the highest). However, these operators are less relevant for our service matching problems because in our case, most often the assignment of the result to a specific service property (e.g., performance) is the significant factor. More complex aggregation operators also enable to combine conjunctive, disjunctive, and averaging operators, e.g., using Fuzzy Pattern Trees [SH11].

2.6 Measures for Validation

Within the scope of our case studies in Section 4.8, we need to measure the *homogeneity* and the *distinguishability* of matching result sets. These properties indicate how well matching results support a user when selecting between a large set of services.

For measuring the homogeneity we use the following formula:

$$hom = \frac{\sum_{k=1}^{K} \binom{N_k}{2}}{\binom{N}{2}}, \tag{2.4}$$

where K is a set of equivalence groups, $k \in K$ is one equivalent group, N_k is the number of items within equivalence group k, and N is the number of all items. In our case, an equivalence group is a set of equal matching result values. This formula corresponds to the probability that, when choosing two times randomly from a set of data (i.e., matching results), the same item will be chosen both times (i.e., two matching results with the same degree). Note that, $\binom{1}{2} = 0$ holds by definition.

Correspondingly, we model distinguishability as the inverted homogeneity:

$$dist = 1 - \frac{\sum_{k=1}^{K} \binom{N_k}{2}}{\binom{N}{2}}. \tag{2.5}$$

Note that there are further measures of statistical dispersion, e.g., the Gini index [Dor79]. We intentionally chose a formula that can also be easily applied to the interval results discussed in Section 4.6.1.

Furthermore, within the scope of our case studies, we work with *precision and recall* metrics known from information retrieval [SM86a]. These metrics have already been

frequently used in service matching to measure the accuracy of matching results [Bru15]. They are defined as follows:

$$Precision = \frac{true_positives}{true_positives + false_positives}, \tag{2.6}$$

$$Recall = \frac{true_positives}{true_positives + false_negatives}, \tag{2.7}$$

where $true_positives$ are pairs of requirements and service specifications that the matching approach correctly identified as successfully matching pairs, $false_positives$ are specification pairs that the matching approach incorrectly identified as successfully matching pairs, and $false_negatives$ are pairs that the matching approach incorrectly identified as not matching pairs.

Recall is often reversely related to precision [BG94]. For example, one can easily construct a very pessimistic or a very optimistic matching algorithm and thereby reduce the likeliness of either false positives or false negatives [Bru15].

True positives, false positives, and false negatives are easy to determine in the case of boolean matching results. However, when working with continuous matching results, it is unclear which results represent positive and which represent negative results. In this case, thresholds can be used to setup ranges of positive or negative results.

Moreover, when reporting experiments, we distinguish between *dependent variables* and *independent variables* [Pre00]. Dependent variables are the values that are measured during an experiment. Subsequent interpretations base on observations of how the dependent variables develop. On the contrary, independent variables are the factors that are intentionally manipulated. The dependent variables depend on these manipulations. The measurements can be adulterated by confounding variables. The influence of confounding variables should be reduced as much as possible.

3

Comprehensive Service Matching

There are already many service matching approaches described in the literature (see [DHC12] or [PvDB+13] for an overview). These approaches are implemented as software components, so-called *Matchers*, and typically focus either on structural properties (e.g., signatures), behavioral properties (e.g., pre- and post-conditions or protocols), or non-functional properties (e.g., performance, reputation, or privacy-related properties).

Combining different matchers considering multiple service properties leads to a higher matching accuracy than only employing single matchers [BOR04, BJPW99]. Especially, both functional and non-functional properties need to be considered during service matching [MRE07]. The reason is that, by considering as many service properties as possible, the requesters' requirements can be covered to the highest possible extent. If large parts of the requirements are not covered by the utilized matching approach, the matching results are misleading and may lead to false negatives and false positives. As an example, consider a situation where only the signatures part of a given requirements specification is matched. Even though the matching result may indicate a high match, the behavioral or the non-functional part of the requirements specification may still lead to a mismatch. As a consequence, the matching result does not reflect the requester's expectation: It is a false positive.

However, in order to cope with these issues by combining multiple matchers, one needs to implement a software component that invokes different matchers in a certain order, that manages data exchange between these matchers, and that aggregates the results returned by each single matcher into one combined matching result. This is a tedious and error-prone task, as matchers may not be compatible with each other and many design decisions need to be made. For example, some requesters may put a special focus on privacy matching. In this situation, the matching should take privacy matching steps into account and privacy matching results should represent a significant part of the end matching results. As another example, large markets usually require especially fast matching to achieve a scalable service discovery. In this situation, the matching's execution time should be optimized by selecting fast matchers and as few matchers as possible. In general, design decisions can make matching more successful if they are made on the basis of both a requester's individual requirements regarding matching quality and the market's characteristics. [PSA15]

In the following, we call the role in charge of all these matching-related design decisions the *Matching Designer*. In Section 5.4, we discuss why it is not trivial to decide whether the requester, the provider, or any other market participant, e.g., a third-party like a broker, takes this role.

There are already approaches that combine matchers to fixed processes. For example, Jaeger et al. combine input matching, output matching, category matching, and "custom

matching modules" [JRGL+05]. All these matchers are executed in parallel and the results are aggregated at the end. In another approach, Huma et al. present a matching process that executes signature matching, pre- and post-condition matching, and protocol matching consecutively [HGEJ12b, HGEJ12a]. However, these matching processes are not flexible: The control flow and data flow between the matchers as well as the matchers' configurations and the aggregation of matching results are fixed. As a consequence, in order to provide strategically designed processes for different situations, matching designers need to either have many different implementations of combined matchers for different situations, or they have to change their process implementations every time the situation changes. This makes the current state tedious, error-prone, and high-effort. For example, if another matcher is added, the execution order as well as the data exchange of the whole matching process may have to be adapted. The more different matchers have been integrated, the more serious these issues become. As a consequence, a flexible and more adaptive approach to matching processes is needed. [PSA15]

In this chapter, we introduce our solution *MatchBox*, a framework that realizes a concept of *Configurable Matching Processes* and a systematic workflow to work with them. MatchBox enables comprehensive service matching considering a customized variety of service properties in order to provide accurate matching results for complex requirements specifications. In contrast to other service matching approaches, MatchBox supports a matching designer in combining existing approaches based on well-defined matcher interfaces and reusing these approaches as part of configurable process models. As soon as matchers have been integrated, the matching designer can use MatchBox to design and redesign matching processes arbitrarily often at model level without having to reimplement any source code. Thereby, the matching designer is able to quickly adapt to different markets' and different requesters' requirements, providing the most appropriate matching process for each situation with low effort. Additionally, the possibility to work at model level provides her with an abstraction that leads to a better overview than the source code itself. Thereby, this approach also increases matching maintainability. Furthermore, MatchBox allows to generate matching processes completely automatically on the basis of a given market specification. The designed process models are validated, can be executed automatically, and support the matching designer in inspecting as well as validating the returned matching results. This can also be used as a simulation in order to collect feedback about whether a matching process is beneficial with respect to a specific context [PSA15].

Rather than representing a new service matching approach competing with the high amount of approaches already described in the literature, the work presented in this chapter applies at a meta level, providing a way to leverage and combine those matching approaches in an easy way. Processes designed with MatchBox are flexible, providing many different kinds of configuration possibilities, while, at the same time, MatchBox also ensures their validity, i.e., the executability of the created matching processes. Thereby, MatchBox represents one step into the direction of world-wide service markets incorporating comprehensive, dynamic matching approaches. [PSA15]

This chapter is structured as follows. The next section lists the scientific contributions provided by this chapter. In Section 3.2, we describe an example scenario and corresponding matching approaches to be used as a running example throughout the section. In Section 3.3, we derive requirements from this scenario. Section 3.4 introduces the concept of configurable matching processes. In Section 3.5, we give an overview of the workflow to apply MatchBox

in three phases that are described in Sections 3.6 to 3.8, including the integration of matchers and aggregation strategies, the (manual and automatic) process configuration, and the execution of matching processes. Section 3.9 briefly describes how we implemented MatchBox, focusing on technologies and architecture. The described implementation serves as a basis for the case studies described in Section 3.10. We highlight the most important limitations of our approach in Section 3.11. Section 3.12 contains a survey about related work. We conclude the chapter in Section 3.13.

Concepts presented in this chapter have been published in three conference papers [PSA15, APB+14, APM+15], in a tool paper [BBP15], and in three technical reports [APG+14, APB+15, PJvR+16]. The concepts are partially based on the Master's Thesis by Gao [Gao14]. Furthermore, parts of the concepts are joint work with Arifulina. Her thesis [Ari] focuses on the parts related to specification languages, while the thesis at hand focuses on the parts related to matching.

3.1 Scientific Contributions

The scientific contributions of this chapter can be summarized as follows:

- This chapter marries concepts from the area of service matching with the areas of software architecture, component-based software engineering, and model-driven software engineering. This combination represents a completely novel approach to a problem that has been addressed in research for decades.
- With MatchBox, we present the first model-driven approach to create matching process models used to combine and execute multiple service matching approaches in a flexible and modularized manner.
- The MatchBox framework contains both a systematic workflow and well-defined interface definition languages for matchers that allow for a fast, easy, and reliable integration.
- We are the first to integrate and leverage systematically modeled market knowledge into our service matching concepts.
- Our hierarchical and aggregated matching results deliver much more valuable information than the matching results produced by related approaches.

3.2 Example Scenario

Consider an exemplary matching designer who is responsible for service discovery in a service market where providers trade university management services following our running example introduced in Section 1.2. In order to match complex requirements specifications and service specifications considering multiple kinds of service properties (as those depicted in Figure 1.3), she needs matchers that implement matching approaches for signature matching, protocol matching, and privacy matching, amongst others.

Our exemplary matching designer already has access to a set of existing matchers: an Ontological Signature Matcher, a Trace-Inclusion-based Protocol Matcher, and a Privacy Matcher (see Section 2.4). Each of these matchers processes a part of the given requirements specification. Her signature matcher analyzes the parameter types of the provided and

requested service's operations for co- and contravariance. For example, according to co- and contravariance rules, Room Manager's operation bookHall from Figure 1.3 matches the requested operation bookRoom. The matching designer's protocol matcher matches the requested and provided operation call orders. For example, in Room Manager's protocol, we can find the requested sequence "first bookRoom, then sendAck". The privacy matcher matches the privacy-related properties. As a result, Room Manager matches rather well with respect to the signatures and the protocol; however, it does not match the depicted requirements specification with respect to privacy requirements.

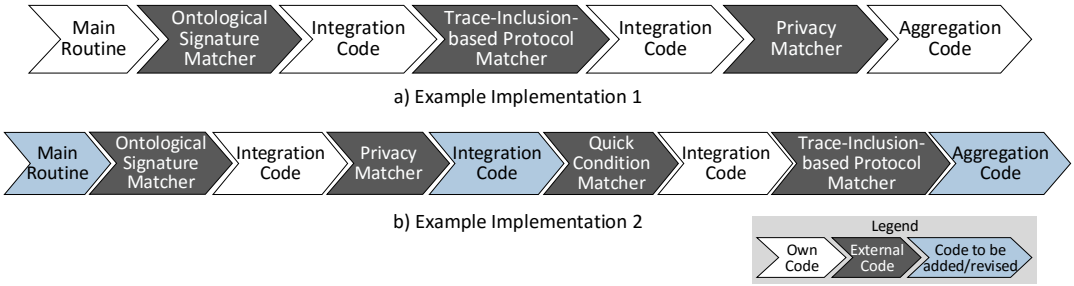


Figure 3.1: Abstract overviews of exemplary matching implementations

Figure 3.1 a) shows an abstract overview of the implementation the matching designer has to construct for combining the three matchers. Using this implementation, she will be able to match the whole requirements specification and to return a matching result to the requester that considers all specified properties instead of delivering three results for different parts of the requirements specification. The three matchers are reused external code and treated as black-boxes. The other parts of the implementation have to be created by the matching designer herself. This includes the Main Routine (serves as a starting point and calls the Signature Matcher), the Integration Code (invokes the Protocol Matcher and the Privacy Matcher), and the Aggregation Code.

The Main Routine and the Integration Code procedures are responsible for control and data flow within the matching implementation. For example, the matchers need to be provided with the data they require, e.g., the corresponding input specifications of the requester and the provider as well as other matchers' results. In addition, the matching designer needs to take care of the data flow between two matchers. For example, the selected protocol matching approach may be based on a signature matching result, as the order of operations occurring in a protocol can be matched only if the single operations have been matched pairwise earlier. In this case, the protocol matcher has to be provided with extra inputs in the correct format. A similar situation holds for the privacy matcher: It depends on the mapping of parameters that is created during signature matching (see Section 2.4.1).

Apart from control and data flow, there are further matcher-specific parameters to be taken care of when invoking a matcher. For example, an ontological signature matcher needs to be provided with a file that contains a domain ontology with information about the parameter types to be matched. Also, depending on the specific signature matcher, the matching designer could configure whether the signature matcher should only consider parameter types or also operation names (see Section 2.4.1).

Furthermore, the matching designer needs to process the output data of all matchers and aggregate it to a final result by writing Aggregation Code. This code implements a certain aggregation operator, e.g., an averaging operator that computes the arithmetic mean. Aggregation is not a trivial task because different matchers may produce results in different ranges. For example, if a provided service matches the requirements quite good with respect to signatures (0.84), completely with respect to the protocol (1.0), and not at all with respect to privacy (0.0), using a simple averaging aggregation would lead to a final matching result of $(0.84+1.0+0.0)/3 = 0.61$. However, result formats could also be heterogeneous, e.g., 84% + $true$ + 0.00. In this situation, it is not clear how to aggregate the results. Even if all results are given in percentage values, the results cannot be aggregated in their current form as they are based on different parts of a service specification: Protocols concern the whole interface of the considered service (interface level), while a signature only concerns one operation within an interface (operation level) [PSA15]. Thus, the inconsistency between the levels of specifications that the matchers work on is a challenge and a potential risk for introducing errors into the matcher implementation.

Finally, after running this implementation, the matching designer can determine that Room Manager from Figure 1.3 satisfies the properties specified in the requirements specification quite well but not perfectly. However, after using the matching process for some time, the matching designer might find out that, in the current market, i.e., the university management market, services differ a lot in their behavior. This is a problem because the current selection of matchers does not cover behavioral properties well. For example, signature matchers are known to deliver false negatives and false positives when it comes to a service's behavior [BJPW99, BV08]. The protocol matcher, on the contrary, addresses the service's interaction, but not the semantics of single operations.

Thus, some time later, the matching designer might want to improve the accuracy of the matching implementation by adding a new matcher that considers behavioral properties of a service, e.g., a Quick Condition Matcher comparing pre- and post-conditions. When integrating this matcher into the implementation, the matching designer has to take into account that a pre-/post-condition matcher typically builds on the results of a preceding signature matching step. Therefore, she has to adapt the control flow and data flow of her implementation as well as the aggregation code, again. [PSA15]

Similarly, the matching designer may serve some new requesters for whom privacy is the most important aspect to be considered. Thus, she needs to adapt the implementation of the aggregation code by assigning a higher weight to the privacy matching step within the aggregation of matching results. Some requesters may even say that all other matching results do not matter if the privacy matching result is not good (e.g., < 0.8). Thus, in order to serve these requesters well, the matching designer needs to move the privacy matching step as much as possible to the beginning of her implementation in order to allow early termination for all services that definitely do not match the privacy requirements. Thereby, a shorter execution time in general is achieved and, as a consequence, most probably, more services can be matched in shorter time [PSA15]. However, she has to keep in mind that the privacy matching step cannot be moved to the very beginning of the implementation because, like the protocol matcher, it depends on the result of a signature matcher. Last but not least, the matching designer decides to change the signature matcher's configuration. Since she detected that the domain terminology is very unstandardized in the current domain of university management, she wants the signature matcher to focus on types; operation

names as well as parameter names should not be considered during signature matching in her process from now on.

The resulting alternative implementation is depicted in Figure 3.1 b). As we can see from the figure, only few parts of the code constructed for Implementation 1 can be reused for Implementation 2: For example, the main routine needs to be revised to invoke the signature matcher with different configuration parameters. The integration code that invokes the condition matcher needs to be newly written and the "< 0.8-filter" has to be added. The aggregation code needs to be revised in order to introduce a weighted aggregation.

The alternative implementation may be better suited for unstandardized domains and for requesters that put a greater focus on privacy than the first one. However, realizing this alternative implementation means for the matching designer to reimplement large parts of her code manually, again. Moreover, other requesters may be served better with the initial implementation or even other implementations. Thus, the matching designer actually either needs several alternative instances of her implementation at the same time, or she needs to adapt her code frequently. In addition, the more complex the matching process, the more time it takes to set it up and the more error-prone it becomes. For example, inconsistent data flow or ill-considered dependencies between matchers are typical problems when combining and running matchers manually.

To sum up, implementing matching processes manually is inflexible, error-prone, and costs a lot of repetitive effort. In order to improve this situation, we need a solution that supports the matching designer by automizing and validating as many parts of this work as possible.

3.3 Requirements

From the example scenario introduced in Section 3.2, we can derive nine requirements for solving the detected challenges [PSA15]:

- (***R1***) *Integrate Existing Matchers:* The matching designer needs to be able to reuse existing matchers.
- (***R2***) *Combine Matching Steps:* Integrated matchers need to be combined in order to solve one matching problem together within one overall matching implementation.
  - (***R2.1***) *Specify Control Flow Between Matching Steps:* By specifying the order of matchers, the matching designer can influence the execution time of the complete matching execution. Furthermore, the control flow determines which matching steps can build on earlier matching steps. Thus, she needs to explicitly define the control flow between different matchers.
  - (***R2.2***) *Specify Data Flow Between Matching Steps:* The matching designer needs to specify data flow between different matchers so that one matcher can use the results of another matcher, if needed. For example, a protocol matcher most often requires the result returned by a signature matcher.
  - (***R2.3***) *Handle Aggregation of Matching Results:* As matchers deliver results only for parts of a specification, the matching designer needs to aggregate these partial results into one final result. Furthermore, the selected aggregation operators highly influence the final result and, thereby, the quality of the results delivered by a matching implementation. Thus, the aggregation has to be configurable.

This challenge is complicated by the different levels of specifications and matching results (e.g., operation level vs. interface level) as well as by different result types (e.g., binary vs. continuous).

(*R3*) *Configure Matcher Properties:* If a single matcher itself provides configuration possibilities, the matching designer must be able to assign concrete parameter values according to these possibilities. For example, she could configure whether the signature matcher should only consider parameter types or also names.

(*R4*) *Run Matching Implementation:* The matching designer must be able to execute the designed matching implementation.

- **(*R4.1*)** *Configure Inputs:* For the actual execution, the matching designer needs to specify the inputs that the matching implementation takes at runtime.
- **(*R4.2*)** *Validate Process:* Automated execution is only possible if the implementation is valid. For example, the specified control flow and data flow must not contradict. The more complex the process is, the more difficult this becomes. Hence, the matching designer must be supported in this task.

(*R5*) *Result Validation:* After having executed a matching implementation, the matching designer needs to be able to inspect and validate the computed matching results in terms of accuracy and execution time.

(*R6*) *Low Effort:* The whole task of providing comprehensive matching should be possible without much extra work for the matching designer.

- **(*R6.1*)** *Reconfiguration:* As her requirements or her context (e.g., the market she operates in) may change, the matching designer needs the flexibility to reconfigure an existing matching implementation with respect to all its properties as listed above with little effort.

Requirements *R1* to *R5* can be summed up to the general goal of providing the matching designer *flexibility* while supporting her in designing valid (correctly executable) matching implementations. However, flexibility always builds a trade-off with *effort* (*R6*): The more flexibility is desired, the more effort is required in the design phase. Thus, the overall goal is to support the matching designer in creating valid matching implementations with as low an effort as possible, while remaining flexible. [PSA15]

3.4 Matching Process Models

MatchBox supports the matching designer in coping with the requirements listed in Section 3.3 in a model-driven way. This means, instead of implementing the code, we work on models of the matching implementation. We call these models *matching processes*.

A matching process consists of one or more *matching step*s. One matching step refers to one matcher that is to be executed as part of the execution of the matching process. We distinguish between *matcher* and *matching step* as there can be different matchers for one *matching type*. A matching type defines a set of matchers with common properties, e.g., matchers that work on the same part of the specification. Thus, this differentiation supports substitutability of matchers applied within a matching process.

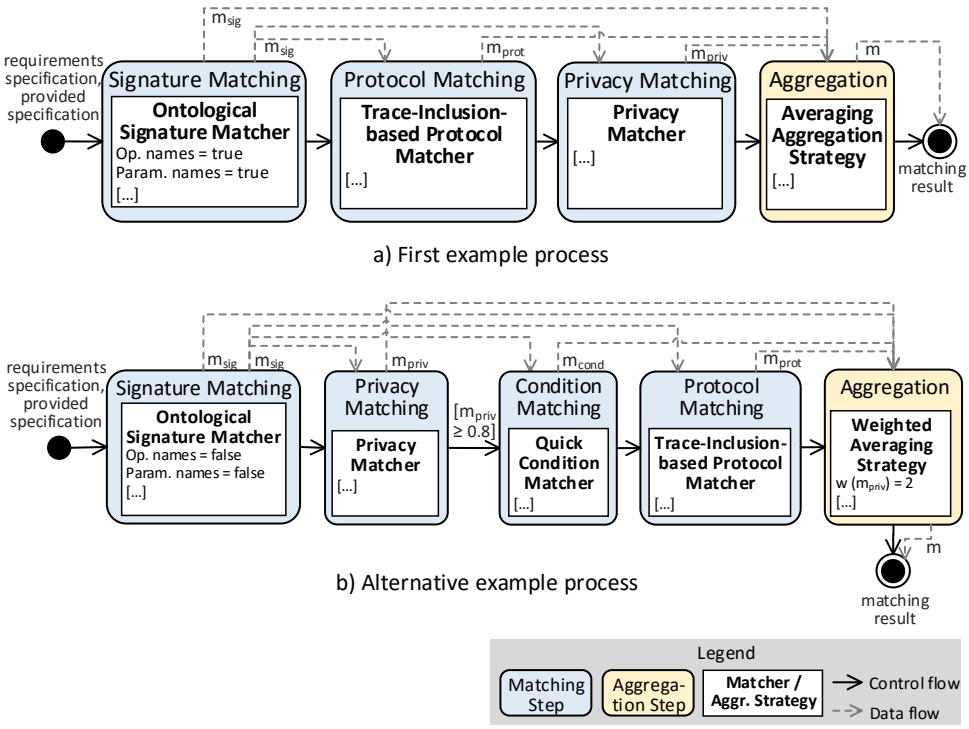


Figure 3.2: Exemplary simplified matching processes

Figure 3.2 a) depicts a simplified matching process developed from the implementation shown in Figure 3.1 a). It consists of three matching steps (Signature Matching, Protocol Matching, Privacy Matching) and an Aggregation step. Each matching step executes one of the matchers described above, whereas each aggregation step executes an *aggregation strategy* based on one or more aggregation operators. The process is simplified with respect to the matchers' configuration possibilities and to the issues that arise from aggregation with different result levels. We discuss this topic in detail in Section 3.6.2.

The steps in this visualization are connected via control flow (black arrows) and data flow (gray, dashed arrows) transitions. The process takes a requirements specification and a provided specification (specification of a provided service) as inputs, e.g., the example specifications from Figure 1.3. Each matching step matches one specification part and computes a matching result for this part (m_{sig}, m_{prot}, m_{priv}). At the end, all matching results are aggregated into a final matching result (m). In addition to the implicit data flow from the initial node to the single matching steps (not shown) and the data flow from these matching steps and to the aggregation step, there is data flow between the signature matching step and the protocol matching step, and between the signature matching step and the privacy matching step due to the functional dependencies as discussed above. Within each matching step, the configuration of the matcher it refers to is contained. The configuration contains a concrete parameter assignment for each defined matcher parameter. For example, the Ontological Signature Matcher should be executed by also considering matching operation names (Op. names = true) and parameter names (Param. names = true). The configurations of the other matchers are omitted for the sake of clarity.

Figure 3.2 b) depicts the matching process corresponding to the implementation in Figure 3.1 b). Compared to Figure 3.2 a), this process consists of one more matching step (Condition Matching), the order of the matching steps has been changed, a guard ($m_{priv} \geq 0.8$) has been introduced, the signature matcher configuration has been changed (only parameter types will be matched, but no operation names and no parameter names), and the aggregation strategy has been substituted.

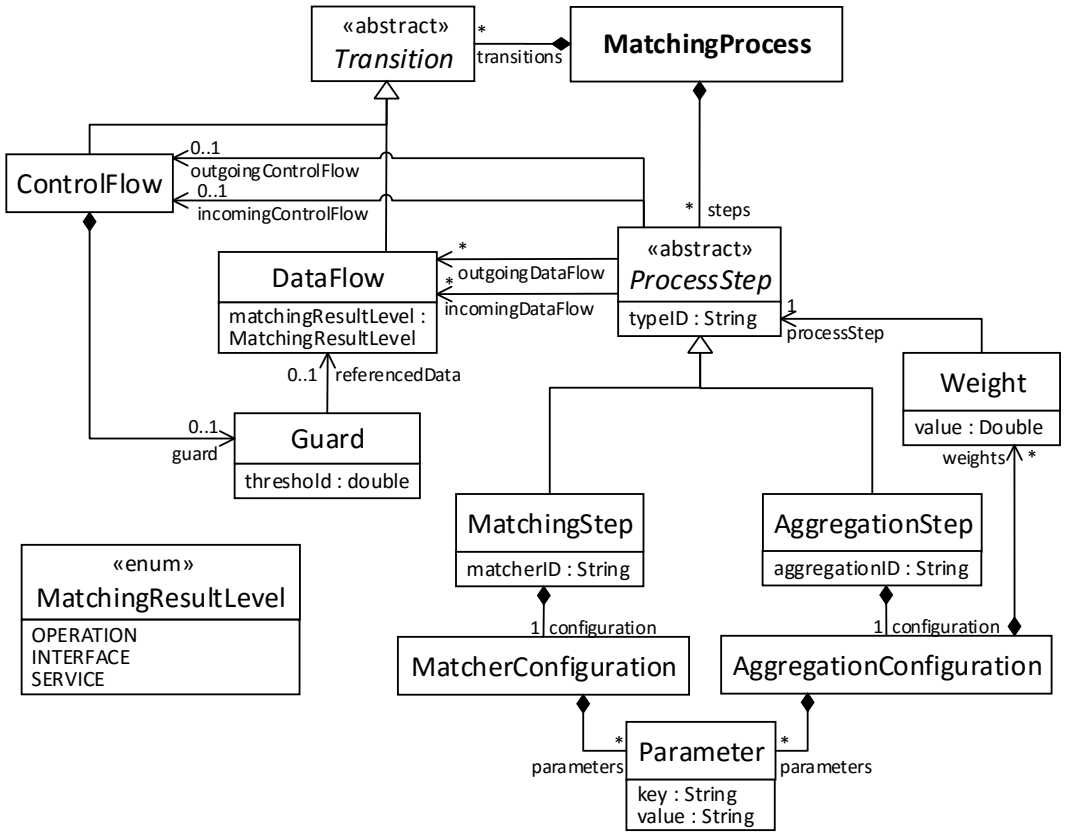


Figure 3.3: MatchBox's metamodel for matching processes

Figure 3.3 shows the metamodel of such matching process models. The metamodel specifies the different kinds of ProcessSteps (MatchingStep and AggregationStep) and Transitions (DataFlow and ControlFlow). Each ProcessStep can have one incoming and one outgoing control flow transition and an arbitrary number of incoming and outgoing data flow transitions. Furthermore, a control flow condition can have a Guard with a threshold. This guard again refers to a data flow condition. Each matching step contains one MatcherConfiguration. Analogously, each aggregation step contains an AggregationConfiguration. Both, matcher configuration and aggregation configuration, can contain an arbitrary number of Parameters, which are key-value pairs.

Designed like this, the metamodel lays the foundations for the fulfillment of *R2* (*R2.1*, *R2.2*, *R2.3*) and *R3*. Note that the depicted version of the metamodel is only an excerpt. The complete version of the metamodel can be viewed in Appendix A.1.

Matching process models can be viewed as executable instances of a domain-specific language defined by the depicted metamodel. Following the terminology of metalevels defined by the OMG [OMG13], the depicted metamodel is on level M2, while the processes created by the matching designer are on level M1. When executing the matching process taking concrete specifications as inputs, we deal with level M0.

As common in MDSD approaches [SV06], there are several benefits of introducing matching process models instead of directly working with the implementation of the processes on code level. One of them is that these models allow us to validate the process for correctness before executing it (*R4.2*). Also, the matching designer is much more flexible because it is typically easier to change the model instead of changing an implementation (*R6.1*). The reason is that the models are an abstract representation that help to handle the complexity. This abstraction also improves the maintainability of the matching process.

Matching Process Models can also be seen as a modularization of complex matching approaches. An alternative to matching processes could also be huge monolithic matching approaches that match many different service properties. However, such a monolithic matcher would be difficult to maintain and to extend. The modularization we achieve by the introduction of our process models helps with handling the complexity because single parts of a huge matcher can now be seen as building blocks whose outputs are combined explicitly using (also modularized) aggregation strategies.

3.5 Overview of the MatchBox Workflow

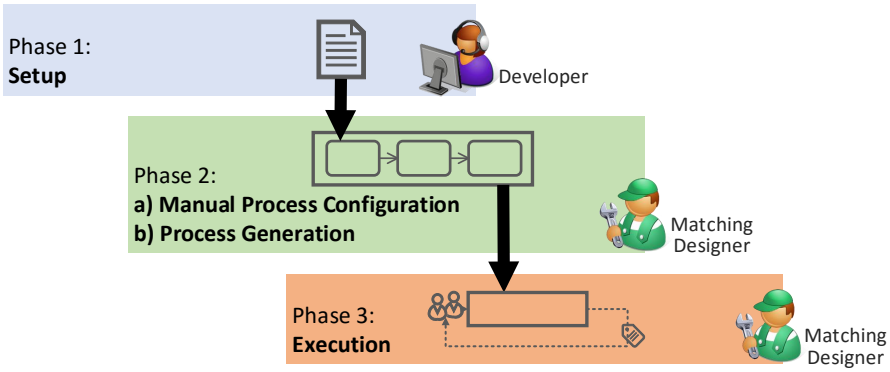


Figure 3.4: Overview of MatchBox' Workflow

The matching process metamodel is the core of the MatchBox framework. In the following, we explain how the matching designer can use instances of this metamodel, i.e., concrete matching process models, as part of the MatchBox workflow within a systematic design process.

MatchBox is a framework in a sense that it provides reusable design and components to be customized and used [Joh97]. For example, the components responsible for creating, validating, and executing processes can be reused. Figure 3.4 depicts an overview that we describe in the following.

Using MatchBox involves three different phases managed by two different roles. First of all, a developer, i.e., a person with technical background, sets up MatchBox. That means

that she integrates existing matchers by using the interfaces provided by the framework. We give details about the setup phase in Section 3.6. The second phase is modeling and configuring a matching process using the matchers that have been integrated in the first phase. The matching designer is in charge of that phase and this is where she can realize her business strategies. This can be done (a) manually supported by MatchBox features like validation or (b) fully automated via a generative approach. Details are explained in Section 3.7. After a matching process has been configured, the matching designer can execute this process with concrete requirements specifications and service specifications as input and validate the returned matching results in Phase 3 (Section 3.8). The execution is performed by a fully automated interpretation of the matching process model created in the previous phase. [PSA15]

The different phases build upon each other. From a developer's point of view, Phase 1 takes place at design time while Phase 2 and 3 take place at runtime. From a matching designer's point of view, Phase 2 takes place at design time, while Phase 3 takes place at runtime. The two roles can also be played by the same actor. For example, the matching designer could integrate matchers themselves before configuring the process and running it. Furthermore, there are many possible iterations and reconfigurations (*R6.1*): For example, the matching designer can run the same process with different specifications as inputs. She can also reconfigure matching processes based on the same integrated matchers in order to compare different matching strategies. In this case, Phase 3 serves as a feedback loop for the matching designer's process configuration in Phase 2. [PSA15]. Thanks to the model-driven design, all these reconfigurations can be done with low effort on an abstraction level that is easy to understand and to maintain (*R6*).

Figure 3.5 gives an overview of the models and languages that are used during the three phases of MatchBox's workflow. In the setup phase, the most important parts within the framework are the AbstractMatcher and the MatcherDefinitionLanguage. The developer integrates new matchers by extending the AbstractMatcher and defining her matcher's properties using the MatcherDefinitionLanguage. In the process configuration phase, the matching designer comes into play by modeling matching processes, including matching steps. Matching process models are instances of the MatchingProcessSpecificationLanguage, which is part of the framework. The execution phase builds on the two previous phases. The Matching Process Execution Engine (MPExecutionEngine) executes the created matching process and creates matching results. The matching results are stored in a model that is an instance of the MatchingResultsMetamodel from the framework. Note that the upper part (modeled by the matching designer) and the lower part (integrated by the developer) are loosely coupled. The only connection is the ID specified in the matching step referring to the MatcherDefinition of the used matcher. Furthermore, the figure shows that the framework is independent from the integrated matchers which provides us with the possibility for extensions and adaptions at runtime. Please note that Figure 3.5 omits aggregation strategies and aggregation steps for a better overview. They are integrated and used in the same ways as matchers are.

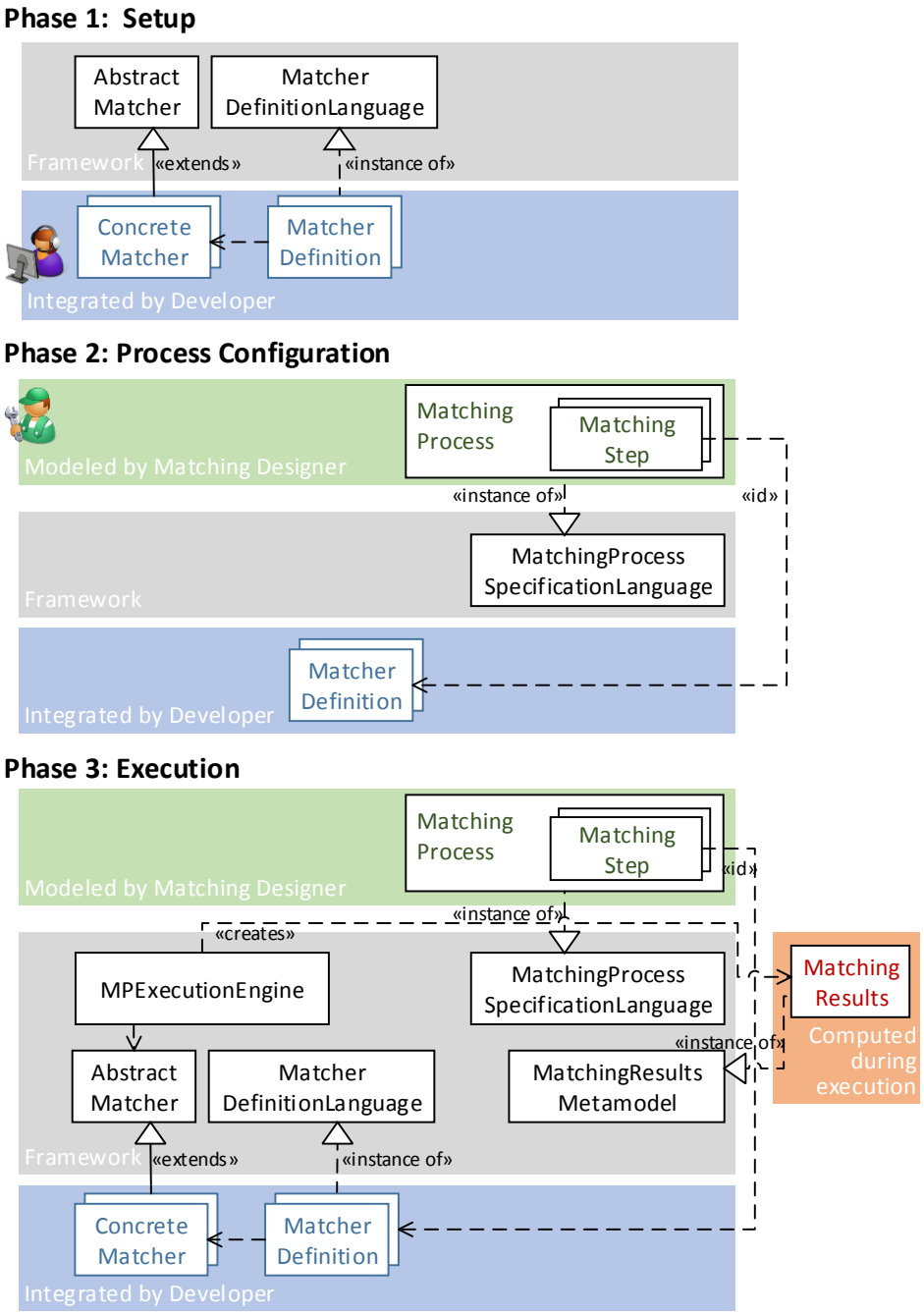


Figure 3.5: Overview of Models and Languages used during the MatchBox Workflow

3.6 Phase 1: Setup

The first phase, setting MatchBox up, comprises the integration of matchers (see Section 3.6.1) and the integration of aggregation strategies (see Section 3.6.2). Both kinds of definitions will be used later in the process configuration phase in order to support the matching designer during the configuration of her process and in order to validate the resulting process. In particular, this section explains how to deal with *R1*. The effort that the setup needs has to be invested once for each integrated matcher. This means much less effort in contrast to the traditional approach, where the integration effort is repeated for each matching process configuration (*R6*).

In the following, we explain the integration on the basis of examples. The metamodel for the definition of matchers and aggregation strategies is depicted and described in Appendix A.1. More examples of matcher and aggregation strategy definitions can be found in Appendix B.

3.6.1 Integration of Matchers

As explained in the following, in order to integrate an existing matcher, the developer needs to (a) define the matcher's interface in a matcher definition and (b) implement delegation code based on a common superclass provided by the framework.

3.6.1.1 Matcher Definition

In order to provide framework-based support for creation and validation (*R4.2*) of matching processes, information about the matchers to be used within these processes needs to be known. In the following, we collect all relevant matcher properties for this purpose on the basis of existing matchers, e.g., the matchers introduced in Section 2.4. A criterion is relevant if it influences the fulfillment of one of the requirements described in Section 3.3. According to all the collected properties, we create a *Matcher Definition Language* (cf. Section 3.5), which can be seen as a DSL (Domain-specific language) for service matching purposes. Two exemplary matcher definitions based on this language are depicted in Figure 3.6. Figure 3.6 a) shows the definition of the signature matcher introduced in Section 1.2, Figure 3.6 b) shows the definition of a protocol matcher. In the following, we explain these matcher definitions and the reasons for the defined properties.

First of all, a matcher should be uniquely identifiable for the framework but also understandable for the human. These issues raise the need for an Id and a natural language Description within the matcher definition.

In addition to their names, their IDs, and their natural language description, a number of further properties are specified per matcher. For example, the most important difference between matchers is that they work on different (parts of) specifications that are taken as inputs at runtime (*R4.1*). The depicted matcher definitions show matchers of two *matching types*: matching.signatures and matching.protocols. A matching type defines a class of matchers that handle the same type of input specification. For example, there can be different matching approaches that deal with signature matching, e.g., the ontological signature matching shown in this figure, or also a string-similarity-based signature matching. Both matching approaches belong to the type "signature matching". In order to automatize the

Name:	***Ontological Signature Matcher***
Id:	*matcher.signatures.ont*
Description:	*The ontological signature matcher leverages the relations specified in a domain ontology. Using these relations, the matcher checks for covariance and contravariance in order to decide whether two signatures comply...*
Type:	*matching.signatures*
Result Format:	*continuous [0, 1]*
Result Level:	*operation*
Parameters:	*Operation_Names: boolean (default = false)* *Parameter_Names: boolean (default = false)*
Required Result:	-
Class:	*de.upb.crc901.matchers. OntologicalSigMatcher*

a) Definition of a Signature Matcher

Name:	***Trace-Inclusion-based Protocol Matcher***
Id:	*matcher.protocols.ti*
Description:	*This matching approach checks if all the traces of the requested service find a matching trace in the ...*
Type:	*matching.protocols*
Result Format:	*continuous [0, 1]*
Result Level:	*interface*
Parameters:	*RollOut : integer (default = 5)*
Required Result:	*Level: operation* *Format: continuous*
Class:	*de.upb.crc901.matchers .TIProtocolMatcher*

b) Definition of a Protocol Matcher

Figure 3.6: Exemplary matcher definitions

execution of matching processes as much as possible (*R4*, *R6*), the delegation of inputs to different matchers needs to be handled. Thus, a matcher's matching type has to be part of the matcher definition.

The introduced protocol matcher requires the matching results of an operation level (one matching result for a pair of two specification parts referring to requested/provided operations) matcher, e.g., of the depicted signature matcher. If the protocol matcher does not receive such results within a concrete matching process model, this model is not valid. The protocol matcher itself, in contrast, delivers results on interface level (one matching result for a pair of two specification parts referring to requested/provided interfaces). See Section 2.1 for connections between these different matching result levels. Another property where matching results differentiate is their format. As an example, the two depicted matchers deliver continuous results within the interval [0, 1], while some other matchers deliver binary results (true or false). In order to process an incoming matching result, its format has to be known. As a consequence, for specifying and validating correct data flow between matchers, the delivered and required Result Level and Result Format have to be specified for every matcher. Specifying properties like the result level and the result format, allows MatchBox to automatically validate whether the data flow and the control flow between process steps have been correctly specified in the configuration phase (*R2.2*, *R2.1*). Furthermore, knowledge about the result levels and the result format is required for the aggregation of matching results (*R2.3*). This issue will be explained in Section 3.6.2.

In addition to the result-related properties, also an arbitrary number of Parameters, i.e., the matchers' configuration possibilities are defined (*R3*). For example, the depicted signature matcher takes the parameters Operation_Names and Parameter_Names, representing the decisions whether operation names and parameter names should be matched, or not. The values these parameters can take when the matcher is instantiated is given by the type. For these two parameters, the type is boolean. Their default values are false. Providing a default value for each parameter allows MatchBox to simplify the process configuration in the next phase so that the matching designer is allowed to work with incompletely specified matching processes to save effort (*R6*).

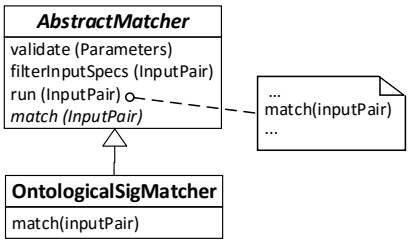


Figure 3.7: Exemplary Template Method application within the MatchBox Framework

The Class property represents the connection between the matcher definition and the implementation of this matcher.

3.6.1.2 Integration on Code Level

MatchBox interprets the matcher definitions when providing the matching designer with the possibility to instantiate and configure (see Section 3.7) and to run (see Section 3.8) a matching process on the basis of the integrated matchers. However, the developer is still left with some effort on code level: As a bridge between the framework code and the integrated matcher's implementation, the developer needs to implement a class according to an abstract common superclass provided by MatchBox. This class delegates to the specific matching algorithm needs to be referenced in the matcher's definition so that the framework can delegate to this class at runtime.

Following Gamma's template method pattern [GHJV95], the common superclass implements the generic part (the same for all matchers) that controls the matcher's data flow. Figure 3.7 shows the class design according to this pattern, taking the OntologicalSigMatcher class as an example and introducing a class AbstractMatcher for the generic parts. If the interface of the matcher to be integrated is different, an adapter class (see adapter pattern [GHJV95]) can play the role that implements the template methods.

Thanks to the matcher definition, AbstractMatcher can save a great deal of work to keep the concrete matchers lightweight. For example, the validation of a matcher's parameters can be done on the generic level as parameter types and default values are known. Furthermore, the AbstractMatcher can filter the input specifications for exactly those specification parts that are specified in the matcher's type. The concrete matcher can overwrite the corresponding method (filterInputSpecs) if special filtering is required. Furthermore, the generic part includes monitors and logging mechanisms as well as data management between the matching logic and MatchBox's user interface, e.g., configuration dialogs and result views (see Section 3.9).

The implementation of the matcher-specific methods build a trade-off between the effort the developer puts in and the knowledge the matching designer gains. For example, if a matching algorithm is adapted so that it extends MatchBox's logging mechanism, the matching designer can monitor the matcher's current internal state while running the matching process (see Section 3.8). However, if the algorithm is used as a black-box, only the start and termination of a matcher can be monitored, though the integration takes less effort. In this case, only the matcher's starting routine needs to be invoked. For even more benefit, the developer can apply instrumentation techniques. For example, she could enhance

the matcher's code using the MatchBox API in order to create more detailed matching results that can be inspected after running the matching process (see Section 3.8) but all this is optional. [PSA15]

The only assumption that the MatchBox framework makes about the matchers to be integrated is that their implementation is executable and written in an object-oriented language for easy integration. Our prototype, however, works with java-based matchers (see Section 3.9). Matchers themselves can also differ in their quality, e.g., their accuracy or their runtime. The matching designer is responsible for selecting appropriate matchers before using MatchBox.

3.6.2 Integration of Aggregation Strategies

One important topic when discussing the combination of several matchers is the aggregation of their matching results (*R2.3*). Aggregation is necessary for several reasons: First, service selection based on multiple matching results leads to a multi-criteria optimization problem where matching results build up a multi-dimensional pareto front. For algorithms, and for humans all the more, it is much more difficult to decide based on multiple criteria than based on one. The more matchers are used, the more complex becomes decision making.

Nevertheless, in some cases aggregation is not reasonable as each step of aggregation loses information. For this reason, MatchBox leaves the decision to the matching designer: On the one hand, MatchBox was designed to be flexible and configurable, it allows to design matching processes with and without aggregation steps. Furthermore, it allows to flexibly configure the aggregation strategies themselves. On the other hand, aggregated matching results in MatchBox are designed in a way that also the original results from the single matchers can still be viewed (see Section 3.8.3).

In some cases, aggregation is simple. For example, if the protocol matcher delivers a result of 1 within a range of [0, 1], and the privacy matcher delivers a result of 0.5 within the same range, an aggregation strategy computing the arithmetic mean would lead to a final result of 0.75. However, aggregation is not always as trivial as in this example for multiple reasons. In this section, we explain some general aggregation concepts for matching results determined by MatchBox building on the foundations within the topic of aggregation explained in Section 2.5. On the basis of these concepts, we explain the integration of aggregation strategies into MatchBox such that they can be used as part of the matching process.

3.6.2.1 Result Formats and Result Levels

As already explained in Section 3.6.1, two important properties of a matching result regarding its aggregation are (a) the format of the matching result and (b) the level a matching result has been calculated for. In the examples introduced above, the aggregation was simple because the results were homogeneous regarding the format and the level. For example, the selected protocol and privacy matchers both deliver results on interface level. However, this is not the case for the example processes depicted in Figure 3.2. In fact, the usage of the aggregation strategy in the processes depicted in Figure 3.2 is invalid. The reason is that the signature matcher's result is defined on operation level and not on interface level as the other matchers' results.

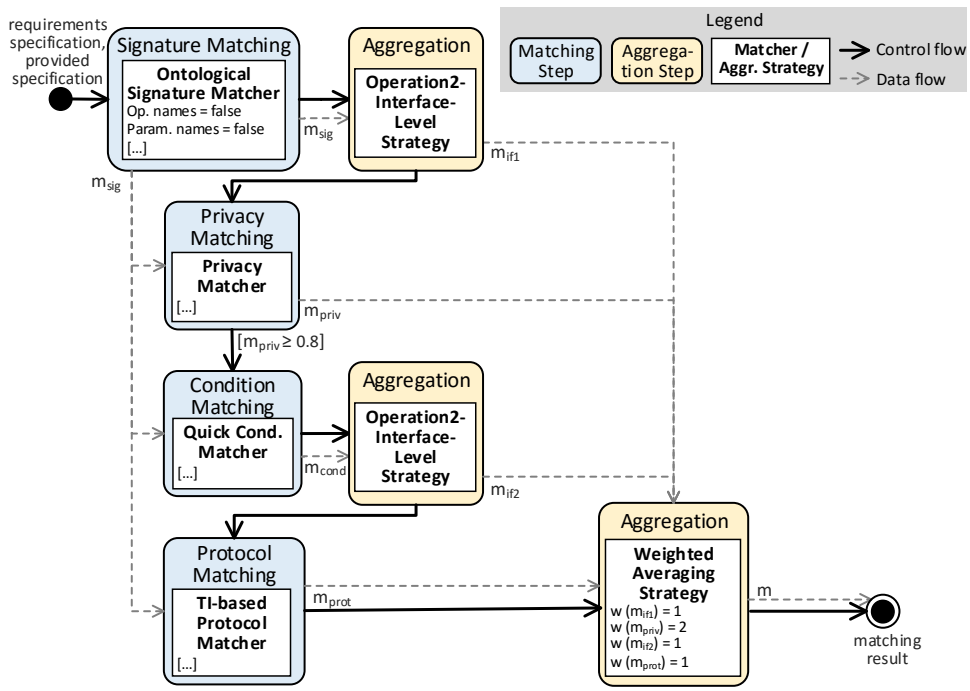


Figure 3.8: Complete Example Process

In order to aggregate the signature matcher's results with the protocol or privacy matcher's results, the signature matcher's results have to be transformed from operation level to interface level by another aggregation step. This is possible as the different result levels refer to parts of a service that are related in a "contained"-relationship with each other (see Section 2.1). For example, one sensible strategy for this transformation is as follows: A required interface matches if all its operations match operations that are part of the provided interface. This strategy corresponds to the *partial module match* described by Zaremski and Wing [ZW95] who laid the foundations for signature matching.

Figure 3.8 shows the valid version of the process depicted in Figure 3.2 b), including the additional aggregation steps. In this figure, the above-mentioned aggregation strategy is called Operation2InterfaceLevel Strategy and it is executed two times. The first occurrence of the strategy takes the signature matching result m_{sig} as an input and delivers an interface level matching result m_{if1} as an output. The interface level matching result can then be taken by the final aggregation step executing the Weighted Averaging Strategy and it can be treated equally as the other matching results on interface level. The same holds for the second occurrence with respect to the condition matching result (m_{cond}) because it is on operation level, too, just as the signature matching result.

One could also argue that, also without the transformation step, the signature matching result is already indirectly part of the aggregated matching result because it influences the results of other matching steps due to functional dependencies. It is again the matching designer's design decision whether the signature matching result is sufficiently important that it should nevertheless be considered within the aggregated matching result for its own, or if

the indirect influence is sufficient in the current context. In the second case, the additional aggregation steps are not needed.

In the same way as we transform between different matching result levels using such transforming aggregation steps, we can cope with transformations between different matching result formats, e.g., between boolean and percentage results. For example, when dealing with boolean and percentage results, the boolean "false" could be transformed to 0% and the boolean "true" could be transformed to 100%. A transformation into the other direction is possible based on thresholds (e.g., everything lower than 50% could be transformed to "false"), but this strategy obviously leads to information loss. In general, transformations between matching result formats are relevant only if an aggregation strategy's implementation is not able to deal with such transformations itself.

3.6.2.2 Definition of Aggregation Strategies

Name:	***Weighted Averaging Aggregation Strategy***
Id:	*aggregation.weighted_average*
Description:	*This aggregation strategy computes the mean value of all incoming matching results under consideration of weights assigned to each matching step.*
Result Format:	*continuous [0,1]*
Result Level:	*interface*
Parameters:	*"weight" : double (default = 1.0)*
Required Result:	*Level: interface; Format: continuous [0,1]*
Class:	*de.upb.crc901.aggregation.strategies.WeightedAggregationStrategy*

Figure 3.9: Exemplary definition of an aggregation strategy

Before aggregation strategies are used within a matching process, they have to be integrated similarly as matchers have to be integrated. This means their properties, e.g., accepted input and output formats and levels, have to be defined.

Figure 3.9 shows the definition of the weighted averaging strategy processed as the final process step of the matching process depicted in Figure 3.8. The weights are defined as Parameters. The current strategy is defined with interface as the result level (the output of the aggregation strategy) and of the required result level (the inputs of the aggregation strategy). However, it is also possible to adapt the same strategy's implementation to other result levels, e.g., on operation level, if both levels have to be the same (see Section 3.6.2.1). In order to transform an operation level result into an interface level result, we need to define further aggregation strategies as explained above.

Internally, MatchBox applies the strategy pattern [GHJV95] to handle the set of integrated aggregation strategies. Typically, weighted averaging aggregation strategies are used in order to generate a final result that considers the results of all matching steps. Such a final result can be used as a first indicator for the requester about the extent to which a service specification matches a requirements specification. Simple alternative aggregation strategies are minimizing and maximizing strategies (see Section 2.5). The matching designer has to decide which strategies fit best to the context. For example, maximizing strategies compensate situations where outliers with bad matching results exist, while minimizing strategies are a better choice for pessimistic service selection, when no risk is acceptable.

In addition to these obvious, relatively simple aggregation strategies, more complex aggregation strategies can be integrated, too. Examples are hierarchical aggregation strategies [YFH09].

3.7 Phase 2: Process Configuration

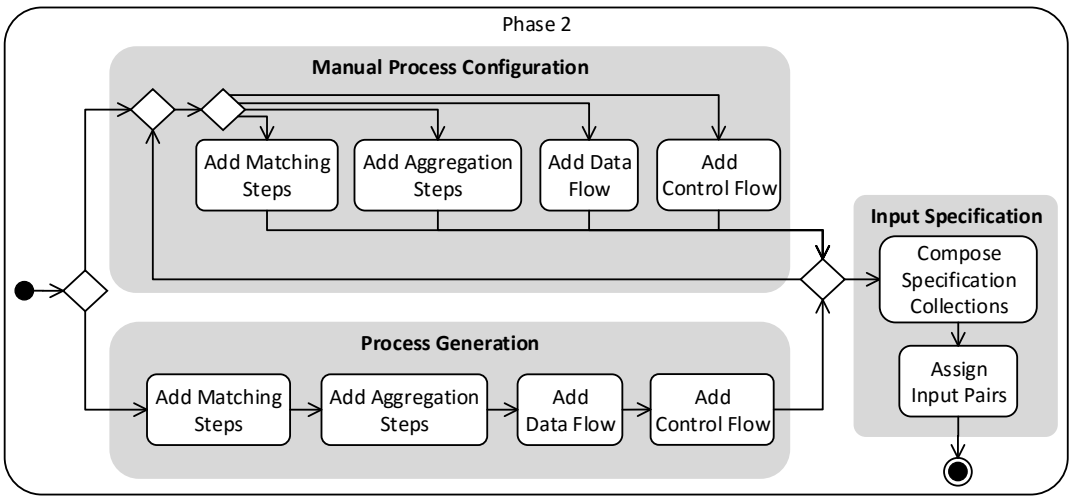


Figure 3.10: Process Configuration Phase

The setup of MatchBox, as described in Section 3.6, enables the usage of matchers as matching steps within a matching process. On the basis of these matching steps, comprehensive matching processes can be configured. Figure 3.10 gives an overview of the steps to create and configure a matching process performed by the matching designer and supported by MatchBox. The process configuration can be performed either manually (see Section 3.7.1), or automatically (see Section 3.7.2), or by using a mixture of both, starting with an automatically generated process and then modifying it manually.

3.7.1 a) Manual Process Configuration

The upper part of Figure 3.10 (Manual Process Configuration) shows which steps the matching designer has to perform in order to model a matching process. First, matching steps can be selected and added. All matching steps that have been integrated in the previous phase are available for selection here. For example, after the developer integrated a signature matcher and a protocol matcher in Phase 1 as explained above, she can now add a signature matching step as well as a protocol matching step. Both steps can then be configured on the basis of the parameters that have been specified during the integration (*R3*). For example, if the signature matcher has been specified with a boolean parameter to decide whether operation names should be considered or not, now the matching designer can choose either "true" or "false" for one specific process instance. [PSA15]

After the matching designer selected one or a collection of matching steps and configured them, the data flow between these matching steps may be configured (*R2.2*). For example, if

a matching designer has added the protocol matcher mentioned earlier, she has to choose a signature matching step the protocol matcher can take the results from. MatchBox supports the matching designer in configuring data flow by validating the dependencies to be created in order to prevent circular dependencies (if Step A requires Step B, Step B must not require Step A) or conflicts with the control flow (if Step A requires Step B, Step B must not be executed subsequent to Step A). Note that the framework automatically takes care of the data flow regarding the propagation of the input specifications: In contrast to the data flow between process steps, the user does not need to explicitly model the data flow going out from the initial node. However, she needs to specify the data flow from matching results to aggregation steps and in between aggregation steps as these flows are not automatically derivable and part of the design decisions. [PSA15]

On the basis of the data flow, the matching designer can determine the order of the matching steps, i.e., add the control flow (*R2.1*). Most often, some sequences are already given by the dependencies created by the specified data flow (e.g., configured as explained above, the protocol matching step cannot be performed before the signature matching step). When adding control flow, each control flow transition can also be constrained with a guard. For example, the guard "$[m_{priv} \geq 0.8]$" (cf. Figure 3.2 b) means that the matching process terminates if the privacy matching result is less than 0.8. Note that Figure 3.2 shows our simplified notation where the paths for early termination, i.e., control flow from a decision node to the final node, are not shown. Such an early termination strategy can speed up the matching process significantly because, when executing a matching process for a high amount of input pairs, guards like this work as a filter. [PSA15]

Apart from implicit decisions at runtime regarding control flow produced by guards, at the moment, MatchBox only allows pure sequential ordering of steps. A comparison and discussion of further possible structures can be found in [Gao14]. Although our matching processes are inspired by UML Activity Diagrams [Obj10], we intentionally decided against the support of alternative or parallel control flow within matching processes for several reasons: For alternative control flow, we could not find any good application examples as matching steps are, typically, not alternative to each other. For example, it makes no sense to decide between a protocol matching step and a privacy matching step because they handle different aspects of a service. A more useful example is to decide between two different matchers of the same property, e.g., between two different signature matchers or between two different signature matcher configurations. However, such a decision is usually not made on the basis of the information a matching process holds during the execution. Instead, such decisions are typically based on further information (e.g., market size) that already can be utilized offline, during the modeling phase. Furthermore, the comparability of matching results matched by the same matching process decreases as soon as the matching process contains alternatives to be evaluated at runtime. We also refrained from supporting parallel control flow within a matching process because we already expect the whole matching process to run parallel on multiple input pairs such that the introduced complexity is expected to be too high in comparison to the gained benefit.

Furthermore, we introduce *Aggregation Steps*. As noted earlier, aggregation steps are similar to matching steps as they are part of the matching process and can be connected to other matching or aggregation steps using control flow and data flow transitions. An aggregation step always refers to an aggregation strategy integrated in Phase 1 (see Section 3.6) in order to aggregate the matching results from the matching steps (*R2.3*).

All tasks that are part of the manual modeling of a matching process can be repeated arbitrarily often. The matching designer can reconfigure the matching process whenever she needs to (*R6.1*). This flexibility is one of MatchBox's benefits because the specification of the matching process influences the runtime as well as the quality of matching results computed in the execution phase.

3.7.2 b) Process Generation

As an alternative to the manual matching process creation as described in Section 3.7.1, we developed an approach to automatically generate matching process models completely automatically from a given *market specification* [APB+14, APM+15].

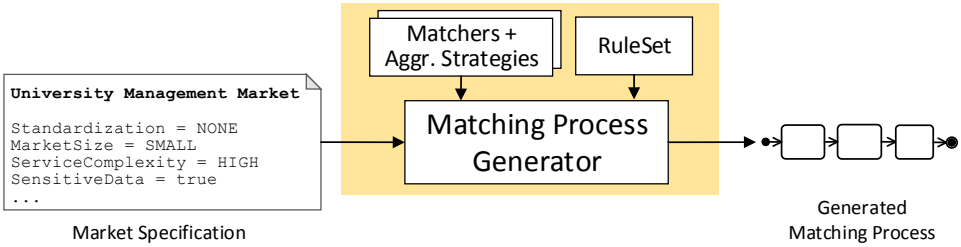


Figure 3.11: Matching Process Generation Overview

Figure 3.11 shows an overview of the Matching Process Generator (MPG). It takes a Market Specification as an input and produces a Generated Matching Process as an output. For this purpose, MatchBox needs to be set up with a set of concrete Matchers and Aggregation Strategies. Furthermore, a RuleSet needs to be defined.

The MPG is at the same metalevel as MatchBox itself. However, the rule set is specific to a certain setup of MatchBox. The reason is that a rule needs to refer to a specific matcher and its configuration possibilities. Furthermore, the usable aggregation strategies need to be known. For this reason, a rule set can only be created after the setup phase (see Section 3.6). In the following, we illustrate the approach and the artifacts of the generation under the assumption that a specific set of matchers and aggregation strategies has been integrated.

Market Specification

By formalizing the current market's properties in the form of a well-defined market specification, we can leverage market knowledge using a systematic, repeatable approach.

The left side of Figure 3.11 shows an exemplary specification of the university management market mentioned in Section 1.2. The university management market has certain characteristics that influence the performance and quality service matching. For example, university structure and its management varies significantly from university to university. There are only few common traits if compared globally. Therefore, neither terminology nor business processes are standardized for services and their specifications in this market [APB+15]. Thus, Standardization is set to NONE. In addition, the UM service market's MarketSize is SMALL. This means, there are currently only few service offers. The ServiceComplexity property has been set to HIGH because most of the services offered in this market offer a complex functionality. SensitiveData is true because university management services typically deal with sensitive data of students and employees whose privacy must be

secured. The metamodel for the market specification language is depicted and described in Appendix A.2.

The ranges of market properties such as MarketSize or ServiceComplexity consisting of fuzzy terms like SMALL and HIGH need to be standardized using universal metrics so that they are comparable. This way, we can prevent situations where a certain market is "small" in the opinion of one matching designer but the same market is judged at "medium-sized" by another matching designer. Examples for such metrics are discussed in [APG+14] and in [Ari].

Generator Rules

Table 3.1 shows some example rules applied within the generation of a matching process based on the given market specification. Most of these example rules are created with the goal to optimize the trade-off between matching accuracy and matching runtime.

Each rule has a type, e.g., Selection, Matcher Configuration, Control Flow Configuration, and Aggregation Configuration. The generation procedure will determine the order in which the rules are applied according to their types.

As an example, consider Rule 1 and Rule 2. These rules denote that only a signature matching step is created if there is Standardization in the considered market. Otherwise, a pre-/post-condition and a protocol matching step are added as well. The rationale behind this is that well-established terminology can replace behavioral specifications of single operations because the semantics of the used names is commonly understandable. If the processes are standardized, matching of the invocation order of operations is not needed because the behavior of equally named services is understood in the same way. In other words, if services in the market are highly standardized, then signatures alone already result in accurate matching because the common terminology and processes prevent behavioral mismatches to a great extent. In this case, we can save runtime by omitting expensive matching steps like pre-/post-condition or protocol matching. Otherwise, we need these additional matching steps that are better able to cover behavioral properties because accuracy would suffer so badly that the lower runtime is not worth it. [APB+15]

The reasoning behind such rules can be very disparate. For example, Rule 3 specifies that reputation matching should be added to the matching process, if the considered market is small, i.e., if it consists of a small amount of service. The idea behind this rule is that in small markets, most service providers are known from earlier purchases. Thus, reputation data is transparent and especially reliable for matching. In contrast, the probability that we have reliable reputation data for every provider and every service is lower in large markets. [APB+15]

Rule 4 focuses on privacy: If a market deals with sensitive data, e.g., student IDs or other personal data that can foster user profiling, then privacy matching should be part of the matching process.

Rule 5 is an exemplary rule for the setting of a matcher's configuration parameters. The idea is that, due to the lack of standardization in the market, signature matching cannot rely on operation and parameter names. Instead, it should focus on well-defined types, which can be matched ontologically.

In Rule 6, the matching process' control flow is configured based on the market size property. This rule states to decrease all thresholds within guards by 0.2 in the matching process for a market with a small size. This results in more services running through the

whole matching process because of a weaker filter effect. The reason for this rule is that, in a small market, the probability of a perfect match is lower than in a large market. By decreasing the thresholds, we can receive more matching results. This increases the mean runtime of the matching process but we can afford this because there are only few services to be matched in small markets.

An example for the configuration of the aggregation of matching results is Rule 7. It says that, if there is sensitive data, we assign a higher weight to the privacy matching step within the results' aggregation. This rule assumes that the final result is determined by a weighted averaging aggregation strategy.

Table 3.1: Exemplary Generation Rules

Rule	Rule Type	Rule Definition
Rule 1	Selection	Standardization = BOTH → select Signatures Matcher
Rule 2	Selection	Standardization = NONE → select Signatures & Pre-/post-conditions & Protocols
Rule 3	Selection	Market size = SMALL → select Reputation
Rule 4	Selection	Sensitive data = true → select Privacy
Rule 5	Matcher Config.	Standardization = NONE → configure Signature Matcher: Op. names = false; param. names = false
Rule 6	Control Flow Config.	Market size = SMALL → decrease all thresholds by 0.2
Rule 7	Aggregation Config.	Sensitive Data = true → multiply Privacy result weight with 2

Such rules have to be created by a consortium of domain experts after evaluating a set of domains. This challenge is also addressed in Section 3.11.

Although examples for standardized and comparable metric ranges are discussed in [APG+14] and in [Ari], these example rules still show the heuristic nature of such a generation. The generalisability of these rules and the metrics they are based on needs to be evaluated within different kinds of service markets. The rule set metamodel and its description are depicted in Appendix A.2. For a larger table with more example rules including an initial evaluation of these rules, we refer to our Technical Report [APG+14].

Generation Procedure

The generation procedure applies rules like the ones in Table 3.1 to a market specification like the one depicted in the left part of Figure 3.11. The lower part of Figure 3.10 (Process Generation) shows the order of modeling steps the generator follows:

- **Add Matching Steps:** In this step, the generator applies all rules of type Selection and all rules of type Matcher Configuration. For each applied selection rule, a matching step with a corresponding matcher configuration is created. The configuration parameters are set as defined in the matcher configuration rules. Parameters that are not set by these rules get the values specified as default values in the matcher definition. This feature also allows MatchBox to cope with incomplete matcher configurations, which again keeps also the manual configuration effort low.
- **Add Aggregation Steps:** Next, aggregation steps are generated. The generation strategy used in this generation step depends on the kind of integrated aggregation strategies. As a default strategy, the generator adds one final aggregation step applying a weighted averaging strategy in order to gain expressive final results and aggregation steps that transform operation level matching results into interface level matching results where needed. In this way, the number of such transforming

aggregation steps equals the number of added matching steps that deliver operation level matching results. Furthermore, the generator applies rules of type Aggregation Configuration. Such rules typically address the weights considered by final aggregation step performing the weighted averaging aggregation strategy.

- **Add Data Flow:** In this step, the data flow between matching steps and the data flow required by aggregation steps is generated. For this purpose, the definitions of matchers and aggregation strategies are analyzed with respect to required matching results. Depending on the integrated matchers, heuristics can be applied. An appropriate heuristic in our example is: Each matcher that requires operation level results will be provided with the matching results of the signature matching step as this is the most basic matching step that delivers results on operation level. As typically all matching steps are to be considered in the final matching result and the final result in our examples is on interface level, data flow from each matching step that delivers operation level results to a transforming aggregation step is generated. In addition, the generator adds data flow from every matching step that delivers interface level results to the final aggregation step.
- **Add Control Flow:** On the basis of the created process steps and the data flow between them, the control flow, i.e., the execution order, has to be determined. An arbitrary ordering is not possible due to dependencies between process steps. Hence, the data flow already determines part of the control flow. However, if there are independent steps within the process, there is still room for strategical decisions regarding their order. MatchBox allows to integrate different strategies to select the best order of matching steps. One example strategy is to move to the front matching steps with simple matchers that are usually finished quickly. The idea behind this strategy is, if the matching process contains guards, this strategy increases the possibility to avoid expensive matching steps by early termination and thereby speeds up the matching process execution in general. For this strategy, we incorporate heuristic estimations of the expected mean runtime of each matching step. This mean runtime may again be influenced by the matcher configurations. Furthermore, the rules of type Control Flow Configuration are applied. For example, guards are created.

Applying this generation procedure to the market specification from Figure 3.11 and the rules from Table 3.1, a matching process similar to the process depicted in Figure 3.8 is created, except that the reputation matching step is missing.

3.8 Phase 3: Execution

The steps of the execution phase are depicted in Figure 3.12. After having finished the matching process configuration described in Section 3.7, the requirements specifications and service specifications the matching process will consume and handle at runtime need to be assembled and assigned to the process in the form of input pairs. Afterwards, the matching designer is provided with an executable matching process. After the execution, matching results can be inspected manually or using automated matching result validation methods.

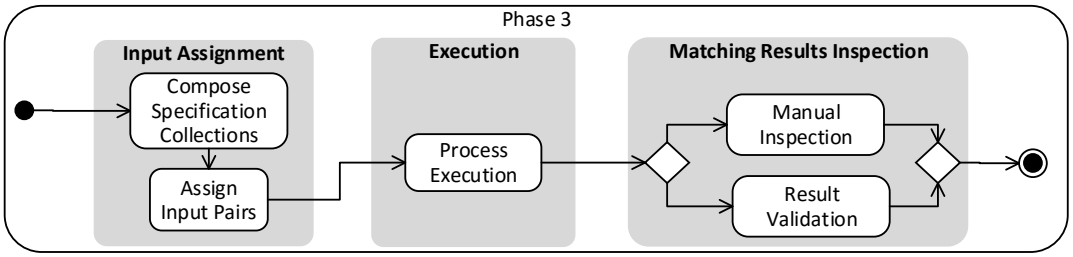


Figure 3.12: Process Execution Phase

3.8.1 Input Assignment

As soon as a matching process has been created either manually or automatically, it needs to be prepared for the execution phase. Thus, the next step configures the concrete inputs the matching process consumes at runtime (*R4.1*). For this, first, the given inputs have to be composed into *specification collections* representing either requirements specifications or provided services' specifications. Depending on the matching types that are used in the matching steps configured earlier, a specification collection can consist of several specification parts. For example, in order to match the specifications shown in Figure 1.3, specifications of signatures, protocols, and privacy properties should be a part of the specification collection. [PSA15]

The composition of the single specification parts within requirements specifications and service specification parts to one specification collection per request and per service is required because of the potentially loose coupling of these specifications. Allowing loosely coupled specification parts to contribute to one requirements or service specification together, allows to use various specification languages (potentially described on different specification levels, e.g., operation level and interface level) in combination with each other. This also clarifies the semantics of an aggregated matching result taking into account all specification parts that belong to one requirements specification respectively to one service.

After a specification collection has been composed, pairs of requirements specifications and service specifications can be assembled and assigned to the matching process. In the following, we call these pairs *input pairs*. In our running example, the input pair consists of the requirements specification for the room reservation service and the specification of the provided service Room Manager. The execution of the matching process processing this input pair will determine to which extent the provided service's specification satisfies the requirements specification.

This procedure also enables to assign several input pairs to a process. This possibility becomes interesting if (a) the matching results of several provided service specifications matched to the same requirements specification are to be compared with each other, or if (b) the matching results of several requirements specifications matched to the same service specification are to be compared with each other. In this case, the process matches the input pairs in parallel, independent from each other, and returns one matching result per pair.

Figure 3.13 shows an excerpt of the MatchBox metamodel for the configuration of input pairs. This metamodel is designed such that it allows us to be independent from a concrete specification languages. All information specified in the input configuration phase is encapsulated in an InputSpecification object which is contained in a MatchingProcess. An

InputSpecification consists of an arbitrary number of requesters and providers, both of type SpecificationCollection. A SpecificationCollection refers to at least one Object. Here, Object serves as a placeholder for the common superclass of all model elements that can be part of a service specification or requirements specification. Furthermore, an InputSpecification consists of an arbitrary number of InputPairs. One InputPair refers to one requester and one provider. It can be enabled or not. Disabled input pairs will be ignored during the execution phase. In addition, an InputPair keeps an ontologyPath. This path defines the location of the ontologies that a pair's contents (e.g., the data types) refer to. After the execution, each InputPair will be assigned with a MatchingResult. More information about matching results and their structure are given in Section 3.8.3.

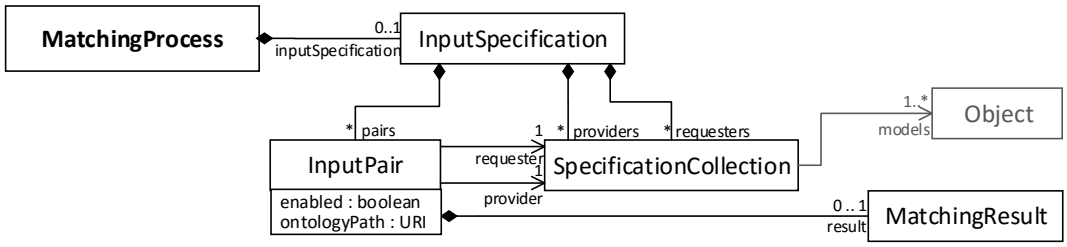


Figure 3.13: Metamodel for Input Assignment

3.8.2 Execution

MatchBox takes care of performing all selected matching and aggregation steps in the defined order fully automatically (*R4*). For this purpose, the instance of the matching process metamodel (see Section 3.4) that has been created during the configuration phase is interpreted. [PSA15]

We call the component responsible for the matching process execution *Matching Process Execution Engine*. The matching process execution engine contains an interpreter for the matching process models. In model-driven software development, an alternative to using an interpreter is code generation [SV06]. In our case, an interpreter is more suitable as it analyzes the model not at design time but at execution time. Furthermore, interpreters are better suited for models that describe behavioral aspects rather than structural aspects [SV06]. The drawback is that interpreters are usually slower than generated code. However, in our case, the time to interpret the matching process is insignificant compared to the time the invoked matchers take (see Section 3.10).

Figure 3.14 shows on an abstract level how the matching process execution engine (MPExecutionEngine) interacts with other parts of the framework in order to execute a matching process: After the MPExecutionEngine has been triggered, it instantiates an MPValidator. This validator checks the given matching process model with respect to modeling flaws, e.g., conflicting control flow and data flow or illegal values for configuration parameters. For valid processes, the execution then starts by iterating over all given input pairs. For each input pair, a RootResult is created. This root result represents the highest level of the hierarchical matching result constructed during the execution of the matching process. After that, the engine iterates over all process steps according to the control flow specified by the matching designer within the matching process model. Depending on

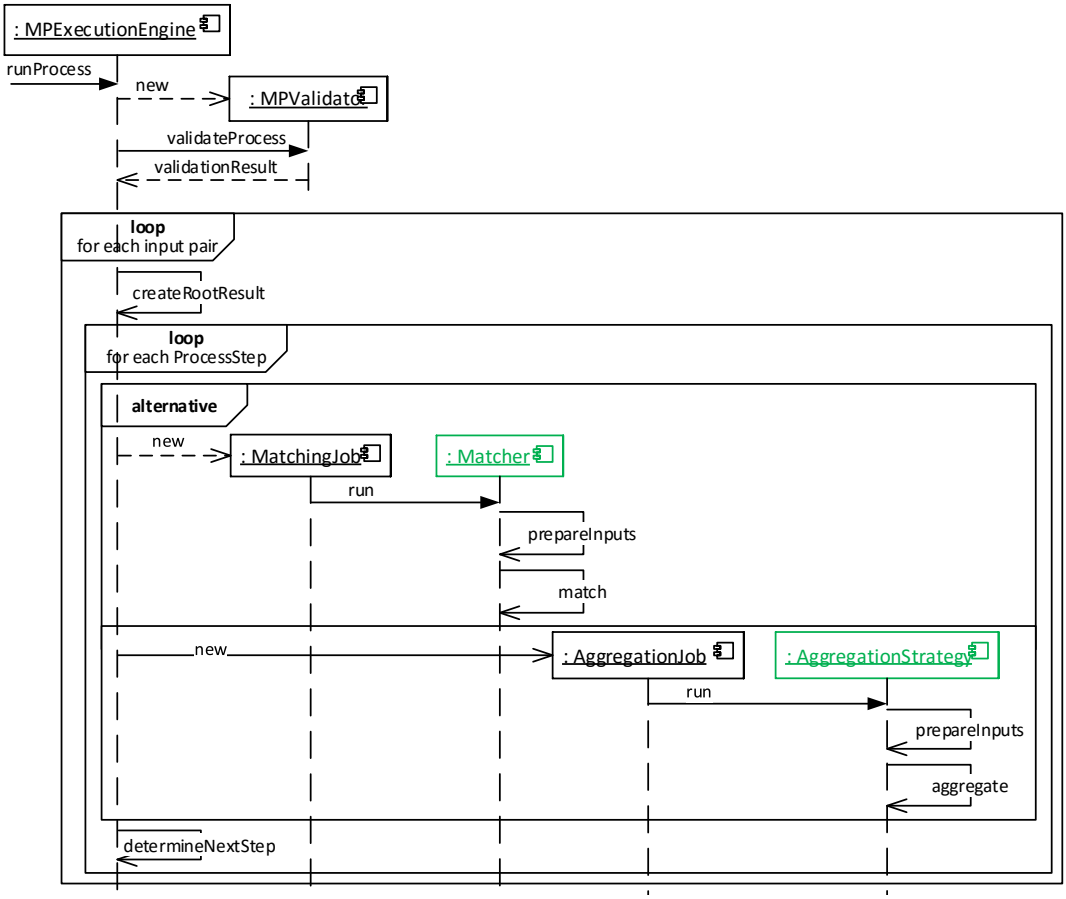


Figure 3.14: Overview of Interactions during the Matching Process Execution

whether the currently considered process step is a matching step or an aggregation step, either a MatchingJob or an AggregationJob is instantiated. Both jobs work equally: The matching job initiates a Matcher and the aggregation job initiates an Aggregation Strategy. At this point, the framework connects to "third party" components (i.e., matchers or aggregation strategies). These components extend components that are part of the framework (e.g., the AbstractMatcher class). Such a low coupling is achieved by reflection: The correct matcher class is invoked via the class name specified within the matcher definition. Low coupling is important in order to stay independent from the matchers, such that any matcher can be integrated at runtime without having to change the framework. The matcher starts with preparing its inputs. This means, the parts of the requirements specification and the service specification to be considered are extracted from the given input pairs. Furthermore, if required, matching results that have been created during the execution of earlier matching steps, may have to be extracted, too. As mentioned in Section 3.6, the input preparation can be implemented in the common super class (e.g., AbstractMatcher) or, if the data structures are rather exceptional, this part can be overridden by the concrete matcher. For aggregation strategies, this works vice versa. After a process step has been processed, the engine determines the step to be executed next under consideration of the modeled control flow. Guards are also taken into account here.

3.8.3 Matching Results Inspection

During the execution of a matching process, matching results for the different matching steps are created and composed in a tree structure. On the topmost level, there are the aggregated matching results serving as overall results for the whole process. During the execution of a matching process, matching results for the different matching steps are created and composed in a tree structure. On the topmost level, there are the aggregated matching results serving as overall results for the whole process. On the next level, there are the results for the matching steps. Below, depending on the matcher, there are different intermediate results that have been determined during the execution of one matching step. For example a signature matcher's result may consist of a result for the comparison of the operation names and a result for the comparison of the parameters. The result for a parameter may again consist of a result for the comparison of the parameter's name and a result for the type matching. The included parts of a result depend on the depending on the matcher's configuration. [PSA15]

```
Aggregated Matching Result: 0.61
  └ Signatures Matching Result: 0.91
    └ bookRoom: 0.83
      └ operation name: 1.0
      └ parameters: 0.66
        └ Date time: 1.0
          └ Date: 1.0
          └ time: 1.0
        └ Integer capacity: 0.5
          └ Integer: 1.0
          └ capacity: 0.0
        └ Room room: 0.5
          └ Room: 1.0
          └ room: 0.0
    └ sendAck: 1.0
      └ operation name: 1.0
      └ parameters: 1.0
          └ ...
  └ Protocol Matching Result: 1.0
    └ ...
  └ Privacy Matching Result: 0.0
    └ ...
```

a) Example results for first example process

```
Aggregated Matching Result: 0.5
  └ Signatures Matching Result: 1.0
    └ bookRoom: 1.0
      └ parameters: 1.0
        └ Date time: 1.0
          └ Date: 1.0
        └ Integer capacity: 1.0
          └ Integer: 1.0
        └ Room room: 1.0
          └ Room: 1.0
    └ sendAck: 1.0
      └ parameters: 1.0
          └ …
  └ Privacy Matching Result: 0.0 (*2)
    └ ...
  └ Condition Matching Result: 0.5
    └ ...
  └ Protocol Matching Result: 1.0
    └ ...
```

b) Example results for alternative process

Figure 3.15: Exemplary Matching Results

Figure 3.15 shows such matching result structures for both matching processes depicted in Figure 3.2. Both processes have been executed with our example specifications described in Section 1.2 as an input. Nevertheless, the results differ (0.61 for Figure 3.15 a) and 0.5 for Figure 3.15 b)) because of the differing matching process configurations: The results in Figure 3.15 a) stem from three matching steps, while the results in Figure 3.15 b) stem from four matching steps. The results from the signature matching in b) are higher than in a) because the mismatches in a) are caused by operation and parameter names, which have been ignored in b). The privacy matching result received a double weight, thus, this low result has more impact on the final result in b) than in a). This shows that it is important to provide the user with such hierarchical matching results: even though there is only a slight difference between the two results when considering only the aggregated matching result on the highest level, the lower levels show the difference much clearer.

After the matching process has been executed as specified, for all enabled input pairs, the matching designer (or other roles, e.g., the requesters or the providers) can inspect the

returned matching results (*R5*). MatchBox's data structure for matching results shown above enables the presentation of all the matching results in a tree view and allows to inspect single matching results on a more detailed level because the corresponding excerpt of the MatchBox metamodel uses Gamma's (degenerated) composite pattern [GHJV95]. Figure 3.16 shows this metamodel excerpt. An instance of one of the subclasses of MatchingResult can be the parent of child MatchingResults. Such an instance is either a BinaryMatchingResult or a ContinuousMatchingResult. The number of result types is easily extensible. Each matching result has a title, a message, and a runtime. Furthermore, it refers to the ProcessStep the matching result has been computed for.

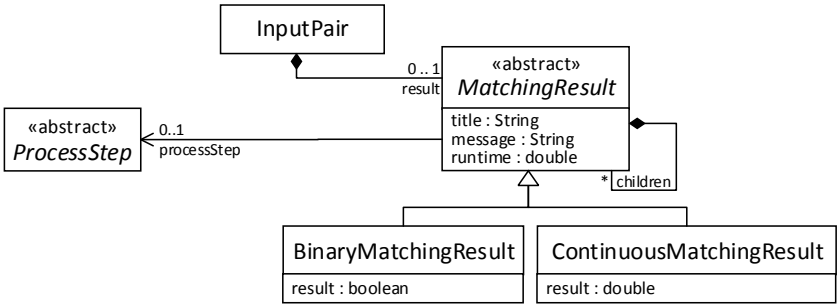


Figure 3.16: Matching Results Metamodel

Comprehensive matching results are required as user feedback [PvDB+13, ZSK15]. They provide requesters with a more informed service selection and indicate possible improvements for their requirements specifications in order to provide more helpful matching results in a next iteration. Furthermore, service providers can benefit from comprehensive matching results in order to create strategies to adjust their offers [PBS14]. The hierarchically structured matching results produced by MatchBox allow a very detailed but also very individual view of matching results for each matching step.

3.8.4 Matching Results Validation

In addition to the possibility to view and validate the matching results manually, MatchBox also provides automatic validation mechanisms. For this purpose, the matching designer needs to specify the expected matching results for the considered input pairs before running the matching process. Expected matching results serve as a "ground truth" here. Usually, they can be determined by domain experts.

After the matching results have been calculated by the process, MatchBox can automatically compare the expected to the calculated results. Thereby, it can determine false positives and false negatives. Based on these measures, MatchBox computes accuracy metrics like precision and recall. In addition, the required execution time for each matching step is recorded by MatchBox so that the matching designer can use it to decide whether the current matching process is appropriate for her application scenario from a performance point of view. [PSA15]

3.9 Prototype Implementation

Our MatchBox prototype has been published within the scope of a tool demonstration paper [BBP15]. It is part of the SeSAME (Service Specification, Analysis, and Matching Environment) tool-suite [AWBP14, UPBe] and is freely available via our website [UPBc]. In the following, we describe the prototype's architecture and its main features.

3.9.1 Architecture and Technologies

MatchBox has been implemented as a set of plug-ins for the integrated development environment Eclipse. It utilizes the extension point mechanism provided by the Eclipse plug-in framework for realizing a component-based construction of the matching processes that enables low coupling. In addition to some user interface related extensions (e.g., configuration wizards, editors, and a result view), MatchBox provides extension points for the matching types, the matchers, and the aggregation strategies. Thereby, every matcher can be integrated very easily by implementing the extension points in its own project using standard Eclipse plug-in development tools. The extension point mechanisms make MatchBox itself easily extensible, too. For example, new matching types can be integrated promptly. [PSA15]

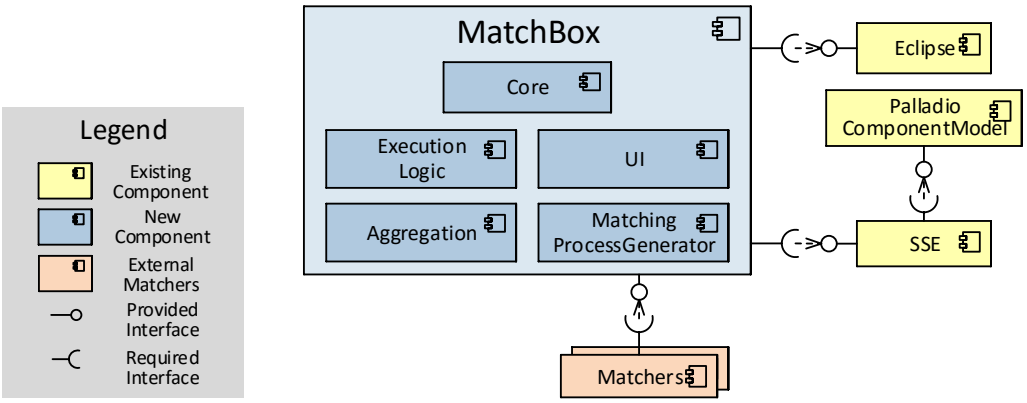


Figure 3.17: Architecture of the MatchBox Prototype

Figure 3.17 shows the components that have been created as part of the implementation. Each component has been realized by a set of Eclipse plug-ins. In the following, we briefly describe each component's purpose with references to the artifacts introduced in Figure 3.5:

Core: The main component Core defines all the models used during the MatchBox Workflow. In total, it includes: the MatchingProcessSpecificationLanguage, the MatchingResultsMetamodel, the MatcherDefinitionLanguage, and the definition language for aggregation strategies. All metamodels have been defined using the Eclipse Modeling Framework (EMF). The definition of matchers and aggregation strategies is integrated via Eclipse Extension Points.

ExecutionLogic: The ExecutionLogic component contains everything MatchBox needs to execute a modeled matching process. This includes the AbstractMatcher and

the MatchingProcessExecutionEngine as well as an abstract aggregation strategy. Furthermore, validation and logging mechanisms are provided.

UI: Amongst others, MatchBox's user interface includes a matching process editor, a dialog for assigning input specifications to a matching process, and the matching results view. The editor has been realized using the Graphical Modeling Framework (GMF), while the other views and dialogs apply other Eclipse visualization frameworks like the Standard Widget Toolkit (SWT) and JFace.

Aggregation: Aggregation strategies can be plugged into MatchBox during its setup as well as matchers. However, MatchBox also comes with a collection of default aggregation strategies, e.g., a weighted averaging strategy, and a minimizing strategy. These default aggregation strategies are part of the component Aggregation.

MatchingProcessGenerator: The MatchingProcessGenerator realizes the generation of matching processes from market properties as introduced in Section 3.7.2.

Eclipse: Eclipse is a foundation of all MatchBox plug-ins. In particular, we apply EMF, GMF, SWT, JFace, and Eclipse's extension point mechanism. The latter is based on the OSGI platform [OSGI].

SSE: The Service Specification Environment SSE includes a collection of languages used for service specification [UPBd]. MatchBox is mostly decoupled from a concrete specification language, but some standard elements from there are used as superclasses to refer to input specification elements.

PalladioComponentModel: SSE depends on the PalladioComponentModel [BKR09]. Thus, MatchBox depends on it indirectly, too.

Matchers: The framework concept realized by MatchBox allows to stay decoupled from any concrete matcher. In principle, the concepts realized within MatchBox are independent from any programming language. However, for easier usage, our current prototype focuses on matchers implemented in Java. Nevertheless, the low coupling principle that MatchBox follows allows us to plug in also components written in other languages, with only a few adaptions.

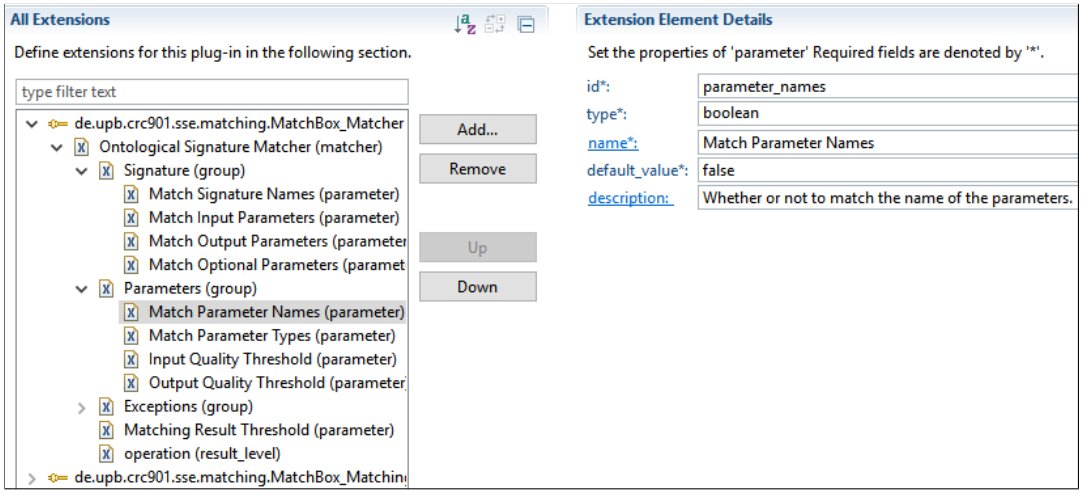


Figure 3.18: Screenshot Matcher Integration

3.9.2 Main Features

Figure 3.18 shows how matcher definitions are specified using Eclipse Extension Points. The depicted example shows the matcher definition of the Ontological Signature Matcher. For example, the definition contains multiple matcher parameters for the consideration of single language constructs, e.g., Signature Names or Parameter Names. The right side depicts more details for the selected matcher parameter. In this example, we can see that the Parameter Names parameter is defined as a boolean parameter (parameter names can either be considered, or not), with the default value false.

Figure 3.19 shows the matching process editor. The shown extract of the process shows two matching steps (Signature Matching and Condition Matching) as well as two aggregation steps (Condition Matching Result 2 Interface Level Result Aggregation and the final Aggregation). Each step contains a white rectangle that depicts the concrete configuration of this step including the parameter values set by the user.

The matching results view depicting hierarchical matching results is shown in Figure 3.20. As we can see there, a matching result can be "opened" to show its child results. For example, within the matching results the signature matcher delivered for the operations search and extensiveSearchRoom, we can also inspect the results for each parameter.

3.10 Validation

MatchBox has been applied and validated within the scope of CRC 901 "On-The-Fly Computing" [SFB901] on the basis of two case studies. In the following, we introduce the validation questions, the case studies, and their results. Last, we discuss the results with respect to threats to validity and the satisfaction of our requirements.

3.10.1 Validation Questions

According to Section 3.3, MatchBox's applicability and success comes down to the fulfillment of the two major goals of (1) flexibility and (2) low effort. Flexibility is achieved by satisfying Requirements *R1* to *R5* (see Section 3.10.5). The impacts of flexibility can mainly be observed by the variability of matching processes that can be constructed using MatchBox. Thus, our first case study focuses on matching process variations. Our second case study focuses on investigating the effort (*R6*) needed to apply MatchBox.

This leads to the following validation questions:

(*VQ1.1*) *Matching Process Variability:* Can MatchBox be flexibly used such that matching processes that significantly vary in their impact are obtained?

(*VQ1.2*) *Matcher Integration Effort:* Is using MatchBox a benefit to the matching designer with respect to effort, compared with traditional matcher combination approaches?

VQ1.1 is addressed in Case Study 1.1 (see Section 3.10.2), while *VQ1.2* is addressed in Case Study 1.2 (see Section 3.10.3). The outcome of both case studies is discussed in Section 3.10.4. While Case Study 1.1 deals with MatchBox' Phase 2 (Process Configuration), Case Study 1.2 focuses on Phase 1 (Setup). Appendix D gives additional information about where to access all data to repeat our case studies.

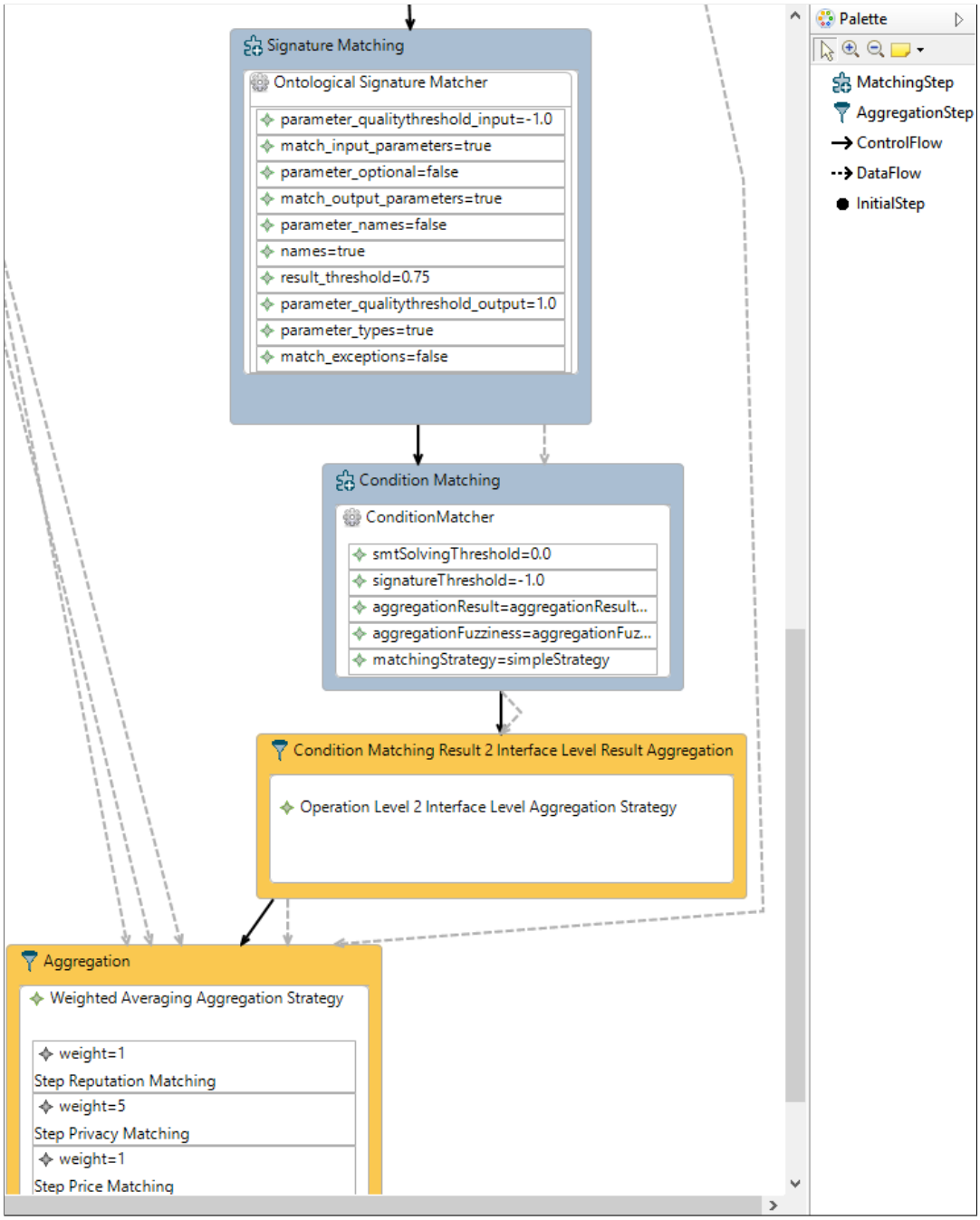


Figure 3.19: Screenshot Process Configuration: Matchers and Aggregation Strategies

InputPair	Requester	Provider	Result	Message
Input Pair SimpleReservation	Request for Reservation Ser...	SimpleReservationService		
Reputation Matching Resu				
Signature Matching Result				
Operation Signature	[RequestForReservationSer...	[simpleReservationService]...	0.8592	
Operation Signature	[RequestForReservationSer...	[simpleReservationService]...	0	
Operation Name	search	extensiveSearchRoom	0.5774	Similarity of...
Input Parameter Ty			0	1 matching ...
Graph Result			0.6	
Parameter N	IN time: dateTime	IN time: dateTime	1	
Parameter N	IN participants: integer	IN participants: integer	1	
Parameter N	IN building: Building	IN building: Building	1	
missing ma		INOUT hasProjector: boole...	false	missing ma...
missing ma		INopt course: Course	false	missing ma...
Output Parameter			false	Matching f...
Operation Signature	[RequestForReservationSer...	[simpleReservationService]...	0	

Figure 3.20: Screenshot Hierarchical Matching Results

Note that the validity of service matching approaches in general is already difficult to validate. Correspondingly, Bruns [Bru15] showed in her survey of evaluation strategies utilized for existing service matching publications that most matching approaches have only been validated to a very limited extent. One reason for the difficulty of evaluating service matching approaches is, for example, that – for most evaluation strategies – we need a collection of suitable example specifications for both the requirements and a variety of provided services. There are already multiple test collections for signature matching [Bru15], but as soon as a matching approach is based on a specification language that is less common, there is a lack of example specifications. The manual specification always leads to the risk of biased results because the generality of these specifications is not ensured [Pre00]. Especially if the evaluator herself constructed the specifications already with a research hypothesis in mind, the external validity of the case study decreases. In addition to the example specifications, for common evaluation strategies, a "ground truth" is needed, i.e., expert results that the computed matching results can be compared to. Sometimes, results already published for another matching approach can be used as reference results, however, this is only feasible if the approaches are comparable to a certain extent.

The specifications we used as inputs for matching in Case Study 1.1 are manually created within the scope of our Collaborative Research Center 901 "On-The-Fly Computing" [UPBb] by ourselves serving as domain experts. Thereby, we overcome the issue that there are no published example specifications or benchmarks for service matching based on comprehensive service specifications available, while accepting the lowered external validity of our case study for now. This and further threats to validity are discussed in Section 3.10.6.

3.10.2 Case Study 1.1: Matching Process Variability

In order to validate MatchBox with respect to *VQ1.1*, we applied the Goal Question Metric (GQM) approach [vSBCR02]. Table 3.2 shows the corresponding GQM goal definition template.

This goal definition refines *VQ1.1*. Following GQM, we derived a question and corresponding metrics (depicted in Table 3.3) from this goal. We measure variability by

Table 3.2: GQM Goal Definition

Goal	Definition	
Low Effort	*Purpose*	Demonstrate to which extent
	Issue	MatchBox can be used to produce
	Object	variable matching processes
	Viewpoint	from a matching designer's viewpoint

comparing between different matching process. In detail, we compare the best matching results per request on a given test collection of input pairs and the mean variance of the matching result scores per request. Furthermore, we determine the mean runtime for the computation of the matching results per input pair. The mean runtime is again the mean of multiple executions to get more reliable values.

Table 3.3: GQM Questions and Metrics

Goal	Question	Metrics
Variability	To which extent do matching results (and their computation time) vary for different matching processes?	– comparison of top results per request – mean standard deviation per request – mean runtime per input pair

Procedure

In order to compare the outcome of several matching processes, we modeled seven matching process variations using MatchBox:

MatchingProcess 1 (MP1): The matching process depicted in Figure 3.2 a) in a valid version (including additional aggregation steps).

MatchingProcess 2 (MP2): The matching process depicted in Figure 3.2 b) in a valid version (including additional aggregation steps).

MatchingProcess 3 (MP3): Like MP1, but with an additional reputation matching step.

MatchingProcess 4 (MP4): Like MP1, but with an aggregation based on different weights (lower signature matching weight).

MatchingProcess 5 (MP5): Like MP1, but with an aggregation based on different weights (lower protocol matching weight).

MatchingProcess 6 (MP6): Like MP1, but with another signature matcher configuration (op. names = false and param. names = false).

MatchingProcess 7 (MP7): Like MP1, but with another signature matcher configuration (exceptions are also matched).

These processes have been executed with 21 input pairs composed from three example requirements specifications and seven exemplary provided specifications from the domain of university management. Table 3.4 shows the expected results for these input pairs. The expectations have been made based on manual inspection before matching results calculated with MatchBox have been known.

These processes and the input pairs are the independent variables in this case study. The dependent variables are the matching results, the extent to which they deviate, and the runtime needed for their calculation.

Table 3.4: Exemplary Input Pairs from the University Management Example Domain

	Input Pair		Expected Result	
	Request	**Provider**	**Funct. Prop.**	**Non-Funct. Prop.**
1	...	simpleReservationService	good match	medium match
2	for	editableLectureRoomReservationService	good match	bad match
3	a	fancyRoomBookingService	perfect match	medium match
4	Reser-	examRegistrationService	bad match	medium match
5	vation	examRegistrationDeregistrationService	bad match	medium match
6	Service	overviewOfGradesPrinter	bad match	medium match
7		historicalOverviewOfGradesPrinter	bad match	medium match
8	...	simpleReservationService	bad match	medium match
9	for	editableLectureRoomReservationService	bad match	bad match
10	a	fancyRoomBookingService	bad match	good match
11	Regis-	examRegistrationService	perfect match	medium match
12	tration	examRegistrationDeregistrationService	good match	medium match
13	Service	overviewOfGradesPrinter	bad match	medium match
14		historicalOverviewOfGradesPrinter	bad match	medium match
15	...	simpleReservationService	bad match	medium match
16	for	editableLectureRoomReservationService	bad match	bad match
17	a	fancyRoomBookingService	bad match	good match
18	Prin-	examRegistrationService	bad match	medium match
19	ting	examRegistrationDeregistrationService	bad match	medium match
20	Service	overviewOfGradesPrinter	good match	medium match
21		historicalOverviewOfGradesPrinter	good match	medium match

As a control experiment, we used the same setting (including the same matching processes) for one further domain: image processing services.

Results

Table 3.5 shows the matching results for the seven matching process variants. The best results per request are depicted in bold font. Here we see that, depending on the requirements specification, the variability regarding the matching results between the matching processes differs a lot. For example, for the request for a reservation service, always the same service (fancyRoomBookingS.) received the best result. However, in the other two categories, the best services differ. Also the mean standard deviations of the matching results differ. In particular, the standard deviations differ more for MP1-MP3, where the processes differ in their steps, as expected. The same holds for the mean runtime.

Comparing the calculated matching results to the expected matching results from Table 3.4, we can see that the results match each other to a great extent. For example, the fanceRoomBookingS. was expected to be the best match within the first group of input pairs, i.e., the best match for the reservation service request. This finding is also reflected in the matching results for each matching process. Considering the calculated matching results for the two printing services overviewOfGradesP. and histOverviewOfGradesP., we can see that they make the best matches for the printing service request for most matching processes, however, their results are not as convincing as the best results in the other two groups.

In general, some matching processes fit better to the expectations than others. For example, MP2 only reflects the expected results in a very abstract way because of the high number of full mismatches. MP4 reflects the results better, however, it produces a noticeable false

Table 3.5: Results for Matching Process Variations for Inputs from the University Management Example

	Input Pair		Aggregated Results						
	Request	**Provider**	**MP1**	**MP2**	**MP3**	**MP4**	**MP5**	**MP6**	**MP7**
1	...	simpleReservationS.	0.416	0	0.503	0.317	0.433	0.445	0.366
2	for	editableLectureRoomResS.	0.056	0	0.172	0.067	0.033	0.056	0.056
3	a	fancyRoomBookingS.	**0.656**	**0.593**	**0.742**	**0.627**	**0.753**	**0.722**	**0.656**
4	Reser-	examRegistrationS.	0.056	0	0.091	0	0	0.056	0.056
5	vation	examRegistrationDeregS.	0.111	0	0.1	0	0	0	0
6	Service	overviewOfGradesP.	0.333	0	0.25	0.4	0.2	0.333	0.333
7		histOverviewOfGradesP.	0.333	0	0.25	0.4	0.2	0.333	0.333
8	...	simpleReservationS.	0.333	0	0.375	**0.4**	0.2	0.333	0.333
9	for	editableLectureRoomResS.	0	0	0.188	0	0	0	0
10	a	fancyRoomBookingS.	0	0	0.125	0	0	0	0
11	Regis-	examRegistrationS.	**0.656**	0	0.375	0.2	**0.4**	**0.667**	**0.667**
12	tration	examRegistrationDeregS.	**0.656**	**0.4**	**0.625**	**0.4**	**0.4**	0.333	0.333
13	Service	overviewOfGradesP.	0.333	0	0.25	**0.4**	0.2	0.333	0.333
14		histOverviewOfGradesP.	0.333	0	0.25	**0.4**	0.2	0.333	0.333
15	...	simpleReservationS.	0.111	0	0.233	0.133	0.067	0.111	0.111
16	for	editableLectureRoomResS.	0.056	0	0.192	0.067	0.033	0.056	0.056
17	a	fancyRoomBookingS.	0.056	0	0.042	0.067	0.033	0.056	0.056
18	Prin-	examRegistrationS.	0.056	0	0	0	0	0.056	0.056
19	ting	examRegistrationDeregS.	0.111	0	**0.334**	0	0	0	0
20	Service	overviewOfGradesP.	**0.333**	0	0.25	**0.4**	**0.2**	0.667	**0.333**
21		histOverviewOfGradesP.	**0.333**	0	0.25	**0.4**	**0.2**	**0.833**	**0.333**
	Mean standard deviation per request:		0.206	0.125	0.175	0.202	0.176	0.278	0.201
	Mean runtime per pair in seconds:		0.187	0.151	0.151	0.113	0.117	0.114	0.122

positive regarding simpleReservationS. for the registration service request. In contrast to the other processes, MP3, MP6, and MP7 always lead to distinct decisions with one significantly best service per request.

Table 3.6 shows the results for two example image processing services matched to the same requirements specification using the same seven matching processes. Here we can see that the results again differ very much between the different matching process even though the inputs were the same. For example MP6 determined a full match for IPService2, while MP2 determined a match of only 0.667.

Table 3.6: Results for Matching Process Variations with Input Specifications from the Image Processing Example Domain

Input Pair		Aggregated Results						
Request	**Provider**	**MP1**	**MP2**	**MP3**	**MP4**	**MP5**	**MP6**	**MP7**
IPRequest	IPService1	0.4	0.333	0.558	0.267	0.533	0.5	0.4
IPRequest	IPService2	**0.9**	**0.667**	**0.708**	**0.933**	**0.867**	**1**	**0.9**
Mean runtime per pair in seconds:		0.088	0.07	0.124	0.105	0.098	0.091	0.084

In addition, we measured how long it took to adapt the matching processes as explained. These times are as follows:

- Creation of MP1: 103.66 sec

- Adaption of MP1 to MP2: 31.43 sec
- Adaption of MP1 to MP3: 19.40 sec
- Adaption of MP1 to MP4: 6.40 sec
- Adaption of MP1 to MP5: 6.22 sec
- Adaption of MP1 to MP6: 12.57 sec
- Adaption of MP1 to MP7: 6.53 sec

We discuss these results in Section 3.10.4.

3.10.3 Case Study 1.2: Matcher Integration Effort

Table 3.7 shows the GQM goal definition template for the goal of low effort.

Table 3.7: GQM Goal Definition

Goal	Definition	
Low Effort	*Purpose*	Analyze whether
	Issue	effort is reduced in creating
	Object	configurations of matching processes
	Viewpoint	from a matching designer's viewpoint

This goal definition refines *VQ1.2*.

The most significant effort in using MatchBox is located in Phase 1: Setup. Thus, Case Study 1.2 compares the effort of integrating matchers based on an experiment involving test persons and a comparison to the traditional combination approach and based on another experiment that compares MatchBox's effort with the most related work, SME2 (cf. Section 3.12).

Following GQM, we derived a question and corresponding metrics depicted in Table 3.8 from our goal. We measure effort by counting lines of code needed for integrating specific matchers as well as the time the developers required for this integration. [PSA15]

Table 3.8: GQM Questions and Metrics

Goal	Question	Metrics
Low Effort	How much effort does it take to setup MatchBox?	– LOC per matcher – Hours per matcher

3.10.3.1 Experiment 1.2.1: Integration by Test Persons

This experiment reports how test persons use MatchBox, with focus on the integration of matchers. This experiment's results have been published in [PSA15].

Procedure

In order to measure the metrics introduced above, we let students of computer science integrate example matchers developed within the scope of Bachelor's Theses, Master's Theses, or student jobs. In total, these students integrated 11 matchers (7 matching types). Each matcher has been integrated once:

Ontological Signature Matcher: This matcher has already been partly introduced in Section 2.4.1. In addition to operation names, parameters, and exception types, configuration possibilities also include special quality thresholds.

Validated Ontological Signature Matcher: This matcher represents an extended version of the ontological signature matcher. In addition to the concepts implemented there, the validated version also implements concepts on the basis of fuzzy sets to cope with transformation-induced fuzziness (see Chapter 4). This matcher has been developed within the scope of Bunse's Bachelor's Thesis [Bun14].

Quick Condition Matcher: This condition matcher matches pre- and post-conditions in a quick way so that it can be used as a filter. If this matcher determines that the conditions are too complex to be matched like this, it delegates to an SMT solver. This matcher has been developed within the scope of Börding's Bachelor's Thesis [Boe15].

Trace-Inclusion-based Protocol Matcher: This matcher matches protocol specifications on the basis of finite automata by calculating how many paths defined in the required protocol are also covered by the provided protocol.

UPPAAL-based Protocol Matcher: Here, the model checker UPPAAL [LPY97] is used to match protocols.

Henshin-based Protocol Matcher: This matcher matches protocols by checking for graph morphisms using the Henshin graph transformation tool [ABJ+10].

Privacy Matcher: As already partly introduced in Section 2.4.3, this matcher addresses specifications of privacy-related properties.

Reputation Matcher: The reputation matcher takes ratings stored in a reputation system, calculates reputation values, and matches these values. All this is done on the basis of fuzzy logic (see Section 4.6.1). This matcher has been implemented within the scope of Neumann's Bachelor's Thesis [Neu15].

Simple Price Matcher: This matcher matches simple prices, i.e., single integer numbers.

Price Model Matcher: This matcher considers more complex price models in comparison to the simple price matcher. These price models can include a collection of price-related concepts, like accounts and flat rates. This matcher was realized by Merschjohann within the scope of his Master's Thesis [Mer14].

Keyword Matcher: The keyword matcher takes lists of keywords and matches them ontologically.

As a control experiment, we compared the effort to use matching processes created with MatchBox to the effort produced by a manual matching process implementation that has been used for half a year within a service market simulation created within the Collaborative Research Center 901 "On-The-Fly Computing" [UPBb]. In this market simulation, 200 service specifications for two domains (water network optimization and image processing) have been traded among 6 service providers, 2 brokers, and one requester within a distributed peer-to-peer system. This matching process used for service discovery in this system integrated three of the matchers listed above: the Ontological Signature Matcher, the Privacy Matcher, and the Simple Price Matcher. The integration code has been written and maintained by three students which were not the students that developed the matchers.

The dependent variables are the lines of code and the time needed for their integration. The lines of code have been identified by manual code inspection, while the integration time has been determined based on interviews.

Results

Table 3.9 shows the values per matcher for the two metrics defined above. The lines of code are lines in the matchers that are responsible for integrating them into MatchBox and not contributing to the matching algorithms themselves. The time was determined by interviewing the students. The rightmost column shows our own judgement regarding the experience level of the subject, i.e., the student (low, medium, high). We assigned the level "low" to subjects that were experienced with Java but not with Eclipse technologies, "medium" to subjects that had some experience with Eclipse technologies but not with MatchBox, and "high" to subjects that already worked with MatchBox before.

To sum up the results from the table, lines of code for matcher integration varied between 14 and 84 lines. Integration times start with 15 minutes and ends by 8 hours, however, the latter is an outlier. The average time (excluding the 8-hour-outlier but including all experience levels) is 47 minutes.

Table 3.9: Measured Metric Values [PSA15]

Matcher	**Integration**		Experience
	LOC	**Time**	
Ontological Signature M.	84	0:30	med.
Validated Ontological Sig. M.	19	8:00	low
Quick Condition M.	28	0:30	low
Trace-Inclusion-b. Protocol M.	18	1:15	low
UPPAAL-based Protocol M.	15	0:15	med.
Henshin-based Protocol M.	15	0:30	med.
Privacy M.	36	0:15	high
Reputation M.	13	0:15	high
Simple Price M.	14	0:15	high
Price Model M.	65	0:15	high
Keyword M.	24	3:50	low

In contrast to the results for the integration of matchers into MatchBox, the control experiment lead to the following results: During the time that the manually implemented matching process was in use, we got 10 change requests. These change requests addressed different process steps, e.g., changes in the requested aggregation strategy and changes in the configuration of the signature matcher. With each change request, we needed to change and test the integration code manually. This situation was even complicated because the changes had to be performed by different developers and each new developer needed to get used to the implementation. As a consequence, the adaptation work became time-consuming and error-prone: In total, it took multiple working days.

3.10.3.2 Experiment 1.2.2: Comparison with SME2

In this experiment, we integrated a third-party matcher into MatchBox. The purpose of this experiment was twofold: First, we wanted to investigate whether a matcher implementations that was taken from an unrelated project can be integrated without any problems. Second, we wanted to compare the effort this integration took with integration into comparable frameworks.

Procedure
We integrated OWLS-MX [KFS06], a semantic web service matcher for services specified in OWL-S into MatchBox. OWLS-MX was developed by Matthias Klusch et al. and is the foundation for various further matchers, e.g., OWLS-MX3 [KK09a, KK12a]. All these matchers successfully participated in the S3 Service Selection Contest [Klu12].

We selected OWLS-MX because there is example code available for integrating the matcher into SME2 [KP], a framework related to MatchBox (see comparison in Section 3.12.2). Within the scope of our experiment, we compared this integration code with the code used for integration into MatchBox.

Results
For the integration of OWLS-MX into MatchBox, we needed – in addition to the matcher definition – 10 lines of code. These lines included the instantiation of the matcher, its setup, and the creation of a matching result according to the MatchBox metamodel. The integration took two hours. Most of this time had been used for the integration of the various java libraries OWLS-MX uses and their integration as an Eclipse plug-in.

Similar to this result, the java class for integrating OWLS-MX into SME2 that was provided for download together with SME2 contains 60 lines of code of which 24 lines were relevant for the integration.

3.10.4 Discussion

The results we collected during Case Study 1.1 confirm our assumption that different matching processes lead to considerably different matching results even if they have been executed with the same input specifications. In some cases, the selected service is expected to be the same for all matching processes, but the certainty of this choice according to the difference from the best service to the second best differs a lot: For example, in the image processing domain, IPService2 is a clearly better match than IPService1 in MP4 ($0.933 - 0.267 = 0.666$) while the results of MP3 do not show such a clear contrast ($0.708 - 0.558 = 0.15$). Furthermore, these results show that a matching designer using MatchBox is able to perform matching process adaptions with such significant consequences in little time (seconds to minutes).

The mean matching runtime measured during Case Study 1.1 differed only a bit. However, summed up within a market with hundreds of services to be matched before an end result can be delivered to the requester, some milliseconds make a great change.

These results for Case Study 1.1 have been reflected in both test domains. However, the sensitivity of the matching results regarding process reconfigurations obviously depends on the considered input pair. Some pairs (e.g., the pairs involving the request for a reservation service as well as the image processing pairs) are more “stabel” in their results than others.

On the contrary, for the input pairs involving the request for a registration service, the expected selection of the service with the best matching result potentially changes according to the used matching process: Using MP4, the simpleReservationService is an actual option, while for all other matching processes, this service is not among the best choices. The same holds for the examRegistrationDeregService: as an offer to the request for a printing service, this service would only be a choice when MP3 is used. From these observations, we can learn that the reconfiguration of a matching process is more likely to have impact in some markets than in others.

An alternative way to measure variability is to create feature models defining the different matching process configurations. A first attempt to this can be viewed in our Technical Report [APG+14]. The problem with this method is to capture the infinite configurations regarding certain matching process constructs, like aggregation possibilities and guards. Furthermore, the variability is always determined by the set of currently integrated matchers to a great extent.

Our results of Case Study 1.2 show that MatchBox supports the matching designer in terms of integration effort. Even though the integration times from Experiment 1.2.1 highly correlated with the subject's experience, the average integration time was less than an hour, which we view as acceptable when taking into account that the integration only has to be done once. After this integration, as a benefit, MatchBox offers a large variety of configuration possibilities and user support with less effort (seconds to minutes for a process reconfiguration according to Case Study 1.1) afterwards.

Our experiences with our manually implemented matching process support this finding. Compared to the control experiment, reconfiguring matching processes on a model level using MatchBox seems to be always easier, faster, and less likely to induce errors.

Additionally, Experiment 1.2.2 shows that the integration of third-party matchers into MatchBox is possible with acceptable effort and that the effort is comparable with the well-known environment SME2.

As a consequence, both validation questions can be answered positively. We take this as an encouraging result. However, in the future, more extensive evaluations are required and the list of threats to validity needs to be reduced, as explained below.

3.10.5 Satisfaction of the Requirements

In the following, we take up the requirements collected in Section 3.3 and discuss the extent to which they are satisfied by the MatchBox framework.

- (***R1***) *Integrate Existing Matchers:* MatchBox' setup phase addresses the integration of existing matchers into the framework. Case Study 1.2 shows that the matching designer is able to integrate existing matchers.
- (***R2***) *Combine Matching Steps:* In order to combine matchers, matching process models have been introduced. Case Study 1 shows that multiple matchers can be combined using matching process models. The matching process models investigated there describe control flow (*R2.1*), data flow (*R2.2*), and deliver aggregated matching results (*R2.3*).
- (***R3***) *Configure Matcher Properties:* The matching process models also contain matcher configurations where parameters defined in the matcher definitions can be set. As

investigated in Case Study 1.1, configuration of the parameters of the single matchers also has an impact on the matching results.

(*R4*) *Run Matching Implementation:* Matching process execution is handled in the third phase of the MatchBox workflow. The matching processes used within Case Study 1.1 have been executed correctly on the basis of the specified input specifications.

(*R5*) *Result Validation:* The matching results model based on a tree structure provides all details about a matching result needed for validation. As shown in Case Study 1.1, we were able to inspect and validate the computed matching results in terms of metrics including matching accuracy and execution time.

(*R6*) *Low Effort:* As the results of Case Study 1.2 indicate, the effort for using MatchBox turned out to be acceptable. When compared to manually implemented matching processes, MatchBox saves extra effort for reconfigurations thanks to the model-driven approach. Further reasons are the supported validation possibilities, the automated generation of matching processes, and the ability to handle incomplete matcher configurations.

3.10.6 Threats to Validity

There are several threats to the validity of our two case studies. In the following, we discuss the most important issues.

First of all, the origin of the example specifications used as an input for matching in Case Study 1.1 is a threat to validity. We created these specifications ourselves, within the scope of our Collaborative Research Center 901 "On-The-Fly Computing" [UPBb]. There was no alternative because the test collections for service matching that are publicly available only cover very simple service specifications based on signatures or on single numbers representing quality properties. However, this solution lowers the external validity [Pre00] because we cannot ensure that these specifications are in fact good examples for the addressed domain and we can be biased due to our research goals.

Furthermore, the time needed to create and adapt matching processes with MatchBox depends not only on the complexity of the process but also on the user's experience. Thus, the adaption times collected in Case Study 1.1 are only partially representative. This experiment needs to be repeated with multiple users with varying grade of experience in MatchBox and modeling tools in general. From this, we expect that the mean time is a bit higher as the times we measured because we had a rather experienced user. However, we also expect no huge changes in the outcome as MatchBox' graphical matching process editor already provides its users with good support. Another idea to investigate this topic into more detail is to also compare the time needed for manual matching process creation with the time to setup and use MatchBox' generative solutions.

The same issue also holds for Case Study 1.2: The time spans required to integrate a matcher as collected based on the students' judgments are hardly comparable. Reasons include that students have different experiences with technologies MatchBox builds on, like Eclipse extension points. On the one hand, some students did not have any experience with Eclipse at all (e.g., cf. time to integrate the Validated Ontological Signature Matcher in Table 3.9). On the other hand, some students integrated multiple matchers one after another

and became more experienced with each one (e.g., cf. Protocol Matchers). These varieties explain outliers. [PSA15]

Manual inspection also showed that both lines of code and time depended on the number of configuration possibilities. For example, our Ontological Signature Matcher has many parameters that can be configured, while the integrated Reputation Matcher has none. Furthermore, the effort in terms of time and lines of code depends on how "deeply" a matcher is integrated into the framework. For example, if very detailed hierarchical matching results are to be produced, this takes more effort as if only one number is returned as an output of the matcher. Also, optional features like logging and input validation can be implemented in more or less complex ways. In addition, some students simplified their task by grouping several matchers. For example, the three protocol matchers all inherit from one abstract super class, which is responsible for most of the integration code. [PSA15]

Furthermore, our control experiment, where a manually implemented matching process has been used, highly depends on the context to which the matching process has been applied. More valid results could be extracted from a larger number of manually implemented matching processes, such that their development effort can be compared with each other.

However, all in all, the collected times and LOC are all still small compared to the effort it would take to implement different matching processes based on existing matchers manually.

Another threat to validity is that only a few metrics have been taken into account to validate the identified research questions. In a more advanced validation, multiple analytical metrics should be used in order to compare their results [BR08].

In general, there is a realistic possibility that our experiments lead to significantly other results when applied to another context, e.g., to another domain, to another set of example specifications, or to other matchers. Future research activities should include more repetitions of these experiments to ensure the validity with respect to this threat.

Furthermore, we only present a level-I-validation here [BR08]. To extend this by a level-II-validation, more extensive user studies are needed as discussed in Section 6.2.

3.11 Limitations

The most important limitation of our concepts is that we assume the presence of complex specifications. However, the creation of such specifications costs a lot of effort and knowledge. In reality, requesters and providers may not be able or not be willing to provide all these details. For example, in today's app markets, discovery is only based on simple strings and offers are specified using a more or less large amount of natural language text and pictures. There are methods to derive specifications, for example, behavior protocols [BIPT09], however, these methods do not work for all aspects we consider to be part of a service specification. In Chapter 4, we address this issue by providing more means to work with incomplete specifications. Furthermore, in Section 5.3, we show how we can apply our concepts to natural language specifications. The current lack of complex and formal specifications is also caused by the lack of functioning service markets: It is a chicken and egg problem. Our approach does a first step in addressing this problem for future service markets.

Furthermore, the effort to integrate a matcher as part of Phase 1 still provides room for improvements. For example, there are classes of similar matchers that only differ in

some characteristics. For such matcher classes, there could be an approach (e.g., based on templates) offering even more integration support for repetitive parts. In Chapter 4.7, we describe how we realized this for one kind of matching approach: fuzzy-logic-based matching approaches.

Another limitation is that the matching process generation from market specifications requires rules and metrics defined by domain experts on the basis of heuristics. This situation should be improved by empirical work as well as standardization: On the one hand, it needs to be evaluated which of these metrics perform well in multiple markets and, on the other hand, we need to provide some more standardized measuring methods that can be used in order to set the metric values for a specific market. For example, regarding our example rules, one could ask: "When is a market regarded as *small*?". The issue of well-defined metrics and ranges is addressed in [APG+14] and in [Ari]. Another possibility to cope with this limitation is to apply approaches for developing a consensus for the interpretation of fuzzy terms [HLCY06, HLL+05]. This problem is also discussed in Chapter 4, where fuzzy terms are used within requirements specifications and service specifications.

Moreover, our concepts end with the delivery of the comprehensive matching results. However, in order to support users of these matching results (e.g., requesters or providers, or automated composition algorithms) in the most optimal way, we need to investigate service recommendation strategies that recommend services fully or semi-automatically based on such matching results. For example, Jungmann et al. integrated machine learning techniques for the purpose of automated service recommendation [JKK13]. Furthermore, strategies and tools for SLA (service level agreement) negotiations [YWK+11] based on such matching results need to be developed. Alternatively, empirical studies about how human user's deal with such comprehensive matching results are needed.

A smaller issue is that, at the moment, we consider three levels of matching results: operation level, interface level, and service level. Also more fine-grained classifications are possible. For example, actually, most condition matchers and our privacy matcher do not require the whole operation level result but only the parameter mapping. Thus, we could also introduce a parameter level result. For now, we did not introduce a parameter level result because such results are not as commonly used for signature matching results. Instead, we focused on the classification common in related matching approaches starting with operation level results.

Our prototype is also limited with respect to several aspects: Currently, control flow within matching processes modeled with the MatchBox prototype is restricted in a way that guards can only be used in combination with gradual matching result formats. Furthermore, our prototype is based on Eclipse's (respectively OSGI's) extension point mechanism. Thus, the matching processes cannot be used standalone by now. As Eclipse-based implementations usually come with a lot of overhead, an application of MatchBox within the context of a larger market architecture requires some reengineering effort.

3.12 Related Work

Approaches related to the work described in this chapter have been surveyed according to the guidelines for systematic literature reviews by Kitchenham et al. [KBB+09, KC07, BKB+07]. This well-established method aims at the construction and documentation of

objective, unbiased, and repeatable results. In the following, we give more details about the survey procedure we followed and the results we produced based on this procedure.

3.12.1 Survey Procedure

One important step in Kitchenham's method is the construction of appropriate survey questions. In line with our requirements, flexibility and effort are relevant comparison criteria. However, the surveyed approaches differentiate in very basic principles when it comes to flexibility. As the evaluation of effort is relative to the provided flexibility, this property is hardly comparable on the basis of a survey. Thus, we focused on flexibility, leading to the following survey question: *How flexibly can matching approaches be combined in related work?*

The literature has been collected during the time period from October 2011 to November 2015. For the collection, we primarily utilized the meta search engine Google Scholar [Gooc], which also includes search results from the most important digital libraries in software engineering, e.g., ACM Digital Library, IEEE Xplore, SpringerLink, and Science Direct. Papers that have been identified as relevant have also been used as a source for snowballing, i.e., scanning the references lists for more relevant papers.

Papers were included according to lists of predefined keywords. We distinguish between *Service/Component Keywords*, *Matching Keywords*, and *MatchBox Keywords*. Service/Component keywords are substantives representing the matched software entities. Examples are "Service", "Web Service", or "Software Component". Matching keywords are substantives and verbs denoting matching tasks, e.g., "Matching", "Matchmaking", "Discovery", and "Retrieval". MatchBox keywords are keywords that indicate (a) any kind of service matching framework, (b) the combination of matchers targeting multiple service aspects or multiple matching algorithms, or (c) configurable and flexible matching approaches. Examples are "Framework", "Hybrid", "Process", and "Rich Descriptions". The keyword sets have been identified in our primary surveys [Pet13, PvDB+13] and later refined iteratively on the basis of the scanned abstracts. The complete keyword lists can be found in Appendix C.

The keywords have been used in different steps of our survey process explained in the following:

1. In the first step of our survey process, we collected all service matching publications where at least one service/component keyword and at least one matching keyword appeared in the title. The matching keyword had to refer to the service/component keyword.
2. In the second step, we scanned all the abstracts for MatchBox keywords that refer to matching keywords. As the collected MatchBox keywords are rather broad and ambiguous, the third survey step was required to get rid of false positives.
3. Next, we excluded further papers according to the exclusion criteria (see Appendix C) by considering the full text.
4. Furthermore, similar papers have been grouped. We classified two papers as similar, if the authors were the same or almost the same and if the same or similar contributions are discussed.

Using this strategy, we started with a number of **410** publications in the first step. **271** papers had been excluded via title and abstract. Further **71** papers had been excluded by further exclusion criteria. Thus, we ended up with **50** papers (grouped into **31** approaches) for detailed reviews. These approaches are discussed in the following.

These numbers already indicate that it was reasonable to use the Kitchenham method for systematic literature reviews as we started with a very high number of potentially related publications. Following this method allowed us to narrow down the search space in a systematic, repeatable way.

3.12.2 Comparison of Processes in Service Matching Approaches

MatchBox is not competing with the high amount of service matching approaches already described in the literature. Instead, it applies at a meta level, providing a way to leverage and combine existing matching approaches in an easy way. Also, the set of papers selected for this survey can be clustered into two groups: *Meta Matching Approaches* [KvdHD06] and further matching approaches. The Meta Matching Approaches are closer related to MatchBox and are compared in Table 3.10. These approaches do not describe concrete matchers but ways to combine existing matchers, similar to how MatchBox does. The other approaches also combine several matchers but not in a flexible way: The user is restricted to given matchers and has no or only few customization possibilities. A comparison of such approaches is depicted in Table 3.11.

For each listed approach, the tables depict an assessment with respect to the following comparison criteria:

- Integratable Matchers (only Table 3.10): This criterion compares the restrictions to the kind of matchers that can be integrated. This refers to Requirement *R1*.
- Combined Matching Steps (only Table 3.11): With this criterion, we compare which matching steps are part of the matching processes of the different approaches. Values include matching steps for different aspects (e.g., "Keywords", "QoS", inputs and outputs "IO", inputs and outputs and preconditions and effects "IOPE") and matching steps with different strategies for the same aspect (e.g., "Logic-based", "Hybrid"). We use the value "flexible" if any kind of matching step can be part of the process. This criterion refers to Requirement *R1*, too.
- Configurability of Matching Steps: This criterion indicates to which extent matching steps are configurable. Values include "extensible" (if matching steps can be extended), "addition" (if completely new matching steps can be added), "parameterized" (if a matching step can be customized by setting parameters), and "-" (if there is no configuration possibility for matching steps in this approach). This criterion refers to Requirements *R1*, *R3*, and *R6.1*.
- Configurability of the Process: This criterion indicates whether the matching process is configurable, i.e., whether control flow and/or data flow can be modified by the user. Possible values include "flexible" (if the whole process can be modified), "order" (if the order of steps within the process is configurable), "extension" (if the process is extensible, i.e., if steps can be added), "reduction" (if the process is reducible, i.e., if steps can be removed or ignored). This criterion refers to Requirements *R2.1*, *R2.2*, and *R6.1*.

- Configurability of Aggregation: To which extent is the aggregation of matching results configurable? Values are "weights" (if weights can be set for a weighted aggregation strategy), "selection" (if the user can select between several aggregation strategies). This criterion refers to Requirement *R2.3*.

Meta Matching Approaches

Table 3.10 lists all approaches that we classified as meta matching approaches. We describe these approaches with respect to the comparison criteria in the following.

Table 3.10: Comparison of Meta Matching Approaches

Approach	Integratable Matchers	Configurability M. Steps	Configurability Process	Configurability Aggregation
[KvdHD06]	flexible	addition	flexible	weights
GLUE2 [CCC+08]	flexible	addition	flexible	-
S2M2 [KK12b]	restricted to matching expressions in a given meta model and text similarity	addition	extensible	replaceable strategies
SME2 [Klu12, KP]	Matchers based on standard ontological service descriptions	addition	-	-
[VPAH07]	Ontological matchers	addition	-	weights
MatchBox	flexible	flexible	flexible sequences	flexible

The meta matching approach proposed by Kokash et al. [KvdHD06] is described in only a sketchy way. Matchers can be integrated and the user is also able to influence the control flow. The approach provides three compositional operators to combine matching approaches: parallel, sequences with thresholds, and switching (decisions between parallel and sequential execution by predefined criteria, e.g., based on the number of services). The only way to influence the aggregation is to configure the weights.

GLUE2 [CCC+08] is a web service discovery engine that allows to integrate matchers for matching functional and non-functional properties. The user can define a matching process ("execution workflow") and thereby change the order of matchers. GLUE2 has been designed on the basis of WSMO [DBBD+05]; however, the authors claim that it can also be used in combination with other specification languages with some effort. The major limitations compared to MatchBox are that, in GLUE2, there is no data flow allowed between matchers. Furthermore, the aggregation is not configurable.

S2M2's [KK12b] main purpose was to abstract from multiple specification languages used to specify semantic web services, e.g., OWL-S or SAWSDL. In S2M2, the user can add new matchers but these matchers have to be composed of matching expressions following a predefined metamodel. Alternatively, also text-similarity expressions are allowed. S2M2 does not allow to configure control flow and data flow at all. However, it is possible to implement own ranking strategies including aggregation.

SME2 [Klu12, KP] is the extensible *Semantic Service Matchmaker Evaluation Environment* used within the Semantic Service Selection contest (S3) [Klu12, S3C]. Matchers can be integrated in a flexible way into SME2; however, these matchers are not combined as matching steps that contribute to one final matching results. Instead, these matchers are executed separately and their effectiveness and efficiency is compared using several evaluation strategies (e.g., precision and recall or runtime). Similar to the MatchBox

workflow, users can integrate Matchers into SME2 by providing an implementation of a provided interface and a matcher specification. The implementation mediates between the framework and the concrete matcher implementation. The matcher specification is much simpler than the one used by MatchBox: It focuses on technical details like paths and versions. The reason that this is sufficient is that, in contrast to MatchBox, SME2 only considers matchers based on standard ontological service description languages like OWL-S, SAWSDL, or WSML. These are the most familiar web service specification languages, however, they only take into account inputs and outputs and (to a limited extent) preconditions and post-conditions (referred to as "effects"). MatchBox needs more information to integrate a matcher into a matching process as it allows much more heterogeneous matchers that address very different service specification aspects and languages. Furthermore, MatchBox allows integrated matchers to be configurable and to set up their parameters within the framework. This is not addressed by SME2. Additionally, SME2 does not support the creation and execution of matching processes but only executes the available matchers separately for comparison. As a consequence, neither an aggregation of matching results from different matchers nor reconfiguration of existing matching processes are supported. However, SME2's comparing evaluation functionalities would make a useful addition to the MatchBox framework.

The approach presented by Vu et al. [VPAH07] allows the integration of customized matchers, too. It, however, assumes these matchers to be based on a derived version of a given QoS ontology. Furthermore, it distinguishes between mandatory and optional matching results. This feature is not explicitly available in MatchBox; however, it can be simulated using guards, which leaves the matching designer with more flexibility. Apart from that, the control flow is not configurable: All matchers are executed in parallel, followed by a ranking phase based on an aggregation. A supported configuration possibility with respect to aggregation is the modification of weights. Furthermore, reputation can be taken into account explicitly. However, the reputation-based step is not treated as a flexibly applicable matcher. Instead, it is fixed and only the extent to which it contributes to the final result is customizable via weights.

In addition to the listed restrictions, the listed related approaches are not model-based like MatchBox: Any new matcher has to be integrated completely on code level and the matching processes are not as easily usable and maintainable as in MatchBox. However, there is not much information available about how to integrate a matcher into the framework in these approaches. SME2 is the only approach where we actually know what kind of information needs to be specified in order to integrate a matcher. In the other cases, we assume that the integration is done on code level completely. There is also no information about whether the integrated matchers are still configurable, as in MatchBox, or not. Furthermore, the approach proposed by Kokash et al. and GLUE2 are the only ones that do not restrict the kind of matchers that can be integrated.

Matching Processes in Further Matching Approaches

Table 3.11 shows matching approaches that perform several matching steps in a predefined ways. These approaches do not mention matcher descriptions or define an interface for matchers to be integrated. The matching steps are fixed to the largest part, and so are their order and their configuration, as well. On reason for lacking flexibility in these approaches could be that they just do not require complex process concepts as they do not contain many

Table 3.11: Comparison of Matching Processes in Further Matching Approaches

Approach	Combined Matching Steps	Configurability		
		M. Steps	Process	Aggregation
[AMM10]	Keyword, QoS	-	-	weights
[BW03]	Keywords, IO	-	-	-
[CTO10]	Syntactic, Semantic (IOPE)	-	(thresholds)	(thresholds)
[ESAEA04]	Protocols, Reputation	-	-	-
[GTRRC08] [GRRC+07]	QoS, Non-QoS	addition	order, extension, two modes	weights
[GNM+96]	Keywords, Profiles, Signatures, Conditions	-	-	-
[vdHYP01]	Semantic, Capability, Syntactic	-	-	-
[HGEJ12a] [HGEJ12b]	Signatures, Conditions, Protocols	- -	- -	- -
[JRGL+05] [JT04]	IO, Category, Custom (e.g., QoS)	addition, extensible rules	extension	weights
[KKF08, KFS09]	Logic-based IOPE, Hybrid IOPE	metrics replaceable	-	-
[KK12b, KK10]	Logic-based IOPE, Hybrid IOPE	-	-	-
[KK12a, KK09a]	IO (logic-based, text-sim., structural-sim.)	-	-	-
[KK08, KKZ09b]	IO (logic-based + text-sim.)	-	-	-
[KK09b, KK06]	IOPE: types, relations, constr., syntactic, parameters, intentional	-	reduction	weights
[KKZ09b, KKZ12]	IO (logic-based, text-sim., structural-sim.)	-	-	-
[Kon95] [KCL+95] [Kon96]	Screening, Evaluation, Analysis (Functional, Non-Functional, Strategic, Domain, Architecture)	-	-	weights
[LASG07, LA07]	IOPE, Ranking Variants	extensible, parameterized	-	-
[MSZ11] [ZSDS13, SZ11]	Structural, Behavioral, Constraint	-	-	weights
[MHB+12]	Servers, Text, Taxonomy, Cond.	-	reduction	weights
[DA09, DKRA08] [DAS08b]	IOPE, improved IOPE, QoS	-	-	weights
[MXB10]	Signatures, Protocols	-	-	-
[WS03b, WS03a] [SW05]	IO (Word-Net, semantic/structural)	-	-	-
[SWKL02]	Context, Profiles, sim., Signatures, Constraints	-	4 modes	-
[SMSA+05]	Domain-indep., Domain-specific	-	-	-
[WWWB11]	Name, Text, IO, Semantic	extensible, reducible, substitutable	-	selection
[WW05]	Name, IO, QoS	parameterized	flexible	-

steps (most often two to four) and they are functionally dependent such that another kind of combination is not possible anyway. As an example, take the approaches by Huma et al. [HGEJ12a, HGEJ12b] and by Motahari-Nezhad et al. [MXB10]. Like the example in our foundational chapter (see Section 2.4), these approaches first execute signature matching and use the matching results for a preceding protocol matching (and for condition matching in Huma's case). Any reconfiguration of this process would be deep interventions into the matching algorithms and their underlying logic. This fact makes these approaches difficult to extend.

Other approaches do not require flexible matching process concepts as they work on very simple service specifications that can be matched homogeneously to a high extent without much effort. Examples for such specifications are pure integer vectors (e.g., in [DA09, WW05]). Some of these approaches even work on many different service properties to be matched in the same way based on these simple specifications. Some of these matching approaches then only consist of one matching step (e.g., [MS04]). In these approaches, service matching is mapped into a pure multi-criteria decision-making problem. Matching problems addressed by MatchBox are also multi-criteria decision-making problems. However, MatchBox also addresses heterogeneous service specifications, where different service properties are specified using different specification languages. This functionality raises further issues that go beyond pure multi-criteria decision-making as illustrated the preceding sections of this chapter.

As we can learn from Table 3.11, three kinds of matching processes are common: (a) There are processes starting with syntactic/structural matching (e.g., signature matching approaches) and ending with semantic/behavioral matching (e.g., condition matching approaches) [CTO10, HGEJ12a, MSZ11, MHB+12, MXB10, WWWB11]. (b) Other processes start with matching steps that match functional properties and then perform matching based on non-functional properties [ESAEA04, JRGL+05, Kon95, DA09, VHA06, WW05]. The reason for this could be, on the one hand, that syntactic/structural matching approaches are viewed as less complex and, therefore, can be used as a filter before executing more expensive matching approaches. On the other hand, they are often seen as hard constraints, while good matching results for other matching steps are "nice-to-have" but not mandatory, which again supports filtering actions. Accordingly, there are also many approaches in addition to the listed ones, that first perform matching and then ranking (e.g., [AL05, PKCH05, SZ11]). (c) Furthermore, another common combination of matching approaches is logic-based vs. non-logic-based approaches [KKF08, KK12b, KK12a, KK08]. These approaches come from the area of semantic service matching based on ontologies. "Logic-based" refers to matchers based on description logic, which is the logic underlying the most common ontology description language OWL2 [GHM+08]. These matchers are usually combined with "non-logic-based" approaches like text-similarity matching in order to reduce false positives or false negatives. However, these approaches all only address input-output matching and sometimes also conditions (PE, preconditions and effects). They have not been combined with matching approaches from another category, like protocol matching or QoS matching.

In some of the listed approaches, the control flow can be changed in many simple ways. For example, [KK09a] and [MHB+12] allow a reduction of the process, meaning that single steps can be ignored. Sycara et al. [SWKL02] provides four "modes" of matcher

combinations. Compared to MatchBox, these modes are equivalent to four different (but fixed) processes.

All these discussions explain why the Configurability columns in Table 3.11 are rather empty. The configuration possibility appearing most often is weighted aggregation (e.g., [AMM10, GTRRC08, JRGL+05, KK09b, MSZ11]). It is common for service matching approaches considering multiple service properties to allow the user to configure the weights determining the final matching result. By providing the possibility to integrate various configurable aggregation strategies, MatchBox goes substantially beyond these approaches.

Our reviews also showed that many of the approaches we sorted out use the terms "matching framework" or "discovery framework" in a broader sense. Those approaches addressed matching in combination with tasks apart from matching like monitoring the service execution [BPDRL07, CDG+06, MSZ11]. MatchBox focuses on matching only, however, in future more, extensions in these directions could become interesting in order to cover more activities relevant for service markets.

On a more general level, there are process languages like BPEL (Business Process Execution Language). BPEL is executable and allows specifying business processes and invoking web services. Based on BPEL, Mietzner and Leymann presented a generative approach [ML08] that derives processes from a specification of variations and conditions. As such approaches are much more general than MatchBox, they provide much flexibility but integrating matchers and the possibility to configure them leads to a much higher effort. One of the reasons is that the matcher interfaces are not yet defined there. In comparison, MatchBox is tailored to matching processes explicitly and, thereby, able to provide exactly those features the matching designer needs. For example, our matcher definition language is the result of extensive research regarding the variety of matcher characteristics.

3.13 Conclusion

In this chapter, we presented MatchBox, a framework that allows a matching designer to integrate and combine multiple service matching approaches. This has been achieved by a model-driven approach that incorporates well-defined matcher descriptions and matching process models that can be configured and executed. Thereby, we maintain a modularization of matchers on an abstract level to handle the complexity that comes with the combination of very different functionality. Nevertheless, aggregation strategies allow the computation of one holistic but user-friendly matching result based on the results of all combined matchers.

Our two case studies showed that MatchBox provides the matching designer with the flexibility to create many different matching variations, while the effort for integration and adaption is kept low. This benefit makes our approach broadly applicable. Future research challenges are listed in Section 6.2.

4

FUZZY SERVICE MATCHING

In Chapter 3, we present an approach to match complex service specifications using several matchers. This allows us to gain holistic and, thereby, more accurate matching results compared to a single matcher because more parts of the requirements specification can be taken into consideration. However, the more complex a specification becomes, the higher the risk of imperfection within this specification becomes. Furthermore, we have to expect even more imperfect specifications because the requirements specifications and service specifications we are dealing with are typically created by humans. As a result, in current service markets, service descriptions are very often lacking, incomplete, or unclear [HS07, SO11].

Some service properties and requirements are imperfect in their nature and cannot be exactly evaluated and specified [ŞT06]. Additional reasons for requesters to create imperfect requirements specifications include indecisiveness, high specification effort, lacking ability, as well as tolerance regarding service variations [Pla13]. For example, a requester may require a service to be "fast" (see Figure 1.3) without specifying what "fast" exactly means. Providers may provide incomplete specifications of the services they offer intentionally or unintentionally [Pla13]. As an example, recall Figure 1.3. Here, the provider did not publish details about her service's performance. This could be because some providers do not want to provide all details about their services in order to protect their business interests. Alternatively, they may not be able to specify details because they do not know them. For example, if a service depends on other (third-party) services, the provider might have difficulties to provide reliable information about quality properties like performance or availability. In general, especially quality properties of a service are typically specified vaguely [YT97].

Furthermore, requesters and providers make different abstractions when creating requirements and service specifications due to their different contexts [MPMM98]. For the same reason, we cannot assume that all requesters and all providers use the same specification languages. Much more likely, they use different specification languages, e.g., the web service description language (WSDL) or the unified modeling language (UML) because they have different backgrounds. As a consequence, model transformations into a common language are applied in order to make the specifications comparable [Ari]. Depending on the expressiveness of this common language, these transformations can be lossy such that the target specification (i.e., the transformation result) becomes imperfect. Moreover, service matching itself does not always have to be precise [ŞT06]. One reason is that service specifications are often described by complex specification languages (e.g., first-order logic for pre- and post-conditions and state charts for protocols). For such languages, solving the matching problem can be inefficient or even not decidable [PvDB+13]. However, during

service discovery or service composition, typically a wide range of services needs to be matched. Thus, the efficiency of each single matching run is crucial for the scalability in a large service market. There are matching approaches that try to cope with these issues by simplifying the matching problem [PvDB+13], e.g., approximative matching approaches or approaches based on heuristics. These simplifications are at the expense of the accuracy of the matching results. However, this is regarded as acceptable compared to the risk of getting no results at all [MPMM98].

In all these cases, matching becomes *fuzzy* because information needed to calculate a precise matching result is missing. Note that we use a broad definition of the term "fuzziness" as a collection for different types of imperfect information including gradedness, vagueness, imprecision, variability, incompleteness, approximations, and uncertainty. Current matching approaches lack in coping with this problem as they make unrealistic assumptions, i.e., that the available information is specified and processed in a complete and perfect way [PvDB+13]. As a consequence, existing matching approaches deliver adulterated matching results that do not notify the users about the fuzziness that emerged. This illusion of certainty leads to an unassessable risk of false positives (mismatching services incorrectly determined as good matches) or false negatives (well matching services incorrectly determined as mismatches). For example, if a matching approach delivers "50%" as a matching result, this can mean a) "the service specification matches 50%" or b) "we are 50% certain that the service specification matches". These two cases need to be distinguished because Case a) denotes that we certainly know that half of the requirements specification is satisfied, while Case b) denotes that we cannot be sure to which extent the service matches. Taking signature matching as an example, Case a) could mean that the service's input parameters match but the outputs do not. Case b) could mean that either inputs or outputs are completely unknown such that it is not possible to assess the risk in buying and using that service. An example of an approach that mixes these two cases is presented by Zilci et al. [ZSK15]. This approach is pessimistic in a way that, a missing part of a specification is treated in the same ways as a mismatch even though the extent to which the service matches is actually unknown, which is not communicated to the user. Furthermore, existing approaches do not properly distinguish between different sources and different forms of fuzziness as shown in our examples. This complicates the understanding of the matching results once more. According to these issues, we need to find a way to deal with unknowns, and how to support the user in making decisions in the presence of these unknowns [RS03].

The Fuzzy Matching concepts proposed in this thesis introduce a more transparent way to cope with imperfect information and imperfect information processing. Fuzzy matching deals with fuzziness occurrences that (potentially) lead to fuzzy matching results, classifies them on the basis of well-defined *fuzziness sources* and *fuzziness types*, and quantifies them. This is accomplished by incorporating formal concepts and tools from fuzzy logic. These allow for capturing and modeling various types of imperfect information within a unified and coherent mathematical framework. The matching results computed based on these principles inform the user about the fuzziness that emerged. Thereby, fuzzy matching improves the user's decision-making based on the computed matching result. In particular, it allows us to distinguish between the two statements a) and b) discussed above. Psychological and economic studies underline the importance of this benefit (see [ANW09] for an overview). For example, Ellsberg's studies [Ell61, Ell15] show that people prefer taking a known risk rather than taking an unknown risk even though the probability for a better outcome may

be low. Lipshitz and Strauss point out that coping with uncertainty "lies at the heart of making a decision" because uncertainty constitutes a "major obstacle to effective decision-making" [LS97]. Also Frisch and Fox agree that human choices often depend on how much relevant information is missing [FB88, FT95]. Our results address this issue by turning the unknown risks of adulterated matching results into known risks by informing about induced fuzziness. All in all, we enable service matching under more realistic circumstances compared to the related work: We are able to explicitly deal with imperfect service specifications, which makes our approach well applicable in practice.

In general, our approach extends traditional approaches from the areas of service discovery and component retrieval. The targeted end users of our approach depend on the service market's scenario as either service requesters, service providers, or intermediaries (brokers) could be in charge of the matching approach. In particular, requesters get the benefit of an improved decision-making when selecting between services due to the awareness of present fuzziness. Corresponding to their risk aversion and ambiguity aversion [Haz89], they become able to choose either services that are certainly a good match or services that may potentially (but not necessarily) be perfect matches. On the other hand, if providers obtain these matching results too, they could profit from additional information regarding how to raise interest in their services. For them, the matching results indicate whether service specifications need to be improved in order to lead to less fuzzy matching results, satisfying a greater range of requesters. In this chapter, we condense the roles interpreting the matching result into one role called the *user*. The user can be represented by different market participants and it can be a human or also a machine, i.e., an artificial intelligence.

Concepts presented in this chapter have been published in [Pla13, PvDB$^+$13, PAPS15] and are under revision in [PSB$^+$]. They are based on the Master's Theses by Vijapurwala [Vij14] and Merschjohann [Mer14], and the Bachelor's Theses by Bunse [Bun14], Börding [Boe15], Neumann [Neu15], and Bruns [Bru15].

This chapter is structured as follows. In Section 4.1, we summarize the most important scientific contributions provided by this chapter. Section 4.2 introduces foundations about fuzzy logic needed to follow the concepts described in this chapter. In Section 4.3, we derive requirements for our fuzzy matching concepts. The classification of fuzziness sources and types is described in Section 4.4. Based on this, we introduce a general fuzzy matching procedure in Section 4.5, while Section 4.6 builds on this procedure and describes two concrete approaches for fuzzy matching. We briefly present our prototype implementing the introduced concepts in Section 4.7. The validation described in Section 4.8 is based on this prototype. Section 4.9 highlights limitations of the concepts presented in this chapter. Section 4.10 presents our survey about related approaches and we draw conclusions in Section 4.11.

4.1 Scientific Contributions

The scientific contributions of this chapter can be summarized as follows:

- This chapter connects concepts from the area of service matching with the areas of fuzzy logic from mathematics and theoretical computer science as well as decision theory and uncertainty theory from humanities and social sciences like psychology.

Thereby, we brought together research areas that seldom met before, leading to new results.

- We are the first to distinguish between different fuzziness types in service matching, e.g., fuzziness in the sense of graded user satisfaction and fuzziness in the sense of incomplete information about service properties.
- We constructed a new fuzzy matching method that combines these types of fuzziness in a methodologically sound manner. This method provides matching results consisting of a degree of necessity and a degree of possibility of meeting the requirements, i.e., a lower and an upper bound regarding the degree of user satisfaction. In contrast, related methods produce a single degree as a matching result, that is often chosen arbitrarily and lacking a clear semantics.
- Furthermore, we are the first to quantify the extent of further fuzziness sources, e.g., transformation-induced fuzziness. Thereby, we support the user in decision-making based on her own risk/ambiguity aversion.
- We are the first in the area of service matching to explicitly investigate and discuss the characteristics a fuzzy matching result should have, incorporating concepts from decision theory.
- We evaluate our fuzzy matching approach on the basis of real service data.

4.2 Foundations of Fuzzy Modeling

In the following, we discuss the foundations of our approach with a focus on fuzzy sets and possibility theory.

4.2.1 Fuzzy Sets and Fuzzy Logic

A fuzzy subset A of a reference set U is identified by a *membership function* μ_A [Zad65]. For each element $x \in U$, μ_A specifies the degree of membership of x in the fuzzy set A. Membership degrees $\mu_A(x)$ are in the interval $[0, 1]$. Based on this, fuzzy sets formalize the idea of graded membership. Considering the set of services with good reputation as an example, any sharp boundary in the form of a threshold on the average rating will appear rather arbitrary. Modeling the concept as a fuzzy set A, we are able to express non-sharp boundaries. For example, a service with rating 4.5 could be viewed as completely good ($\mu_A(4.5) = 1$), a rating of 3.7 as 'more or less' good (e.g., $\mu_A(3.7) = 1/2$), and 2.5 as definitely not good ($\mu_A(2.5) = 0$). Fuzzy sets are often associated with natural language expressions and used to capture their meaning in a precise, mathematical form. Thus, they represent a human-machine interface between a symbolic and a numeric level of modeling.

However, a fuzzy set can have different semantic interpretations [DP97]: On the one hand, the membership degrees of a fuzzy set can be interpreted in terms of *preference*, on the other hand, they can represent *uncertainty*. As an example, consider the fuzzy set A of services with a good reputation, formalized by a membership function μ_A on $U = [0, 5]$. As a requirement by a service requester, $\mu_A(x)$ can be interpreted as the degree to which the requester is satisfied with a service having an average rating of x. As opposed to this, if 'good' is given by a service provider as (imprecise) information about the reputation of a

provided service, $\mu_A(x)$ is the degree to which x is considered *possible* as the true value of the average rating.

To operate with fuzzy sets in a formal way, fuzzy logic offers logical connectives, e.g., a class of operators called *triangular norms* (t-norms) [KMP02]. A t-norm $\top$ is a $[0,1] \times [0,1] \longrightarrow [0,1]$ mapping. It can be used as a generalized logical conjunction and to define the intersection of fuzzy subsets A, B as follows: $\mu_{A \cap B}(x) = \top(\mu_A(x), \mu_B(x))$ for all $x \in U$. The standard negation operator $\alpha \mapsto 1-\alpha$ can be used to model the set-theoretical complement: $\bar{A} = U \setminus A$ of A in U: $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$ for all $x \in U$.

4.2.2 Possibility Theory

Possibility theory is a general uncertainty calculus with a close connection to fuzzy logic because possibility distributions are often derived from fuzzy sets by interpreting membership degrees in terms of degrees of possibility [Zad78]. Formally, a *possibility distribution* π is a mapping $U \longrightarrow [0,1]$. A distribution of that kind induces a *possibility measure* defined by the supremum of π for all $A \subseteq U$. The measure Π captures uncertain information about $u_0 \in U$: For each subset $A \subseteq U$, $\Pi(A)$ is the degree of plausibility that $u_0 \in A$.

A a lack of information can be captured more adequately by means of possibility than by probability distributions. In particular, complete ignorance is adequately formalized by the distribution $\pi \equiv 1$. For example, if nothing is known about the true reputation u_0 of a service, for example, because the service is completely new in the market, then each value $x \in U = [0,5]$ is fully plausible. In probability theory, this situation could be modeled by the uniform distribution with density $p(x) \equiv 1/5$, the same distribution that also models perfect knowledge about the equal probability of each value. Thus, in this situation, standard probability theory cannot distinguish between a complete ignorance and complete knowledge. In contrast, in possibility theory, implausibility can be captured by a *necessity measure*. A subset A is considered necessary to the same extent to which the complement of A is considered implausible. The domain of possibility/necessity measures for fuzzy subsets are defined as follows:

$$\Pi(A) = \sup_{x \in U} \min\big(\pi(x), \mu_A(x)\big) \ , \tag{4.1}$$

$$N(A) = 1 - \sup_{x \in U} \min\big(\pi(x), 1 - \mu_A(x)\big) \ . \tag{4.2}$$

All in all, in contrast to probability theory, possibility theory can model several kinds of uncertainty [DFMP04], e.g., vagueness and incompleteness. As shown above, given a uniform probability distribution, we cannot tell pure randomness and ignorance apart, i.e., we cannot distinguish between uncertainty due to variability and uncertainty due to missing information. In contrast, possibility distributions can be used to explicitly model full or partial ignorance, i.e., the lack of knowledge. In particular, the possibility and necessity measures defined in possibility theory are useful for representing matching results as we will show later in this chapter

When the available information to be processed is frequentist, probabilistic modeling is natural [DFMP04]. In contrast, possibility theory is better suited than probability theory if we do not have statistical data. The latter is most often the case in service matching.

Furthermore, in contrast to probability theory, possibility theory does not require a numerical setting.

Another important criterion in our context is efficiency: Possibilistic problems can be solved much more efficiently than stochastic problems [IR00]. One reason for this is that the possibilistic representation is weaker because possibility measures are based on an ordinal structure rather than an additive one. [DFMP04]

However, there are transformations possible between both kinds of measures due to the Possibility/Probability Consistency Principle [Zad78]. A transformation from a probability measure to a possibility measure looses information. However, according to Dubois et al. [DFMP04], turning a probability measure into a possibility measure may be useful if we deal with other weak sources of information. We make use of such a transformation in Section 4.6.1.3.

4.3 Requirements

From the challenges described in the beginning of this chapter, we can derive seven requirements that an appropriate fuzzy matching solution has to satisfy:

(*R1*) *Deliver Unadulterated Matching Results:* In the presence of fuzziness, an appropriate matching approach should still deliver matching results that are not misleading in a way that they inevitably lead to false positives and false negatives. No false certainty should be pretended.

(*R2*) *Handle Different Types and Sources of Fuzziness:* A fuzzy matching approach has to cope with different sources of fuzziness, e.g., requester-induced fuzziness, provider-induced fuzziness, algorithm-induced fuzziness, and transformation-induced fuzziness. Furthermore, different types of fuzziness also need to be distinguished, e.g., vagueness, gradedness, and uncertainty.

(*R3*) *Make Source of Fuzziness Transparent:* The user needs to know which source of fuzziness a matching result is inflicted with, in order to be able to undertake potential countermeasures, if needed. For example, in the presence of provider-induced fuzziness, the provider could check whether she can and should provide more detailed information within the provided service specifications. Thus, the source of fuzziness needs to be returned along with the matching result.

(*R4*) *Make Extent of Fuzziness Transparent:* In order to assess the usefulness of a returned matching result, the user needs to know how much fuzziness emerged. For example, if the fuzziness is very high, she may decide to not take the risk and turn to matching results for other services.

(*R5*) *Support Diverse Matching Approaches:* There is a variety of matching approaches that work in very different ways. They are based on different specification languages (e.g., automata vs. text-based specifications vs. numerical specifications) as well as on different theoretical principles (e.g., graph matching vs. first-order logic vs. fuzzy logic). Thus, we need fuzzy matching concepts not only for one specific kind of language and one kind of underlying theory. Instead, we need a more generalizable concept that works for several approaches with common properties.

(*R6*) *Support Complex Specification Languages:* As shown in the example scenario, specifications are not trivial, e.g., they do not only consist of single numbers and a simple integer comparison is not enough. In reality, they are heterogeneous and consist of a more complex structure. Our fuzzy matching concepts need to be applicable to such complex specifications.

(*R7*) *Efficiency:* Our matching approaches need to be utilized within different kinds of service markets, including large, dynamic markets. Thus, these matching approaches have to be sufficiently fast, such that service discovery in such markets is still scalable.

(*R8*) *Non-Invasiveness:* As many matching approaches for service specifications already exist and it is not practical to "reinvent the wheel", we need concepts for fuzzy matching to be used as an extension for existing non-fuzzy approaches.

All these requirements can be summed up to the general goal of providing a general approach that provides the user with valuable information about the matching result while imperfect information is taken into account.

Note that this thesis does not aim at fully satisfying all listed requirements. For example, a detailed evaluation of efficiency is beyond the scope of this thesis and, therefore, left for future work. Nevertheless, it is important to take such requirements into account when developing appropriate concepts. We also describe one experiment that addresses efficiency indirectly (see Section 4.8.2.3).

4.4 Fuzziness Classification

In order to investigate different manifestations of fuzziness as well as different sources inducing fuzziness, we conducted an initial systematic literature review about how related matching approaches consider fuzziness [PvDB+13]. On the basis of this survey and the market scenario described in Section 1, we define *fuzziness types* as well as *fuzziness sources* in this section. Furthermore, we discuss *fuzziness occurrences* that can be seen as instances of fuzziness sources and types. Later sections build on this classification.

4.4.1 Fuzziness Types

We refer to *Fuzziness Type* as a kind of imperfect information or imperfect information processing. Figure 4.1 shows an overview of different fuzziness types and their interrelations. In the following, we describe each fuzziness type with reference to service matching and the involved roles.

4.4.1.1 Vagueness / Imprecision

Vagueness, respectively Imprecision, can occur in the requirements specification as well as in the provided service specification. An expression is vague if it has more than one possible interpretation [Zha98].

For example, requirements on services are often specified vaguely, e.g., using natural language expressions such as "close to 4 stars" (see *Requester-induced Fuzziness*, Section 4.4.2). Especially non-functional requirements are usually vague and imprecise in

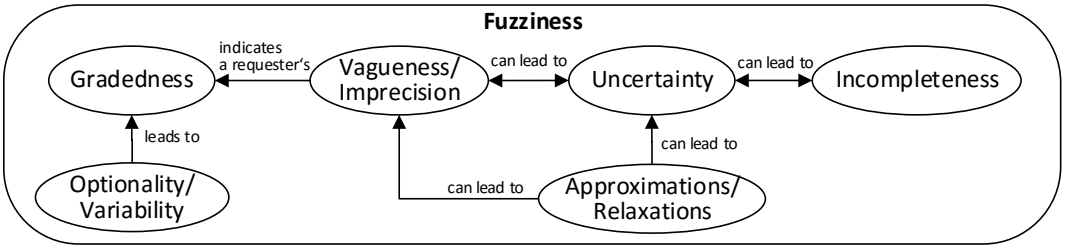


Figure 4.1: Fuzziness Types and their Interrelations

nature [YT97]. Such vague expressions are not immediately amenable to computational processing. In the provided service specification, vagueness may appear if a provider is not sure about the precise value of a certain property, or if she intentionally tries to obscure certain details (see *Provider-induced Fuzziness*, Section 4.4.2.1).

Vagueness within a requirements specification may indicate a requester's Gradedness. Furthermore, vagueness may lead to Uncertainty.

4.4.1.2 Optionality / Variability

We refer to Optionality, respectively Variability, if a requester indicates that there are several options to satisfy her requirements or if the provider offers several alternatives in her service specification.

For example, a requester may state that she wants a service with a price of either "100 Euros" or "10 Euros per month" or "one Euro per invocation". In another example, a provider may offer a service with either "a response time of 1ms for a price of 10 Euros per month" or "a response time of 50ms for free".

Optionality leads to Gradedness if multiple options satisfy the requester's requirements to different degrees.

4.4.1.3 Gradedness

In this thesis, the term Gradedness always refers to someone's satisfaction. Within the scope of service matching, it is the requester whose satisfaction is the main subject. Thus, in the following, gradedness always refers to the requester.

The Vagueness of a requirements specification is in direct correspondence with the Gradedness of a requester's satisfaction. Typically, a requester will not only distinguish between good and bad services. Instead, a service can match the requirements *to some degree*, such that the requester can be *partially* satisfied with a service. [PSB$^+$]

For example, requesters are often tolerant with regard to slight deviations from the requested characteristics (see *Requester-induced Fuzziness*, Section 4.4.2.2).

4.4.1.4 Incompleteness

Incompleteness in a provided service specification shows cases, where the provider did not deliver all possible information on a service. Incompleteness within specifications can reach various dimensions. For example, the provider might not offer a behavioral specification of the service, including pre- and post-conditions and protocols (high amount

of incompleteness); alternatively, single values within a privacy specification could be unspecified (low amount of incompleteness).

Such incomplete specifications can be due to multiple reasons (see *Provider-induced Fuzziness*, Section 4.4.2.1). Furthermore, incompleteness often leads to Uncertainty during the matching procedure as information needed to determine a correct matching result may be missing.

Incompleteness in requirements specifications can be interpreted as irrelevance of certain service properties. These properties then do not need to be matched as the requester has no constraints related to them. Thus, only incompleteness within a provided service specification is relevant for fuzzy matching.

4.4.1.5 Approximations / Relaxations

Unlike the other fuzziness types, Approximations and Relaxations do not refer to the underlying imperfect information but to the imperfect processing of (potentially completely precise) information. Algorithms in general and matching algorithms in particular apply approximations and relaxations due to multiple reasons (see *Algorithm-induced Fuzziness*, Section 4.4.2.3).

Approximations within an algorithm potentially introduce Uncertainty as they lead to the risk that the result has been distorted because of the approximations during its calculation. In addition, a requirement relaxed during the matching phase can become a vague requirement.

4.4.1.6 Uncertainty

In the literature, there are various definitions of Uncertainty or, in other words, partial ignorance [WHR+03]. We adopt a general definition given by Walker et al. [WHR+03] and taken up by Perez-Palacin and Mirandola [PPM14]: Uncertainty is "any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system.". This definition was intended for the research area of software modeling and it also fits within the scope of our service matching concepts as they are based on models of services and requirements.

The literature in decision-making theories reports that uncertainty constitutes a major obstacle to effective decision-making [LS97]. This is also true for the user of service matching approaches. Uncertainty is a problem during service matching in particular because it leads to the fact that the user's interpretation of the matching result is not based on "correct" data any more. In current matching approaches, uncertainty is not transparent to the user, such that a matching result may become misleading and the user's risk is not assessable.

There are various reasons for why properties of a service or their specification can be afflicted with uncertainty. As explained above, most of the already introduced fuzziness types lead to uncertainty. Moreover, uncertainty literature distinguishes between *aleatory uncertainty* (or *variability uncertainty*) and *epistemic uncertainty* [PPM14, WHR+03]. While the first kind refers to data that is stochastic in nature (e.g., reputation data), the second refers to a lack of knowledge (e.g., incompletely specified privacy policies). This distinction is required as it influences the possibilities to deal with uncertainty: Epistemic uncertainty can be reduced by providing more knowledge, which is not the case for aleatory uncertainty. However, in both cases, there is a benefit of making the user aware even of

irreducible uncertainty as the awareness enables the user to assess her risk when acting on the basis of a given matching result.

Current literature distinguishes between five orders of ignorance/uncertainty [PPM14, Arm00], of which the first three can be immediately transferred to fuzziness in service matching:

- **0th order of uncertainty:** The lack of uncertainty, i.e., the user knows exactly how well a service matches the requirements.
- **1st order of uncertainty:** The lack of knowledge (i.e., known uncertainty), i.e., the user knows that she does not know exactly how well a service matches.
- **2nd order of uncertainty:** The lack of knowledge and the lack of awareness, i.e., the user does not know that she does not know how well a service matches.

The 0th order represents the ideal case. However, this case cannot always be achieved, due to the reasons explained above. Existing (fuzzy) service matching approaches lead to the 2nd order of uncertainty because there, uncertainty is not reflected within the matching result and the user is lead to believe in a potentially adulterated matching result. By estimating and presenting the extent of induced uncertainty to the user, our fuzzy matching concepts presented in this thesis reduce the level of uncertainty from the 2nd order to the 1st. As a benefit, the user obtains the opportunity to cope with the known uncertainty and to assess the risk of an uncertain matching result. Furthermore, the user may even get to the 0th order by eliminating the known uncertainty, provided that we deal with epistemic uncertainty. Reaching the 1st order is essential to enable strategies to completely eliminate uncertainty. Thus, our goals are also in line with Garlan, who states that "uncertainty needs to be considered as a first-class concern to be dealt with systematically because we live in a world where we cannot hope to achieve perfection, so, we must rethink many of the ways in which we conceive, engineer, and validate our software-based systems" [Gar10].

4.4.1.7 Further Properties of Fuzziness Types

Different fuzziness types can be specified and determined using different mathematical concepts, e.g., fuzzy logic or probability theory. Different concepts con cope differently well with different fuzziness types. Section 4.2 goes into more detail regarding these concepts and their advantages and disadvantages.

To sum up, many fuzziness types lead to Uncertainty. Hence, we need to cope with all these fuzziness types because uncertainty can lead to serious problems in service matching, as discussed. Nevertheless, we need to distinguish between different fuzziness types because they all come with different characteristics and require different actions

From decision-making literature, we can learn that coping with uncertainty can be done applying three basic strategies: reducing uncertainty, acknowledging uncertainty, and suppressing uncertainty [LS97]. Later in this chapter, we adopt these uncertainty coping strategies to coping with fuzziness types in service matching including especially (a) to inform the user and (b) to find and apply appropriate strategies to reduce fuzziness, if necessary and if possible. By these two countermeasures, we can support the user in decision-making.

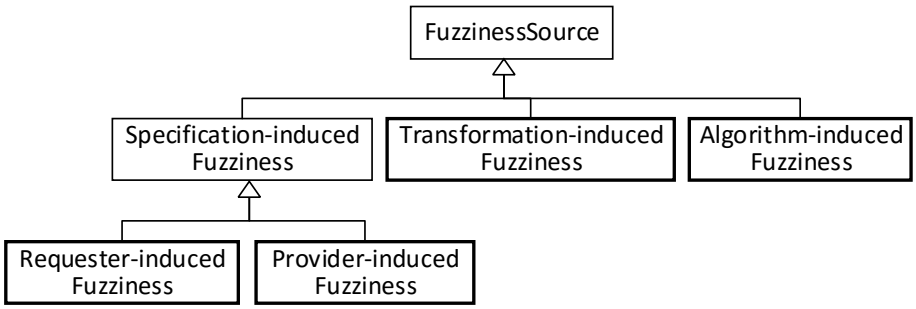


Figure 4.2: Classification of Fuzziness Sources

4.4.2 Fuzziness Sources

The distinction between different fuzziness types is related to the distinction between different fuzziness sources. A *Fuzziness Source* is a potential cause for fuzziness. It describes a specific situation that possibly may lead to a fuzzy matching result. Depending on occurring fuzziness types (see Section 4.4.1), a fuzzy matching result is either a gradual matching result, an uncertain matching result, or both.

In this thesis, we distinguish between four main fuzziness sources depicted in Figure 4.2: Provider-induced Fuzziness, Requester-induced Fuzziness, Algorithm-induced Fuzziness, and Transformation-induced Fuzziness. Provider-induced Fuzziness and Requester-induced Fuzziness can be subclassified again into Specification-induced Fuzziness because they both originate within the input specifications. In the following, each fuzziness source is explained on the basis of our initial definitions given in [Pla13].

4.4.2.1 Provider-induced Fuzziness

Provider-induced fuzziness originates within the service specification delivered by the service provider. Service providers do not always provide complete and precise service specifications [KKRKS07].

There are various reasons for providing imperfect service specifications [Pla13]:

- Providers do not want to present details of the realization of their offered services in order to protect business interests,
- they do not know all details about their services' characteristics themselves because some characteristics depend on a variety of influencing factors,
- they only know their perspective and have no overview of all options of how a service might be used by other parties, or
- they are not able or not willing to specify their knowledge in an appropriate (e.g., a machine-interpretable and comprehensive) specification language as this requires a lot of effort and extra knowledge.

For example, map service providers, such as Google, may not be willing to provide the precise resolution of their maps within a publicly available specification. In another example, a room management service provider cannot give details about the service's response time or

its availability because it relies on third-party servers providing room databases [BBM10]. In addition, providers often do not specify non-functional properties, e.g., privacy policies, formally. In reputation matching, provider-induced fuzziness occurs for new services with only few ratings so far, or because of unrateability of a service that has only been used as part of a composition [dSZ13]. For example, if a service consists of several image processing services, not each of these sub-services' effects may be immediately visible to the user.

Summing up, the origin for provider-induced fuzziness is the providers' intentional aversion as well as their disability or missing expertise regarding specific service properties or specification languages. Accordingly, service specifications are often expected to be incomplete or imprecise.

As a consequence, especially a provider can benefit from the knowledge whether provider-induced fuzziness emerged in a matching result because this source for uncertainty can potentially be eliminated or at least be reduced by improving the provided service specifications. Depending on the reason for the provider-induced fuzziness, this can be done by the service provider herself, or also by other roles performing additional analysis, e.g., different kinds of performance prediction [SDMIS04, BKR09], if suitable models are available. A service provider may be willing to improve her specifications if this effort promises better revenues. However, note that this is only possible in the case of epistemic uncertainty and not in the case of aleatory uncertainty (see Section 4.4.1). For example, if the uncertainty is due to the stochastic nature of reputation data, the provider is not able to improve the situation.

4.4.2.2 Requester-induced Fuzziness

Requester-induced fuzziness refers to a requirements specification delivered by a service requester that contains vague requirements or further indications that she tolerates certain variations [Pla13].

As an example, the requester could specify soft thresholds for QoS properties. For example, she could request a maximal response time of 5 seconds for the required room management service, but the best matching service could have a maximal response time of 5.25 seconds. In this case, this service could nevertheless be of interest for the requester, at least to some extent. Similarly, the requester can use a fuzzy term and just specify the requested response time to be "fast". Further reasons for requesters to specify their requirements incomplete or imprecise include high specification effort, indecisiveness, and lacking ability [EKM11].

In contrast to provider-induced fuzziness, requester-induced fuzziness does not lead to uncertainty in the general case. Instead, it may reflect a requester's gradedness towards the satisfaction of her requirements specification. As a consequence, requester-induced fuzziness does not need to be eliminated in general. However, in order to cope with gradedness, a gradual matching result format is beneficial.

Accordingly, requester-induced fuzziness is the only fuzziness source that does not inevitably lead to uncertainty during the matching procedure. However, requester-induced fuzziness can occur in combination with transformation-induced fuzziness (see Section 4.4.2.4). For example, the transformation of fuzzy terms like "fast" into a formal construct could be uncertain. In these cases, requester-induced fuzziness is indirectly involved into the emergence of uncertainty.

4.4.2.3 Algorithm-induced Fuzziness

Fuzziness may not only be introduced by the requester or the provider, but also by the matching approach itself [Pla13]. For example, some matching algorithms are based on heuristics or on other kinds of approximations or simplifications [PRVM13]. In these cases, the matching results inevitably become uncertain. One reason to introduce such a fuzziness is that complex specification languages require matching algorithms with a high computational complexity. For example, some matching approaches rely on subgraph matching, which is NP-hard [CF11]. Similarly, pre- and post-condition matching is often based on expensive SMT-solving [WHdM09]. In such cases, approximations are necessary to keep the matching process efficient and the whole service discovery scalable.

Just like Requester-induced Fuzziness and Provider-induced Fuzziness, also Algorithm-induced fuzziness induced into a matching procedure should be made transparent to the user. Only then can she assess the risk within the uncertain matching result that is produced by the matching algorithm.

4.4.2.4 Transformation-induced Fuzziness

Usually, requirements specifications as well as service specifications need to be (partially) transformed into the same specification language in order to enable matching without having to define matching algorithms for numerous combinations of specification languages [Ari]. One reason is that the different backgrounds of the involved requesters and providers may lead to the usage of different specification languages that are not immediately comparable with each other. For example, an exam management service's protocol could be described with hierarchical state charts, while the required room management service could be described with a protocol on the basis of petri nets.

The specification language used for matching needs to adhere to certain characteristics. For example, it needs to be machine-readable such that automated matching is possible. In addition, it should not be too expressive such that specifications written in this language can still be matched using efficient algorithms. The expressiveness of the common target language and the degree to which matching can be (efficiently) automated become a trade-off. [Pla13]

As a consequence, even if both the requirements specification and the service specification are instances of the same specification language, transformations may become essential. This issue becomes even more serious as specifications are typically constructed by human users, which raises the need for languages that are easy to learn and easy to read for humans. However, such languages are often not appropriate for service matching. For example, natural language requirements specifications are easy to use for humans as no additional knowledge is required, but they are not interpretable by most matching approaches. Such specifications need to be transformed into a language with formally defined semantics (see Section 5.3).

However, each transformation between specification languages contains risks. For example, the target language of the transformation could be less expressive than the source language (which is realistic considering the trade-off regarding expressiveness and efficiency mentioned above). For example, petri nets, which allow parallel control flow, are not totally mappable to the simpler (but easier matchable) flat automata. In such a case, information

gets lost during the transformation. [Pla13] This is a problem because the matching result is based on the transformed specification and does not reflect the compliance of the original specifications. Thus, the user cannot rely on a matching result based on such specifications. This situation is worsend as transformations are often invisible to the user so that she has no idea of the adulteration of the matching result.

Yet, this is not the only situation where transformation-induced fuzziness leads to uncertainty. Consider the case where the target language is more expressive than the source language. Even though there is no information loss, the mapping of language constructs might still be unclear. For example, using the fuzzy reputation matching approach presented in Section 4.6.1, a requirement "approx. larger than 4 stars" is transformed into the membership function for a fuzzy set. However, it is unclear what the membership function should exactly look like because different users may have different opinions and preferences [HLL+05]. For example, is 3.5 stars still accepted in the above mentioned case or not? In this example, the target language is more expressive than the source language, but the transformation still leads to uncertainty.

In general, transformation-induced fuzziness can be subdivided into fuzziness induced within the transformation of the requirements specification and fuzziness induced within the transformation of the provided service specification. It can occur in combination with requester-induced fuzziness or with provider-induced fuzziness, but also separately.

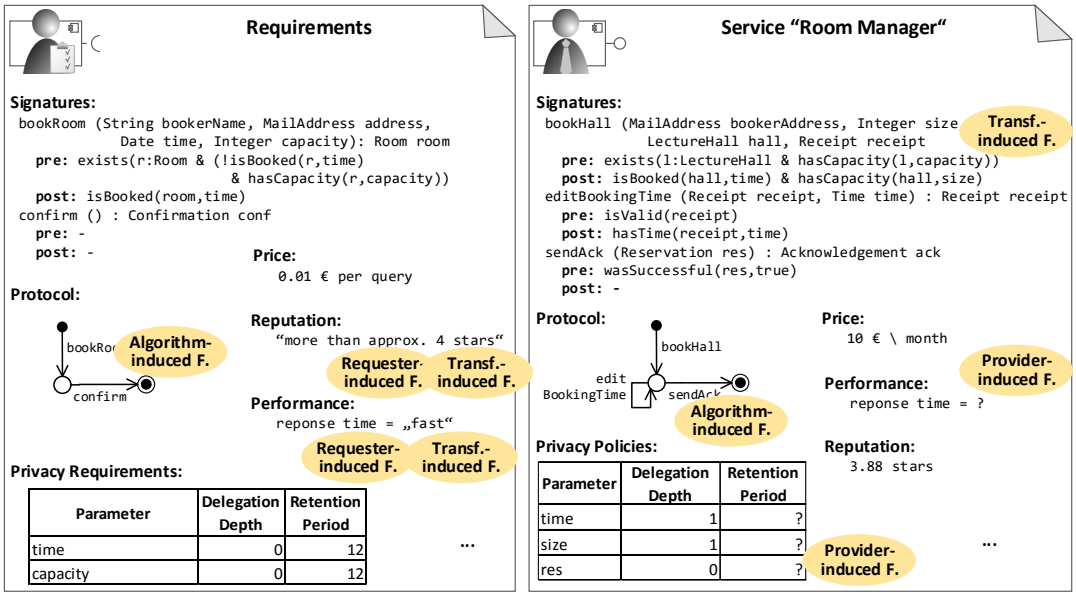
Transformation-induced fuzziness can only be detected on instance-level, i.e., considering concrete specifications and not only the definition of the transformation itself on language-level [Bun14]. Accordingly, we can detect it only at runtime but not at design time. However, the transformation itself indicates *potential* occurrences. This issue is explained in detail below.

4.4.3 Fuzziness Occurrences

The fuzziness types and sources introduced above are detected semi-automatically within the scope of a fuzzy matching approach as we propose in Section 4.5. If a fuzziness source is detected and if this source leads to a fuzzy matching result, we speak of a *Fuzziness Occurrence*. There can be multiple fuzziness occurrences of the same or of different fuzziness sources and of different fuzziness types in one matching procedure.

As an example, consider Figure 4.3, which shows the example specifications described in Section 1 with annotated fuzziness sources. Each annotation represents a potential fuzziness occurrence in a simplified way. For example, Requester-induced Fuzziness occurs where the requirements specifications contain imperfect information, i.e., the soft threshold in the reputation requirements ("approx. 4 stars") and the fuzzy term in the performance requirements ("fast"). On the other hand, Provider-induced Fuzziness occurs where the provider delivered imperfect information, i.e., the missing knowledge in the performance properties ("response time = ?") and the incomplete privacy specification (retention period = [?,?,?]).

Please note that Figure 4.3 abstracts from the fact that Algorithm-induced Fuzziness and Transformation-induced Fuzziness cannot be detected within the input specifications alone. As their names suggest, the matching algorithm and the transformations respectively need to be considered, too. Given a protocol matching algorithm that applies simplifications, the protocols may lead to Algorithm-induced Fuzziness. Similarly, given a provider's



(a) Requirements Specification (b) Service Specification

Figure 4.3: Example Specifications with Annotated Fuzziness Sources

specification language that is able to describe signatures more expressively than the language used for matching, the signatures may lead to Transformation-induced Fuzziness.

Potential fuzziness occurrences of different fuzziness sources and types occurs in different points in time during the matching scenario. An example scenario is shown in Figure 4.4. The depicted sequence diagram shows at which times during the interactions which fuzziness source occurs. Provider-induced fuzziness occurs directly during the provider's specification of her services. Analogously, requester-induced fuzziness occurs while the requester specifies her requirements. Only after that can transformation-induced fuzziness occur. The fuzziness source that comes into play last is algorithm-induced fuzziness.

Furthermore, we need to distinguish between *potential fuzziness occurrences* and *relevant fuzziness occurrences* because a potential fuzziness occurrence is not always problematic during the matching procedure. Fuzziness occurrences may have no effect at all, so that the matching result is not inflicted with fuzziness/uncertainty even though there is a potential fuzziness source involved. For example, the transformation of the specification language a provider uses for signatures may be lossy with respect to a specific language construct (e.g., cardinalities of parameters). However, if the specifications to be transformed do not use this language construct, then no fuzziness arises. In other words, you can only detect a relevant fuzziness source on the instance level (within the concrete specifications) and not on the meta level (within the language definition) [Bun14].

Another example for the difference between potential fuzziness occurrences and relevant fuzziness occurrences is provider-induced fuzziness: Even though a provider may provide an incomplete specification lacking certain information about a provided service, this incompleteness has no effect if the requester's requirements specification does not address these information anyway. For example, if a provider gives no or only imprecise information on performance, fuzziness only emerges if the requester has no performance-related

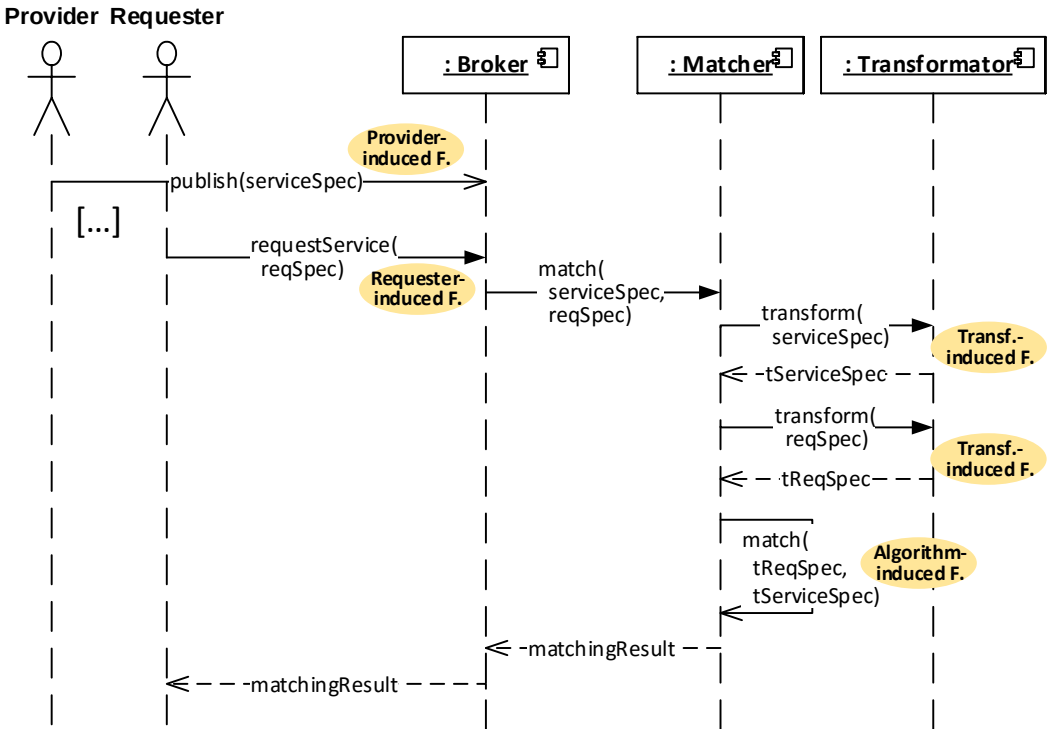


Figure 4.4: Temporal Order of the Origins of Potential Fuzziness Occurrences

requirements. As a consequence, the relevance of a fuzziness occurrence can only be judged during matching or on the basis of knowledge of the matching algorithm to be performed.

Although the fuzziness occurrences annotated in Figure 4.4 are not necessarily relevant fuzziness occurrences, we can learn that the number of potential fuzziness occurrences depends on the concrete market scenario. For example, consider the following four cases: (1) In scenarios, where the matcher gathers data from further parties or components, further fuzziness occurrences could come into play. (2) The broker could reject all imperfect requirements specifications or all imperfect specifications of provided services such that the corresponding fuzziness occurrences can be avoided in the first place. (3) Transformation-induced fuzziness does not emerge in a market scenario where transformators are not included such that requesters and providers must use fixed specification languages. (4) If the selection based on the matching result is not made by the human but by an algorithm (e.g., as part of a composition algorithm), this could induce additional fuzziness occurrences within this algorithm.

Independent from the temporal order of their origins, fuzziness occurrences can influence each other: For example, a specification that is incomplete due to provider-induced fuzziness contains less risks with respect to transformation-induced fuzziness because less constructs have to be transformed. Thus, if a provider manages to reduce provider-induced fuzziness, i.e., if she extends the specification, transformation-induced fuzziness could become worse. Similarly, if algorithm-induced fuzziness is reduced by modifying the algorithm, this could mean that provider-induced fuzziness occurrences could become more relevant if the modified algorithm addresses more missing or imprecise elements from the provided

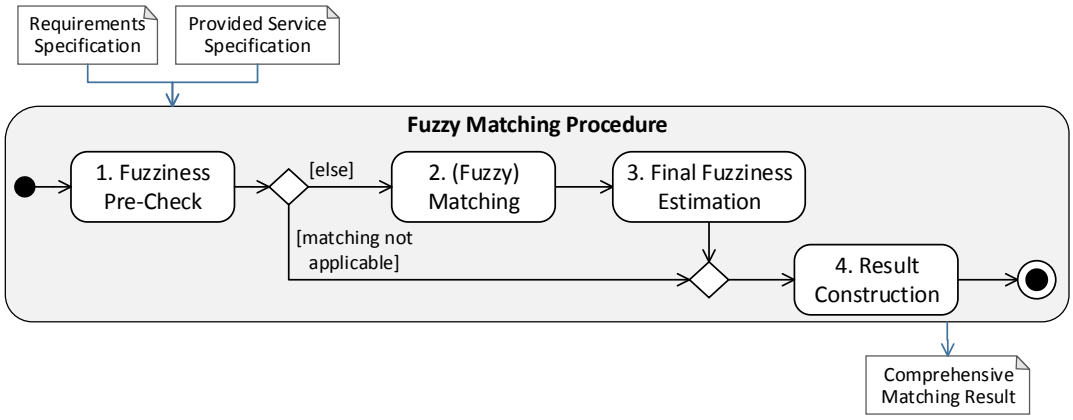


Figure 4.5: Overview of the Fuzzy Matching Procedure

specification, as the original algorithm. Nevertheless, in both examples, the general situation was improved because the overall extent of potential fuzziness is reduced and the occurrences that became relevant can potentially be extinguished in an easier way.

4.5 Fuzzy Matching Procedure

As discussed above, we need to take care of fuzziness induced into the matching procedure. For this purpose, we developed a general fuzzy matching procedure to be applied in combination with any matching approach. Figure 4.5 depicts the steps to be performed within this procedure in their general form. For actual usage, this procedure needs to be "instantiated" as shown on the basis of two examples in Section 4.6.

1. Fuzziness Pre-Check

In order to judge whether matching is applicable and useful at all, a first estimation about the presence of potential fuzziness occurrences is performed. If the fuzziness is too high (e.g., the provided specification is so superficial that none of the requester's requirements can be verified), we can save time by omitting the matching step from the beginning. The judgment whether the fuzziness is too high cannot only be made on the basis of the input specifications, but must also take further sources into account, e.g., the matching algorithms to be executed or the user's risk or ambiguity aversion. The general users' risk aversion again may depend on the domain: For example, in the domain of banking service, only perfect matches may be interesting. So, in this domain, fuzzy matching may not be applicable at all. In other domains, fuzziness can be tolerated to a higher extent.

As the fuzziness pre-check's result determines whether the following steps are performed or not, it keeps the fuzzy matching procedure from performing unnecessary computations. Thereby, it contributes to efficiency (*R7*). As a consequence, this check is more useful if it is based on simple computations, too. A very simple example check is only considering very significant fuzziness occurrences, e.g., cases where a specification part is not specified at all. For this check, no internals, e.g., lists of language constructs, need to be checked.

2. (Fuzzy) Matching

Step 2 of the general fuzzy matching procedure is the actual matching action. Here, we can insert different kinds of matching approaches. We show an example for inserting a fuzzy matching approach in Section 4.6.1. However, also any other ("unfuzzy") existing matching approach can be performed here, as shown in Section 4.6.2. Enabling the integration of existing matching approaches is important in order to satisfy *R5* (support diverse matching approaches) and *R8* (non-invasiveness).

Accordingly, this step can also lead to different kinds of matching result formats, e.g., gradual result formats and interval result formats (see Section 4.6).

3. Final Fuzziness Estimation

Next, the extent of fuzziness that emerged is determined. This is a foundation in order to satisfy *R4* (make extent of fuzziness transparent). In many cases, not a precise measurement but only an estimation is possible. The extent is classified into one group of a fixed set of groups. For example, a set of three groups could be: *High*, *Medium*, and *Low*. Fuzzy terms are suited for the representation of fuzziness estimations because they do not require an exact measurement and they are user-friendly [Zad96].

Furthermore, it is beneficial to distinguish between different fuzziness sources here, such that the user is well-informed and can select between multiple coping strategies (see result construction step), if necessary. Thus, each detected fuzziness occurrence is classified by the fuzziness sources presented in Section 4.4.2. This is a foundation in order to satisfy *R2* and *R3* because the distinction between fuzziness sources is required in order to support the user in finding appropriate countermeasures to reduce uncertainty, if needed.

Regarding the format in which fuzziness estimations are stored, we can refine *R4* into further sub-requirements [PAPS15]:

(*R4*.1) *Normalization:* The user has to be able to judge whether the fuzziness is low or high. This means that the maximum and minimum of the scale used to present fuzziness have to be known.

(*R4*.2) *Comparability/Commensurability:* Fuzziness measures for different matching runs have to be comparable. For example, a requester should be able to choose between different alternative services on the basis of the fuzziness within the matching results. This way, she can choose the service with the result that is less fuzzy, if the matching results are similar apart from that.

(*R4*.3) *Relevance:* Fuzziness without impact should not be reflected in order to not burden the user with irrelevant information.

4. Result Construction

In order to inform the user about detected fuzziness occurrences as well as about their source and extent, the matching result to be returned to the user is annotated with the information gathered in the previous steps. Thereby, *R3* and *R4* can be fulfilled.

The annotated matching results can be leveraged in order to give automatic recommendations for coping with fuzziness, e.g., strategies that could potentially reduce fuzziness. In the following, we call such strategies *fuzziness coping strategies*. Table 4.1 shows four recommendations for exemplary fuzziness occurrences:

Table 4.1: Exemplary Recommended Fuzziness Coping Strategies

Fuzziness Occurrence		**Fuzziness Coping Strategy**		
Concerned Spec. Part	**Fuzziness Source (Type)**	**Action & Artifact (Role)**	**Possible Side Effects**	**Prio-rity**
Prov. Performance Spec. is missing an execution environment	Provider-induced (Epis. Unc.)	Deliver additional monitoring data (Provider)	Increased Transf.-ind. F. (target language less expressive wrt. exec. environments than source language), ...	high
Prov. Reputation data based on Ratings	Provider-induced (Alea. Unc.)	No actions possible	-	-
Req. Performance = “fast”	Requester-induced (Vagueness)	Concretize Performance Req. wrt. Response Time, e.g., as a fuzzy set (Requester)	Increased Prov.-ind. F. (Provided Response Time info is incomplete), ...	medium
Reputation Req. specified as a fuzzy set	Requester-induced (Gradedness)	No actions required	-	-

In the first example, there is a provider-induced fuzziness occurrence of type epistemic uncertainty within the performance concerning the performance aspect. The recommended strategy suggests the provider to extend her provided service specification with more information. As discussed above each reduction of fuzziness comes with possible side effects. Here, the side effect concerning transformation-induced fuzziness is mentioned as one example. The second example addresses provide-induced fuzziness of type aleatory uncertainty. As discussed in Section 4.4.1, such an occurrence cannot be reduced by providing more information as it refers to data that is stochastic in nature. The third example is a fuzziness occurrence where the requester specified “fast” as a performance requirement. Such vagueness can be reduced by concretizing the requirements specification. One option to do this suggested here is to specify the property response time as a fuzzy set. As a possible side effect, provider-induced fuzziness could increase if the data about response time on the provider side is fuzzy, too. In the last example, requester-induced fuzziness of type gradedness is addressed. As discussed in Section 4.4.1, this fuzziness type does not lead to uncertainty. Thus, there is no need to reduce this occurrence. Furthermore, a prioritization of the suggested fuzziness coping strategies can be automatically derived on the basis of the estimated extent of a fuzziness occurrence. Also the number and severity of possible side effects could contribute to the derived priorities. Such a prioritization is important because, in real-world examples, the user can expected to be provided with a much longer list of such recommendations depending on how complex the requirements specification is.

4.6 Fuzzy Matching Approaches

In the following, we present two approaches that both follow the general fuzzy matching procedure described in Section 4.5. The choice of which approach should be applied depends on (a) the nature of the specification language to be matched and (b) the grade of intended “invasiveness”.

Regarding (a), we cannot use one concrete approach in order to support any matching approach based on any service specification language (*R5*, *R6*) because service specification languages can be very different. For example, there are matching approaches that operate on numerical specifications, while others operate on graph-based specifications.

Regarding (b), there are two possibilities when integrating a fuzzy matching approach: Inventing it from scratch or extending an existing matching approach. The first solution has the advantage that the approach can be optimized from the beginning with respect to dealing with fuzziness. However, there are already a lot of service matchers out there [PvDB+13]. Thus, it often makes sense to extend one of the existing approaches in order to enable fuzzy matching. Such an existing approach could be already available as an implemented matcher either with its source code or as a black-box component, (e.g., ”matching-as-a-service“). As a consequence, we also need a “non-invasive” approach to deal with fuzziness in the presence of black-box matching approaches (*R8*).

Section 4.6.1 presents a fuzzy matching approach invented from scratch and optimized to dealing with fuzziness. It is applicable for specifications that comprise a (hierarchical) collection of numerical values or simple linguistic terms. In contrast, Section 4.6.2 describes an approach to extend a given (“unfuzzy”) matching approach with fuzziness quantifications. This approach is applicable for various kinds of specification languages.

4.6.1 Fuzzy Matching Based on Fuzzy Logic

The approach presented in the following is based on formal concepts and tools from fuzzy logic: fuzzy sets and possibility theory (see Section 4.2). The concrete meaning of vague expressions can be captured by fuzzy sets. In the context of the matching problem, the fuzzy set will then serve as a soft constraint. Gradedness is naturally captured by the membership function of a fuzzy set, which can assume intermediate values between 0 (complete dissatisfaction) and 1 (full satisfaction). Modeling a linguistic expression in terms of a membership function (i.e., a precise mathematical object) is sometimes referred to as a process of “precisiation” [Zad08]. The possibility to specify the expressions in linguistic terms, however, is often more intuitive for users, than specifying fuzzy sets on their own [TT08, PRVM13].

For the purpose of illustration, we describe our approach using the example of reputation matching (cf. Section 2.4.4) because reputation “plays a key role” in today’s software markets [SO11]. Moreover, we show how the same approach can be applied to the matching of other service properties, e.g., performance matching.

Our approach operates on specifications that allow for describing service properties and requirements using a list of conditions that argue about numerical values. It addresses requester-induced fuzziness, provider-induced fuzziness, and transformation-induced fuzziness, as well as vagueness, gradedness, incompleteness, and uncertainty.

This section is based on our paper [PSB+] and extends it with more explanations, illustrations, and further concepts. These further concepts include estimating transformation-induced fuzziness and considering more complex requirements specifications.

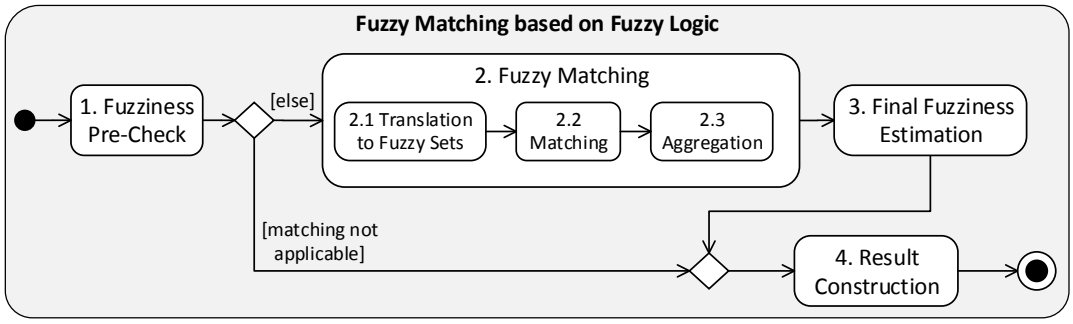


Figure 4.6: Procedure for Fuzzy Matching Based on Fuzzy Logic

c1: Rep(Service) ≥ 4
c2: Rep_{RT}(Service) $\gtrapprox$ 4 based on *many* ratings (w = 2)
c3: Rep_{Sec}(Service) $\gtrapprox$ 3 based on *many* ratings of the last 3 months
c4: Rep(Provider) $\gtrapprox$ 4

a) Requirements Specification

Entity	Context	# Ratings	
Service: RoomManager	Overall	200	Rep(200 ratings) = 4.5
Service: RoomManager	ResponseTime	80	Rep (80 ratings) = 3
Service: RoomManager	Security	40	Rep (40 ratings, 3 months) = 2.75
Service: BookARoomPro	Overall	80	Rep (80 ratings) = 3.35
Service: BookARoomPro	ResponseTime	20	Rep (20 ratings) = 1.5
Service: BookARoomPro	Security	10	Rep (10 ratings, 3 months) = 4.5
...	...	...	...
Provider: UBServices Inc.	Overall	130	Rep (130 ratings) = 3.5

b) Available Ratings & Exemplary Reputation Values

Figure 4.7: Exemplary Fuzzy Reputation Requirements and Reputation Values

4.6.1.1 Overview

Figure 4.6 visualizes an overview of the general procedure applied using fuzzy matching based on fuzzy logic. In addition to the steps of the general matching procedure shown in Figure 4.5, the matching step has been refined into three substeps: 1. Translation into Fuzzy Sets, 2. calculation of Matching results, and 3. Aggregation.

In the following, we explain each step of the procedure on the basis of the example requirements specification depicted in Figure 4.7 a). The depicted requirements specification is a fuzzy version of the requirements specification depicted in Figure 2.4 a). In contrast to the unfuzzy version, in Figure 4.7 a), requester-induced fuzziness is immediately visible because there are two language constructs that indicate the requester's vagueness: the soft threshold denoted by $\gtrapprox$ and the fuzzy term many.

In addition to requester-induced fuzziness, provider-induced fuzziness emerges in reputation matching because of the uncertain reputation data available at the provider side: A reputation value is always just an expected value developed from the aggregation of single ratings. Special cases, e.g., a service new in the market that did not receive any ratings yet, lead to the highest possible amount of uncertainty.

4.6.1.2 Step 1: Fuzziness Pre-Check

One of the most severe sources for fuzziness in this approach is the provided service's specification, i.e., the user ratings collected for a service to be matched. As provider-induced fuzziness may lead to uncertainty, the fuzziness pre-check becomes relevant.

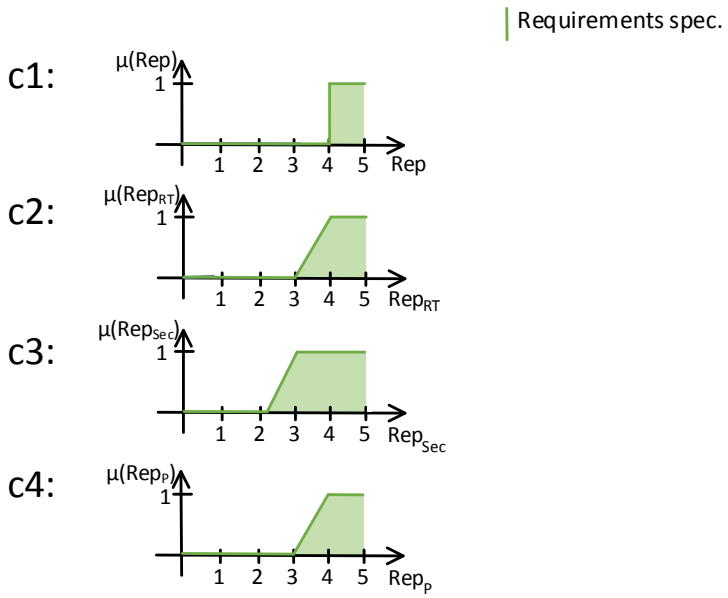


Figure 4.8: Example Fuzzy Sets for the Requirements Specification from Figure 4.7

As explained in Section 4.5, the fuzziness pre-check's purpose is to improve efficiency by suppressing unnecessary matching computations (*R7*), the check itself should be able to be performed quickly. Thus, it is based on simple heuristics.

For example, we can compare the number of available ratings per context to the mentioned ratings in the requirements specification. If there are no appropriate ratings available in order to evaluate the majority of the conditions in the requirements specification, we can directly return the most uncertain matching result and skip Steps 2 and 3.

4.6.1.3 Step 2.1: Translation into Fuzzy Sets

In order to perform the main matching algorithm, the input specifications, i.e., the requirements specification as well as the reputation value based on the available set of ratings, need to be transformed into fuzzy sets modeled as membership functions. The benefit of a translation into fuzzy sets is that it enables us to use a coherent mathematical framework that is able to cope with several fuzziness types, in particular, both vagueness and uncertainty. [PSB$^+$]

Creation of fuzzy sets from requirements specifications

Fuzzy parts of the requirements specifications, i.e., conditions containing a soft threshold, are transformed into membership functions denoting fuzzy sets. On the contrary, all other conditions are transformed into conventional sets. Conventional sets are actually special cases of fuzzy sets with $\{0, 1\}$-valued membership functions. [PSB$^+$]

Figure 4.8 depicts membership functions of the (fuzzy) sets created from Conditions `c1` – `c4` from Figure 4.7 a) in green color. "The x-axes denote reputation values in a scale from 0 to 5, while the y-axes represent the membership as a number between 0 and 1. For example, the lower threshold for the requested reputation in `c1` is 4. Thus, the membership is 0 from 0 to 4 and 1 between 4 and 5. This means that, if a service's reputation value is higher than

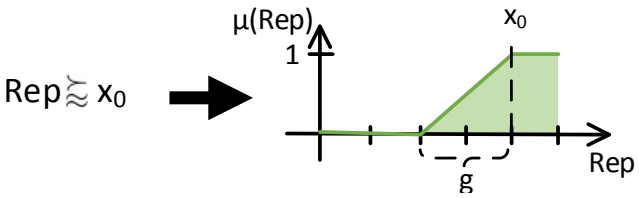


Figure 4.9: Transformation into the Membership Function in Function 4.3

a reputation of 4, it matches completely. As there are only "hard" transitions between the membership of 0 and the membership of 1, we speak of a "crisp" set." [PSB+]

In contrast to c1, the conditions c2, c3 and c4 are transformed into membership functions denoting fuzzy sets as they indicate soft thresholds. Thus, we specify a set of rules that define how to turn the expressions included in these conditions into membership functions. This includes the rules to

1. turn an expression "$reputation_value \gtrapprox x_0$",
   - for example: "$Rep_{RT} \gtrapprox 4$", see c3 in Figure 4.7
2. turn an expression "$fuzzy_term$ ratings",
   - for example: "many ratings", see c3 in Figure 4.7

into fuzzy sets.

The first case, i.e., an expression "$reputation_value \gtrapprox x_0$", is transformed into a fuzzy set with a membership function μ defined as follows in (4.3):

$$\mu(x) = \begin{cases} 0 & x \leq (x_0 - g) \\ \frac{x - x_0 + g}{g} & (x_0 - g) < x < x_0 \\ 1 & x \geq x_0 \end{cases}, \tag{4.3}$$

where g represents a configurable parameter ≥ 0 for the gradient of the fuzzy set: The size of the interval between the point where the fuzzy part begins ($x = x_0 - g$) and the point where the fuzzy part ends ($x = x_0$), i.e., the interval where $0 < \mu(x) < 1$. In order to obtain a crisp set, we transform into the same function with $g = 0$. Figure 4.9 visualizes these values correspondingly.

Furthermore, Figure 4.8 shows the transformation results for the four example conditions from Figure 4.7:

- For c1, the expression $Rep(Service) \geq 4$ is transformed with $g = 0$ and $x_0 = 4$.
- For c2, the expression $Rep_{RT}(Service) \gtrapprox 4$ is transformed with $g = 1$ and $x_0 = 4$.
- For c3, the expression $Rep_{Sec}(Service) \gtrapprox 3$ is transformed with $g = 1$ and $x_0 = 3$.
- For c4, the expression $Rep(Provider) \gtrapprox 4$ is transformed with $g = 1$ and $x_0 = 4$.

Function (4.4) shows the membership function obtained from an expression "$fuzzy_term$ ratings".

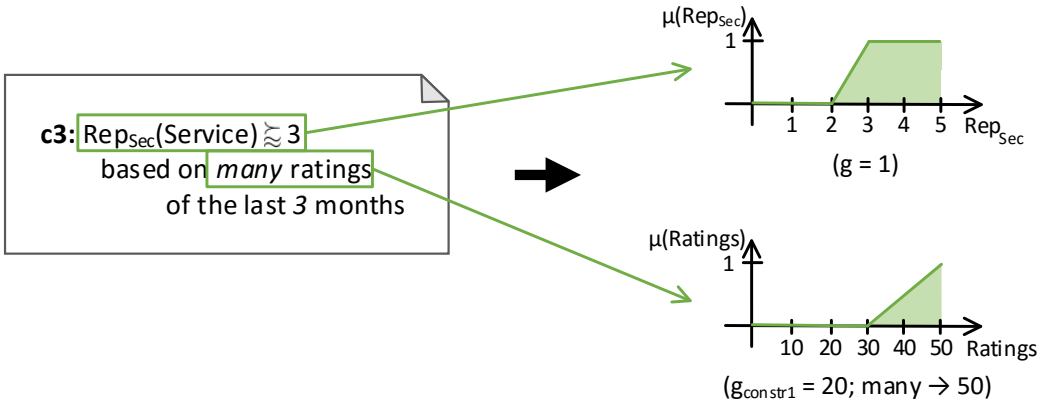


Figure 4.10: Example Transformation Results for Condition c3

$$\mu_{constr1}(x) = \begin{cases} 0 & x \leq (x_0 - g_{constr1}) \\ \frac{x - x_0 + g_{constr1}}{g_{constr1}} & (x_0 - g_{constr1}) < x < x_0 \\ 1 & x \geq x_0 \end{cases}, \tag{4.4}$$

where g represents a parameter ≥ 0 for the gradient of the fuzzy set and x_0 is a value transformed according to a given mapping from fuzzy terms to values. For example, there could be a mapping with $many\ ratings \rightarrow 50\ ratings$ in large markets with many customers or a mapping with $many\ ratings \rightarrow 10\ ratings$ for rather new markets. The utilized mapping should be based on the knowledge and the experience of a domain expert. Using the same method, fuzzy terms could also be used for the reputation value (e.g., "high").

The functions shown above have been derived from the standard trapezoidal function. In the example of reputation matching, (4.4) and (4.3) are monotonically increasing. The reason is that the reputation value as well as the number of ratings becomes better with decreasing numbers. These values are to be *at least* as high as the given threshold but higher is always accepted, too. However, when applied to other kinds of specifications, these functions have to be adapted to monotonically decreasing functions in some cases. As an example, take properties related to a service's performance, where the service's mean response time is to be minimized [BLB13].

Using these transformations, we obtain one to two fuzzy sets per condition. For example, for c3, using the example mapping mentioned above, we transform into the fuzzy sets depicted in Figure 4.10. Note that the expression "of the last 3 months" has not been taken into account so far. We discuss this matter in the next subsection.

The key parameters (including the mapping from fuzzy terms) within these transformation rules are configurable. However, we actually never know how well these transformation rules truly reflect the user's satisfaction. Some user's satisfaction might be reflected better with steeper fuzzy sets, some others better with less steep fuzzy sets. Similarly, different users would choose completely different mappings for the fuzzy terms (e.g. starting at a lower or higher reputation value). For this reason, transformation-induced fuzziness emerges at this point and introduces uncertainty into the matching procedure. In order to inform the user

about the induced uncertainty, we quantify these fuzziness occurrences, see Step 4 described in Section 4.6.1.6.

Creation of fuzzy sets from ratings

The first step when creating a fuzzy set from ratings is to select which ratings will be considered. Here, the requirements specification needs to be considered again. As an example, recall the "of the last 3 months" expression. This expression cannot be handled like the other expressions from the requirements specification: While the required rating value and the required number of ratings are constraints related to the whole set of considered ratings, the expression about the age is a constraint related to each single rating. The degree to which the constraint about the age is satisfied can be adapted by deselecting single ratings from the set of considered ratings. Thus, it will be used to filter the set of ratings considered in the other two constraints. Accordingly, this expression will be taken into account when considering the data for the provided service and is not part of the fuzzy sets that represent the requirements specification. Thus, we call such a constraint a *service data restriction*. In general, service data restrictions are expressions that influence the reputation modeled for the provided service on the basis of a set of considered ratings. Other examples for such restrictions are a "newer > older" constraint (i.e., newer ratings should get a higher priority than older ratings when calculating the reputation value [JBPP14a]) or a constraint defining the aggregation strategy to be used, e.g., a minimum aggregation of ratings.

Recall that our current matching problem involves both fuzziness and (frequentist) statistical information. We expect that the requester will be interested in a characteristic value like the expectation, i.e., the average rating of a service. She then specifies a soft constraint in the form of a fuzzy set A on this characteristic value. For example, the mean rating should be $\gtrapprox 4$ in c2. In practice, the true expectation m is unknown. Instead, we can only assume to get an estimation $\hat{m}$ of this value, e.g., the arithmetic mean of a finite set of ratings. Correspondingly, there are two problems to be considered. First, we need to characterize the uncertainty about m, i.e., the reliability of the average $\hat{m}$, in an appropriate way. For this purpose, we use a statistical resampling technique, so-called *bootstrapping*, that allows for estimating a probability distribution for the expectation [ET93]. The larger the sample size, the more peaked this probability density function $p(\cdot)$ will be around the average $\hat{m}$. In particular, the distribution provides an idea of the variance of the estimation. [PSB$^+$]

Figure 4.11 shows an example bootstrapping procedure. Example ratings are shown on the left. On the basis of these ratings, the samples in the middle are drawn. In this simple example, the sample size is $n = 4$. For each sample, we determine the expectation e. Next, we count how often each expectation occurs, as depicted on the right. These numbers form the probability distribution, visualized in the graph below. Note that this example is largely simplified. For example, in reality, sample sizes should be much larger, as explained later.

The variance of the resulting probability distribution in our example application does not only depend on the overall sample, i.e., the number of ratings. There are various other characteristics that lead to the fact that users perceive and evaluate the same service more or less divergently. For example, some services come with a wide range of functionality such that the probability that each user uses a different part with different quality increases. Other services may be simpler such that users evaluate on a more unique basis. Furthermore, some services may target a very heterogeneous user base that evaluates services very heterogenously. For example, Google Maps is used by business people as well as teenagers.

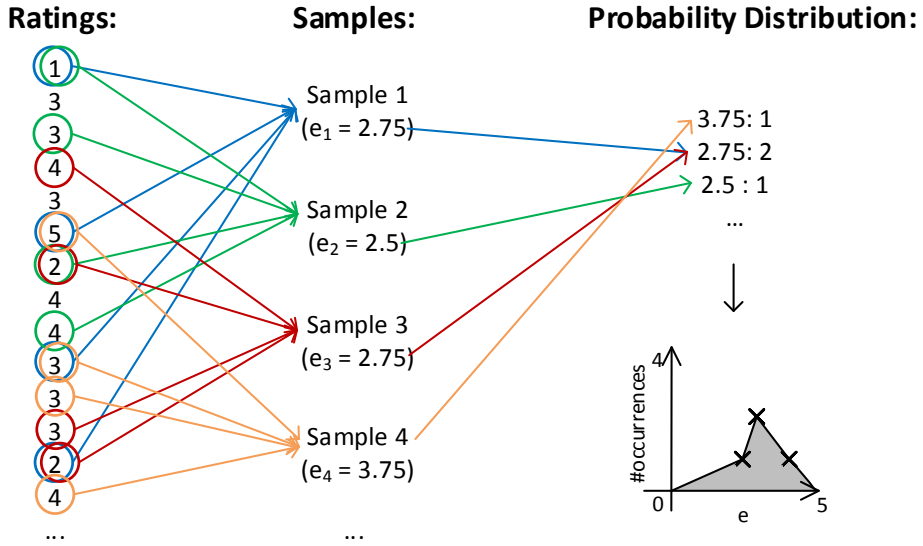


Figure 4.11: Example for a Simple Bootstrapping Procedure

Other services have a smaller user base where users have similar background and evaluate on a common basis. Examples for such services are pure business services in a very specific domain. Furthermore, most services are not static but depend on a rapidly changing data base. This might also influence typical evaluation tendencies varying over time.

As a second step, the information on the ratings should be represented in a way that makes it amenable to further processing within our approach. To this end, the probability function $p(\cdot)$ is to be represented in the form of a possibility distribution. We realize this by applying an established probability-possibility transformation [DFMP04] that interprets possibility as upper probability:

$$\pi_m(x) = \sup \left\{ \alpha \in [0,1] \,|\, x \in C_{1-\alpha} \right\} \ , \tag{4.5}$$

where $C_{1-\alpha}$ is the $1-\alpha$ confidence interval derived from the distribution $p(\cdot)$. Thus, it represents the shortest interval around $\hat{m}$ covering a probability mass of $1-\alpha$. [PSB+]

Two special cases of the above problem that allow for a simplified specification of π_m are (a) a very small and (b) a very large sample of ratings. Regarding (a): If only a few ratings are available (e.g., only 2 or 3), bootstrapping will hardly make sense. The same holds if no rating at all is available. In such a case, we define $\pi_m \equiv 1$, thereby reflecting the highest grade of uncertainty, i.e., complete ignorance, about m. Regarding (b): If the number of ratings is very large, $\hat{m}$ will approximate m so well that m can be assumed to be known with a high certainty. As an example, consider a situation where there are more than 1000 ratings for the service ImagePro1 in c1, with an average of $\hat{m} = 4.5$. Then, this value will be extremely close to the true expectation. Pretending full certainty, information about m can then be modeled in terms of the possibility distribution π_m with $\pi_m(x) = 1$ if $x = \hat{m}$ and $\pi_m(x) = 0$ if $x \neq \hat{m}$. In general, the more ratings are available, the more peaked π_m will be around the average $\hat{m}$. Thereby, the sample size directly influences the specificity of the possibility distribution π_m. Thus, the reliability of the information about m is appropriately reflected by π_m. [PSB+]

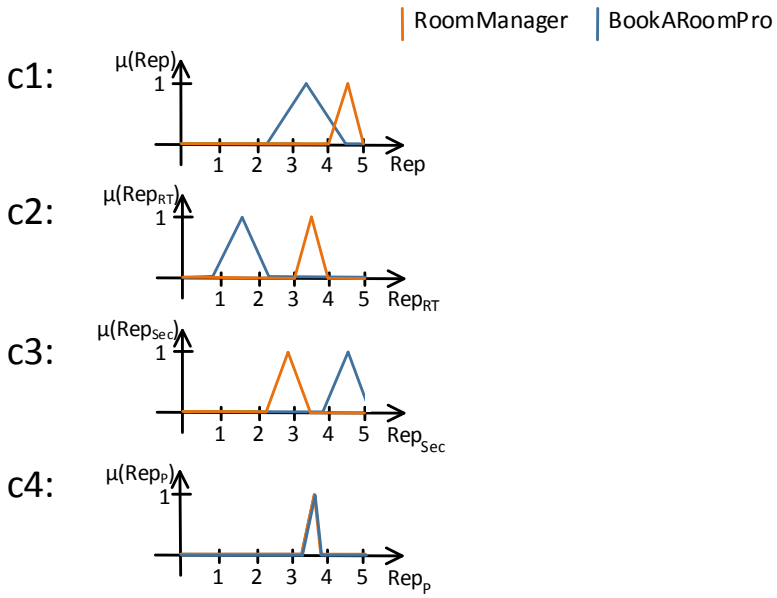


Figure 4.12: Example Fuzzy Sets for the Ratings for the Provided Services from Figure 4.7

The transformation from the information needed to match the "minimum number of ratings constraint" is simpler. The reason is that this information is typically known precisely. Therefore, it can be represented in terms of a possibility distribution π_s that yields a value of 1 for the current number of ratings and 0 otherwise. If the sample size is s, the requester will be satisfied to the degree $\mu_C(s)$.

In Figure 4.12, the possibility distributions informing about ratings (B_c) of the services RoomManager and BookARoomPro are shown in orange and blue, respectively.

4.6.1.4 Step 2.2: Matching

In the previous sections, we have explained how each of the expressions within the requester's conditions c is translated into a corresponding fuzzy set A_c with the membership function μ_{A_c}. Furthermore, we showed how information B_c about a service can be formalized in terms of a possibility distribution π_{B_c}. This distribution usually reflects uncertainty about the true value. In our running example of reputation data, π_{B_c} is derived from user ratings.

Figure 4.13 shows the corresponding fuzzy sets for the requested reputation values from our running example put on top of each other. In order to determine to which extent the information about a provided service satisfies the requirements specification, we need to compare A_c and B_c for each of the four conditions c1 to c4. For this purpose, we apply possibility theory by computing the possibility and the necessity of the condition A_c under the distribution π_{B_c} according to (4.1) and (4.2):

$$\Pi_{B_c}(A_c) = \sup_{x \in U} \min \left(\pi_{B_c}(x), \mu_{A_c}(x)\right) \ , \tag{4.6}$$

$$N_{B_c}(A_c) = 1 - \sup_{x \in U} \min \left(\pi_{B_c}(x), 1 - \mu_{A_c}(x)\right) \ . \tag{4.7}$$

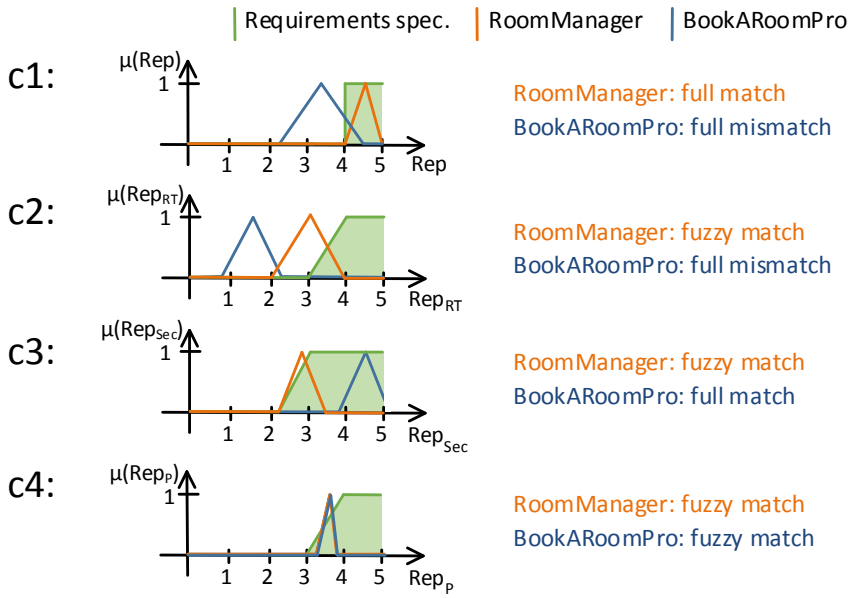


Figure 4.13: Matching the Example Fuzzy Sets from Figures 4.8 and 4.12

Thus, the satisfaction of the requester will be characterized by two values representing an interval $[n, p] \subseteq [0, 1]$ such that

$$n_c = N_{B_c}(A_c) \leq \Pi_{B_c}(A_c) = p_c \ . \tag{4.8}$$

This means the requester is certainly not satisfied in cases where $[n_c, p_c] = [0, 0]$ and $A_c \cap B_c = \emptyset$. The requester is definitely satisfied in cases where $[n_c, p_c] = [1, 1]$ and $B_c \subseteq A_c$. Otherwise, if either A_c or B_c are fuzzy, both n_c and p_c may take intermediate values within [0,1]. This means, the requester is possibly but not necessarily satisfied to some degree. In the extreme case where $[n_c, p_c] = [0, 1]$ we know absolutely nothing about the necessary and possible satisfaction of the requester. [PSB$^+$]

Accordingly, the necessity and possibility degrees in (4.8) represent a lower and an upper bound on the satisfaction of the requester: n_c is the degree to which the requester is *necessarily* satisfied, and p_c the degree to which she is *possibly* satisfied. The length of the interval reflects the level of ignorance regarding the service properties, here, reputation. If this uncertainty is reduced because more precise information is acquired, the interval will shrink. In the special case where the provider offers precise information, the interval degenerates to a point that reflects complete certainty about the satisfaction of the requester's requirements specification.

For some of the conditions depicted in Figure 4.13, matching is trivial: For example, we have a full match for RoomManager in the case of c1, and for BookARoomPro in the case of c3. Here, $B_c \subseteq A_c$ and $n_c = p_c = 1$. Likewise, we have a full mismatch for BookARoomPro in the cases of c1 and c2. In this case, $B_c \cap A_c = \emptyset$ and, therefore, $n_c = p_c = 0$.

In the other cases, the fuzzy set derived from the requirements specification cuts the fuzzy set derived from the ratings of a provided service within the interval where the membership changes from zero to one in a smooth transition, i.e., within the gradient. These cases are fuzzy matches. Thus, we need to obtain necessity-possibility intervals $[n_c, p_c]$ according to

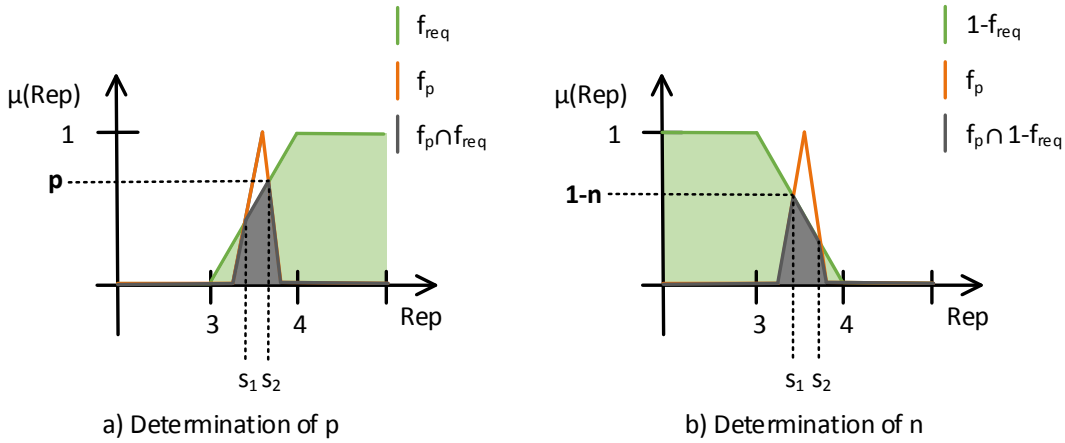


Figure 4.14: Example for Calculation of p and n for Condition c4

(4.6 – 4.7). The complete results for our running example in the form of these intervals are depicted in Table 1.

Requirements Spec.		**Matching Results**	
Condition	Expression	RoomManager	BookARoomPro
c1	≥ 4	$[1, 1]$	$[0, 1]$
c2	$\approx\!\!\!\gtrsim 4$	$[0.0, 0.5]$	$[0, 0]$
c3	$\approx\!\!\!\gtrsim 3$	$[0.5, 0.828]$	$[1, 1]$
c4	$\approx\!\!\!\gtrsim 4$	$[0.385, 0.615]$	$[0.385, 0.615]$

Table 4.2: Results for the Sets in Figure 4.13

In the following, we show the calculation of n and p step-by-step on the example of c4 from the requirements specification and the ratings for the provided service RoomManager. The function f_{req} derived from c4 is depicted in (4.9) and the function f_p derived from the provided ratings is depicted in (4.10):

$$f_{req}(x) = \begin{cases} 0 & x \leq 3 \\ \frac{x-3}{1} & 3 < x < 4 \\ 1 & x \geq 4 \end{cases} \tag{4.9}$$

$$f_p(x) = \begin{cases} 0 & x < 3.2 \\ \frac{x-3.2}{3.5-3.2} & 3.2 \leq x \leq 3.5 \\ \frac{3.8-x}{3.8-3.5} & 3.5 < x < 3.8 \\ 1 & x \geq 3.8 \end{cases} \tag{4.10}$$

Next, we construct the intersection set $f_{req} \cap f_p$ in order to determine p. The x values for the intersection points are $s_1 = 3.286$ and $s_2 = 3.615$, also depicted in Figure 4.14 a).

In fuzzy set theory, the intersection set is defined as the minimum function (cf. Section 4.2):

$$\mu_1 \cap \mu_2(x) = MIN(\mu_1(x), \mu_2(x)) \tag{4.11}$$

Thus, $f_{req} \cap f_p$ is defined as follows:

$$f_{req} \cap f_p(x) = \begin{cases} 0 & x < 3.2 \\ min(x-3, \frac{x-3.2}{0.3}) & 3.2 \leq x \leq 3.5 \\ min(x-3, \frac{3.8-x}{0.3}) & 3.5 < x < 3.8 \\ 0 & x \geq 3.8 \end{cases} \tag{4.12}$$

Figure 4.14 a) shows the resulting function in gray color. As defined in (4.1), the supremum of the minimum function denotes the possibility p. For our kinds of functions (a monotonically increasing or decreasing function on the one hand vs. a function with exactly one local maximum on the other hand), the supremum always needs to be one of the two intersection points. In this case, $p = f_{req} \cap f_p(s_2) = 0.615$.

In order to determine n, we need to intersect f_p with the inverse function of f_{req}:

$$\mu^C = 1 - \mu(x) \tag{4.13}$$

$$(1 - f_{req}) \cap f_p(x) = \begin{cases} 0 & x < 3.2 \\ min(1-(x-3), \frac{x-3.2}{0.3}) & 3.2 \leq x \leq 3.5 \\ min(1-(x-3), \frac{3.8-x}{0.3}) & 3.5 < x < 3.8 \\ 0 & x \geq 3.8 \end{cases} \tag{4.14}$$

Figure 4.14 b) depicts the resulting function in gray color. The intersection points are $s'_1 = 3.385$ and $s'_2 = 3.714$. Again, we need to determine the supremum, which is at $x = s'_1$, but this time, we need the inverse. Here, $n = 1 - ((1 - f_{req}) \cap f_p(s'_1)) = 0.385$. Thus, we get the interval $[0.385, 0.615]$ as the matching result for `c4`.

The most certain matching results following this approach are $[n, p] = [0, 0]$ (we have a complete match with full satisfaction) and $[n, p] = [1, 1]$ (we have a complete mismatch with full dissatisfaction). In both cases, the result is precise and unambiguous. In practice, the result will often be more fuzzy for two reasons:

- First, there is uncertainty about the requester's satisfaction because of uncertainty on the side of the provider, i.e., the interval $[n, p]$ will get wider.
- Second, the result is more ambiguous in the sense that the interval moves closer to the middle point 0.5, such that we neither have a clear match nor a clear mismatch, but something in-between.

In the second case, the location of the interval $[n, p]$ is mainly influenced by the requester: The more 'fuzzily' she specifies her requirements, the more 'mediocre' the evaluation will be. [PSB$^+$]

Figure 4.15: Matching Results for the Ratings Constraints in c2 and c3

The fuzzy sets for the other constraints, e.g., the requested number of available ratings, are matched in the same way. However, since there is no uncertainty on the provider side in those cases, always $n = p$ holds. For example, the 80 response time ratings of the service RoomManager fully match the $many_ratings$ constraint in c2. In contrast, the 40 security ratings for the same service only partially match the $many_ratings$ constraint in c3. Figure 4.15 visualizes the fuzzy sets for these constraints. In Section 4.6.1.5, the matching results for the single constraints are aggregated into a matching result for the whole condition c.

4.6.1.5 Step 2.3: Aggregation

For the next step, recall the foundations on aggregation introduced in Section 2.5. There, we compare different aggregation operators based on different criteria. This comparison is used for a selection of operators in the following.

Furthermore, recall that n_c and p_c are interpreted as a lower and upper bound of the requester's satisfaction on condition c. The set of possible degrees of overall satisfaction can be obtained by aggregating, respectively, the lower bounds and the upper bounds [PSB$^+$]:

$$\begin{aligned}[n,p] &= \{\, \mathrm{AGG}(s_1,\ldots,s_q) \mid s_i \in [n_{c_i}, p_{c_i}] \} \\ &= [\, \mathrm{AGG}(n_{c_1},\ldots,n_{c_q}),\ \mathrm{AGG}(p_{c_1},\ldots,p_{c_q})]\end{aligned} \tag{4.15}$$

In our matching approach, we need two different aggregation tasks: a) the aggregation of several constraints within one condition into one matching result per condition and b) the aggregation of the matching results per condition into one matching result per one pair of requirements specification and service.

Aggregation of Matching Results per Constraint

We suggest to aggregate the matching results of a condition's constraints using a conjunctive operator based on a t-norm, e.g., the minimum [PSB$^+$]. Combining the constraints on m and s, the overall satisfaction of the requester is then characterized by the interval

$$[\, \min(N_m(A), \mu_C(s)), \min(\Pi_m(A), \mu_C(s)) \,] \ .$$

As an example, take Condition c3 from our running example: As known from Table 4.2, the reputation value matches with an interval $[0.5, 0.828]$. Furthermore, the constraint

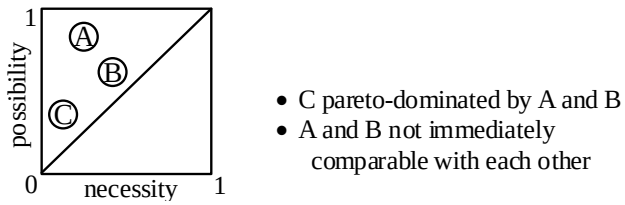


Figure 4.16: Services associated with degrees n/p to which they necessarily/possibly meet the requirements specification (based on [PSB$^+$])

about the minimum number of ratings matches with $[0.5, 0.5]$ (see Figure 4.15). Then, the aggregated matching result for c3 is $[0.5, 0.5]$.

Aggregation of Matching Results per Condition

In order to obtain an overall matching result, the intervals need to be aggregated. Here, we suggest to use the arithmetic mean.

However, two services are not necessarily comparable to each other if the overall evaluation of a service is given in the form of an interval. Instead, one service can *dominate* another one only in a Pareto sense (cf. Figure 4.16). Nevertheless, a total order on services can be enforced by reducing intervals to scalars, e.g.:

$$r = \alpha\, n + (1 - \alpha)\, p \in [0, 1] \ , \tag{4.16}$$

where the parameter $\alpha \in [0, 1]$ reflects the risk aversion of the requester. For $\alpha = 1$, the requester is completely risk-averse and fully focuses on the lower bound of the matching result. For $\alpha = 0$, she is completely risk-affine and, thus, will be decide on the basis of the upper bound. Requesters can decide for an α based on different factors. One example is the domain of the service that they are searching: If a banking service is searched, the requester assumable does not want to be very risky. Another example is the market size: If a requester knows that there are many services provided on the current market that probably fit her interests, so she probably does not need to accept high risks. [PSB$^+$]

4.6.1.6 Step 3: Final Fuzziness Estimation

Next, we quantify the fuzziness that has emerged during the matching. As noted above, we need to distinguish between occurrences of different fuzziness sources:

Provider-induced Fuzziness

Provider-induced Fuzziness has already been taken into account during matching. In particular, it is already reflected in the matching result: The size of the interval, i.e., $p - n$, indicates how fuzzy (in terms of *uncertain*) the matching procedure has been: If the provided information is precise, then $p = n$ holds and there is no uncertainty. This is the case, for example, for the constraints on the number of ratings. If $p \neq n$ holds, there has been uncertainty induced by the imprecise information on the provided side, i.e., provider-induced fuzziness.

In particular, the interval between n and p usually grows with the variance of the fuzzy set for the provided value. Figure 4.17 a) shows this fact on an example.

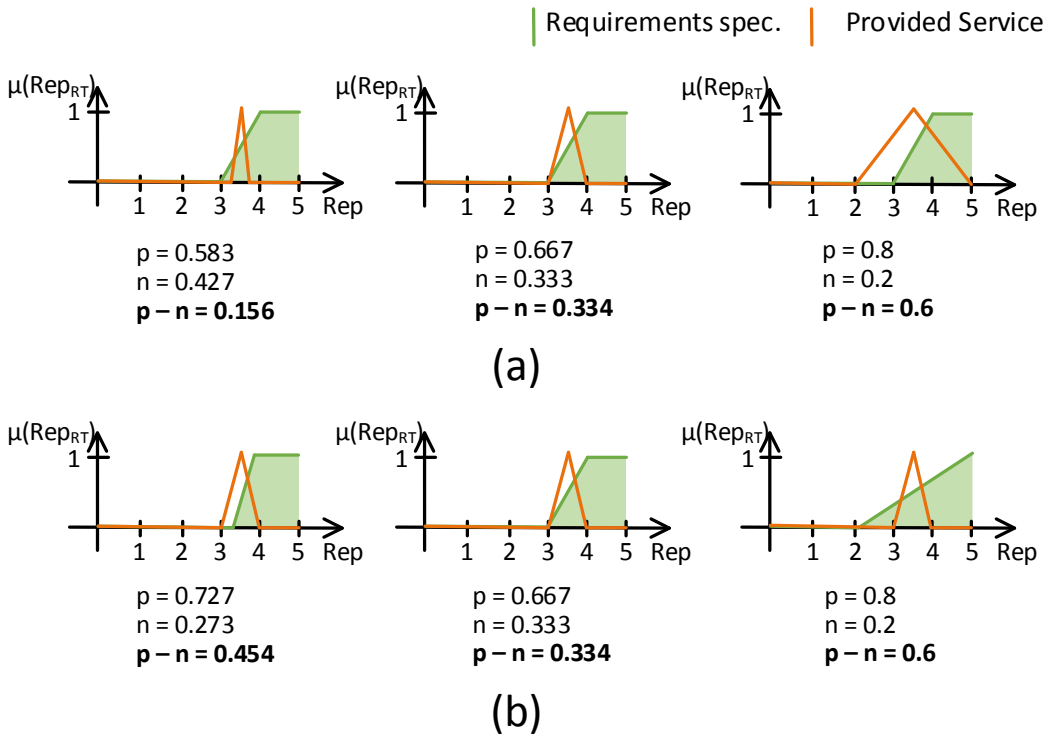


Figure 4.17: Examples for Varying n-p Intervals

As the minimum size of the interval (0) and the maximum size (1) are both known, *R4*.1 is satisfied. The size of the interval is also easily comparable, which satisfies *R4*.2. As the interval directly represents the impact of fuzziness, also *R4*.3 is satisfied.

Requester-induced Fuzziness

Note that there is also requester-induced fuzziness in our example reputation matching approach. This is induced by allowing the requester to use soft constraints and fuzzy terms within her requirements specification. However, this fuzziness source does not induce uncertainty but the requester's gradedness towards the satisfaction of her requirements specification. Thus, there is no need to measure such occurrences (cp. Section 4.4.2).

For example, Figure 4.17 b) shows, how the interval between n and p becomes sometimes even smaller while the requester's fuzzy set becomes "fuzzier".

Transformation-Induced Fuzziness

As noted in Section 4.6.1.3, also transformation-induced fuzziness occurs in this example. More precisely, the transformations from the conditions specified by the requester into fuzzy sets are uncertain in a way that we do not know whether the resulting fuzzy set really represents the requester's opinion [HLL+05]. Referring to the transformation defined in Formula 4.3, the value of parameter g is unknown, while the others are given. This means, we do not know where the smooth transition from a membership of 0 to a membership of 1 begins.

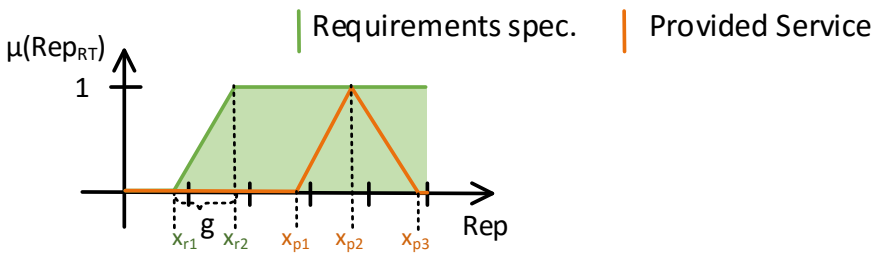


Figure 4.18: Variables used to Estimate Transformation-induced Fuzziness

In order to estimate how much uncertainty has been induced due to this transformation, we investigate each fuzzy set derived from the fuzzy expressions that are part of the conditions in the requirements specification. However, not each of the fuzzy sets leads to transformation-induced fuzziness. As an example, consider the green fuzzy set in Figure 4.18. The size of g determines the interval between x_{r2} and x_{r1}. While x_{r2} is given within the investigated expression, g and, therefore, x_{r1} are uncertain. However, as g can only be assigned with values between 0 to x_{r2} in order to define a valid fuzzy set, we know that, for the depicted fuzzy set, x_{r1} must be somewhere in between 0 and x_{r2}, too. Based on that and the fact that $x_{r2} < x_{p1}$ holds in this example, also $x_{r1} < x_{p1}$ has to be true for every g. As a consequence, in this example, it does not matter which g is chosen, the expression certainly leads to a full match.

This is, obviously, not always the case. In general, the induced uncertainty depends on the location of x_{r2}. There are four cases, as shown in Table 4.3. Case 1 is the case explained above and shown in Figure 4.18. In Case 2, always $p = 1$ holds, while n is dependent on the concrete position of the currently uncertain x_{r1}. Similarly, in Case 3, $p > 0$ always holds, while n is dependent on the uncertain x_{r1}. In Case 4, p and n can take any value. Furthermore, the larger x_{r2} becomes, the higher is the risk for false negatives and false positives. The reason is that, with an increasing x_{r2}, the number of cases where x_{r1} makes the difference between a full mismatch ($p = n = 0$) and a partial/fuzzy match increases, too.

Table 4.3: Different Cases of Transformation-induced Fuzziness

	Position	**Explanation**	**Fuzziness Score**
Case 1:	$x_{r2} \leq x_{p1}$	Full match ($p = n = 1$).	NONE
Case 2:	$x_{p1} < x_{r2} \leq x_{p2}$	$p = 1$ holds, n is uncertain	LOW
Case 3:	$x_{p2} < x_{r2} \leq x_{p3}$	$p > 0$ holds, n is uncertain	MEDIUM
Case 4:	$x_{r2} > x_{p3}$	p and n uncertain (as long as $p \leq n$). The larger x_{r2}, the higher the risk for false negatives and false positives.	$x_{r2} \leq x_{p3} + \frac{x_{max} - x_{p3}}{2}$ $\Rightarrow$ HIGH else $\Rightarrow$ VERY HIGH

It is noticeable that the uncertainty continuously increases from Case 1 to Case 4. More generally, the uncertainty increases with an increasing x_{r2}, which is natural as we consider the upper bound to be fixed but the lower bound to be highly variable.

On the basis of this finding, we assign fuzzy terms to be used as fuzziness scores to each case. A fuzzy term makes the extent of uncertainty immediately assessable even for inexperienced human users. Furthermore, it indicates that the result of the quantification should not be viewed as a precise value but as an estimation. Alternatively, more fine-

grained mappings with more cases or a continuous function depending on the size of x_{r2} (e.g., $f(x) = min(0, \frac{x_{r2}-x_{p1}}{x_{max}})$) are possible in order to improve the comparability. We chose the more coarse-grained mappings based on the four (respectively five) cases so that the semantics of each fuzziness score is clear and easy to understand. As a consequence, the scores are easily distinguishable and interpretable for more experienced users. The fourth case has been divided into two cases as this case seems to be the most serious one for the reason stated above. As a benefit, the distinction leads to a slightly better comparability.

Later, we aggregate the fuzziness scores for each expression into a fuzziness score for a condition using the maximum such that the final fuzziness score represents the highest grade of contained uncertainty such that it can be used to assess the risk of using this matching result. After that, the fuzziness scores for each condition can be aggregated into a fuzziness score for the whole requirements specification using the same strategy. In our running example, the scores are as follows:

- c2: RoomManager ⇒ MEDIUM; BookARoomPro ⇒ VERY HIGH
- c3: RoomManager ⇒ MEDIUM; BookARoomPro ⇒ NONE
- c4: RoomManager ⇒ HIGH; BookARoomPro ⇒ HIGH

Note that, for a clearer illustration, these values have only been derived from the fuzzy sets for the rating values. In fact, we can apply the same procedure to the fuzzy sets that represent the constraints.

Still, this approach is to be viewed as a heuristic. It abstracts from further possibilities for fuzziness. For example, we make the assumption that the upper bound for the interval that defines the smooth transition (x_{r2}) is always correctly chosen as it is directly derived from the threshold given by the requester.

The minimum fuzziness score here is obviously NONE. The maximum is VERY HIGH. If the latter is made transparent to the user, *R4*.1 is satisfied. The fuzziness scores are comparable—at least to some extent—, which partially satisfies *R4*.2. By design, *R4*.3 is satisfied due to the derivation from the four cases which take into account the relevance.

4.6.1.7 Step 4: Result Construction

In order to support the user in the best possible way, we provide both, aggregated overall values for quick comparison as well as detailed results in a hierarchical structure (see Chapter 3). This hierarchical structure allows the user to inspect all intermediate results, i.e., the matching results and fuzziness scores for every condition and for every expression within these conditions. In this way, the user can inspect the results and find out the reasons for a bad matching result. On the highest level, the n-p-interval and the aggregated fuzziness score is presented.

The fuzziness coping strategies relevant for this matching approach have already been presented in Section 4.5. They can be applied on the basis of these matching results.

4.6.1.8 Further Applications

The reputation matching approach is only one example the fuzzy matching approach described above can be applied to. As a second example, we selected performance matching

Requirements:

c1: ReponseTime $\lessapprox$ 3 sec
based on Arch_A & Ctx_A

c2: MTTQR < 30 sec

Service Information:

Entity	Metric	Archi-tecture	Context	Value
ImagePro1	RT	Arch_A	Ctx_A	2,86 sec
ImagePro1	RT	Arch_A	Ctx_A	1,82 sec
ImagePro1	RT	Arch_A	Ctx_A	...
ImagePro1	RT	Arch_B	Ctx_A	3,28 sec
ImagePro1	RT	Arch_B	Ctx_A	6,91 sec
ImagePro1	RT	Arch_B	Ctx_A	...
ImagePro1	MTTQR	Arch_A	Ctx_A	35,51 sec
ImagePro1	MTTQR	Arch_B	Ctx_A	75,53 sec
...	...	...	...	...

Figure 4.19: Performance Matching Example [PSB$^+$]

based on (model-based) performance prediction [BKR09, SDMIS04]. Figure 4.19 shows an example explained in the following.

The representation of requirements and information about a service here is similar to the representation in Figure 4.7: Again, requirements consist of several conditions, as depicted in the left part of the figure. In this example, the conditions address the performance metrics response time (RT) and mean time to quality repair (MTTQR) [BLB15]. RT is the time that a service requires to serve a response for a call. MTTQR is a metric for measuring the elasticity of a software service, i.e., the mean time that a service needs to adapt to increasing or decreasing workload by, for example, scaling its execution environment out or in. [PSB$^+$]

Information about how services perform wrt. these metrics are depicted in the right part of Figure 4.7. The table contains different rows for different architectures and different contexts because performance metrics are always specific to a service's internal architecture and its external context. The internal architecture is determined by the software architecture of the service, its implementation, and its execution environment. The external context is represented by the service's input (work) and call rate (load). For example, the response time of an image processing service can expected to be much lower if the service is executed in a high-performance compute center in contrast to the service being executed on a standard mobile phone. Furthermore, the response time depends on the number of concurrent service invocations. The representation of execution environments and usage scenarios is heavily simplified in Figure 4.7: All abbreviations (e.g., "Arch_A", "Ctx_A") refer to more complex models as known from performance engineering (e.g., [BKR09]). [PSB$^+$]

Fuzziness may occur in different ways. For example, Condition c1 contains an approximation operator which potentially leads to requester-induced fuzziness. In addition, the requester did not specify any internal architecture for c2. Often, the internal architecture may be even not fixed but vary in case of elastic services, e.g., in cloud computing settings where the execution environment can be scaled out or in. Accordingly, the value for the service to compare with is uncertain. If a value is not available for a certain environment or for a certain usage scenario, provider-induced fuzziness may occur. For all these reasons, similar as in our reputation example, it is recommendable to model performance specifications as fuzzy sets, as described in [BLB13], and to apply our approach as described in this Section.

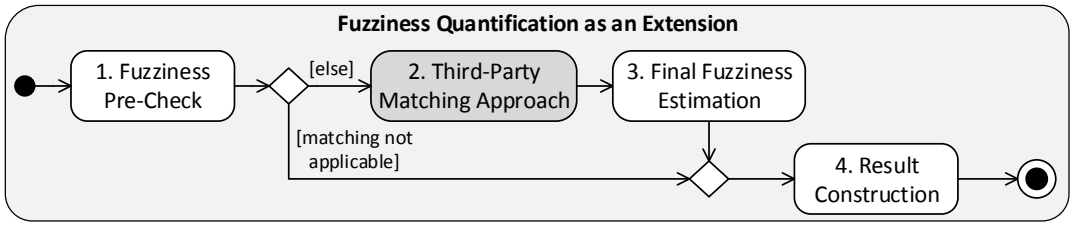


Figure 4.20: Procedure for Fuzziness Quantification as an Extension

4.6.2 Fuzziness Quantification as an Extension

The fuzzy matching approach based on fuzzy logic described in Section 4.6.1 is designed to build a new matching approach from scratch: The whole matching algorithm has been constructed for the purpose of fuzzy matching based on fuzzy logic. However, for several reasons, this cannot be the whole solution in order to enable fuzzy matching in practice:

On the one hand, a lot of service matchers already exist [PvDB+13]. Thus, it makes sense to extend one of the existing approaches in order to enable fuzzy matching. On the other hand, we can hardly use the approach based on fuzzy logic if we do not want to reinvent the whole matching approach. The presented fuzzy matching approach based on fuzzy logic can hardly be integrated into already existing matching approaches as the underlying logic of the whole main matching procedure would need to be adapted. The sources of an obtained third-party matcher may not even be available to the responsible person. As a consequence, we need a "non-invasive" approach to deal with fuzziness in the presence of black-box matching approaches (*R8*). Furthermore, there are specifications based on concepts that are not compatible to fuzzy logic, e.g., pre- and post-conditions specified on the basis of predicate logic. This requires a more general approach (*R5*).

Thus, we propose a second fuzzy matching approach that can be used on top of existing matchers. This approach is similar to the fuzziness estimation step from Section 4.6.1.6. But instead of an integration into the deep logics of a matching approach, we propose to examine the specifications a matcher takes as an input and to annotate the matching result the matcher delivers as an output. Further notes about the applicability of this approach are given in Section 4.6.2.3.

4.6.2.1 Overview

Figure 4.20 visualizes an overview of the general fuzzy matching procedure applied as an extension. In addition to the steps of the general matching procedure shown in Figure 4.5, the matching step is represented by a Third-Party Matching Approach to be viewed as a black-box. In our example, this step applies our exemplary privacy matching approach exactly as described in Section 2.4.3.

As Step 1, 2, and 4 do not contain any new concepts compared to the ones explained earlier, in the following, we focus on Step 3, the fuzziness estimation. The estimation is explained using the example specifications depicted in Figure 4.21. The example is based on the example specifications depicted in Figure 2.3. In contrast to the specification for the RoomManager service from Figure 2.3 b), the specification for BookARoomPro in

Requirements

Signatures:

```
bookRoom (String bookerName, MailAddress address, Date time, Integer capacity): Room room
...
```

Privacy Requirements:

Parameter	Purpose	Delegation Depth	Retention Period	Visibility	Location Limit	Sensitivity
bookerName	RoomBooking	1	12	-	Europe	High
address	Contact	0	12	-	Europe	Very High
time	RoomBooking	1	48	Notification Services	Europe	Medium
capacity	RoomBooking	1	∞	-	Europe	Low
room	Confirmation	1	48	Notification Services	Europe	Medium

(a) Requirements Specification

Service "BookARoom Pro"

Signatures:

```
bookRoom (String bookerAddress, Time time): LectureHall hall, Receipt receipt
...
```

Privacy Policies:

Parameter	Purpose	Delegation Depth	Retention Period	Visibility	Location Limit
bookerAddress	Contact	1	?	-	Germany
time	RoomBooking	0	?	-	Germany
hall	Confirmation	1	?	-	Germany
receipt	Confirmation	1	?	-	Germany

(b) Service Specification

Figure 4.21: Example Specifications for Fuzzy Privacy Matching

Figure 4.21 b) is incomplete in a sense that retention periods are unknown. Thus, provider-induced fuzziness occurs.

In the example of privacy, a provider often must provide policies for legal reasons. However, this neither means that she also provides a formal representation as required for matching, nor that this representation includes all properties used during the matching process. Thus, privacy properties make a realistic example [PAPS15].

4.6.2.2 Fuzziness Estimation

After the matching step, in our example, both services, RoomManager and BookARoomPro, receive a very similar matching result according to the specifications depicted in Figures 2.3 and 4.21. The reason is that, in both cases, retention periods do not match. However, considering RoomManager, there is certainly a mismatch because the requested retention period is smaller than the retention period specified for the provided service (12<24). Considering BookARoomPro, we are uncertain about whether the specifications match or not. The service could potentially match well with respect to retention periods. As a consequence, for a requester, BookARoomPro would be a better choice as it matches at least as good as RoomManager, but probably even better. However, on the basis of standard matching results, this difference is not visible to the user. Thus, due to such uncertainties, we need

to determine fuzziness scores reflecting the confidence a requester can have regarding the delivered matching result. [PAPS15]

In the following, we focus on estimating provider-induced fuzziness based on our approach described in [PAPS15]. To this end, we examine the input specifications the matching approach processes. Note that, although provider-induced fuzziness occurs in the specification of the provided service, the fuzziness estimation needs the requirements specification, too, in order to find out which language constructs are relevant, as we will see in the example discussed below.

We iterate through the specified language constructs in a similar way as the matching approach itself (see Section 2.4.3). For each requirement, the algorithm checks whether one of the requested properties is not specified at the provider's side. In general, the more values are missing at the provider's side, the higher the fuzziness score for a service, $fScore(service)$. [PAPS15] For simplicity, we use the same scale of fuzziness scores as in Section 4.6.1.6. If nothing is missing, $fScore(service) =$ NONE is assigned. If everything is missing (the provider didn't specify privacy policies at all), $fScore(service) =$ COMPLETE is assigned.

The fuzziness score for a service's whole privacy specification $fScore(service)$ is an aggregation of fuzziness scores $fScore(pol)$ for the different policies the specification of the provided service consists of. The score $fScore(pol)$ for a policy is constructed on the basis of the fuzziness scores $fScore(prop)$ for each property that is part of the policy. $fScore(prop)$ increases with respect to (a) the requester's *sensitivity* $sens$ for the requirement corresponding to this policy and (b) the *severity* sev of the requester's restrictions derived from the specified value for a certain property, e.g., retention period.

The rationale for taking sensitivity into account is that sensitivity denotes how important a requirement is to the requester. For example, if a policy p cannot be checked due to incompleteness and the requester's sensitivity $sens_i$ regarding the requirement corresponding to this policy is VERY HIGH, the fuzziness for this property is critical; if her sensitivity for this policy is rather low, fuzziness is less critical. [PAPS15]

The rationale for taking the severity of the requester's restrictions into account is the possibility for false negatives or false positives: If a requester allows an infinite retention period (or delegation depth respectively) for a certain policy, this requirement will always match; but the stricter the requester, i.e., the lower the value for the required retention period, the more likely this value will lead to a mismatch. [PAPS15]

Considering the fact that the privacy matching approach that is extended in our example is a *pessimistic* approach if fuzziness occurs (uncertain conditions always lead to mismatches), the severity sev_r should increase with an increasing required value r. For example, the required retention period for the time parameter in the requirements specification in Figure 4.21 is $r_{time} = 48$, while the required retention period for the bookerAddress parameter is $r_{ba} = 12$. The corresponding provided values p_{time} and p_{ba} needed for comparison are unknown. Following the matching rules described in Section 2.4.3, all values $p_{time} \geq 48$ (respectively $p_{ba} \geq 12$) would lead to a match instead of the mismatch the privacy matcher assumes for these cases. Consequently, a false negative is likely in both cases but even more likely for p_{ba} than for p_{time}.

An exemplary fuzziness score assignment for severity sev based on fuzzy sets is depicted in Figure 4.22. There, we see four membership functions for the fuzzy sets LOW, MEDIUM, HIGH, and VERY_HIGH. In cases where the requested retention period value r is located in a

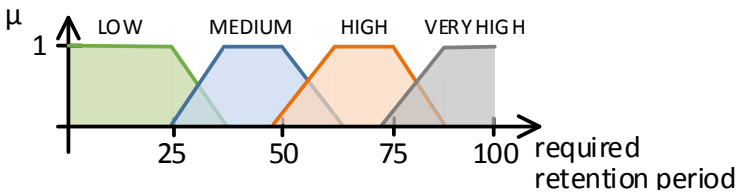


Figure 4.22: Exemplary Mapping from Retention Period Values to Severity Fuzziness Scores

region where membership $\mu = 1$ holds for one of the four functions, the assignment is clear. The regions where the functions become fuzzy, i.e., $0 < \mu < 1$ for more than one function at the same x-location, we need further information in order to assign a fuzziness score. For example, the severity sev could be determined depending on the membership degrees and the requester's risk aversion.

Note, that also other, e.g., more or less fine-grained mappings, are possible. The most appropriate mapping for a certain domain needs to be determined in an empirical way. In our case, we simplified the decision for the granularity by sticking to the same scale as used for the sensitivity in order to simplify the aggregation of both values done later. Also note that, for *optimistic* matching approaches (treating uncertain conditions as matches), the mapping would be the other way around: The fuzziness score should decrease with an increasing required value.

On the basis of this mapping, we can now calculate fuzziness scores per property, per policy, and per service using Formula 4.17:

$$
\begin{aligned}
fScore(prop_r) &= \begin{cases} \text{NONE} & \text{provided value available} \\ \text{NONE} & \text{required value r} = \infty \\ AGG(sens_r, sev_p) & \text{else} \end{cases} \\
fScore(pol_p) &= AGG(fScore(prop_{p..n})) \\
fScore(service_s) &= AGG(fScore_{s..n})) \qquad (4.17)
\end{aligned}
$$

with $prop_r$ referring to one property within a policy/requirement, pol_p referring to one policy within a service specification, and $service_s$ referring to a service specification. Which aggregation operator AGG to choose is a design decision: We could argue to always use a maximizing aggregation operator here because the fuzziness score will be used to assess the risk of a bad decision. This risk should always be handled pessimistically and no compensations should be made, e.g., the risk per service is the highest risk per policy. However, using an averaging operator (here: e.g., the median) would lead to the advantage that we get more distinguishable results. In general, the best decision also depends on the number of the available offers and their specifications.

For our example specifications in Figure 4.21, using the maximum aggregation operator leads to the fuzziness scores depicted in Table 4.4. As we can see there, we get a final fuzziness score of VERY HIGH, although only one property has been evaluated with a fuzziness score of VERY HIGH and the severity of this property is even LOW. This is reasonable if the very high sensitivity of the corresponding policy is taken very seriously as it indicates that violations to this property are very crucial to the requester.

Table 4.4: Estimated Fuzziness Scores for the Running Example

Requirement / Policy	**Sensitivity**	**Severity** (Retention Period)	**Fuzziness Score** (max)
bookerAddress / bookerAddress	VERY HIGH	LOW	**VERY HIGH**
time / time	MEDIUM	MEDIUM	MEDIUM
room / hall	MEDIUM	MEDIUM	MEDIUM

Measuring incompleteness, as is done here, has already been proposed by decision-making theorists, as decisions generally can depend on how much relevant information is missing [FB88, FT95]. But up to now, it has never been applied to service matching (see survey in Section 4.10).

As the fuzziness scores have the same format that we used above for transformation-induced fuzziness (see Section 4.6.1.6), again, *R4*.1 and *R4*.2 are satisfied. *R4*.3 is satisfied because severity is taken into account.

4.6.2.3 Further Applications

Again, the extension of the privacy matching approach is only an example. In general, the fuzziness quantification as an extension is broadly applicable to various matching approaches. For example, Bunse developed in his thesis [Bun14] a signature matcher where transformation-induced fuzziness on the example for parameter multiplicities has been considered. Moreover, Merschjohann [Mer14] developed a price model matcher including a fuzziness estimation step that quantifies algorithm-induced fuzziness. Börding [Boe15] worked out a concept for estimating algorithm-induced fuzziness occurring during the matching of pre- and post-conditions.

The application of our concepts to matching approaches based on specification languages for protocols is challenging as such specification languages often do not reveal whether the specification is complete or not. For example, for a protocol specified as an automaton, there is no possibility to decide whether it is complete or not only on the basis of this specification. For provided service's, we could compare the specification with runtime behavior in order to detect some flaws in the specification, but also this approach does not ensure completeness. Furthermore, for requirements specification, this approach is not feasible. Current research activities [SFB901] include the application of evolutionary algorithms to synthesize complete behavioral specifications.

The same challenges hold for pre- and post-conditions. For example, if a precondition is left empty, this could mean that the service is always applicable. However, an empty precondition could also be understood as an unknown precondition. These two options have a totally different impact on matching.

One possibility to apply our concepts to such specifications is to extend these specification languages with special language constructs that indicate a missing entity, e.g., a missing condition, or a missing part of a protocol. In case of conditions, a language construct that defines a pre- or a post-conditions as "not defined" needs to be introduced [Vij14, Boe15]. This way, the matcher can distinguish between cases where a precondition of a provided service is "not defined" or just not existent. The first case leads to fuzziness, while the second case leads to a full match with any required precondition.

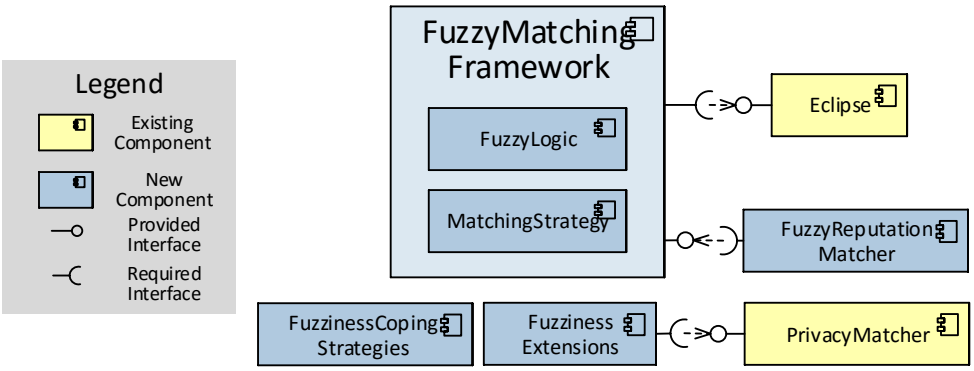


Figure 4.23: Architecture of the Fuzzy Matching Prototypes

Usually, we can identify cases where we need such language constructs within the meta model of the considered specification language: Anywhere, where an arbitrary number of model elements can be inserted (e.g., an arbitrary number of transitions within a protocol, or an arbitrary number of parts within a condition), there is potential uncertainty about completeness. However, there are also many languages where the size of a specification is fixed. For example, in the privacy specification constructs used as an example, there is a fixed set of properties to be specified for each parameter: The cardinality for both delegation depth and for retention period is "1".

4.7 Prototype Implementation

In order to realize fuzzy matching based on fuzzy logic as described in Section 4.6.1, we realized a fuzzy logic framework. The framework part is independent of the particular matcher, e.g., the framework can be used for reputation matching, but also for other approaches, e.g., performance matching. This has been achieved by designing an architecture based on three substitutable layers: the fuzzy logic layer, the matching strategy layer, and the concrete matcher. The approach presented in Section 4.6.2 is implemented by further extensions.

4.7.1 Architecture and Technologies

Figure 4.23 depicts the components that have been created as part of the implementation. Each component has been realized by a set of Eclipse plug-ins, briefly described in the following:

FuzzyLogic: The FuzzyLogic component provides an API to help with concepts from fuzzy logic, e.g., different membership functions and operations defined on them. This component can be substituted by third-party fuzzy logic frameworks.

MatchingStrategy: The MatchingStrategy component is a layer between the FuzzyLogic component and concrete matchers. It realizes most of the concepts described in Section 4.6.1 but is independent from a concrete matcher and a concrete specification language.

FuzzyReputationMatcher: In order to apply fuzzy matching based on fuzzy logic, we implemented a FuzzyReputationMatcher that uses the FuzzyLogicFramework as an example fuzzy matcher. By the help of the framework, this matcher realizes the example fuzzy reputation matcher used as an illustration in Section 4.6.1.

Eclipse: The FuzzyLogicFramework uses the Eclipse Modeling Framework (EMF).

FuzzyExtensions: The FuzzyExtensions contain the fuzziness estimation as explained in Section 4.6.2.2.

FuzzinessCopingStrategies: An extensible set of FuzzinessCopingStrategies as explained in Section 4.5.

PrivacyMatcher: The PrivacyMatcher is used as an example application for our implementation of FuzzyExtensions. It realizes the privacy matching approach described in Section 2.4.3.

4.7.2 Main Features

The fuzzy matching implementation is highly configurable. For example, membership functions of fuzzy sets can be derived in a more soft or in a more strict way. Figure 4.24 shows a screenshot that depicts the fuzzy reputation matcher's configuration parameters: ValueFSFuzzyPartSize refers to the size of the interval where the membership within the fuzzy sets that represent the required reputation value turns from 0 to 1 or from 1 to 0 (parameter g in Figure 4.9). ConstraintFSFuzzyPartSize refers to the same parameter for the fuzzy sets constructed from additional constraints within the requirements specification. ConstraintFSThreshold refers to the point where these fuzzy sets begin/end. RiskAversion is used for mapping the resulting n-p-interval to a scalar as described in Section 4.6.1. Bootstrapping defines whether the fuzzy sets for the provided values are constructed via bootstrapping (see Section 4.6.1.3) or using standard values. Note that each integrated fuzzy matcher could extend this list of parameters, for example, in order to realize the performance fuzzy matcher described in Section 4.6.1.8.

Below, the matching results view shows some matching results calculated with this matcher configuration. In particular, we can see the intervals in the column Result and the estimated fuzziness within the column Fuzziness. For example, in Input pair 2, we have a low provider-induced fuzziness of type aleatory uncertainty (al. unc.) leading to a result interval of [0.3393;0.5536]. In contrast, Input pair 3 got a result of [1;1] where no fuzziness has been induced, while the matching result of Input pair 4 is completely uncertain.

4.8 Validation

Our fuzzy matching concepts have been applied and validated within the scope of CRC 901 "On-The-Fly Computing" [SFB901] on the basis of two case studies. In the following, we introduce the validation questions, the case studies, and their results. Finally, we discuss the results with respect to threats to validity and the satisfaction of the collected requirements. Appendix D describes where to find all evaluation data, e.g., utilized example specifications.

In Section 3.10, we discussed the difficulties regarding service matching evaluation in general. For fuzzy matching approaches, even more difficulties can be added, as surveyed by Bruns [Bru15]. For example, for fuzzy matching results, there does not always exist a

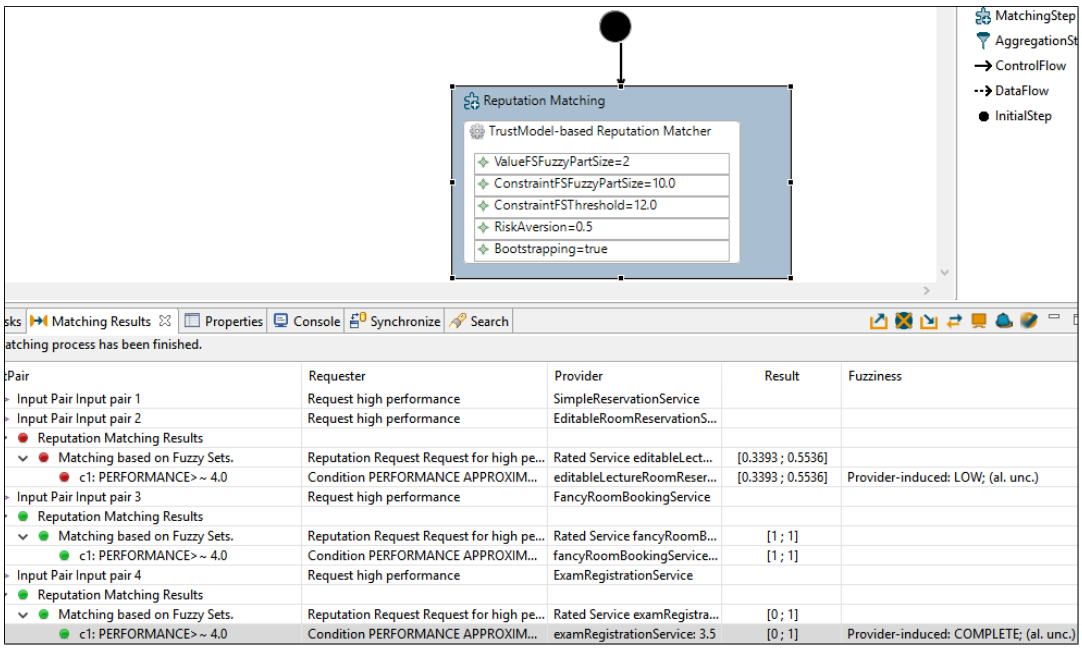


Figure 4.24: Screenshot of Some Fuzzy Matching Results for the Reputation Matcher

"ground truth" regarding how well a specification should match. In addition, fuzzy matching results, as we propose them in this chapter, have an extended format compared to traditional matching results, which makes fuzzy matching approaches hardly comparable to traditional matching approaches on the basis of the matching results. In addition to all this, there is the general trade-off between expressiveness of a validation metric and its applicability [BR08]: validation metrics that a very expressive are most often not applicable, while metrics that are easily applicable often only capture the behavior to be measured to a limited extent.

In the following, we present our validation attempts taking into account these difficulties and focusing on the most important validation questions. The threats to the validity of our validation are discussed in Section 4.8.6, while ideas for more advanced validation strategies are sketched in Section 6.2.

4.8.1 Validation Questions

Inspired by studies about uncertainty and risk by Letier et al. [LSB14], our evaluation goal was to investigate whether modeling fuzziness explicitly and reflecting it in the final matching results improves decision-making of the involved users.

Following the guidelines of evidence-based software engineering by Kitchenham et al. [KDJ04], we formulated our evaluation question as follows:

(***VQ2.1***) *Decision-Making:* Do fuzzy matching results improve the user's decision-making in the presence of fuzziness?

This question needs to be investigated with respect to the fuzzy-logic-based approach presented in Section 4.6.1 and also with respect to the fuzziness quantification approaches

presented in Sections 4.6.1.6 and 4.6.2.2. The first is addressed in Case Study 2.1 (see Section 4.8.2), while the latter is addressed in Case Study 2.2 (see Section 4.8.3).

4.8.2 Case Study 2.1: Evaluation of Fuzzy Reputation Matching based on Fuzzy Logic

In the following, we report about four experiments we performed using the implementation presented in Section 4.7. In order to answer the evaluation question, within the scope of these experiments, we applied our approach to a set of handmade specifications as well as real-world specifications. Following the example by Esfahani et al. [EKM11], we compared our new fuzzy approach to an alternative approach that does not take fuzziness into account in any way. The alternative approach is represented by the traditional reputation matching approach described in Section 2.4.4. The traditional approach ignores all emerged fuzziness by turning fuzzy terms into hard thresholds and using the arithmetic mean as an estimated reputation value instead of representing it in terms of a possibility distribution. As a consequence, in the traditional approach, we only distinguish between mismatching conditions (result $= 0$) and matching conditions (result $= 1$). The results per condition are aggregated using the average – the same aggregation operator as used for aggregating the results in our new approach. Because of this aggregation of the boolean results computed per condition to an overall result, the overall traditional results can be gradual to a specific extent.

Predominantly, our experiments are intended to show two important advantages of the interval matching results delivered by our fuzzy-logic based matching approach in comparison to the traditional results:

- *Distinguishability*: Specifying requirements and services using fuzzy sets will often allow for distinguishing services that matched equally well using the traditional approach. Given the evaluation of a set of services, we define the degree of distinguishability as the probability that two services randomly selected from all pairs (and a level of risk aversion in $[0, 1]$) do not have the same score (see Section 2.6).
- *Uncertainty representation*: The width of the interval $[n, p]$ informs about the uncertainty of the matching result, which is expected to be important for decision-making. In particular, the upper bound p informs about the potential of a service. The suboptimality of a service with a matching result $[n, p]$ is not proven unless there is another service with a result $[n', p']$ such that $p < n'$; in this case, the former is *dominated* by the latter. To make sure the best service is not missed, the requester should in principle check all non-dominated ones, or modify the requirements until a clear winner is found.

In addition, we investigated further properties that contribute to the applicability of our approach: scalability in terms of the amount of matched data in Experiment 2.1.3 and transferability to other specifications as well as correctness of matching results compared to the literature in Experiment 2.1.4.

4.8.2.1 Experiment 2.1.1: Toy Example

The first experiment is a toy example. The main purpose of this experiment was to test the evaluation setting.

Procedure

In this experiment, we applied our approach to 10 services within the domain of university management with 500 ratings in total as well as one requirements specification including five conditions. All specifications in this experiment have been constructed manually. The service specifications are the independent variables in this experiment. The dependent variables are the matching results and their distinguishability as defined above.

Results and Discussion

Figure 4.25 a) shows the matching results for each service. One interesting result is that s2 and s3 both have a matching result of 0.4, and s8 and s9 both have a matching result of 0.8. Selecting one of the services on the basis of these results becomes a difficult task for the requester because she cannot differentiate between services with the same matching result. This situation occurs especially often when considering the traditional results. This is also reflected within the computed degree of distinguishability for the traditional approach: 0.8. [PSB$^+$]

In contrast, the degree of distinguishability for the fuzzy approach is higher, namely 1. The degree of 1 means that all services can be distinguished from each other on the basis of the $[n, p]$ intervals. Figure 4.25 a) shows the $[n, p]$ intervals as well as the best and worst ranks possible for each service. The best rank is obtained by considering a service's upper bound p while considering all other services' lower bounds n. The worst rank is obtained by considering a service's lower bound n while considering all other services' upper bounds p. For example, s10 could be assigned with rank one, if complete information would lead to a matching result similar to the upper bound of 0.85. However, complete information could also have negative consequences for this service, e.g., the matching result could develop towards the lower bound of 0.1 while the other services develop towards higher results. In this situation, s10 would get the very worst rank. In contrast to these high variances between possible ranks for s10, Service s8 —having a smaller interval— is a good choice in any way as it could be ranked first, but it will never be worse than Rank 4. In addition to this knowledge especially valuable for the service requesters, this information is also of a high importance for service providers because it allows them to evaluate whether they should spend effort in providing more information.

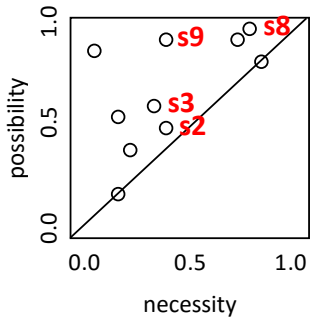
Moreover, Figure 4.25 b) depicts the results in a possibility/necessity diagram. In this diagram, each service is represented as a point. The upper right corner represents services that can be viewed as certainly good matches (both n and p are high). In contrast, results in the upper left corner are potentially but not necessarily good. The more certain the matching result $[n, p]$ of a service, the closer it is located to the diagonal (because $n \approx p$, forming a narrow interval). The currently depicted matching results contain a high amount of fuzziness reflected by large intervals. This indicates that the representation on the basis of $[n, p]$ intervals provides more information than the traditional results in the table and, thereby, supports decision-making much better. [PSB$^+$]

4.8.2.2 Experiment 2.1.2: TrustRadius Ratings

The setting for Experiment 2.1.2 is similar to the first one except for the use of real instead of manually specified input data. Furthermore, this experiment handles much more input data.

Service	Trad. Result	New Result	Best/Worst Ranks
s1	0.6	[0.7, 0.9]	1 / 5
s2	0.4	[0.4, 0.5]	4 / 9
s3	0.4	[0.35, 0.6]	4 / 9
s4	0.8	[0.8, 0.8]	1 / 5
s5	0.2	[0.2, 0.2]	8 / 10
s6	0.4	[0.25, 0.4]	4 / 9
s7	0.6	[0.2, 0.55]	4 / 10
s8	0.8	[0.75, 0.94]	1 / 4
s9	0.8	[0.4, 0.9]	1 / 9
s10	0.6	[0.1, 0.85]	1 / 10

a) Comparison of traditional and our new results



b) New results depicted in a necessity/possibility diagram

Figure 4.25: Results of Experiment 2.1.1 (based on [PSB+])

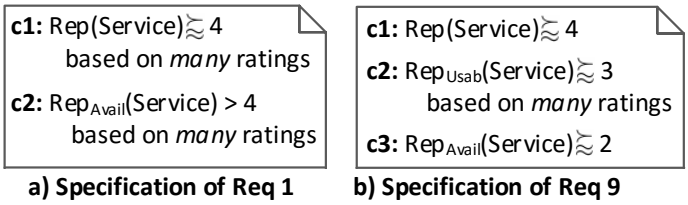
Table 4.5: Degrees of Distinguishability (Exp. 2)

	A&B (F)	A&B (T)	CMS (F)	CMS (T)
Req 1	0.802	0.498	0.652	0.471
Req 2	0.447	0.435	0.223	0.116
Req 3	0.087	0.000	0.116	0.000
Req 4	0.763	0.237	0.898	0.508
Req 5	0.170	0.087	0.622	0.114
Req 6	0.968	0.569	0.936	0.568
Req 7	0.826	0.660	0.813	0.570
Req 8	0.842	0.498	0.886	0.487
Req 9	0.941	0.743	0.909	0.729
Req 10	0.648	0.597	0.447	0.514

Procedure

In order to perform this experiment on the basis of real data, we collected ratings from the software rating website *TrustRadius.com* [Tru], where software is classified into different categories regarding the functionality. Ratings are given by real people that used the software in the past and thereby judge different properties of the software, e.g., its usability, its availability, and its performance. At time of evaluation, TrustRadius listed 34 services with in total 520 Ratings in the category 'Content Management Systems' (CMS) and 23 services with a total of 615 Ratings in the category 'Accounting and Budgeting Services' (A&B).

We matched these services against 25 requirements specifications (including one to five conditions) that have been specified by computer science students and PhD students. None of these test persons had prior knowledge about the internals of the matching algorithm and how the matching results are produced exactly. Figure 4.26 shows two examples of the requirements specifications produced by the test persons. The contexts allowed to be used within the conditions in these requirements specifications were limited to the kinds of ratings TrustRadius contained most data for at time of evaluation: Overall Rating, Usability Rating, Availability Rating, and Performance Rating. In addition to the Content Management Systems, we repeated this experiment for the A&B category. In total, we analyzed 340 matching results for CMS and 230 matching results for A&B. [PSB+]



a) Specification of Req 1 **b) Specification of Req 9**

Figure 4.26: Example Requirements Specifications for Experiments 2.1.1–2.1.3 [PSB+]

Results and Discussion

Table 4.5 provides a summary of the degrees of distinguishability of the matching results for the first ten requirements specifications within the two data sets. The results are shown, both for the new, fuzzy (F) and for the traditional approach (T). As can be seen, the fuzzy approach consistently improves distinguishability across all requesters. This finding is in line with our expectations. In the following, we discuss a selection of the results in more detail. For the complete list of results, refer to Appendix D.

As an illustration, Figure 4.27 shows the diagrams for Requester 1, 6, 8 and 9 for the A&B and for the CMS services. As can be seen, for the specification of Requester 1, the points are located closer to the diagonal than those of the three others. This means that the matching for Requester 1 lead to more precise evaluations of the services. In line with Figure 4.26 a), the diagram also shows that the highest possibility degrees do not exceed 0.5. This fact indicates that `Req 1` is quite demanding. The requirements of Req 9 (see Figure 4.26 b)) are lighter (at least for the availability ratings), as can be seen from the many services with matching results having a possibility degree of 1. Nevertheless, there are only a few services having a high necessity degree, too, suggesting some obvious choices for this requester. In order to achieve even more useful results, she may consider tightening her requirements and starting a new matching iteration. As we can learn from these facts, the new matching results are more informative and, thus, much more useful than the simple scores of 0 that are likely to be obtained in the traditional approach.

Taking a closer look at the matching results, we can find further indications for the benefit of our new interval results. For example, when matching requirements specification Req 6 to the CMS services, the traditional results are only able to distinguish between three groups of services: Services that do not match at all (result of 0), services that match with a result of 0.25, and two services that match with a result 0.5. The services with the result of 0.5 include the well-known CMS services Joomla! and Drupal. In contrast, considering the interval results, Joomla! having a result of [0.4,0.65] dominates Drupal having a result of [0.217,0.55]. According to these intervals, Joomla! is potentially the better match and more certainly a good match as well. Interestingly, the interval results also reveal that some services among those with a traditional result of 0.25 are potentially a better choice. Among these is WordPress which is potentially a much better match according to its interval result of [0.375, 0.75], even though Joomla! is more certainly a good match: p is lower but n is higher. On the basis of these results, a risk-averse requester would choose WordPress, and a risk-affine requester would rather choose Joomla!. Thereby, the interval results support requesters in finding the most appropriate choice according to their own user-specific characteristics much better than the traditional results. In general, when considering the traditional results, some services often hide within a group of equal matching results, complicating the selection,

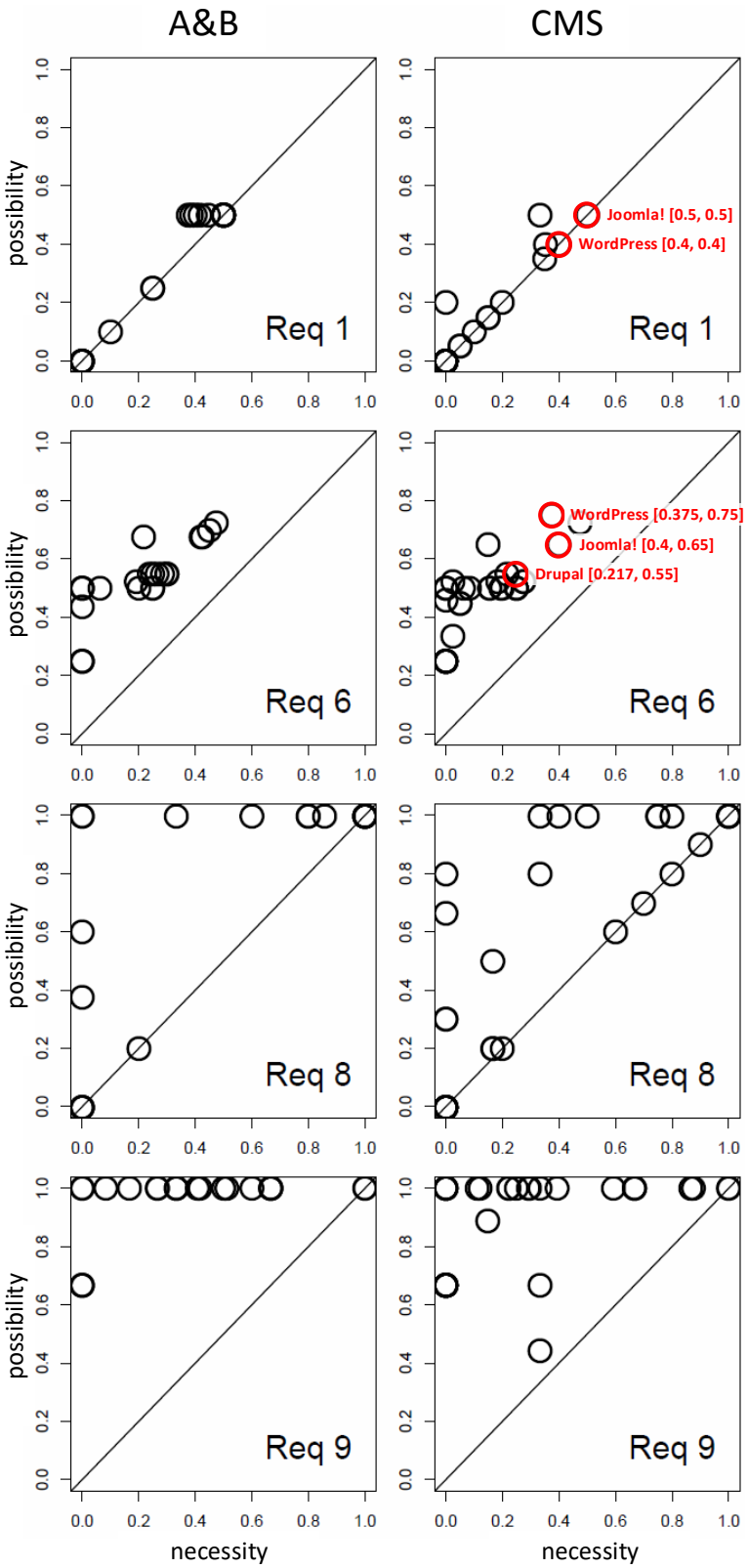


Figure 4.27: Results of Experiment 2.1.2 [PSB$^+$]

while the interval results more clearly identify single services as good candidates depending on the amount of risk a requester is willing to take. Furthermore, the matching results strongly depend on the requirements specification again: In the results for most of the requirements specifications, Joomla! clearly dominates WordPress, but not for Req 6. This indicates that it is useful to work with more complex requirements specifications allowing the requesters more freedom in specifying the services they are satisfied with than related approaches do. [PSB+]

4.8.2.3 Experiment 2.1.3: Scalability

Unlike the first two experiments within this case study, the next experiment addresses scalability.

Procedure

In a third experiment, we tested how our approach scales to a broader range of input data, i.e., a high number of requirements specifications as well as a high number of services and a high number of ratings. For this purpose, we randomly generated 1000 requirements specifications and matched them against all the services in all the categories available from TrustRadius. At evaluation time, these were in total 924 services with altogether 10907 ratings. Based on these 924 services, we executed 924000 matching runs and measured the runtime on a Dell Latitude notebook operating Windows 8.1, 64-bit with 8GB RAM and an Intel(R) Core(TM) i7-2640M CPU @2.80GHz. [PSB+]

Results and Discussion

The mean total runtime for all 924000 matching runs took 642 seconds (10.7 minutes). With respect to the high amount of input data, we take this number as a satisfying result because, in general, we do not expect that so many services need to be matched at the same time in most usage scenarios. The matching on the basis of reputation data can often be ommitted for a number of services because they can be filtered out by matching steps being executed earlier within the matching process with respect to structural or behavioral properties. Furthermore, manual inspection showed that much of the measured time is spent on saving matching results in the form of MatchBox' hierarchical matching result trees that are constructed in order to provide the user with as much information about the matching as possible, including intermediate matching results. For example, the result file for the 924000 matching results constructed within the scope of this experiment was 2.28 GB. [PSB+]

4.8.2.4 Experiment 2.1.4: Performance Matching

This experiment is based on data from a performance simulation of self-adaptive services, as described in [BLB13]. In [BLB13], three different service variants have been simulated with varying architectures (5s vs. 10s vs. 20s batches) of a cloud-based web service. In the following, we call these three services S_a, S_b, and S_c. [PSB+]

Procedure

The performance simulation described in [BLB13] predicts response times and the MTTQR for each self-adaptive service architecture with an exponentially distributed interarrival rate $\lambda = 4\frac{1}{second}$ for 100 seconds. Within these 100 seconds, 393 respectively 376 predicted

response times as well as one MTTQR value for each of the services have been obtained. According to the results reported in [BLB13], S_a is the best performing service, dominating S_b, while S_b again dominates S_c.

Within the scope of our experiment, we compared the reported results according to this simulation data to the matching results computed by our fuzzy performance matcher. The simulated data was matched against four manually specified requirements specifications Req 1', Req 2', Req 3', and Req 4'. These requirements specifications increase in their strictness starting with a requested response time lower than 3 seconds and a requested MTTQR lower than 40 seconds in Req 1' down to a requested response time lower than 1.5 seconds and a requested MTTQR lower than 10 seconds in Req 4'.

Results and Discussion

Table 4.6: Matching Results for Experiment 2.1.4 [PSB+]

	S_a	S_b	S_c
Req 1'	[1.0 , 1.0]	[1.0 , 1.0]	[0.0 , 0.041]
Req 2'	[1.0 , 1.0]	[0.649 , 0.725]	[0.0 , 0.0]
Req 3'	[0.9 , 1.0]	[0.300 , 0.467]	[0.0 , 0.0]
Req 4'	[0.599 , 0.765]	[0.146 , 0.39]	[0.0 , 0.0]
Best/Worst Ranks	1/1	1/2	3/3
Rank in [BLB13]	1	2	3

Table 4.6 shows the matching results for Req 1', Req 2', Req 3', and Req 4' matched to S_a, S_b, and S_c. As we can see, Service S_a receives the best rank for all requirements specifications, while S_c always receives the worst rank. S_b is on Rank 2 for Req 2' to Req 4' and on Rank 1 for Req 1' together with S_a because Req 1' is very undemanding.

Thus, all in all, our matching results are in line with the results reported in [BLB13]. However, in addition, our results also provide the extra information of fuzziness. For example, regarding Req 3', S_a is not necessarily a full match. Due to imperfect information, we can only determine that the matching result is somewhere between 0.9097 and 1. This information is helpful for a very risk-averse requester when selecting a service. But also the provider of S_b can learn a lot from the results: Providing more information can potentially improve her ranking in general. However, providing more information is definitely not sufficient to rank her service best for most of the example requirements specifications. To improve her revenue, she also needs to do internal changes. [PSB+]

4.8.3 Case Study 2.2: Evaluation of Fuzziness Quantification

In the following, we describe how we evaluated the fuzziness quantification concepts introduced in Section 4.6.1.6. This includes three experiments. While the first one focuses on reputation matching, the others deal with privacy matching.

4.8.3.1 Experiment 2.2.1: Reputation Matching

Following the ideas presented by Esfahani et al. [EKM11], we applied a sensitivity analysis in order to evaluate our fuzziness estimation regarding transformation-induced fuzziness in fuzzy reputation matching.

Procedure

In order to evaluate the fuzziness scores assigned during the fuzziness estimation in fuzzy reputation matching (see Section 4.6.1.6), we simulated set of matching executions by systematically varying the fuzzy set parameters x_{r1}, x_{r2}, x_{p1}, x_{p2}, and x_{p3} (see Figure 4.18). This is done by iterating different values using a fixed increment (e.g., 0.5 and 0.15) in nested loops. x_{r1} has been incremented in the most inner loop so that we could investigate *result groups* where x_{r2}, x_{p1}, x_{p2}, and x_{p3} are equal but x_{r1} varies. This is important because x_{r1} is the uncertain parameter while the other parameters are known, as explained in Section 4.6.1.6.

Accordingly, the independent variables in this experiment are the different fuzzy sets. The dependent variables are the result groups as defined above. For each result group, we measured the dispersion regarding the p-values, the n-values, and the difference $p - n$. This allows us to aggregate average differences per fuzziness score. The higher the average differences, the more sensitive to uncertainty are the matching results for this fuzziness score.

Furthermore, we measured the differences in the result groups with fuzziness scores HIGH and VERY HIGH. This is done by measuring homogeneity as introduced in Section 2.6. The more homogeneous the results are, the more balanced the proportion of mismatches to fuzzy matches within a group. For example, if all results within one result group are full mismatches, then homogeneity is 1 and there is no risk that the uncertain location of x_{r1} leads to a false positive or a false negative. In contrast, the more imbalanced the proportion of mismatches to fuzzy matches is, the higher the probability that the actual returned result is wrong. Accordingly, a lower average homogeneity indicates a higher risk for false negatives and false positives cased by uncertainty. Like in the above described experiment, we repeated this experiment for different increments.

Results and Discussion

Table 4.7: Statistics for Fuzziness Scores in Reputation Matching

	Fuzziness Score	avg PDiffs	avg NDiffs	avg DiffDiffs	Mismatch&Fuzzy Cases	all Cases
incr = 0.5	NONE	0	0	0	0	330
	LOW	0	0.1966	0.1966	0	484
	MEDIUM	0.27	0.4362	0.311	0	506
	(VERY) HIGH	0.5656	0.4682	0.4223	130	130
incr = 0.15	NONE	0	0	0	0	46376
	LOW	0	0.2451	0.2451	0	51486
	MEDIUM	0.2993	0.4787	0.385	0	53234
	(VERY) HIGH	0.5927	0.4804	0.4562	21760	21760

As Table 4.7 shows in the first five columns, the differences increase with an increasing fuzziness score. Especially the p-differences show this development: There are no

differences for p in all matching executions where the fuzziness score is LOW as p is always 0; there is an average difference of 0.27 (resp. 0.2993) within matching executions with the fuzziness score MEDIUM, so p varies a bit in those cases; for cases with the fuzziness score (VERY) HIGH (including HIGH and VERY HIGH), p varies with a far higher average difference of 0.5656 (resp. 0.5927).

The rightmost column shows how many result groups have been assigned to each fuzziness score. Remember that each result group has a different assignment of x_{r2}, x_{p1}, x_{p2}, and x_{p3}. The second column from the right shows the amount of result groups that contain both full mismatches ($n = p = 0$) and fuzzy matches ($0 \leq n \leq p \leq 1$). These are interesting cases as these are the cases where the uncertain x_{r1} makes the difference between a full mismatch and a fuzzy match, i.e., cases where serious false negatives and false positives can occur. As we see in the table, such cases only occur for matching executions with the fuzziness score (VERY) HIGH. This finding demonstrates the correctness of our discussion about these cases in Section 4.6.1.6. Moreover, the table shows that all cases with a fuzziness score of (VERY) HIGH are such cases (130 from 130, respectively 21760 from 21760). This makes sense as $x_{r1} < x_{p3}$ and $x_{r1} > x_{p3}$ can both hold, independent from x_{r2}, x_{p1}, x_{p2}, and x_{p3}, as long as $x_{r2} > x_{p3}$ holds.

Regarding the homogeneity measurements, we found the following: For an increment of 0.5, we measured an average group homogeneity within HIGH cases of 0.6394 and within VERY HIGH cases of 0.5754. For an increment of 0.15, we measured an average group homogeneity within HIGH cases of 0.7384 and within VERY HIGH cases of 0.6116. As a result, both times, the homogeneity for the VERY HIGH cases is higher. This means that, taking the increment of 0.15 as an example, matching results with a fuzziness score of VERY HIGH lead to more false negatives and false positives than matching results with a fuzziness score of HIGH in approximately 12% of the cases.

These findings indicate the effectiveness of our fuzziness scores: The scores truly reflect the uncertainty within matching results and are, thus, useful for the user's decision-making. We also performed this experiment with further increments lower, higher and in between 0.5 and 0.15 and the findings were the same.

4.8.3.2 Experiment 2.2.2: Privacy Matching with Expert Results

In this experiment, we performed our fuzzy privacy matching approach on 21 pairs of exemplary requirements specifications and provided service specifications from the university management domain. This experiment has been published in [PAPS15].

Procedure

For all these pairs, we first determined the matching results manually by expert knowledge, on the basis of complete specifications. After that, we ran our fuzziness estimation approach introduced in Section 4.6.2.2 on incomplete versions of these specifications. Next, the experts' matching results based on full knowledge were compared to the results the matcher determined for the incomplete specifications. Thereby, we could determine correct matches (the calculated matching result was equal to the expert's result, i.e., a true positive or a true negative) as well as mismatches (the calculated matching result was not equal to the expert's result, i.e., a false positive or a false negative). Furthermore, we measured fuzziness scores and compared them to the matching results. [PAPS15]

Results and Discussion

Table 4.8: Matching Results and Fuzziness Scores from Experiment 2.2

	Result	Expected Result	Actual Result	Fuzziness Score
IP 1	TP	1	1	LOW
IP 2	TP	1	0.583	LOW
IP 3	FP	0	1	MEDIUM
IP 4	TN	0	0	LOW
IP 5	TN	0	0	LOW
IP 6	TN	0	0	NONE
IP 7	TN	0	0	NONE
IP 8	TN	0	0	LOW
IP 9	TN	0	0	NONE
IP 10	TN	0	0	NONE
IP 11	TP	1	0.589	MEDIUM
IP 12	TP	1	1	MEDIUM
IP 13	TN	0	0	NONE
IP 14	TN	0	0	NONE
IP 15	TN	0	0	NONE
IP 16	TN	0	0	NONE
IP 17	TN	0	0	NONE
IP 18	TN	0	0	NONE
IP 19	TN	0	0	NONE
IP 20	TP	1	1	VERY_HIGH
IP 21	FP	0	1	VERY_HIGH

Table 4.8 shows the expected matching results determined by expert knowledge, the actual matching results our matching approach calculated, and the estimated fuzziness scores. Furthermore, the results are classified as true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). On the basis of these results, we determined precision (0.714) and recall (1) as defined in Section 2.6.

All in all, the results show that the fuzziness scores for correct matching results (true positives and true negatives) were rather low while the results for false positives (and false negatives) were comparatively high in 95% of all cases. Assuming that a user uses the matching results in combination with the fuzziness scores, false positives are reduced. For example, with a selection strategy that does not select services with a high (> 0.5) fuzziness score, the precision increases to 0.857. [PAPS15]

From these results, we learn that fuzziness scores are worth considering when selecting a service based on a matching result, at least in the context of the analyzed example situation.

4.8.3.3 Experiment 2.2.3: Optimistic/Pessimistic Privacy Matching

For this experiment, remember the difference between an optimistic and a pessimistic matching approach. "(1) The optimistic case is to consider all missing information on the provider side as the most strict case, e.g., retention period = 0. This potentially increases

the number of false positives. This means, the matcher returns a high matching result although, in reality, the service does not match. (2) The pessimistic case is to substitute missing information with the least strict case, e.g., retention period = *. This can lead to false negatives because, in uncertain cases, the matcher might return low matching results although, in reality, the service would be a good match." [PAPS15]

Procedure

In reality, the user is not able to influence a third-party's matcher in a way that she can choose between optimistic and pessimistic execution. For evaluation purposes, we used a configurable matching approach and executed it in both strategies on the basis of 28 exemplary input pairs in order to measure the differences. In particular, we expected a correlation between the differences and the estimated fuzziness scores, as explained below.

Results and Discussion

Table 4.9 shows the results from both matching runs and their difference for each pair. Furthermore, the estimated fuzziness is depicted. As the results show, high fuzziness scores do not always correlate with a high difference. This is due to the fact that the fuzziness also incorporates characteristics like sensitivity and not only the likelihood for false positives or false negatives. However, when comparing the average difference for low fuzziness scores (NONE, LOW, MEDIUM) to the average difference for high fuzziness scores (HIGH, VERY_HIGH, COMPLETE), we find that the difference for low scores is significantly lower (0.416) than for high scores (0.714). This is in line with our expectations because it shows that false positives and false negatives are more likely for matching results assigned with higher fuzziness scores than for results with lower scores.

4.8.4 Discussion Summary

All in all, from our evaluation results, we can conclude that a user can make a more informed decision based on the results delivered by our new approach compared to traditional results.

Especially Experiment 2.1.1 and Experiment 2.1.2 show the effectiveness of the intervals calculated in our fuzzy-logic-based approach. Furthermore, we showed in Experiment 2.1.3 that this approach is also able to handle a big amount of data, which makes it applicable in practice. The broad applicability to further specifications as well as the correctness of our matching results compared to reference results from the literature have been shown in Experiment 2.1.4.

Experiment 2.2.1 to 2.2.3 address the fuzziness scores from the fuzziness estimation step and show that these scores indicate the dimension of the risk the usage of a fuzzy matching result is inflicted with in an appropriate way.

However, there are several threats to the validity of our evaluation as discussed in Section 4.8.6.

4.8.5 Satisfaction of the Requirements

As discussed in the following, the requirements that we collected in Section 4.3 are satisfied to a high degree:

Table 4.9: Matching Results and Fuzziness Scores from Experiment 2.3

	Result Opt.	Result Pess.	Diff	Fuzziness
IP 22	1	0	1	MEDIUM
IP 23	1	1	0	NONE
IP 24	1	1	0	NONE
IP 25	0.75	0	0.75	VERY_HIGH
IP 26	1	1	0	NONE
IP 27	0.1819	0.1819	0	NONE
IP 28	0.5	0.5	0	NONE
IP 29	1	0.5	0.5	LOW
IP 30	1	0	1	MEDIUM
IP 31	1	0.3572	0.6428	LOW
IP 32	0.75	0.125	0.625	HIGH
IP 33	1	0.2	0.8	MEDIUM
IP 34	0.5	0	0.5	HIGH
IP 35	0.6364	0.3637	0.2727	LOW
IP 36	1	0	1	MEDIUM
IP 37	1	1	0	MEDIUM
IP 38	1	1	0	NONE
IP 39	0.875	0	0.875	VERY_HIGH
IP 40	1	0	1	VERY_HIGH
IP 41	0.5	0	0.5	COMPLETE
IP 42	0.5	0.3637	0.1363	LOW
IP 43	1	0.5	0.5	LOW
IP 44	1	0	1	MEDIUM
IP 45	1	0.3572	0.6428	LOW
IP 46	0.75	0	0.75	VERY_HIGH
IP 47	1	0	1	VERY_HIGH
IP 48	0.6819	0	0.6819	COMPLETE
IP 49	0.6364	0.1819	0.4545	HIGH

(***R1***) *Deliver Unadulterated Matching Results:* Fuzzy Matching approaches constructed in a way as described above do not deliver adulterated matching results. The reason is that they explicitly distinguish between the matching result and the quantified fuzziness and present them in a separate way.

(***R2***) *Handle Different Types and Sources of Fuzziness:* In general, our fuzzy matching concepts are constructed in a way that they cope with all fuzziness types and fuzziness sources introduced in Section 4.4. The concrete approaches presented in Section 4.6 cover gradedness, vagueness, uncertainty, and incompleteness as well as requester-induced, provider-induced, and transformation-induced fuzziness. Other types and source, e.g., algorithm-induced fuzziness are left for future work, as discussed in Section 6.2.

(*R3*) *Make Source of Fuzziness Transparent:* Step 3 (Final Fuzziness Estimation) and Step 4 (Result Construction) take care of determining and presenting the arisen fuzziness sources.

(*R4*) *Make Extent of Fuzziness Transparent:* Step 3 (Final Fuzziness Estimation) and Step 4 (Result Construction) take care of determining and presenting the arisen extent of fuzziness. The evaluation presented in Section 4.8 shows that the determined fuzziness scores form an appropriate basis towards supporting a more informed decision-making.

(*R5*) *Support Diverse Matching Approaches:* This requirement is not completely satified: While the fuzzy matching approach presented in Section 4.6.1 is tailored to approaches based on fuzzy logic, the approach introduced in Section 4.6.2 is used for approaches based on (numerical) specifications where incompleteness is identifiable (see Section 4.6.2.3). Future work should also target further matching approaches. For example approaches based on graph matching still represent a challenge.

(*R6*) *Support Complex Specification Languages:* The satisfaction of this requirement is related to *R5*. As mentioned above, future work should cover further kinds of specification languages, e.g., graph-based languages for protocol specifications.

(*R7*) *Efficiency:* The experiment described in Section 4.8.2.3 indicated the scalability of our fuzzy logic-based approach. The scalability of an approach extended with our fuzziness estimations depends on the concrete matching approach. A detailed evaluation of the efficiency is subject to future work.

(*R8*) *Non-Invasiveness:* Section 4.6.2 describes an approach where fuzzy matching is applied as an extension to a non-fuzzy approach. The experiment described in Section 4.8.3.3 showed that it was able to treat that approach as a black-box and still get useful fuzzy matching results.

Section 4.10 compares the extent to which our approach satisfies the requirements with related work.

4.8.6 Threats to Validity

In the following, we discuss the most significant threats to the validity of our validation.

Although we were able to apply our reputation matching approach to a lot of real data taken from the TrustRadius website, we still have rather few ratings per service and per property. This leads to a number of large intervals (almost [0,1]). However, in these cases, the user is at least provided with the information that the matching result is unknown. This is not the case for the traditional results where the user is lead to believe in an adulterated result which may consequently cause a bad choice. Using the more informance interval results, the user is provided with the possibility to decide on her own how much risk she is willing to take when selecting a service based on a matching result. [PSB$^+$]

Another threat to the validity of our validation is the fact that the traditional matching approach we used as a baseline could also perform differently. For example, another (unfuzzy) matching approach leading to more gradual results, also for the single conditions, would also make an interesting comparison. The same holds for the results gained from performance simulations and used for comparison to our matching results. For future evaluation activities, we need to find more possibilities to compare our matching results to other approaches in spite of issues like availability of example specifications

and comparability of fuzzy matching results to results in simpler formats complicating the evaluation of our approach. [PSB+]

Similar to Section 3.10.4, another threat to validity is that only a few metrics have been taken into account. For example, instead of using one metric to measure distinguishability, multiple analytical metrics should be used in order to compare their results to each other [BR08].

Some of our experiments also included manually specified specifications. As already discussed in the beginning of this section, such a procedure is always inflicted with a risk for biased results. The same holds for manual interpretation of matching results.

Furthermore, also in this part of the thesis, we only present a level-I-validation [BR08]. To extend this by a level-II-validation, more extensive user studies are needed as discussed in Section 6.2.

In general, we cannot exclude the possibility that our experiments lead to significantly other results when applied to another context, e.g., to another domain, to another set of example specifications, or to other matchers. Thus, there is a need for more repetitions of these experiments.

4.9 Limitations

Currently, we investigated fuzzy matching concepts only for a small set of matching approaches. In the future, this research needs to be extended with further matching approaches covering other service properties.

For example, the fuzziness quantification as an extension that we presented in Section 4.6.2 currently only works for numerical values. We need to extend it in order to cover further properties. Examples are ontological terms like the visibility property from the considered privacy specifications, or graph-based specifications used for protocols or conditions. The latter is especially difficult: For protocols specified as automata, the specifications themselves do not reveal whether they are complete or incomplete. The same holds for pre- and post-conditions specified in formalisms like first-order logic. We propose a first step towards a solution to deal with such specification languages in Section 4.6.2.3. However, this solution requires extra knowledge and effort spent by the requesters or providers. Future studies should also address the more general case.

Furthermore, the applicability of the fuzziness quantification as an extension is limited also in another way: The integrated matching approaches can be used as a black-box, however, the one adding the fuzziness extension still needs knowledge about how the matching approach works in order to deal with the arisen fuzziness in an appropriate way. For example, she needs to know whether the integrated matching approach follows an optimistic or a pessimistic strategy, which is not always realistic.

Possibilities for the improvement of our fuzzy-logic-based approach include the refinement of service data restrictions (see Section 4.6.1.3). At the moment, we work with hard thresholds here. However, also for such constraints, fuzzy thresholds could be more useful because requesters' do not necessarily have hard constraints regarding properties like the age of ratings.

Furthermore, in our fuzzy-logic-based approach, we exclusively use trapezoidal fuzzy sets. However, sigmoidal fuzzy sets could be even more appropriate, as the requester's gradedness does not necessarily increase linearly. It could also increase exponentially.

The derivation of fuzzy sets is only one example where our approach is based on domain knowledge. Another example is the selection of aggregation operators. We can propose a certain aggregation operator based on its characteristics, like compensation, however, in order to determine which aggregation really works best in a real-world scenario, user studies are needed.

Our prototype is limited to the main concepts and does not realize every detail described in this chapter. For example, we also did not fully implement fuzziness pre-checks and fuzziness coping strategies as their realization is trivial and they have not been used within the experiments.

4.10 Related Work

Just like in Section 3.12, approaches related to the work described in this chapter have been rigorously surveyed according to the guidelines for systematic literature reviews by Kitchenham et al. [KBB+09, KC07, BKB+07]. In the following, we give more details about the survey procedure we followed and the results this procedure lead to.

4.10.1 Survey Procedure

The survey procedure used for this section is similar to the survey procedure described in Section 3.12. Thus, only the varying parts are highlighted in the following.

The survey described in the following is based on our earlier published surveys [Pet13, PvDB+13] which serve as a primary study. This primary study already showed that the survey procedure was applicable and lead to useful results. In addition to the primary study, we have now concretized our definitions of fuzziness. Our survey question is: *To which extent do existing approaches deal with fuzziness with respect to our requirements?*

Like in Section 3.12, papers were included according to lists of predefined keywords. In addition to *Service/Component Keywords* and *Matching Keywords*, here, we also use *Fuzziness Keywords*. Fuzziness keywords describe terms that indicate one of the fuzziness types. Examples are "fuzzy", "uncertain", "approximate", and "vague". The complete keyword lists can be found in Appendix C.

We used the same survey process as in Section 3.12. Only the second step changed:

1. see Section 3.12
2. In the second step, we scanned all the abstracts for the fuzziness keywords. Only papers with at least one fuzziness keyword mentioned in the abstract have been included into the next step. Furthermore, the fuzziness keyword had to refer to the matching keyword and to the approach described in the publication. For example, if not the matching but other activities, e.g., composition, are described to be fuzzy, the paper is excluded.
3. see Section 3.12
4. see Section 3.12

Using this strategy, we started with a number of **409** publications in the first step. **340** papers had been excluded via title and abstract. Further **31** papers had been excluded by further exclusion criteria listed in Appendix C. Thus, we ended up with **49** papers (grouped into **33** approaches) for detailed reviews and to be discussed in the following.

4.10.2 Comparison of Fuzzy Service Matching Approaches

We reviewed the selected publications with respect to seven criteria that reflect the requirements described in Section 4.3:

- Considered Fuzziness Types: This criterion answers, which of the fuzziness types introduced in Section 4.4.1 have been considered in the described matching approach. Possible values range from Gradedness ("Gra"), over Optionality/Variability ("Opt/Var"), Vagueness/Imprecision ("Vag/Imp"), Uncertainty ("Unc"), and Approximations/Relaxations ("App/Rel"), to Incompleteness ("In"). Note that always only the most significant fuzziness types are listed. For example, even though vagueness may lead to uncertainty in most cases, only vagueness is listed. This criterion refers to Requirement *R2*.
- Considered Fuzziness Sources: This criterion answers, which of the fuzziness sources introduced in Section 4.4.2 have been considered in the described matching approach. Possible values are Requester ("Req"), Provider ("Prov"), Algorithm ("Algo"), and Transformation ("T"). This criterion refers to Requirement *R2*.
- Result Format: This criterion indicates how comprehensive the matching results returned by the described matching approach are and to which extent they reflect induced fuzziness. Possible values include "score" (one numerical value out of a continuous range of values, e.g., percentage values), "degree" (one result out of a fixed number of result classes), "boolean" (e.g., "select/reject" results), and others. This criterion refers to Requirements *R1*, *R3*, and *R4*.
- Aspect: Which kind of service properties are matched? Possible values are Non-Functional Properties including Quality Properties ("NFPs"), inputs/outputs ("IO"), inputs, outputs, preconditions, effects/post-conditions ("IOPE"), ("Keywords"), and ("various"). This criterion refers to Requirement *R6*.
- Complexity: This helps to answer the question of how complex or how expressive the underlying specification language is. Possible values are "low" (e.g., used for purely numerical or simple and strongly restricted linguistic representations), "medium" (for more complex representations, e.g., incorporating ontologies), and "high" (very expressive languages, e.g., including a flexible description of preconditions and effects). This criterion refers to Requirement *R6*.
- Kind: The kind of specification language supported by the described matching approach. Possible values are numerical ("num."), "ontological" ("ontol."), linguistic terms ("ling."), fuzzy sets ("FS"), natural language ("NL"), and "various" (including several of the other values). This criterion refers to Requirements *R5* and *R6*.
- Logic: Which kind of logic underlies the proposed matching approach? Possible values include Fuzzy Logic ("FL"), Ontologic, respectively Description Logic ("ont/DL"), and others. This criterion refers to Requirement *R5*.

Table 4.10: Comparison of Fuzzy Matching Approaches: Fuzziness and Result

Approach	Considered Fuzziness Types	Considered Fuzziness Sources	Matching Result Format
[AAY11]	Gra, Vag/Imp	Req	score
[BBM09, BBM10]	Gra, Vag/Imp, App/Rel	Prov, Req, Algo	score
[BL08, BL11] [LSH+12, LSHY09]	App/Rel	Algo	score
[CBF05]	App/Rel	Algo	score
[DCCH07]	Vag/Imp, Gra	Req	score
[CCL+05]	Vag/Imp	Req, Prov	score
[CYLT05, HCL05] [HLCY06, HLL+05] [LLCY06, LLCY08]	Vag/Imp	Req	score
[FLS08]	App/Rel	Req, Algo	multiple scores
[JCL09]	App/Rel	Algo	score
[KFS06, KF07] [KKF08, KFS09]	App/Rel	Algo	degree
[KKR09, KKRKS07] [KKRSK07, KKR07]	Opt/Var, Vag/Imp, Gra	Req, Prov	score
[KK12b, KK10]	App/Rel	Algo	degree
[LFT12]	Vag/Imp, Gra	Req	score
[LL08]	App/Rel	Algo	score
[MZH08]	App/Rel	Algo	score
[PCP09]	In	Req, Prov	score
[PHC08]	Vag/Imp	Req, Prov	boolean
[PRVM13]	Vag/Imp	Req, Prov	score
[QWO13]	Vag/Imp	Prov	score
[RS03]	Vag/Imp	Prov	Reuse decisions
[SCS10]	App/Rel	Algo	boolean
[SFHC13]	App/Rel	Algo	score
[SS08]	In	Req, Prov	degree
[ŞT09, ŞT06]	Vag/Imp	Req, Prov	(weak/strong) select/reject
[SWKL02]	App/Rel	Algo	score
[TAS11]	Vag/Imp	Req	score
[TCM08]	App/Rel	Algo	score
[TGRBD07]	App/Rel	Algo	score
[TOAD07]	App/Rel	Algo	score
[Wan09, WCL+06]	Vag/Imp, In	Req, Prov	score
[WLH07]	Vag/Imp	Req, Prov	score
[XF07]	Vag/Imp	Req, Prov	score
[ZSK15]	Opt/Var, Vag/Imp	Req, Prov	score
Our approach	All	All	Interval / Score+Fuzziness

Table 4.11: Comparison of Fuzzy Matching Approaches: Specifications and Complexity

Approach	Specified Aspect	Specification Complexity	Specification Kind	Matching Logic
[AAY11]	NFPs	low	num.	FL
[BBM09, BBM10]	NFPs	low	ling.	FL
[BL08, BL11] [LSH+12, LSHY09]	IOPE	medium	ontol.	FL
[CBF05]	IOPE	high	num. + NL	ont/DL
[DCCH07]	NFPs	low	FS	FL
[CCL+05]	NFPs	low	ling., num.	FL
[CYLT05, HCL05] [HLCY06, HLL+05] [LLCY06, LLCY08]	IOPE	medium	num., ontol.	FL
[FLS08]	IOPE	medium	ontol.	FL
[JCL09]	NFPs	low	ling.	-
[KFS06, KF07] [KKF08, KFS09]	IO	medium	ontol.	ont/DL
[KKR09, KKRKS07] [KKRSK07, KKR07]	various	high	ontol.	FL
[KK12b, KK10]	IOPE	medium	ontol.	ont/DL
[LFT12]	NFPs	low	ling.	FL
[LL08]	IO	low	ontol.	ont/DL
[MZH08]	keywords	low	NL	Prob. Theory
[PCP09]	NFPs	high	ontol.	ont/DL
[PHC08]	keywords	low	ontol.	ont/DL + FL
[PRVM13]	NFPs	medium	ling., num.	FL
[QWO13]	Ratings	low	ling.	FL
[RS03]	various	medium	various	FL
[SCS10]	IO	medium	ontol.	ont/DL
[SFHC13]	various	high	NL	ont/DL
[SS08]	IO	low-medium	ontol. FL	ont/DL
[ŞT09, ŞT06]	NFPs	low	ling.	FL
[SWKL02]	various	high?	ontol.	ont/DL
[TAS11]	NFPs	low	ling.	FL
[TCM08]	various	medium	ontol.	ont/DL
[TGRBD07]	IO	medium	ontol.	ont/DL
[TOAD07]	various	medium	ontol.	ont/DL
[Wan09, WCL+06]	NFPs	low	num. / ling.	FL
[WLH07]	NFPs	low	ling.	FL
[XF07]	NFPs	low	ling.	FL
[ZSK15]	NFPs	medium	various	Constraints, Opt.
Our approach	various	high	various	FL

An overview of the comparison with respect to these criteria is given by Tables 4.10 and 4.11. From the comparison, we can conclude that related approaches are limited with respect to multiple issues:

Multiple fuzziness types and sources

During our reviews, we noticed that the majority of approaches motivated fuzzy matching by the case that exact matching approaches do not find any service because no service description matches the requirements exactly but only partially (e.g., [KKRKS07]). By introducing gradual matching results, they want to reduce the number of false negatives. These motivations indicate that these approaches have another understanding of fuzzy matching than we do: They only consider gradedness but do not target uncertainty. Thus, these approaches are not interested in reflecting further types of fuzziness in their matching results. However, for us, a gradual result is not enough. The reason is that, as explained earlier, there is a difference between "a service matches 50%" and "we are 50% sure that a service matches". Existing approaches either only consider the first case or they mix both semantics within one value, leaving the user misinformed. For example, [ZSK15] treats missing specifications as a mismatch, mixing mismatches with uncertainty.

It is striking that we did not find any approach that explicitly takes into account uncertainty induced during the matching procedure. Correspondingly, no approach quantified or even reflected uncertainty in the matching result. In general, most of the approaches hardly mentioned the result format and its properties at all.

Furthermore, unlike our approach, no approach explicitly distinguished between multiple types of fuzziness. Most of them handle either fuzzy requirements or fuzzy service properties (e.g., [BL11, RS03, Wan09, TAS11]), or they address fuzzy data that the service takes as inputs [CYLT05, HCL05]. In some of the selected publications, we detected several types, e.g., vagueness and gradedness or vagueness and uncertainty; however, the approaches did not consider the difference or ignored one of the types (most often uncertainty). The same holds for fuzziness sources. In many approaches, provider-induced as well as requester-induced fuzziness emerges; however, the approaches do not treat them in different ways (if they treat them at all).

Only Bacciu et al. [BBM09, BBM10] mention that fuzziness on the requested side is more about preferences, while fuzziness on the provided side is more about approximate estimates and that both can be specified using fuzzy sets. This is similar to the principles our fuzzy-logic based approach follows. Furthermore, they emphasize that their approach works with fuzzy numbers throughout the whole process. Unlike other approaches that start with fuzzy requirements to be transformed into crisp values for determining the matching result, Bacciu et al. only transform their result into a crisp result at the very end. This idea comes closest to our idea of fuzzy matching; however, we keep the fuzzy values even longer as we reflect the fuzziness also within the matching result. The matching results determined in Bacciu et al.'s approach neither reflect the fuzziness (uncertainty) that is left, nor do they leverage the distinction of different fuzziness types.

Furthermore, in their approach, Bacciu et al. [BBM09, BBM10] consider a *confidence level*. This is a value that can be specified in order to indicate a probability that the provided specification accurately reflects the reality. This refers to an inherent uncertainty within the provider's specification. Moreover, they use a proximity-based index in order to assess the uncertainty of the similarity between two fuzzy sets. This comes close to our approach of

quantifying the difference between n and p. However, as they still apply approximations within their matching algorithm that are not reflected within the matching result, the results are still adulterated.

In [KKRKS07], the user is able to specify *fuzzy missing strategies* that encode what to do in general when a service specification does not provide some kind of information that has been specified in the requirements specification. Possible strategies are to ignore the missing information or to assume to a certain degree that a requirement will be met by a service not specifying it. These additional configuration parameters increase the user's awareness of possible fuzziness within the matching result and gives her some control. However, there is still no information whether fuzziness actually emerged during one specific matching process and to which extent.

Algorithm-induced fuzziness potentially occurs in approaches that apply approximate subsumption reasoning (e.g., [SCS10, KKF08, KK10]). They aim at improving the matching's recall, i.e., reducing the number of false negatives, however, precision inevitably decreases, i.e., the number of false positives increases. Following our argumentation, the user's decision-making is complicated if the matching results do not reflect whether and to how much extent approximations have been applied within a concrete matching process. One step into addressing this issue is made by LARKS [SWKL02]. There, in case of a *relaxed match*, the user is provided with a numerical distance value that shows how "far away" the match is from an exact match. By configuring a threshold, the user is able to define how large the distance needs to be in order to be considered as a match. Similarly, like in [TCM08], the user could define on her own, which requirements may be relaxed and which not.

Note that the listed Considered Fuzziness Sources/Types in Table 4.10 are not necessarily equal to the actually arisen fuzziness sources/types, i.e., fuzziness that should have been considered in order to avoid adulterated matching results. For example, many approaches potentially lead to algorithm-induced fuzziness, but they ignore it (e.g., [BBM10]).

One exception is the approach described by Fenza et al. based on ontologies [FLS08]. In this approach, the user provides a *precision degree* as an input. This degree defines the allowed distance a requested ontological type may have from a provided ontological type in order to lead to a match. The ontological types may appear within the specified inputs, outputs, preconditions, post-conditions, or further describing text in natural language. The distance can be viewed as algorithm-induced fuzziness because it is an approximation during the calculation of the matching result.

Moreover, none of the collected approaches quantified transformation-induced fuzziness although it emerges in several approaches. For example, Almulla et al. [AAY11] transform ("fuzzify") crisp QoS values into fuzzy sets using a simple algorithm. This algorithm takes as an input the borders of the interval that limit the "fuzzy part"; however, the actual function, e.g., the gradient is uncertain. This issue is not taken into account in Almulla et al.'s approach. Similar issues hold for other approaches, e.g., [Wan09, WCL+06, CYLT05, LFT12, PRVM13]. The only approach that addresses similar problems applies methods to reach a consensus about different fuzzy sets for the same concept using group decision resolution strategies [HLL+05, HLCY06]. Here, the fuzzy sets specified by different users build the basis for a majority vote. Torres et al. [TAS11] apply a clustering approach in order to solve the requesters' problem to select fuzzy terms for quality requirements in dynamic markets, where quality judgments vary all the time. All these methods may be applicable to reduce transformation-induced fuzziness, but they do not eliminate it

completely. Furthermore, they neither make the users aware of the fuzziness' existence nor of its extent. Approaches where the values are directly specified using membership functions, e.g., [BBM10, DCCH07], do not suffer from transformation-induced fuzziness.

Constantinescu et al. [CBF05] transform the request into a relaxed request that is used for approximated matching. This means that they intentionally induce transformation-induced fuzziness. However, in this case, fuzziness does not lead to adulterated matching results: The relaxations assure that no false negatives or false positives are produced as they are followed by a refined, up to an exact matching if necessary. Because of this *pruning* approach, mismatching services can be sorted out very early without suffering from inaccuracy. However, if there are many well matching services, the approach is inefficient as it requires many iterations in this case. Furthermore, the approach is tailored to subsumption reasoning as a matching technique, i.e., only IOPE descriptions can be matched this way.

Also apart from transformations into fuzzy sets, some of the listed approaches include transformations into a specific specification language, like [PCP09]. However, they do not consider concepts like transformation-induced fuzziness, either.

We see the differentiation of different types of fuzziness as an important benefit of our approach that distinguishes it from related approaches. Informing about fuzziness on the basis of different sources gives the requesters and providers more insights about the matching results. Consequently, they can overcome different fuzziness targeted by different means (e.g., adapting the requirements specification or the provider specifications).

Informativeness of the matching result

Most of the matching approaches mentioned return either a boolean (e.g., [CCL+05, SCS10]) or a scalar value (score) as a matching result (e.g., [AAY11, BL08, LSHY09]). Our approach delivers more information along with the matching result, in order to give feedback about the extent of fuzziness that had been introduced into the matching process.

The result format used most often are scores, i.e., single numerical values that represent how well a service specification matches the given requirements specification. Exceptions are [FLS08] as explained above, [RS03], and [ŞT06]. In [RS03], the matching approach results in one of four different *reuse decisions* that denote to which extent the matched component has to be adapted. This can be explained by the fact that this approach comes from the domain of component-based software engineering and not from service-oriented computing. They assume that the component's implementation can be adapted after discovery. For services, this is not true, as the user typically does not have access to the service. Adaptations are only possible as non-invasive extensions (e.g., [MXB10]). Sora and Todinca's approach [ŞT09, ŞT06] provides as output either "select" or "reject" combined with either "weak" or "strong". The latter addition helps to assess the certainty of the result, but only to a limited extent. The reason for the uncertainty, i.e., its source, as well as its extent is not known to the user.

Almost all approaches based on fuzzy logic transform into a crisp matching result after having matched on the basis of fuzzy sets. This transformation leads to a loss of information. In particular, the crisp matching result does not reflect fuzziness anymore. Bacciu et al. [BBM10] criticize this issue, too, and try to overcome it by transforming at the very end of the matching procedure only.

The approach suggested by Fenza et al. [FLS08] mentioned above returns a distance score, in addition to a membership degree. This distance score can be viewed as a calculation of the extent of algorithm-induced fuzziness.

Similar to this, in Thakker et al.'s approach [TOAD07], a *factor of tolerance* is given. This factor defines a tolerance region in relation to the value to be matched in order to receive more flexible matching results. In other words, an approximation within the algorithm leads to algorithm-induced fuzziness. This again leads to uncertainty as the matching result does not reveal to the user, to which extent a specific matching process for a specific service-request-pair has actually been approximated.

Within our selected set of publications, we did not find any approach that returns interval matching results. Up to now, intervals are only considered as inputs, i.e., within the requirements or service specification (e.g., [AAY11, CBF05, ZSK15]).

All in all, publications did not discuss the format of the matching results much. We cannot exclude that the actual realizations of the described approaches additionally provide their users with more comprehensive matching results as described in the papers.

Complexity of considered service specifications

Many of the surveyed approaches deal with single fuzzy numbers used to model QoS properties. For example, Bacciu et al. [BBM10] model a service's response time, availability, and trustworthiness by trapezoidal fuzzy sets and corresponding requirements by triangular fuzzy sets. In another example, Torres et al. [TAS11], match fuzzy requirements with precise, numerical service data.

Our approach goes beyond single numbers or linguistic variables but deals with their composition in the form of more complex conditions aiming to model more complex service properties. In particular, also quality properties need to be described with expressive specification language [PCP09]. For example, a service's response time depends on the context of a service, e.g., the execution environment and the usage behavior [MM07, FGT12, BKR09]. Consequently, a simplification to just one number is too abstract and not realistic.

In addition, there are more existing approaches using fuzzy sets that consider other kinds of simple service specifications. These specifications are often based on keywords only (e.g., [CYLT05, HCL05]), often referring to simple ontologies that consist only of kind-of relations between these keywords (e.g., [BL08, BL11]). Many approaches focus on keywords modeled as linguistic terms (e.g., [Wan09]).

Most of the fuzzy matching approaches target matching of non-functional properties, especially QoS. The reason could be that people agree that non-functional properties are usually vague and imprecise in nature [TAS11, YT97].

All these approaches have in common that they cannot deal with complex and heterogeneous specifications as we need for our scenarios.

Further discussion

Bacciu et al. [BBM09, BBM10] are the only authors that explicitly state that their approach can be applied as an extension of non-fuzzy approaches. All other approaches assume a completely new development.

As we can learn from Table 4.10 and Table 4.11 and the discussion derived from this data, existing fuzzy matching approaches are limited with respect to several of our requirements. In particular, only few fuzziness types and fuzziness sources are considered and neither of

them is reflected within the matching result. All in all, we can say that, although fuzziness is already widespread in service matching, uncertainty is not often dealt with explicitly. Existing approaches focus rather on vagueness and gradedness.

The keyword-based search we used to conduct our survey leads to a few threats to validity as we may have sorted out too many papers: For example, in papers about service composition, matching is also an important part. These matching approaches are not included in our survey. Furthermore, our set of keywords may be too limited and also our abstract-based search involves the risk of excluding relevant papers [BKB+07]. However, our survey is still based on a lot of data such that we, nevertheless, expect our findings altogether to be transferable.

4.10.3 Fuzziness in Related Software Engineering Disciplines

Our survey revealed that only few service matching approaches explicitly dealt with fuzziness in terms of uncertainty although we consider this as an important aspect of our research. Thus, we extended our survey by taking into account literature from related software engineering disciplines.

For example, in his position paper [Gar10], David Garlan lists several sources of uncertainty in software engineering. These sources include (a) the "human in the loop" because humans are not fully controllable or predictable, (b) mobility (variety of platforms), especially if these platforms change dynamically and frequently, and (c) rapid evolution due to an increasing demand on a short time to market and required adaptions to new requirements. These aspects fit well to our service matching problem as we have (a) the human involved in both specifying services and requirements as well as in making the final decisions before acquiring a service on the basis of the matching results, (b) different platforms services can be applied on ranging from the cloud centers to mobile devices, and (c) due to the high competition developing in global markets, service offers and requirements are expected to change often. Though Garlan does not provide any concrete solutions applicable to our domain, he repeatedly emphasizes that software engineers should not "maintain the illusion of certainty" and that "uncertainty needs to be considered as a first-class concern to be dealt with systematically", which is in line with our idea of fuzziness estimation and presentation.

Based on [WHR+03], Perez-Palacin and Mirandola [PPM14] constructed a classification of uncertainty in software models. This is related to the classes of specification-based fuzziness in Section 4.4.2 as our specifications are also software models. Perez-Palacin and Mirandola took into account three dimensions of uncertainties: location (does uncertainty emerge due to the model, due to the metamodel, or due to the meta meta model?), level (how much knowledge is missing?), and nature (is uncertainty due to imperfect knowledge or due to inherent variability?). The relation of fuzziness in service matching to the location dimension has already been discussed in Section 4.4.3, while the relation to the level dimension has been discussed in Section 4.4.1.6. The nature dimension distinguishes between aleatory uncertainty and epistemic uncertainty as discussed in Section 4.4.1.6, too.

Esfahani et al. distinguish between external and internal uncertainty within the field of self-adaptive software systems [EKM11]. Considering the matcher as the software system, we could refer to algorithm-induced fuzziness as a source for internal uncertainty, while all other fuzziness sources refer to the environment, i.e., external uncertainty.

Similar limitations hold for further approaches related to requirements specification for self-adaptive systems under consideration of uncertainty, e.g., RELAX [WSB+09] and FLAGS [BPS10]. Both approaches are based on temporal fuzzy logic. For example, RELAX allows to address uncertainty in requirements specifications for adaptive systems. It deals with two key sources of uncertainty: environmental changes of a system's execution environment and behavioral changes at runtime. RELAX is a potential language for specifying behavioral requirements in the presence of requester-induced fuzziness. However, it does not address how those specifications can be handled during automated matching and how matching results could look like in the presence of relaxed requirements. Furthermore, it typically requires deeper knowledge of the system's internal behavior, which we typically do not have when working with service-based systems.

In reliability analysis (e.g., [YST01, MMAG11, GPK04, CGMS07]), models like fault trees, Markov chains, and stochastic Petri nets are utilized to predict the reliability of a software system. These models contain parameters obtained from uncertain field data [YST01]. Yin et al. [YST01] discuss three ways to present this uncertainty: bounds, confidence intervals, and probability distributions. The confidence intervals are comparable to the n-p-intervals determined in Section 4.6.1. However, the presented analytical approaches are limited to special kinds of models and parameter distributions; they are hard to generalize (see *R5*). In particular, they are not applicable to our specification languages for reputation or privacy.

In general, reliability models require internal information about a system (e.g., failure rate and repair rate) that we typically do not have within the domain of deployed services whose internals are not exposed to potential customers. Furthermore, simulation-based approaches are discussed as a way to compute uncertainty in reliability analysis [YST01, MMAG11]. Simulation-based approaches are infeasible for application in service matching as they require intensive computation which is too cost-intensive in a running service market (see *R7*).

4.11 Conclusions

In this chapter, we presented our concepts for fuzzy matching. On the basis of well-defined fuzziness types and sources, fuzzy matching aims to make fuzziness induced into the matching result transparent to the user. This is achieved by applying methods from fuzzy logic and possibility theory. As a consequence, the user is provided with more knowledge about the risk she is going to take when acting on the basis of a fuzzy matching result. Our approach allows users to participate in and to control the decision making process based on informative matching results.

The results of our case studies indicate that handling fuzziness during the matching process explicitly and reflecting it in the matching result supports decision-making in the analyzed situations to a satisfying extent. As our survey confirms, our approach goes beyond existing approaches because existing approaches do not properly consider and reflect fuzziness within the matching results. Future research challenges are listed in Section 6.2.

5

Integrated and Comprehensive Fuzzy Matching Processes

In this chapter, we combine the matching processes presented in Chapter 3 with concepts for fuzzy matching presented in Chapter 4. This leads to a concept of *Fuzzy Matching Processes*. Furthermore, we discuss matching results created by such fuzzy matching processes with respect to understandability for the user. Last, we propose a method about how to integrate fuzzy matching processes into a service market architecture.

5.1 Scientific Contributions

The scientific contributions of this chapter can be summarized as follows:

- We show how to integrate the two ideas of comprehensive service matching and fuzzy matching in order to get a combined approach that leverages from both.
- We present an approach to back-translate matching results into the original specification language of the service requester in order to improve her understanding. Thereby, we go beyond the state of the art in service matching approaches that only discusses the presentation of matching results in a superficial way.
- We demonstrate how to integrate a matcher into a service market on the architectural level using a systematic process based on architectural tactics. In particular, we discuss the trade-offs complicating the decision-making process.

5.2 Combining Comprehensive and Fuzzy Matching

In order to leverage the benefits of both fuzzy matching and comprehensive matching processes, we bring these two concepts together in this section. The goal is to achieve matching of specifications that are comprehensive on the one hand, but also imperfect on the other hand, at the same time. This combination is not an easy task as demonstrated by our requirements presented in the following.

5.2.1 Requirements

In the following, we derive and explain the requirements for a solution that combines fuzzy matching and comprehensive matching processes.

(***R1***) *Integrate Fuzzy Matching Steps:* Users must be provided with means to integrate fuzzy matching steps into matching processes. This can also have an impact on control- and data flow because now not only matching results are propagated through the matching process but also information about fuzziness.

(***R2***) *Aggregate Fuzzy Matching Results:* Aggregating fuzzy matching results can be more difficult than aggregating traditional matching results because they contain more information, e.g., in terms of interval results and in terms of fuzziness scores.

(***R3***) *Enable Fuzziness Coping Strategies:* Our general fuzzy matching procedure allows to recommend fuzziness coping strategies (see Section 4.5). Thus, this functionality should also become part of the MatchBox framework.

(***R4***) *Optimize Matching Processes on the Basis of Fuzzy Matching:* Matching processes are designed to cope with comprehensive service specifications. Fuzzy service matching usually speeds up matching steps. This characteristic should be leveraged in order to optimize matching processes.

5.2.2 Framework Extensions for Fuzzy Matching

Some parts of the MatchBox framework need to be extended in order to cope with fuzzy matching steps. For example, in order to allow MatchBox to recommend fuzziness coping strategies (*R3*), the matching result model needs to be enhanced by information about the induced fuzziness. The extension is depicted in Figure 5.1. As can be seen there, the MatchingResult class now refers to a class FuzzinessAnnotation. An arbitrary number of fuzziness annotations can be part of one matching result. FuzzinessAnnotation has an attribute score that represents the fuzziness scores. Each FuzzinessAnnotation refers to one FuzzinessOccurrence. A FuzzinessOccurrence is characterized by its type and its source. The types and sources comply with the classification presented in Section 4.4; however, we refined the lists of sources and types by extra information needed to display valuable information to the user and to automatically recommend how to cope with fuzziness as discussed in Section 4.5. For this reason, we distinguish between transformation-induced fuzziness occurring within the transformation of the requirements specification (REQ_TRANS_INDUCED) and transformation-induced fuzziness occurring within the transformation of the provided service's specification (PROV_TRANS_INDUCED). Similarly, we distinguish between aleatory uncertainty

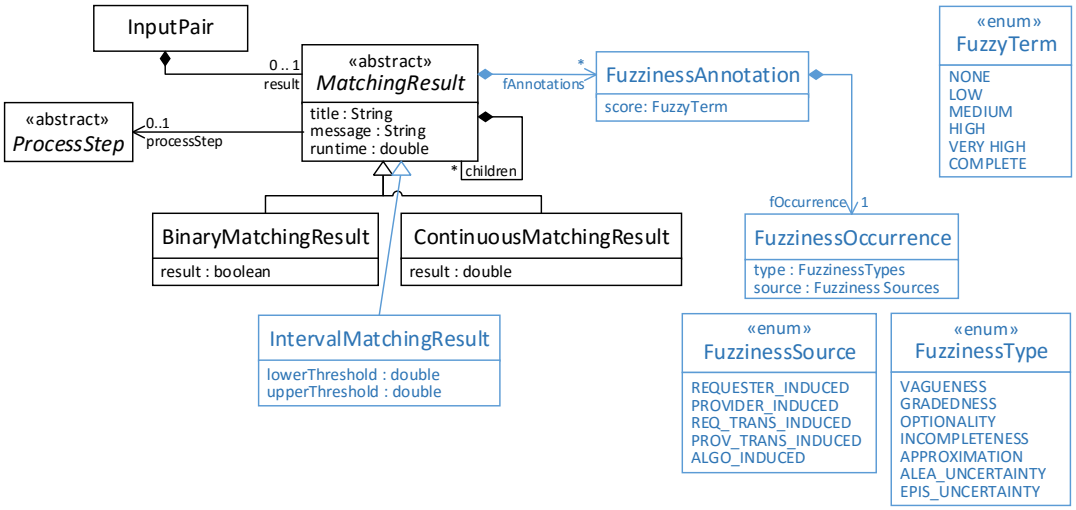


Figure 5.1: Matching Results Metamodel Extended with Fuzzy Matching Results

(ALEA_UNCERTAINTY) and epistemic uncertainty (EPIS_UNCERTAINTY) because only aleatory uncertainty can be reduced by one of the involved roles.

Furthermore, Figure 5.1 shows that another subtype of MatchingResult has been added: the IntervalMatchingResult. IntervalMatchingResult could also be a subtype of ContinuousMatchingResult because it delivers a special kind of continuous result, however, there is no added value as the result attribute in this case has to be represented by an aggregate of the lowerThreshold and the upperThreshold. Such an aggregate could be the mean, or the weighted mean using the risk aversion parameter α as introduced in Section 4.6.1.5, however, it will always lose information.

The integration of the new matching result type is a foundation for the aggregation of fuzzy matching results explained in Section 5.2.3. Furthermore, the integration of the new matching result type also requires the adaption of the Matcher Definition Language (see Section 3.6). As this adaption is trivial, we do not describe it further.

One special case we need to take care of is when integrated fuzzy matching steps deliver their matching result to further matching steps. For example, if a protocol matching step depends on the matching result of a signature matching step that applies a fuzzy matching approach, the protocol matching step will build on potentially uncertain matching results. We suggest two extensions to deal with this issue: (a) The fuzziness score returned by the dependent step should be propagated to the depending step such that the depending step can take into account the fuzziness, too. Note that, as a consequence, the depending step is inflicted with fuzziness, too. This means, the fuzziness score returned by the depending step may have to be aggregated with the fuzziness score of the depending step.

Furthermore (b), the guard element should be extended such that also runtime decisions based on the extent of induced fuzziness can be made. For example, this allows us to specify that the protocol matching will only be executed with signature matching results that have maximum fuzziness “low”. This extension is presented in Figure 5.2. As can be seen there, we introduce two subtypes of the (now abstract) type Guard: MatchingResultGuard

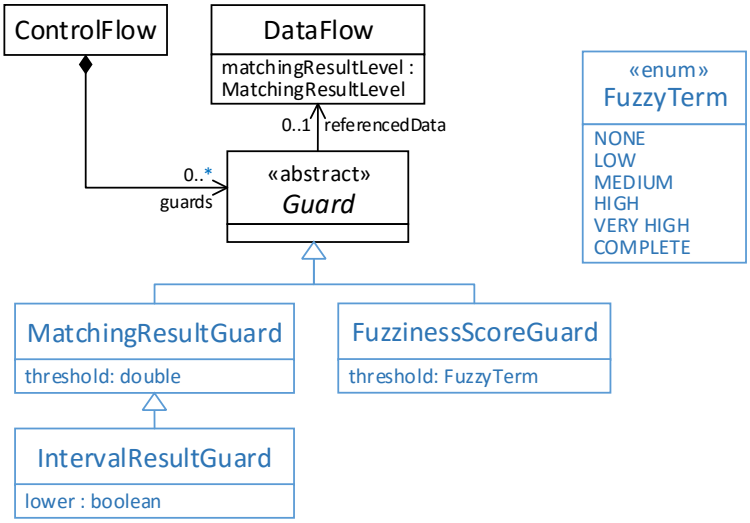


Figure 5.2: Matching Process Metamodel Extended with Fuzzy Guards

(representing the old semantics of Guard) and FuzzinessScoreGuard (representing the new guards refering to fuzziness scores).

Figure 5.2 also shows how to extend the guards such that interval matching results can be considered. For this purpose, we added a type IntervalResultGuard being a subtype of MatchingResultGuard. In addition to the threshold, an IntervalResultGuard also contains an attribute lower that defines whether the guard refers to the lower threshold or, otherwise, to the upper threshold.

All described extensions to the matching process metamodel contribute to the fulfillment of *R1*.

5.2.3 Aggregating Fuzzy Matching Results

Fuzzy matching leads to new aggregation problems to be solved (*R2*): On the one hand, new result formats, e.g., the interval matching results, need to be aggregated. On the other hand, fuzziness scores need to be aggregated.

As mentioned in Section 3.6.2.1, MatchBox provides two ways to aggregate matching results of different formats: (a) A new aggregation strategy needs to be defined and this strategy has to be able to cope with these different formats, e.g., by internally transforming them into one intermediate format. Alternatively (b), these transformations can be made explicit by introducing additional aggregation strategies similar to the "operation level to interface level" strategy (see Section 3.6.2.1).

Since we already proposed a way to transform from binary matching results to continuous matching results and back again, in the following, we only discuss how to transform between continuous matching results and interval matching results. Such transformations are quite trivial and work in both directions: An interval matching result can be mapped to a continuous matching result as explained in Section 4.6.1.5. However, this way, information is lost. Thus, it may be a better way to map a continuous result to a (trivial) interval result by setting $lowerThreshold = result$ and $upperThreshold = result$. This way, the

subsequent aggregation step only deals with intervals. However, the downside of this solution is the lost reusability of matching processes defined without these extensions.

The aggregation of fuzziness scores has already been addressed in Section 4. We can use the same strategies when aggregating fuzziness scores of the same types and sources but from different matching steps to one overall fuzziness score. However, the child results should still be maintained as fuzziness coping strategies could use the knowledge about which matching step has been fuzzy.

As the fuzzy matching results (including fuzziness scores) are the only output of a fuzzy matching procedure (see Section 4.5), all sub steps being part of the fuzzy matching procedure can be hidden within the matching step: The fuzzy matching step then encapsulates the sub steps and the fuzzy matching result serves as an interface. Thus, we also fulfill *R1*.

5.2.4 Coping with Imperfect Information by Self-Adaption

Due to their modular structure and the well-defined matcher interfaces, matching processes modeled and executed in MatchBox can easily be extended to include dynamic self-adaption concepts [Mur04]. Using dynamic self-adaption can then be used to optimize matching with respect to quality and performance criteria (*R4*). Consider the following example: A matching process is used within a market and, over time, it is recognized that matching became rather slow, while market participants are usually very satisfied with the accuracy of matching results. Then, the matching process could adapt itself with respect to a better runtime. For example, it could remove matching steps that did not perform well in terms of performance or quality, or it could substitute matchers by (fuzzy) matchers that are expected to run faster in some cases. This reconfiguration can be performed step-by-step, while the matching process is monitored over the whole time not only with respect to runtime but also with respect to accuracy. This way, the accuracy-runtime trade-off matching is always faced with can be optimized.

In particular, such a self-adaptive matching process can be used to cope with incomplete or imprecise specifications. For example, if the monitor detects that certain matching steps deliver highly uncertain matching results all the time in a certain market, these steps could be removed or their matchers could be reconfigured or substituted.

Another idea could be to observe the typical requesters' risk aversions over time. In domains where the requester's risk aversion has been detected to be extremely high or extremely low, the matching process could adapt the number of integrated fuzzy matching steps and favor more respectively less matching steps that deliver fuzzy results.

As a consequence, runtime monitoring for matching processes has to be extended. According to the examples mentioned above, properties like average runtime, average accuracy, average fuzziness, and typical requester properties could be worth monitoring. As all these properties can develop through time in a dynamic market, self-adaption at runtime becomes more and more important.

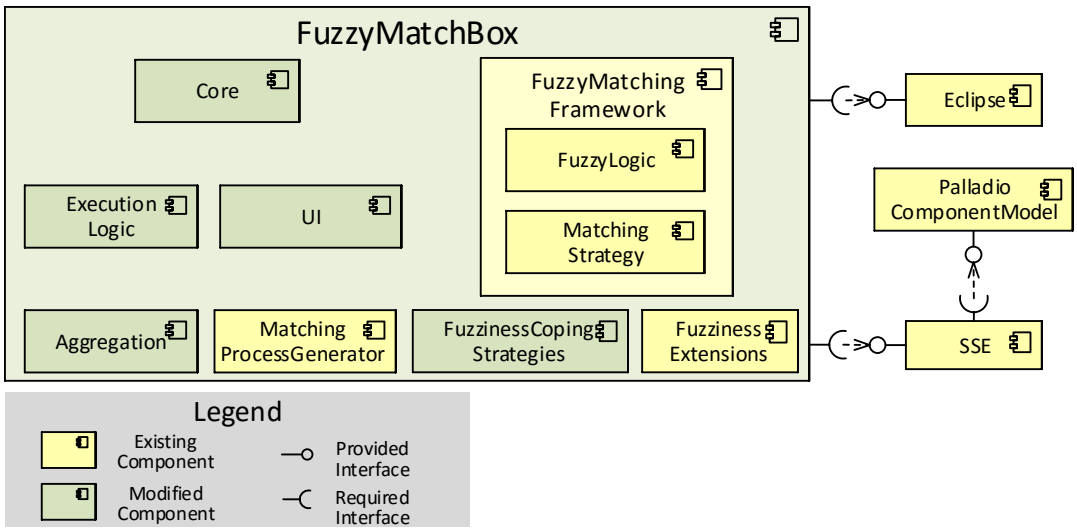


Figure 5.3: Architecture of FuzzyMatchBox

5.2.5 Prototype Implementation

This prototype combines the two prototypes from the previous chapters (see Sections 3.9 and 4.7) as depicted in Figure 5.3. The combination required adaptions in some of the components of the MatchBox framework:

Core: The adaptions within the Core component affect the meta models for matching processes and for matching results.

ExecutionLogic: The ExecutionLogic had to be adapted such that guards can also consider fuzziness. Also the components needed to realize self-adaptive behavior have to be added here.

UI: The extended UI now is also able to display fuzzy matching results.

Aggregation: Aggregation strategies have been extended by a default aggregation strategy to handle the aggregation of interval matching results as well as fuzziness scores.

5.3 Back-Transformation of Matching Results

In order to provide a broad range of service requesters the possibility to discover services using service matching, the effort and the required knowledge to construct appropriate requirements specifications should be kept as low as possible. For this reason, current research [SFB901] investigates more user-friendly possibilities for requirements specifications in contrast to the complex requirements specifications used for service matching within the state of the art. Examples are natural language specifications or specification "by-example" (e.g., [CZC08]). Typically, such specifications are revised and optimized and then transformed into specification languages with formally defined semantics such that they serve as an appropriate input for service matching. This task can be done by the service provider or by an intermediary like a broker or the matching designer.

Furthermore, service requesters need informative matching results not only for successful decision-making when selecting a service. For example, they need feedback about which parts of the requirements specification should be modified in order receive more interesting results in the next search run. MatchBox produces such informative matching results by listing all considered language constructs that contribute to the matching result in form of child results (see Section 3.8.3). The child results and parent results together form a hierarchically structured matching result appropriate for extensive inspection.

Approaches dealing with requirements specifications in natural language are already well-known in the software engineering literature [ACC+02, PR11]. However, there are two problems when applying such approaches to service matching problems: First, the matching results in their current form are on a different level than the user-friendly requirements specifications mentioned above as they are based on different language constructs: The transformations from user-friendly requirements specifications into another, formally defined specification format enable the usage of matchers working on different forms of the specifications than the requesters delivered. This procedure makes matchers more universally applicable. However, as a downside, matchers produce matching results based on the format of the consumed specifications and not in the format of the specifications as defined by the requesters. Thus, there is a gap between the representation of the matching results and the representations the requesters understand.

Figure 5.4 a) shows this situation on an example: The requester searches for "a simple room reservation service that is fast and cheap and does not save personal data". She receives a matching result per provided service which is an aggregate from a signature matching result, a performance matching result, a price matching result, and a privacy matching result. In order to understand such matching results, the requester at least needs to know which part of her initial requirements specification has been transformed into which part of the requirements specification the matcher takes as an input. From the matching result, an expert user (i.e., someone knowing the formal specification language the matcher takes as well as the transformation logic) can understand that the matcher got as an input a signature with the name bookRoom with an input parameter time of type Date and an output parameter room of type Room. Domain knowledge had to be leveraged when transforming the expression "simple room reservation service" into such a signature. An example for such domain knowledge is the knowledge that a room reservation is always based on a defined time slot which can be encoded as a date. The performance-related part of the requirements specification ("fast") has been transformed into a fuzzy set for the performance property

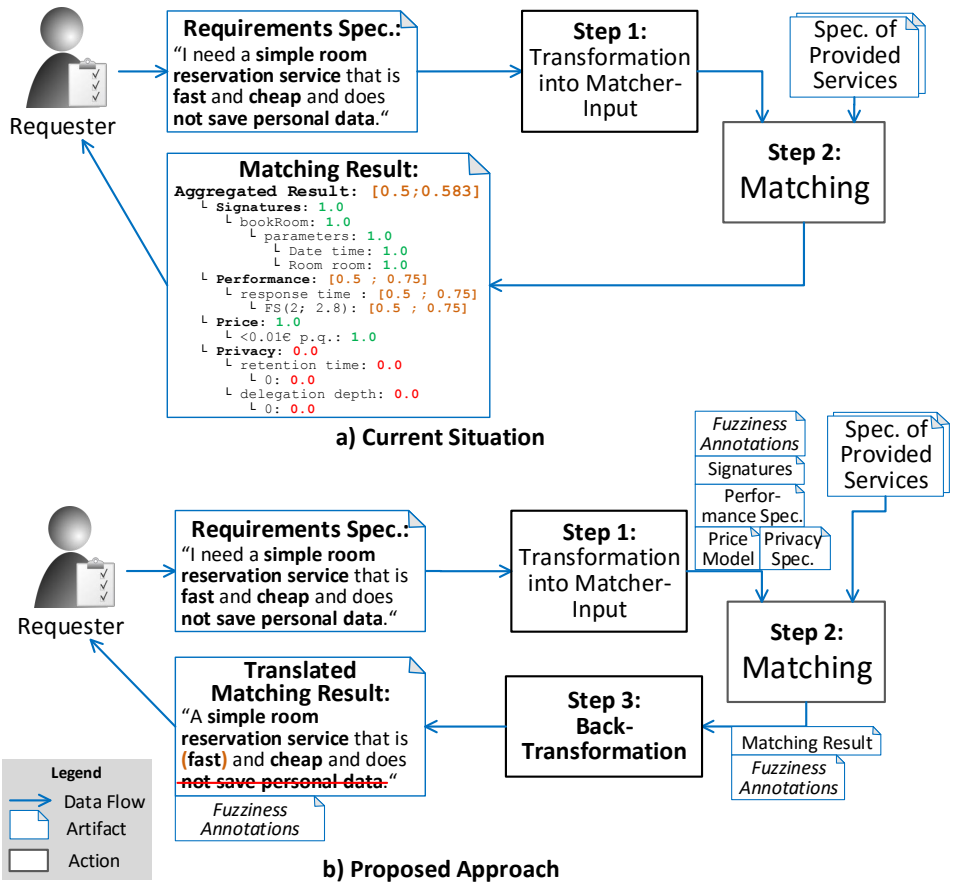


Figure 5.4: Problem of Understanding Matching Results and Proposed Solution

response time. This fuzzy set has a smooth transition from the membership of 1 to the membership of 0 between the values of 2 and 2.8 seconds (FS(2;2.8)). The price-related part ("cheap") has been transformed into a price model where one user's query of the service should cost 0.01 Euros maximum (<0.01E p.q.). Furthermore, the expression "does not save personal data" has been used to derive a specification of privacy requirements where the attribute retention time as well as the attribute delegation depth have been set to 0. Note that the matching result displayed in Figure 5.4 a) is simplified in the sense that the values of the provided service as well as fuzziness annotations are omitted. A requester that initially used natural language to specify her requirements will be unfamiliar and, thus, probably be overstrained with such a matching result.

As a consequence, we need a back-transformation that transforms the matching result returned by the matchers into a representation more understandable for the requester while still preserving the semantics. In particular, we assume that, the more similar the matching result appears to the initial requirements specification, the more understandable it is for the requester who created the initial requirements. Figure 5.4 b) shows this idea using the same example as described above. This time, we added a step Back-Transformation after the matching step. This step takes the original Matching Result (as depicted in Figure 5.4 a) as an input and returns a representation of the matching result that follows the natural

language requirements specification and is therefore more useful for requesters without expert knowledge. For example, the Translated Matching Result, as shown in Figure 5.4 b), could be represented by a copy of the initial requirements specification but highlight parts that are not fully satisfied according to the matching result.

Moreover, a transformation from "informal" specifications (like natural language specifications) into a well-defined, unambiguous specification (like the inputs a matcher expects) can always lead to uncertain results. The reason is that semantics are expressed in another format and it is unclear to which extent these semantics still match the requester's idea of her requirements specified in the first place. This problem is similar to transformation-induced fuzziness emerging from a requirements specification as discussed in Section 4.4.2.

Thus, along with the translated matching result, our approach proposed in Figure 5.4 b) provides a fuzziness annotation that depicts an estimation of not only how fuzzy the matching result is in general but also how fuzzy it is with respect to the initial requirements specification. This fuzziness (in terms of uncertainty) is estimated during the forward transformation based on several heuristics and refined later, when it is known which parts of the transformation are relevant in the concrete matching result determined by the matcher. Hence, the fuzziness annotations suggested here build on the fuzziness annotations introduced in Chapter 4 and are extended to cope with the additional challenge introduced by the transformation as discussed above.

In the following, we proceed by describing our approach using the example of a requester delivering input in controlled natural language. Nevertheless, the concepts are designed in a general way such that they can also be applied to other kinds of user-friendly specifications. The concepts described in the following are based on matching processes introduced in Section 3 as well as fuzzy matching as introduced in Section 4.

In the next section, we derive some basic requirements for the solution to be presented afterwards: Section 5.3.2 details on the transformation into the specifications to be matched and the fuzziness estimation performed along. Section 5.3.3 explains how matching is applied in this scenario. Section 5.3.4 focuses on the back-transformation. Furthermore, we describe the prototypical implementation in Section 5.3.5 and related work in Section 5.3.7.

5.3.1 Requirements

In the following, we derive and explain the requirements for a solution that realizes the approach proposed above. This includes both an appropriate transformation of a user-friendly requirements specification into a requirements specification accepted by a matcher and a corresponding back-transformation from the matching results into a representation of the matching results inspired by the initial requirements specification.

(*R1*) *Matching Trade-Off:* Even though matching in this scenario is based on specifications created from natural language, the matching should still be sufficiently accurate on the one hand and sufficiently fast on the other hand. More extensive inputs may lead to more accurate results but also to slower matching. Thus, the concepts for the transformation into the matcher's input have to take this trade-off into account and optimize it with respect to the current situation.

(*R2*) *Understandable Matching Results:* The main requirement within this section is to achieve the creation of understandable matching results. As noted above, we consider

a matching result as understandable if it can be represented with a similar appearance as the requester's initial input.

(*R3*) *Efficiency:* The forward as well as the back-transformation should not lead to a high overhead in terms of computation time as well as the transformation developer's (maintenance) effort.

(*R4*) *Transparent Fuzziness:* As discussed in Chapter 4, it is important to enable the requester to assess the risk she takes when relying on a matching result. In combination with specifications in natural language this is even more important because of the introduced uncertainties as explained above. Thus, we still need to display fuzziness induced during matching (as explained in Chapter 4) but also fuzziness induced because of the transformations from natural language into the matchers' inputs.

5.3.2 Step 1: Transformation into a Matcher's Input

As a foundation for the transformation, we present an exemplary controlled natural language grammar in the following. Furthermore, within the transformation from the natural language requirements specification into the matcher's input, three tasks have to be accomplished: (a) the creation of a comprehensive specification, (b) the creation of traceability links, and (c) the estimation of fuzziness. In the following, we explain these three tasks on the basis of our running example and the exemplary grammar.

5.3.2.1 Controlled Natural Language for Service Requirements Specifications

Our approach is based on *controlled natural language* (e.g., [Fuc10, WAB+10]) which is limited to a fixed syntax and a fixed vocabulary. In the following, we present an exemplary grammar for a controlled natural language within the scope of service requirements specifications:

$$
\begin{aligned}
CNLRequirementsSpec &::= \text{“I need a”} \quad ServiceKind \quad Restriction* \\
ServiceKind &::= (ServiceModifier) \quad ServiceCategory \\
Restriction &::= (\text{“and”}|\text{“,”})\ \text{“that”}\ (\text{“is”}) \quad (RestrModifier) \quad RestrProperty \\
ServiceModifier &::= \text{... see service modifier ontology} \\
ServiceCategory &::= \text{... see service category ontology} \\
RestrModifier &::= \text{... see restriction modifier ontology} \\
RestrProperty &::= \text{... see restriction property ontology} \\
&\quad | \quad \text{“takes a”} \quad ParameterType \quad \text{“as an input”} \\
&\quad | \quad \text{“returns a”} \quad ParameterType \quad \text{“as an output”} \\
ParameterType &::= \text{... see parameter type ontology}
\end{aligned}
$$

The depicted grammar defines that a template of a requirements specification (CNLRequirementsSpec) always consists of a ServiceKind and an arbitrary number of Restrictions. The placeholder ServiceKind consists of an optional ServiceModifier and a mandatory ServiceCategory. The terminals to insert into these placeholders are defined

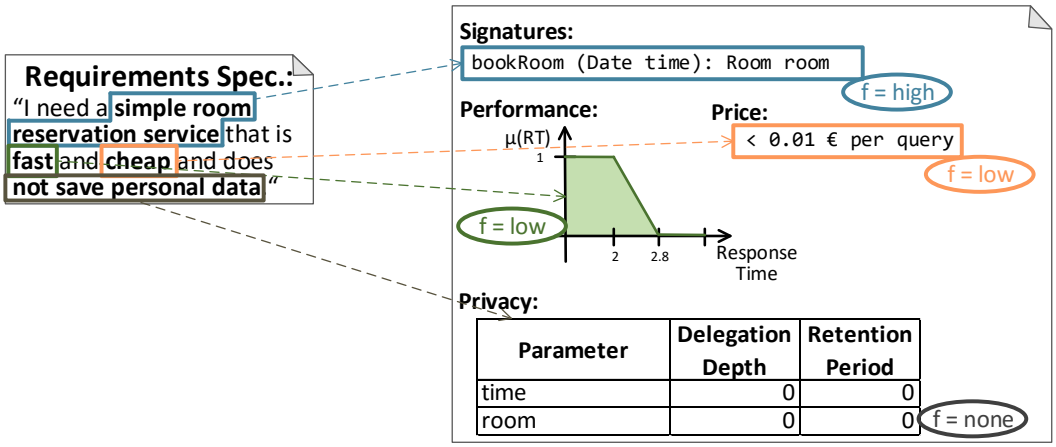


Figure 5.5: Transformation into an Input for Matching

in ontologies. For example, service modifiers could be "simple" (as in our example sentence) or "fancy". Examples for service categories are "Room Reservation Service" (as in our example), "Map Service", "Image Processing Service", and many more. A Restriction consists of an optional RestrModifier and a mandatory RestrProperty. Examples for restriction modifiers are "some" or "very". As exemplary restriction properties, our example specification inserted "fast", "cheap", and "does not save personal data". Alternative restriction properties are also expressions like "that takes/returns XY as an input/output", where "XY" is defined in an ontology, as well.

5.3.2.2 Creation of a Comprehensive Specification

Figure 5.5 shows an example transformation. In accordance with Figure 5.4, four kinds of specifications are created from the natural language sentence: operation signatures, a performance specification, a price specification, and a privacy specification. The transformation is based on heuristics and domain knowledge. For example, in order to transform an expression "simple room reservation service" into a signature bookRoom (Date time): Room room we assume a database with different common variations of room reservation service signatures. Using the modifier "simple" helps to select the best fitting signature from this database. In a more complex example, the requester could also request a "simple room reservation service that takes a date as an input parameter". In that case, we have even more criteria for the selection of the signature.

All in all, the amount of added and refined information, as in this example, is only one of the sources that can be leveraged to derive requirements specifications from natural language requests:

Amount of added information: As noted above, as much information as possible should be extracted from the initial requirements specification. In terms of signatures, input and output parameters are good indicators for the target signature. If the requester also describes more detailed behavior, even pre- and post-conditions could be generated.

Manual inspection: Another source of information used for the selection can be the requester herself if she inspects the transformation's output, i.e., the created specifications. Such

inspections could be introduced in an iterative way, leading to several repetitions of transformations with slightly changed inputs. Such an iterative approach enables the transformation to take into account a requester's judgement like "the signature is exactly what I meant" or "the performance specification does not really fit my interests, the service should be even faster". The feasibility of this source of information depends on the requester's knowledge and on her willingness to spend effort.

Market knowledge: Additionally, also knowledge from the market can be leveraged for the creation of an appropriate specification in different variants:

Earlier feedback: For example, a reputation system could provide information on the feedback earlier requesters gave. Especially requesters that stated similar requests and that were provided with similar transformed requirements specifications are interesting for this purpose. For example, the reputation data indicates that if earlier requesters that requested a "simple room reservation service" were satisfied with the result that has been determined based on the bookRoom signature, the transformation should stay with this approach. If earlier requesters were not satisfied, the transformation should lead to other target specifications in the future. Such an approach can be realized by machine learning mechanisms like reinforcement learning [SB98, KLM96]. However, the reliability is relatively low because of the introduced indirection: a bad feedback regarding a service can also mean that the translated requirements specification was perfect but the market offered only mismatching services.

Available offers: A requirements specification can also leverage knowledge from the offers that are available in the market. For example, if it is already known that all offered services within the category "room reservation" take the parameter time as an input, it is likely that this parameter should also be part of the derived signature. Similarly, if services in the current market all vary in their response time between five and ten seconds, an expression "low" could be interpreted in a different way than in a market where all services respond in under one second.

For the transformation of fuzzy terms like "fast", we follow the principles of "fuzzification" that we already applied within the scope of Chapter 4. Furthermore, the considered structured natural language still contains the risk of ambiguous specifications. For example, a requirements specification could use different modifiers in combination with the same property (e.g., "quite fast" and "very fast"). Our approach resolves such cases by considering the strictest restriction. In the future, we could extend our approach by deriving prioritized alternatives.

The usage of predefined target language patterns, among which has to be selected during the creation of the specifications, also enables us to take care of creating good-quality specifications that take into account the matching trade-off (see *R1*). Reasons include that the available patterns can already be evaluated with respect to how well they can be matched. Furthermore, this restriction can prevent problematic cases like over-specification occurring when depending completely on specifications created by (inexperienced) humans.

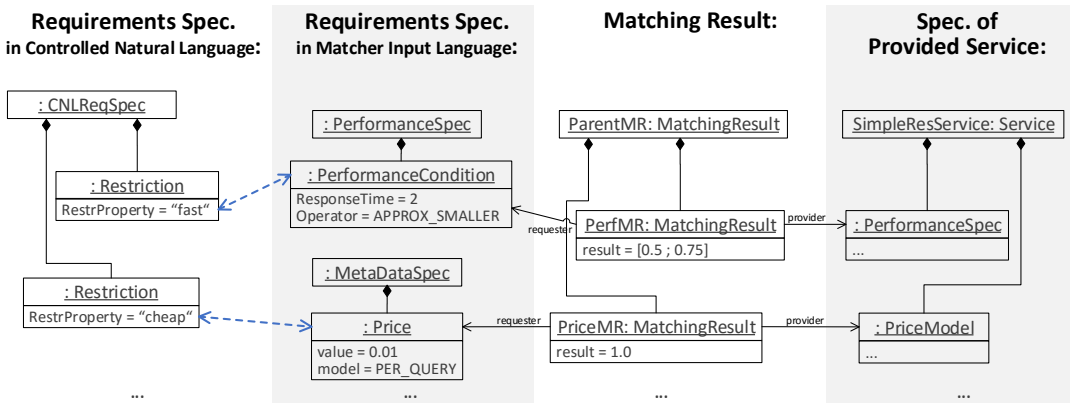


Figure 5.6: Object Diagram View of Transformed and Matched Model Elements

5.3.2.3 Creation of Traceability Links

The dotted lines in Figure 5.5 represent *traceability links* created together with the target models. Based on these links, the back-translation can trace back which model element within the matching result corresponds to which element in the initial controlled natural language requirements specification.

Figure 5.6 shows an extract of the same transformation in abstract syntax. Note that the shown object diagrams abstract from certain details, e.g., concrete subclass names or additional references, for a better overview. Here, we see four levels: the requirements specification in controlled natural language that is transformed into the requirements specification in the matcher's input language. The latter again is an input for the matcher, together with the specification(s) of one or more provided services. Last, the matching result is the matcher's output to be transformed into the controlled natural language used for the initial requirements specification. In Figure 5.6, the blue dotted arrows represent traceability links. As we can see there, for us, a traceability link is a simple one-to-one relation between a model element on the source side (the requirements specification in controlled natural language) and a model element on the target side (the same requirements specified in the matcher's input language).

In the software engineering literature, traceability usually occurs within the area of requirements engineering. There, traceability refers to the ability to describe, follow, and reconstruct a requirement within a software system [GF94]. Thus, usually, traceability links have been used for the purpose of (bidirectional) transformations between a software system's source code and its (requirements) documentation [ACC+02]. In contrast to these approaches, in our case, we have one more level of indirection because we need to trace back the original requirements specification not only from the transformed requirements specification, but from the matching results that have been calculated based on the model. Accordingly, for us, a traceability link actually includes two links: (a) the one between the natural language model element and the model element the matcher takes as an input and (b) the one between the model the matcher takes as an input and the matching result. However, the MatchBox framework already takes care of the latter by directly referring to the matched model elements from the matching result (see Section 3.8.3). A similar approach has also

been applied by Gerking et al. [GDHS15]. There, tracebility links have been utilized within the scope of model-to-model transformations, too, but for model checking purposes.

In Figure 5.6, this is indicated by the requester/provider references that the sub matching results hold to the matched model elements.

5.3.2.4 Estimation of Fuzziness

The potential sources listed in Section 5.3.2.2 to be used for the transformation, e.g., for the selection of appropriate signatures, also influence the fuzziness estimation. For example, the more information the requester delivers, the more certain it is that the transformation result corresponds to her expectation. The signature in Figure 5.5 has been derived on the basis of little information. Thus, its fuzziness has been judged with high. More knowledge about the parameters can reduce this score. For example, additional information like "... that takes a date as an input parameter" can reduce a fuzziness score high to medium, or even lower, depending on how much more information can be extracted. The specifications of some service properties leave less room for uncertainties than others. For example, the expression "does not save personal data" is pretty clear. There is no fuzziness because properties like delegation depth and retention time clearly have to be most strict, i.e., set to zero, in this situation.

Furthermore, if a requester inspected the transformation result and evaluated it to be good, the fuzziness score can also be reduced. Depending on the requester's expertise, this information can get a higher or a lower emphasis.

As already noted, feedback accessible from a reputation system can also be used to select between parts of the transformation target based on earlier experience. Similarly, selections that have been made based on a high reputation for a very similar situation can be judged as more reliable than selections that only lead to bad feedback in the past or selections where reputation is rather unknown. The reason is that, in the latter case, the requirements are totally new and no similar requirements specifications have been received yet.

The information from all these sources can be prioritized based on the concrete situation, e.g., based on the requester's expertise, and then be aggregated to an overall fuzziness score, as known from fuzzy matching (see Chapter 4). We also can use the same concepts for representation of fuzziness scores as known from fuzzy matching. Thereby, our approach fulfills *R4* (transparent fuzziness).

5.3.3 Step 2: Matching Process Generation and Enhanced Matching

The matching process framework introduced in Chapter 3 serves as a basis for matching all those target specifications describing different service properties. Especially the automated generation of matching processes (see Section 3.7.2) can be leveraged for the purpose of automatically deriving an appropriate matching process subsequent to the transformation on the basis of the transformed matching result. An appropriate matching process is a matching process that includes matching steps in order to match all parts of the derived requirements specification.

Since the natural language specifications form the basis for matching in this scenario, we can expect specifications to be imperfect (see Section 4.4.1). This raises the need to add fuzzy matching steps to the matching process. The fuzziness estimated during the transformation

from natural language then contributes to the fuzziness score for transformation-induced fuzziness and propagates to the next step, the back-transformation.

Furthermore, depending on the integrated matchers, the matchers' configuration parameters could be optimized with respect to potentially incomplete input specifications due to the transformation from natural language. For example, if the derived signatures have been annotated with a high fuzziness score, the signature matcher could be configured in a way that it returns not only one parameter mapping but several alternative mappings to proceed with in the dependent matching steps like condition matching (see Section 2.4.1). As a consequence, several alternative matching results will be returned and can serve as a feedback for a better transformation in the next iteration.

5.3.4 Step 3: Back-Transformation and Representation

Because of the traceability links created during the forward transformation, the back-transformation is straightforward: While iterating through the tree of matching results, the collection of traceability links can be accessed for each model element.

An alternative approach here would be to derive the traceability information automatically based on the knowledge that has already been used within the forward transformation. This way, traceability links do not need to be stored, which reduces the complexity of the forward transformation. However, storing the links leads to two important advantages concerning *R3*: (a) efficiency in terms of the developer's effort regarding maintainability of the specification languages and the transformation and (b) efficiency in terms of a reduced computation cost.

- Better Maintainability: Storing traceability links during the forward transformation allows to keep the back-transformation simple and "stupid" because the back-transformation needs no knowledge about the forward transformation. This is a fact of special importance in our scenario because the knowledge leveraged during the forward transformation is heuristic and vague. Thus, the transformation logic is likely to change on the basis of experiences collected over the time. In addition, the source language is also subject to frequent changes as the terminology from the underlying ontologies may be adapted and extended. For example, more synonyms for "fast" could be introduced, or we could introduce a transformation rule that transforms "fast" not into an attribute of the performance model but into a performance context of the reputation model (cf. Section 2.3.4). Using an automated construction of the traceability information during the back-transformation would mean to adapt both transformations if something changes. In contrast, storing traceability links beforehand allows us to keep the logic that is subject to change in only one place which makes future adaptions and extensions easier reduces the effort. Furthermore, such an independent back-transformation allows us to maintain multiple alternative forward transformations and using the same back-transformation for all of them. Multiple forward transformations are useful if different heuristics used for the transformation make sense in different situations, e.g., in different domains. Furthermore, the source language, i.e., the controlled natural language, could look different for different domains.
- Reduced Computation Costs: Usually, the back-transformation can be expected to be executed much more often than the forward transformation. The reason is that one

requirements specification is usually matched to a large number of provided service specifications. For each provided service specification, one matching result is created and each matching result needs to be transformed back. As a consequence, it is beneficial to keep the back-transformation simple and rather increase the complexity of the forward transformation by the construction of traceability links.

An approach to present the results created on the basis of fuzzy set matching in an appropriate way has already been proposed in Section 4.6.1. In addition, standard fuzzification and defuzzification approaches can be applied in order to adjust the representation of the matching result to the requester's terminology.

The back-transformation also propagates the fuzziness scores to the final matching result (*R4*). On the basis of all this information, the requester's original text can be annotated with hints that show which parts of the requirements specification are not satisfied or which parts are uncertain (*R2*). An example for such a representation is shown in Figure 5.4 b).

Note that "back-transformation" is actually not completely correct term as our transformation is not just the reverse direction of the forward transformation: While the forward transformation transforms from Language A (e.g., controlled natural language) to Language B (a matcher's input language), the "back-transformation" transforms from Language C (the language used to present matching results) to Language A. For the same reason, bidirectional model transformation languages like triple graph grammars [GW06, LAS+14] are no solution to our problem. However, as in our case Language C refers to language constructs from Language B, the term "back-transformation" is appropriate.

5.3.5 Prototype Implementation

As shown in Figure 5.7, three plug-ins have been added to the FuzzyMatchBox framework in order to realize the concepts presented in this section:Fuzzy

ControlledNaturalLanguage: A plug-in that represents a controlled natural language realized by an ecore metamodel following the grammar introduced in Section 5.3.2.

ControlledNL2RequirementsSpecsTransformation: Realizes our example transformation including the creation of traceability links. There are different ways to realize traceability links. For example, model transformation languages like QVT [OMG11] or triple graph grammars [GW06, LAS+14] already come with a solution. However, as we have two levels of transformations, we constructed a trivial ecore metamodel as a more flexible solution for this purpose and to keep the integration and maintenance effort low.

MatchingResult2ControlledNLBackTransformation: Realizes a back-transformation based on the traceability links. This functionality has been encapsulated into its own plug-in in order to improve low coupling to foster substitutability for alternative back-transformations.

5.3.6 Limitations

The natural language specifications our solution focuses on are simple controlled natural language texts restricted to a set of predefined constructs. These constructs include a predefined collection of modifiers, a predefined collection of service categories,

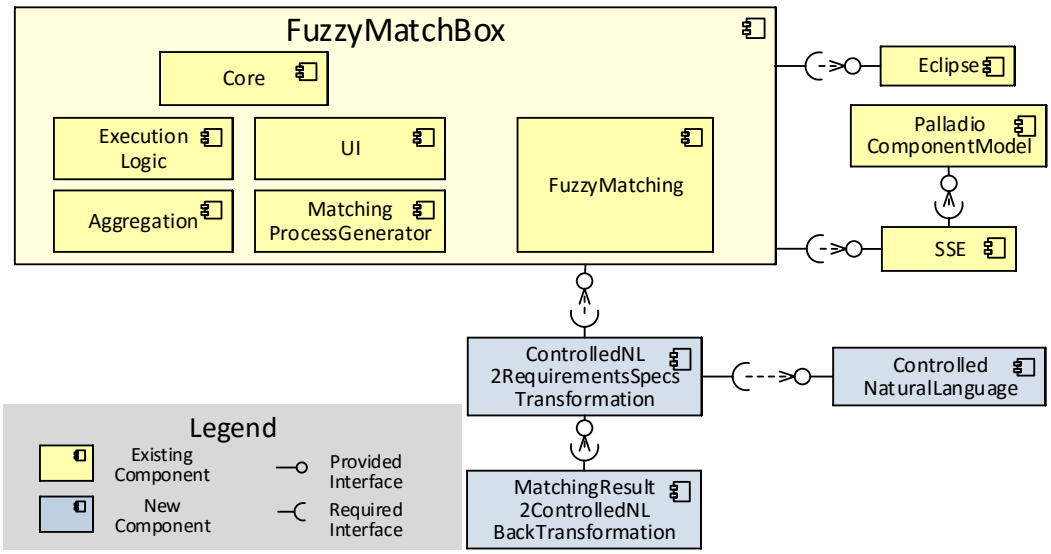


Figure 5.7: Extended Architecture to include Back-Transformations

and a predefined collection of service properties. In order to work with free natural language specifications, various techniques from natural language processing like disambiguation [Les86, IV98] would need to be taken into account. Furthermore, the selection process during the forward transformation would become more complicated. However, the requesters gain more flexibility.

Furthermore, our prototype is also restricted on the target side of the transformation: Some exemplary service specification languages have been selected as target languages for the transformation. An extension of our approach could consider more and other service specification languages for the same or for further specification aspects.

5.3.7 Related Work

There are numerous software engineering approaches working on natural language requirements and their traceability. Most of these approaches work on controlled natural language. Ilieva and Ormandjieva [IO05] as well as Bryant [Bry00] and Moreno [Mor97] transform natural language requirements specifications into object-oriented models. Mencl [Men04] derives behavioral specifications, e.g., automata, from textual use cases, while Li [Li99] generates sequence diagrams and Ravenflow [Man06] generates activity diagrams. Weston et al. [WCR09] automatically transform natural language requirements specifications into feature models. Fockel and Holtmann [FH14] derive a SysML model.

However, all these approaches vary from our concepts with respect to three issues:

(a) None of these approaches is concerned with matching results. This is an important issue because, as discussed above, it introduces an additional indirection. This leads to the fact that no bidirectional transformation can be used (as, e.g., in [FH14]) because the back-transformation significantly differs from the forward transformation. The back-transformation from a model-checker's output into the original input language as reported

by Gerking et al. [GDHS15] comes closest to our approach considering this issue. However, this approach is, nevertheless, limited with respect to issues (b) and (c).

(b) Related approaches only transform into structural models (e.g., [IO05, Bry00, Mor97]) or behavioral models (e.g., [Men04, Li99, Man06]). They do not derive a set of heterogeneous and comprehensive specifications covering both structural and behavioral properties as well as non-functional properties.

(c) Related approaches do not consider the induced fuzziness or uncertainty in any way. The reason for this finding could be that, in the other approaches, the risk of changed semantics due to the transformation is not that high as the specification languages are more similar to each other. Accordingly, they are not relying on other sources of information like heuristics and domain-knowledge as much as our approach does. The only publication of the ones listed above that mentions domain knowledge is the work by Weston et al. [WCR09]. They discuss that a domain expert's knowledge helps to identify irrelevant details in the source specification, i.e., to derive a more abstract target model. In our case, the target model is typically less abstract than the source model.

5.4 Integrating a Matcher into a Service Market Architecture

Established markets today are still limited to a relatively simple, keyword-based search. Keyword matching, however, neither captures the semantics of the services to be searched [MPMM98] nor non-functional properties. Thus, in academia, there is a mass of research for more complex service matching approaches as discussed in Sections 2 and 3. However, integrating a service matcher component implementing such a service matching approach into an existing service market is complicated. The reason is that there are different architectural possibilities with different consequences on market success that makes a trial-and-error approach risky and expensive. For example, integrating a service matcher into the requester's client, instead of integrating it at the provider's side, provides the benefit of customizability without losing comparability between matching results. However, this architecture may lead to a bottleneck that can slow down the whole discovery because requester's usually only have limited computing power accessible such that many matching processes probably have to be performed sequentially. On the other hand, integrating a service matcher into the provider's system can lead to serious security problems allowing service providers to manipulate matching results. However, a better performance may be attainable, at least for market's where providers can be expected to have more computation power accessible than the requesters. This is often true in markets where requester's also take the role of the end users, e.g., app markets. For other markets, e.g., in a business-to-business scenario, this could be different.

As we can learn from these examples, the best solution has to weight many trade-offs and it is context-specific: Different market scenarios lead to different environmental circumstances, which require different solutions. Especially for comprehensive specifications and complex matching approaches as dealt with in this thesis, an appropriate solution is relevant. This leads to the conclusion that a more systematic approach for the integration of service matchers on the architectural level is needed. However, until now, in literature, architectural decisions wrt. service matchers have rarely been discussed and there is no systematic approach for finding a good architecture for the integration of service matchers into a service market. Also applying classic software architecture decision-making methods has not been analysed with respect to service matchers and their influence on market success yet.

In this section, we present a systematic approach for the integration of service matchers into a service market. This includes the definition of requirements and a discussion on architectural tactics based on these requirements. We exemplarily apply our approach to a specific service market taken from the research area of On-The-Fly Computing [SFB901]. As a contribution, a systematic approach that can be used to integrate service matchers into existing markets, but also to support the creation of new service markets from scratch is proposed. Thereby, the general success of service markets influenced by the use of service matching approaches can be improved. [PBS14]

In Subsection 5.4.1, we derive requirements for integrating a matcher into a service market architecture. Based on these requirements, we discuss different architectural tactics in Subsection 5.4.2. Subsection 5.4.3 presents an example application of our approach to service markets in On-The-Fly Computing based on the discussions in the previous chapters. Subsection 5.4.4 deals with related work.

The concepts presented in this section have been published in a conference paper [PBS14] and in a technical report [PBS14].

5.4.1 Requirements

"This section derives requirements for a service matcher's integration into a service market architecture by analyzing the current state of service markets as described in Section 2.2. There may be further, more fine-grained requirements that have to be considered for the matcher's implementation but we focus on the requirements with the largest impact on higher level design decisions.

In the first part of this section, an overview of the requirements and a short explanation of how we collected them are given. After that, the requirements and their derivation are explained in detail. At the end of the section, we discuss trade-offs between different requirements." [PBS14]

5.4.1.1 Overview and Method

"Figure 5.8 depicts an overview of the collected requirements including their dependencies and their trade-offs. A dependency from A to B means that the fulfillment of B supports the fulfillment of A. For example, *R5* and *R6* have an impact on *R4* but their fulfillment alone does not guarantee the fulfillment of *R4*. Also note that neither the list of requirements, nor the depicted dependencies are meant to be a complete collection. We only focused on the ones that are most important in our context.

As we can see in Figure 5.8, the starting point for all requirements is the overall goal: *Market Success*. Only successful service markets will be accepted by a broad range of service requesters and service providers. In order to be successful, service markets need to optimize all market participants' profits while providing all market participants the same prospects [SO11]. Thus, the first two requirements are *market optimality* (*R1*) and *market fairness* (*R2*). The latter is highly determined by *manipulability* (*R3*).

All further requirements are rather technical requirements and can be derived from the market requirements. We derived them by analyzing current markets and the desired operations to be performed in a perfect service market, and looking at a variety of service matchers. We also reflect some of the requirements for a perfect market defined by Schlauderer and Overhage [SO11] but we had to extend them because we put a stronger focus on the technical integration of matcher and not on market mechanisms. Furthermore, we collected our requirements in an iterative process. After obtaining and evaluating an initial set of requirements, some of them had to be refined while others could be aggregated.

In Section 5.4.1.2, we describe all requirements depicted in Figure 5.8 and explain why they are important for our work. Trade-offs between these requirements are explained in Section 5.4.1.3." [PBS14]

5.4.1.2 Requirements Specification

"All market participants' profits are related to the allocation of the services traded on these markets: The requester's goal is to obtain the service which serves her purpose best and the provider's goal is to increase its revenue by selling as many services as possible. The matcher controls the allocation of services by providing matching results that are used to decide which

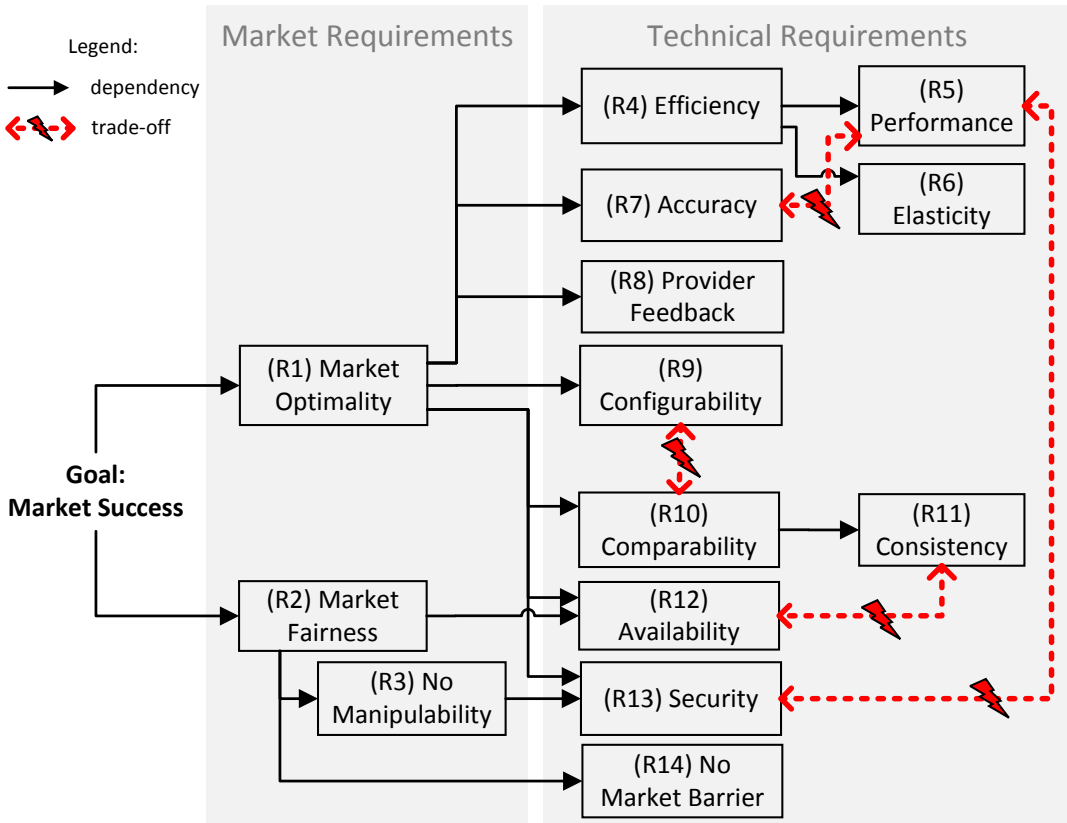


Figure 5.8: Overview of the requirements for integrating a matcher into a market [PBS14]

service is responded for a certain request. Thus, the ability to produce an optimal allocation becomes one of the key requirements of the matcher's integration.

(*R1*) *Market Optimality:* All requesters should obtain those service offers which fit their requirements best.

Being tightly coupled with market optimality is the requirement for market fairness. From market fairness, we can directly derive a requirement for avoiding manipulability.

(*R2*) *Market Fairness:* All market participants should have the same prospects of achieving their goals in the market so that the market remains attractive for them [SO11].

(*R3*) *No Manipulability:* It must not be possible to manipulate matching results such that one market participant improves its prospects over another one, while the service offer stays the same.

As shown in Figure 5.8, in order to satisfy the market requirements, several technical requirements need to be satisfied as well. One of them is the discovery's efficiency which is related to the latency in answering a request. Latency has been proven to be a critical factor for the acceptance of online interactions [Bru09]. Thus, we can assume that requesters do not want to wait long after they have requested a service. Therefore, only service offers that reach the requester after a short time will be considered.

(*R4*) *Efficiency:* Efficiency measures how much resources (e.g., time) are needed to answer a request including the complete search process. The more efficient a discovery runs, the more services can be taken into account and the higher is the probability that the requester can be provided with an optimal service. Thus, the efficiency should be as high as possible.

Since the matcher is part of all transactions during discovery, both failed and successful, the matcher's design has a huge impact on the discovery's efficiency. Especially performance and elasticity are important.

(*R5*) *Performance:* Performance refers to the time to perform one matching process, i.e., the time to determine how much a particular service satisfies the request. It needs to be high in order to gain a good efficiency of the overall discovery process (*R4*).

(*R6*) *Elasticity:* Even if the performance of one matching process is good, the discovery's efficiency is still problematic when a huge amount of matching processes is required. Thus, similar to cloud computing systems, in service markets, the matching system needs to be elastic [HKR13] in a way that it adapts to the amount of required matching processes.

Another essential requirement that is needed to satisfy market optimality is accuracy. A matching result gained fast is still of no value if it is not accurate.

(*R7*) *Accuracy:* The matching result should be as accurate as possible and false positives as well as false negatives should be minimized, i.e., services with a good matching result have to satisfy the request's requirements well and services with a bad matching result must not satisfy the requirements.

The matching result is not only relevant for the requester to select adequate services. It is also useful for the service provider in order to give feedback about missed potential transactions. Thereby, it contributes to market optimality because, based on this feedback, service providers are able to improve their service offers with respect to the customers' needs. In contrast to the provider, the requester does not necessarily need the matching result if she is directly provided with the description of the best service instead.

(*R8*) *Provider Feedback:* Matching results should be returned to the provider as feedback.

The matching result can be an aggregate of matching results for different matching conditions. However, different parties could have different preferences regarding calculation and the weights for different conditions and the aggregated end result (e.g., signatures matching with a higher priority as protocols matching). Thus, a configurable matcher could produce matching results more useful for the requester or the provider.

(*R9*) *Configurability:* It should be possible to configure the matcher wrt. someone's preferences.

However, matching results also need to be comparable so that a requester can be supplied optimally if several adequate services, e.g., from different service providers are available. Furthermore, matching results are only comparable if they are consistent.

(*R10*) *Comparability:* If different services are matched to the same request, services with the same matching result should satisfy the request equally well. Similarly, services with better matching results should satisfy the requester more than services with lower matching results. This has to hold, even and especially, if those services are offered by different providers.

(*R11*) *Consistency:* Dynamic markets, where providers can appear and disappear or change their offers at any time, can lead to situations in which several versions of a service or a service description are available. It has to be avoided that this dynamics leads to inconsistent matching results between different providers so that comparability can be ensured.

Another class of properties needed to achieve optimal and fair markets is dependability. In general, dependability captures several attributes, e.g., availability, maintainability, and reliability [ALR01]. For service matchers in service markets, bad maintainability and bad reliability both lead to a bad availability. Thus, we summarize these properties into one availability requirement *R12*.

(*R12*) *Availability:* A matcher needs to be available as much as possible because unavailable matchers reduce the number of potential service offers that could be forwarded to the requester.

From *R1* and *R2* (respectively *R3*), we can also derive several security requirements. We summarize them into one requirement *R13*. The security requirement is critical. For example, if aspects like reputation ratings of a service are to be matched, service providers have to be prevented from cheating in a way that they claim to have a better reputation than they actually have.

(*R13*) *Security:* Matching results have to be secure so that they cannot be manipulated by any service provider. Similarly, the service specifications the matcher receives as inputs must be secured wrt. manipulability by unauthorized actors: only the provider of the specified service should be able to edit the service specification. However, service specifications may also refer to information that must not be manipulated by the provider itself, e.g., reputation.

In order to create a fair market, market barriers have to be avoided, i.e., the effort to enter the market and to request or to provide a service on this market, should be low. Market barriers are a general problem on service markets [SO11].

(*R14*) *No Market Barrier*: Market barriers need to be avoided to keep a market fair and attractive for all market participants.

However, with the effort related to enter a market also comes the possibility for independence. For example, for some providers, a market in which they are not dependent on some third-party servers but on their own resources, may be more attractive because of their own requirements for qualities like availability and security. Thus, the requirement for avoiding market barriers may not be relevant for all markets." [PBS14]

5.4.1.3 Trade-offs

"As already noted, some of the requirements described in Section 5.4.1.2 result in trade-offs. Figure 5.8 shows these trade-offs by dotted arrows annotated with a flash symbol.

One trade-off is between accuracy (*R7*) and performance (*R5*). The more time the matcher has, the more information can be considered during matching and the more precise algorithms can be run. If only little time is available, [...] *fuzzy matching* can be used to speed-up the matching process [Pla13, PvDB+13].

Since a service market is a large distributed system, the CAP theorem [Aba12] becomes relevant. It means that a distributed system, e.g., a cloud system, cannot provide the three properties *consistency*, *availability*, and *partition-tolerance* at the same time. In our case, especially consistency (*R11*) and availability (*R12*) lead to a trade-off because a high availability can be achieved by replicated data (here: service specifications) which increases the probability of inconsistencies due to different versions in the market.

Furthermore, there is a trade-off between configurability (*R9*) and comparability (*R10*). If a matcher is configurable, matching results are calculated in different ways or based on different inputs. Thus, they are not comparable anymore.

An additional well-known trade-off is between performance (*R5*) and security (*R13*) as adding security-improving mechanisms, e.g., additional encryption, inevitably influences the performance negatively [BHKMT13].

Because of such conflicting requirements, there may not be one general best solution for all service markets. It depends on the specific environment and assumptions, how to balance the different requirements best." [PBS14] As we can see, creating an appropriate architecture always also means to weight of these trade-offs. In addition, the created design decisions can open up new trade-offs, as we will see in the following sections.

5.4.1.4 What makes these requirements special to a matcher?

Many of the listed requirements do not only hold for the matcher but also for further market components. However, some are also specific to the matcher or at least more distinctive for the matcher. Let us consider a certification component as an example. A certification component is responsible for verifying that a service's functional specification is consistent to its actual implementation [SO11, Jak15]. If this is the case, the certification component provides this service with a certificate. This certificate serves as further guarantee for potential requesters and thereby increases the sales opportunities of the certified service.

Requirements like accuracy (*R7*), consistency (*R11*), and security (*R13*) are definitely also relevant for a certification component. Also, requirements like availability (*R12*) and response time (*R5*) play a role but they are not as crucial as in the matching scenario: The matcher is part of much more transactions than the certification component. Each request leads to multiple matching processes, not only for all successfully matched services, but also for all services that do not match. In contrast, the certification is only performed once when a new service version or service specification version appears in the market. Therefore, requirements like availability and response time are much more crucial for the matcher than for a certification component.

Other examples for components in the market with similar requirements as the matcher are an analyzer (analysis service descriptions wrt. functional and non-functional attributes,

e.g., performance predictions) or a monitoring component. Since an analyzer only runs once for each service composition and a monitoring component only runs for each successfully sold service, the same argumentation as for the certification component holds. All in all, at the moment, there seems to be no component, which is as crucial for the overall performance and scalability of the service market and therefore, as important for market success, as the matcher. Thus, design decisions wrt. these requirements are especially interesting when considering the matcher.

Furthermore, a matcher faces special security challenges. We discuss these challenges in one of our earlier publications [PJP+14].

5.4.2 Integrating a Matcher based on Architectural Tactics

Based on the requirements derived in Section 5.4.1, we create architectural tactics [BBK03] regarding how to integrate a matcher into a service market architecture.

When integrating the matcher into the market, based on the given market components, there are different architectural alternatives. Note that there are many different market architectures possible as many different market components can be selected and interaction can vary between different market scenarios. In the following, we focus on one of the most basic scenario in incorporating a requester component, a provider component, and a discovery system mediating between the former two. Three main example alternatives are depicted in Figure 5.9 and discussed in the following.

"These alternatives could be understood as different architectural patterns [BMR+96]. In Alternative A, the matcher is integrated into the requester's system and, therefore, deployed on the requester's hardware. Here, the requester's system accesses the discovery system to get the specifications of the provided services and forwards them to the matcher. Alternative B lets the providers deploy the matcher on their own resources. In this case, the discovery system forwards the request to the providers, where each provider matches its service specifications against the request. In Alternative C, the matcher is part of the discovery system and deployed on the market operator's resources. Instead of the market operator, this role can also be played by another trusted third-party which is part of the market. The specifications of the provided services could still be located at the providers, or, alternatively, stored on the market operator's resources, too. As we can see, the question of where to integrate the matcher is related to the question of who deploys the matcher, whereas, in our case, in particular, the installation phase of the deployment lifecycle is relevant [Dea07].

Each of the three alternatives has different benefits and drawbacks and, in particular, a different impact on the fulfilment of our requirements. Table 5.1 lists these benefits and drawbacks with respect to the requirements collected in Section 5.4.1. A minus in the table means that it is difficult to satisfy the corresponding requirement, whereas a plus means that it is easier to satisfy it. In the minus case, the table also displays the reason. Similarly to the architectural tactics described by Bachmann et al. [BBK03], the table depends on bound and free parameters: Bound parameters are already fixed because their assignment is the same for all service markets. Free parameters (highlighted in italics) need further assumptions to be assigned, i.e., the evaluation can have a different result depending on the properties of a concrete service market. Furthermore, there is a column about weights in order to allow influencing the overall evaluation by assigning special priorities to some requirements.

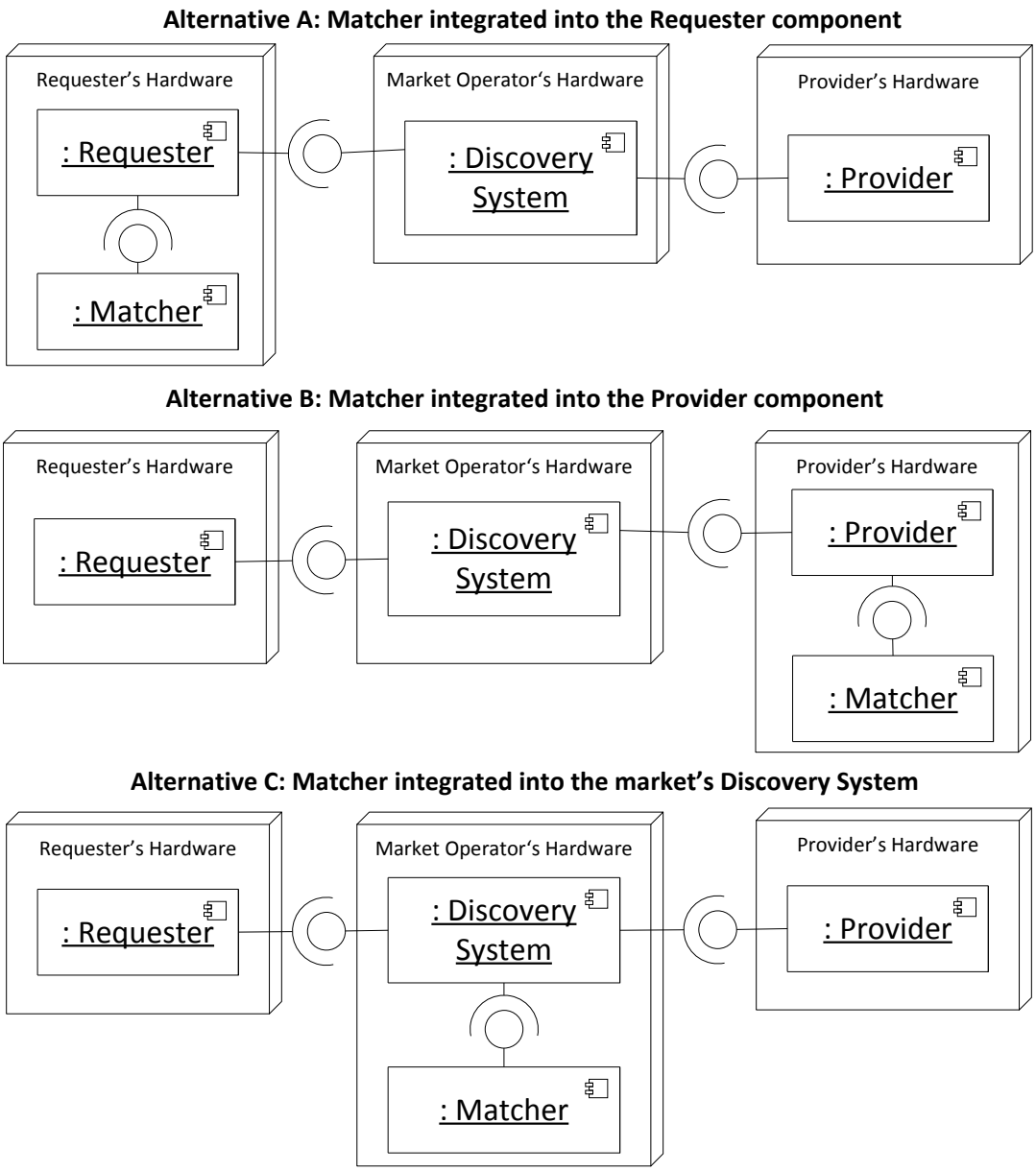


Figure 5.9: Architectural alternatives for matcher integration [PBS14]

Table 5.1: Where to integrate the matcher? [PBS14]

	Alternative A: Requester	**Alternative B: Provider**	**Alternative C: Market Operator**	**w**
(*R4*) Efficiency	- (much transfer)	*depends on location and size of specs*		*?*
(*R5*) Performance	- (no caching)	+	+	*?*
	depends on kind of requesters/providers			
(*R6*) Elasticity	*depends on number of providers and requests at a time*			*?*
(*R8*) Feedback	- (much transfer)	+	- (much transfer)	*?*
(*R9*) Configurability	+	- (conflict *R10*)	- (conflict *R10*)	*?*
(*R10*) Comparability	+	- (conflict with *R9* and *R13*)	- (conflict with *R9*)	*?*
(*R11*) Consistency	- (not insurable)	+	- (not insurable)	*?*
(*R12*) Availability	+	-	+	*?*
		depends on market size		
(*R13*) Security	+	- (high risk for manipulation)	+	*?*
(*R13*) Barriers	- (critical)	+	+	*?*

Weights can be positive (e.g., +1), if a requirement is particularly important, or negative (e.g., -1), if it is less important compared to the other ones. Similar to free parameters, weights depend on properties of concrete service markets, too. Thus, they are not yet assigned in this version of the table. We will discuss an example for a concrete service market with a completed version of Table 5.1 in Section 5.4.3.

The table only covers the technical requirements from Figure 5.8 because these are the ones that we can directly influence with the selection among the three architectural alternatives from Figure 5.9. One exception is accuracy (*R7*) which is not part of the table because it is mostly influenced by the internal design of the matcher and not by its integration into the market architecture. However, we still need to take it into account because of its trade-off with performance.

In the following paragraphs, we describe the evaluation shown in the table for each requirement.

(*R4*) Efficiency

As depicted in Table 5.1, the possibility to satisfy *R4* depends on where the specifications of the provided services are stored and how large they are because the matcher needs the request and the provided specifications as input. The provided specifications could either be stored somewhere in the market or each provider stores the specifications of its provided services. As a conclusion, if we assume that requesters run the matchers, Alternative A, this would lead to a lot of data transfer in any case. Many service specifications (also including the specifications of all *not* matching services) need to be transferred to the

requester before matching can be performed, which costs a lot of bandwidth, increasing with the specifications' size. For Alternative B and C, the evaluation depends on the location and the size of the specifications: If specifications are stored by the providers, having the matcher there, too, would be the most efficient; if specifications have been sent to the market it is more efficient to match them there.

(*R5*) Performance

Also regarding the performance, Alternative A seems not to be a good solution. Matchers located at the service provider or a third party provide the possibility to cache matching results and benefit from it when similar requests from different requesters come in. This possibility is not available for matchers deployed at the requester's because it would only pay off, if one requester repeatedly states similar requests, which is not the case if the discovery scenario works well and the requester already gets a satisfying result after her first request.

Furthermore, if the matcher runs on hardware with high computation power, e.g., compute centers, this could speed up the matching process, too. In contrast, if the matcher runs on a mobile device, a matching process takes longer. Which role can provide the more appropriate resources depends (amongst others) on the domain. For example, service requesters in some technical domain could be assumed to have better hardware available than the typical hotel booking service user, whereas service providers can be expected to have access to more computation power than the requester in both cases.

(*R6*) Elasticity

Elasticity refers to the extent to which the amount of matchers adapts. For Alternative A, the amount of matchers increases with each new requester and thereby indirectly also with the amount of stated requests. This is good if there are many requests but only few service offers. For Alternative B, the amount increases with each new provider, and, thereby, also with the amount of service offers. If we have a market with many providers but only few requesters, Alternative B is preferable. Alternative C suffers from the fact that the market operator has to provide or pay a cloud infrastructure in order to provide an elastic matching architecture.

(*R8*) Feedback

According to *R8*, matching results need to be transferred to the provider. Naturally, this produces the least data transfer costs if the matcher is running on the provider's site, too.

(*R9*) Configurability

As already discussed in Section 5.4.1.3, configurability is in conflict with comparability. This especially holds for Alternative B and C: if each provider uses a different matching configuration, the matching results for service offers of different providers are not comparable for the requester. However, if the requester has the matcher and the possibility to configure it, this is no problem as all services from different providers are matched with the same configuration. In contrast, comparability among different requests is not needed as matching on service markets is one-sided. This means, we are searching for an optimal allocation of services to requests but not the other way around, as software services are

immaterial and there are (almost) no capacity constraints. Thus, for these requirements, Alternative A is the best solution.

(*R10*) Comparability

For comparability, the same argumentation as for configurability holds because of their trade-off. However, in addition, comparability is also influenced by security as in the case of manipulation of matching results, comparability cannot be ensured. This is a disadvantage of Alternative B because it is most susceptible for manipulation, as we will explain below when discussing *R13*.

(*R11*) Consistency

Regarding consistency, Alternative A as well as Alternative C both have the disadvantage that it is not necessarily ensured that they match the service specification describing the provider's current offer. Compared to this, the provider has a better chance to ensure consistency because the provider manages the service specifications herself.

(*R12*) Availability

The availability of the matcher is only certain if it is deployed at the requester's side because if the requester is not available, there is no request to be matched. A similar argumentation holds for the provider: if a provider deploying a matcher is not available, this provider's offers are not matched but all other providers' offers are (by the other providers' matchers). If the market operator deploys the matcher, the matcher as well as the specifications could be unavailable with the consequence that potentially successful transactions fail. However, depending on the number of matchers, the market operator can provide, the invocation of an unavailable matcher can easily be forwarded to another matcher. This option is harder to be realized for Alternative B as the providers are not necessarily connected with each other. Furthermore, for Alternative B, the importance of the availability requirement depends on the size of the particular service market: at least for the requester, it is not a huge problem if some matchers are not available if there are many providers.

(*R13*) Security

Security becomes a problem, in particular, if the provider deploys the matcher. In this case, it is hard to keep the provider from manipulating matching results. In contrast, the requester or the market provider can be assumed not to be interested in faking the results.

(*R14*) Market Barriers

The effort of deploying a matcher (even if the software itself is provided) introduces a market barrier. However, we need to distinguish between the impact of market barriers for the requesters and the impact of the barriers for the providers. If the requester needs to provide hardware to deploy a matcher, the whole market acceptance of customers is questionable. For providers, it seems to be more realistic to assume that they accept a high market barrier induced by the effort of deploying the matcher because they gain profit by selling their services.

Conclusion

As we can see, there is no obvious answer to the question of where to integrate the matcher. Each alternative has its advantages and disadvantages and many aspects depend on the concrete environment, e.g., the domain the market's services belong to or the size of the market." [PBS14]

5.4.3 Exemplary Application: Service Matchers in OTF Markets

"In this [subsection], we take service markets in On-The-Fly (OTF) Computing [SFB901] as an example. OTF Computing investigates markets based on different research areas and addresses different service domains as examples. In this paper, we focus on the service market in the domain of *water network optimization* [DS12a]. We give more information about the assumptions of our example market in Section 5.4.3.1.

5.4.3.1 Context-specific Assumptions

The vision of OTF Computing tries to solve the problem of serving a large variety of customer-specific service requests. Using service composition techniques, basic services provided by service providers on so-called OTF markets can be flexibly combined in order to create complex but individually tailored services that satisfy unique customer requests.

The market for water network optimization services makes a list of assumptions:

A1 The world-wide water network optimization service market is large and highly dynamic with many providers that can appear and disappear at any time. Thus, there are many provided services, while requests can be expected to be stated distributedly over time (because of different time zones a world-wide market has to cope with.

A2 The specifications of the provided services are stored at the providers' systems.

A3 The providers describe their services in a comprehensive way, resulting in detailed service specifications.

A4 Service requesters are mathematicians as well as water network analysts. They request services from desktop computers but their hardware cannot be expected to be high-end.

A5 In addition to the discovery system and the matcher, the market also contains a service composition engine and a reputation system [JBPP14b, JBPP14a].

5.4.3.2 Resolved Dependencies

Based on the assumptions listed before, we can now assign all free parameters of Table 5.1 and create a specifically completed table with respect to OTF Computing. The OTF-specific table is depicted in Table 5.2. [Remember that a minus in the table does not mean the corresponding requirement is not possible to be satisfied. It only means that it is difficult to satisfy it. On the other hand, a plus means that it is easier to satisfy the corresponding requirement and not that the requirement is inevitably satisfied.]

Based on this and the assumptions A1-A5, the table is completed as follows: Regarding efficiency, Alternative B is beneficial because only the request needs to be transferred which

Table 5.2: Adapted version of Table 5.1: Where to integrate the matcher in OTF markets? [PBS14]

	A: Req.	**B: Prov.**	**C: Market O.**	**w**
(*R4*) Efficiency	-	+	-	*+1*
(*R5*) Performance	-	+	+	*+1*
(*R6*) Elasticity	-	+	-	*+1*
(*R8*) Feedback	-	+	-	*0*
(*R9*) Configurability	+	-	-	*+1*
(*R10*) Comparab.	+	-	+	*0*
(*R11*) Consistency	-	+	-	*0*
(*R12*) Availability	+	+	+	*0*
(*R13*) Security	+	-	+	*-1*
(*R14*) M. Barriers	+	-	-	*0*
Sum	5	6	4	*0*
Sum (incl. weights)	5	9	4	*0*

costs less bandwidth than in Alternative A and B, where many large specifications (see A3) need to be transferred (A2). The performance can be assumed to be better in general if the matcher is deployed at the provider's because we cannot assume that the requesters are technical advanced people as their requests can come from many different domains (A4). Regarding Alternative C, a third party like the market operator can be expected to be able to provide resources with high computation power. Regarding elasticity the provider is to be preferred because of A1: With Alternative A, the matcher needs to perform many matching processes for one request. In contrast, with Alternative B, the number of matchers increases with an increasing number of services and thereby with an increasing number of required matching processes. In this case, each matcher only has to take care of the matching processes needed to match the request with services of one service provider. Alternative C can easily become a bottleneck because it cannot be expected to scale with the provided services, nor with the requests (except when the market operator runs a cloud environment). In order to judge availability, we can use Assumption A1, that OTF markets contain many providers. Thus, when deploying the matcher on the provider side, the unavailability of one matcher is not that serious, if there are many substitutable providers, so Alternative B is still a plus.

Due to the market barrier problem, we need to determine that the matching software is always delivered to market participants when they enter the market. Without this assumption, neither Alternative A would be possible as the effort for a requester providing an appropriate matcher is too high and complicates the market entrance drastically, nor Alternative B would be possible because of the conflict with comparability (*R10*).

Based on Table 5.2, we can now make an informed decision. As we can see from the table, Alternative B outweights the other two but the difference is not significant enough to allow a secure decision. For this reason, in the next step, the weights become relevant.

5.4.3.3 Assigned Weights

Next, we need to take into account which requirements are more important to achieve market optimality (*R1*) and market fairness (*R2*) in our application context from OTF Computing and for which drawbacks we can still find acceptable solutions. As shown in Table 5.2, we assigned special weights to efficiency, performance, elasticity, configurability, and security. We discuss these decisions in the following paragraphs.

(*R4*) Efficiency, (*R5*) Performance, (*R6*) Elasticity

In OTF Computing, we put a special focus on the discovery's efficiency (and thereby also on performance and elasticity) because the idea is not only to discover services but also to compose them on the fly (see A5). The composition includes the decomposition of a request into subrequests and the discovery for each subrequest. Leading to a recursive scenario, each subrequest can be decomposed again.

Thus, in order to satisfy one request, not only one but many discoveries and, therefore, multiple matching processes are needed. However, still, a request should be answered in acceptable time. This makes these three requirements important in particular.

(*R9*) Configurability

As described in Section 5.4.1.3, one trade-off to be solved is between accuracy (*R7*) and performance (*R5*). If we give a special weight to performance, what happens to accuracy?

We suggest to cope with this trade-off by using a configurable matcher. Based on the properties of a request's application domain, the matcher can choose a different implementation. Amongst others, different matcher configurations could be achieved by substituting, simplifying, or even omitting matching conditions. For example, in a domain with a very standardized terminology, e.g., touristic, complicated behavioral matching, e.g., pre- and post condition matching, may not be needed because signature matching is already very successful.

However, as we pointed out in Section 5.4.2, configurable matchers may lead to a conflict with comparability (*R10*) when deployed on the provider side: Matching results of different services determined by matchers using different implementations are not comparable anymore. Because of this, we propose that matching configurations only differ between different domains. Thereby, comparability within a domain is given. Comparability between different domains can be neglected because the matching should always fail in such cases, anyway.

(*R13*) Security

We assigned a negative weight to the security requirement because we propose to take actions agains security problems in two ways: (1) The secure interaction with the matcher can be ensured using appropriate security mechanisms, e.g., Service Automata [GMS12]. (2) If proactive security mechanisms fail, a reliable reputation system [JBPP14b, JBPP14a] (A5) could prevent service providers from cheating: Reputation systems provide requesters the possibility to rate a service (or its provider) after using it in order to share their experiences with future requesters. If a service provider manipulates the matcher in a way that it the requester is tricked into buying a service that does not really comply to the request, the requester can submit a bad rating. Thereby, by and by, the service provider's reputation will decrease, lowering its sales opportunities.

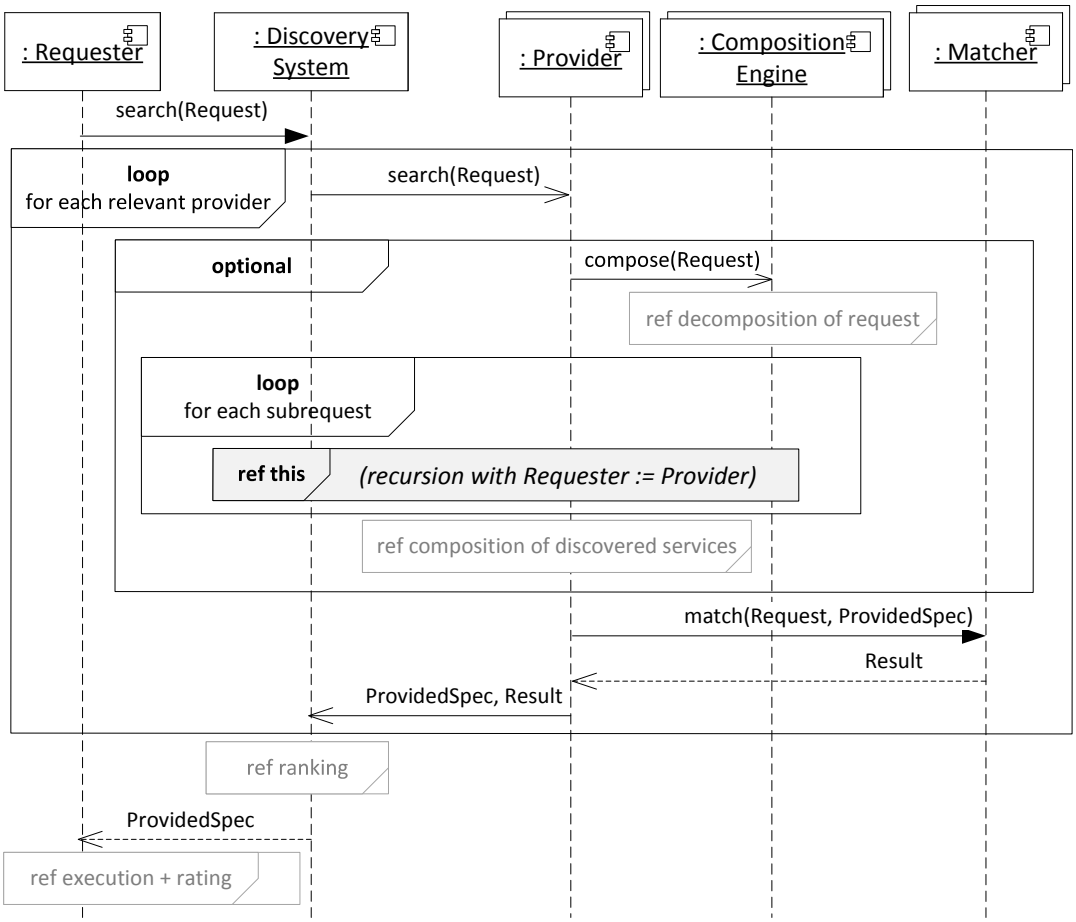


Figure 5.10: Interaction with the matcher on OTF water network optimization service market

5.4.3.4 Resulting System

Evaluating Table 5.2 taking into account the weights, the clear winner is Alternative B: integrating the matcher into the provider's system.

Figure 5.10 shows how the interaction with the matcher integrated into the provider's system works in our exemplary OTF water network optimization market as a sequence diagram. Note that [...] the interaction has been simplified [to achieve a better overview]. For example, the case that a provider provides more than one service has been intentionally neglected here in order to reduce the complexity. However, the sequence diagram nevertheless describes an abstract view of the interaction of the components shown in Alternative B of Figure 5.9 corresponding to the interfaces depicted in Figure 5.9.

As depicted in Figure 5.10, the `Requester`'s request is forwarded to a set of `Provider` components. These providers invoke a `Composition Engine` because in OTF Computing, we assume that for satisfying a request, a composition of multiple services provided in the market is necessary. In this case, the search is performed recursively with the `Provider` becoming a `Requester` and searching for services that can be composed to satisfy the request. Furthermore, each `Provider` forwards incoming requests to its

`Matcher`. At the end, the `Discovery System` obtains many service offers in form of specifications of the provided services `ProvidedSpec` with corresponding matching results. The main search itself is performed asynchronously. Only for services that successfully match, an offer is sent back. After ranking the received service offers, the `Discovery System` sends the best service (or alternatively a set of best services) back to the `Requester` component. An exemplary instance of this sequence diagram for a concrete service composition from the water network optimization domain can be found in [Appendix D.3]." [PBS14] This scenario demonstrates that a more informed decision can be made based on our requirements and tactics.

5.4.4 Related Work

"Our related work is located in the intersection of the research areas of software architectures, service matchers, and service market mechanisms. Even though there is a lot of research for the single areas, the integration of service matchers into a service market has not been addressed on the architectural level, yet. Thus, related work in this intersection is limited to some discussion of general service discovery architectures.

Klusch [Klu08] gives an overview of semantic service discovery architectures and classifies them into centralized and decentralized architectures. However, the alternatives are not discussed with respect to requirements for a matcher's integration nor taking into account market mechanisms. [Centralized architectures have a central service repository which manages the discovery, i.e., contains the matcher. In contrast, the decentralized architecture is characterized by several service repositories, distributed over the network, e.g., peer-to-peer systems.] In their survey [DHC12], Dong et al. build on Klusch's results and classify their surveyed matching approaches regarding discovery architectures into centralized and decentralized architectures, too. In their survey, they categorized most of the SWS matchers as centralized and only some as decentralized, however, they do to go into details regarding how these architectures are realized and which impact they have on the market.

Referring to the architecture description of web services by the W3C [W3C], Jaeger et al. [JRGL+05] describe three main discovery scenarios: a centralized registry, an index, and a peer-to-peer scenario. Based on this, Jaeger et al. distinguish between a (centralized) server-sided and a (index or peer-to-peer) client-sided matching scenario. In the server-sided scenario, the matcher is located at a third-party server, similar to our Alternative C. The requester efforts are described to be low in this scenario. In the client-sided scenario, the requester has the matcher (Alternative A), providing the possibility for the requester to use its own, customized matcher. However, most of our requirements are not discussed in this work. The original architecture description [W3C] briefly mentions some high level advantages of the different architectures but only the requirements wrt. the dynamics and the scalability of the environment are taken into account.

For an overview of general decision-making approaches to choose among different architectural alternatives, we refer to the survey conducted by Falessi et al. [FCKK11]. Many of such approaches could be applied to integrate matchers into service markets as a second step subsequently to our approach. For example, PerOpterix [KKR11] is also about supporting a software architect to make well-informed trade-off decisions when choosing an architecture based on architectural tactics. However, it requires already a rather concrete model of the architecture as input, which we do not have yet in the stage of design where

our approach takes place. PerOpterix could be applied to a service market architecture after requirements and trade-offs are have been explored using our work. In general, a combination of our matching-specific approach and a general approach could thereby provide support on a more detailed level." [PBS14]

5.5 Conclusions

In this chapter, we brought together the concepts we introduced with respect to comprehensive matching processes (Chapter 3) and the concepts we introduced with respect to fuzzy service matching (Chapter 4). This combination has been discussed from three different viewpoints: the development and benefits of fuzzy matching processes, the back-transformation of fuzzy matching results, and the integration of a matcher into a service market's architecture.

For the purpose of user-friendly matching results, we propose incorporating additional sources of information, estimating and presenting fuzziness, and a back-transformation of the matching-results based on traceability links. Our systematic approach for integrating a service matcher into a service market includes the discussion of requirements and architectural tactics in order to enable a more informed decision-making regarding architectural alternatives.

As a benefit of these three working packages, service discovery in service markets becomes more attractive to all market participants as their market entry is supported and their goals are taken into account explicitly. Thereby, service markets become more successful.

All in all, our concepts also represent a first attempt to bridge the gap between real software markets and the mass of existing matching approaches available in the literature.

6

CONCLUSION

This chapter presents the conclusions that can be drawn from this thesis. It also points out future research challenges which can be derived from the results of this thesis.

6.1 Results and Conclusions

This thesis addresses two main research challenges within the area of matching service specifications in service markets: How to deal with complex input specifications and how to deal with imperfect information. In four case studies including altogether ten experiments, we demonstrated the validity of the concepts we proposed to cope with these challenges. In the following, we briefly conclude our research findings.

- **Challenge 1: Dealing with Complex Input Specifications**
  On the one hand, service matching has to be able to manage complex requirements and service specifications in order to provide a holistic and customer-specific evaluation of a service offer. With our model-driven framework *MatchBox*, we introduced the possibility to combine multiple service matching approaches to comprehensive matching processes taking into account a variety of service properties. Our case studies showed that such matching processes provide a flexible and low-effort solution to design and execute matching tailored to diverse scenarios.
- **Challenge 2: Dealing with Imperfect Information**
  On the other hand, service matching has to cope with imperfect information, i.e., requirements and service specifications that are incomplete or imprecise due to effort aversion, indecisiveness, intentional concealment, or lacking knowledge of market participants. As a solution, we propose concepts for *fuzzy matching* that provide decision-making support by quantifying and presenting uncertainty induced by such fuzziness sources. Our case studies showed that such an explicit consideration of fuzziness provides market participants with much more valuable knowledge about offered services than related approaches.

Applying theories and methods from various research areas, allowed us to present interdisciplinary solutions and lead to the fact that our contributions go beyond the state of the art in service matching.

Since we tackle challenges that kept service matching from being broadly applied in practice in the past, both researchers and practitioners benefit from the concepts and tools introduced in this thesis.

6.2 Future Research Challenges

The results of this thesis raise several possibilities for future research. This section discusses a few of these research challenges. Minor possibilities for improvements are pointed out in the discussion and limitations sections of the previous chapters.

Extended Fuzziness Classification

In this thesis, we focused on a fixed set of fuzziness types that we identified as the most important. However, there are further types and their impact on matching results should be investigated. Two examples are *Ambiguity* (i.e., expressions which have more than one semantically unrelated meaning [Zha98]) and *Inconsistency* (i.e., parts of a specification contradict other parts of a specification). These fuzziness types are likely to occur if a requester or a provider is not sufficiently careful while specifying or when she over-specifies her offer. In computer linguistics, there are special methods to eliminate ambiguity. These methods have already been applied to requirements engineering [SJ15]. However, the question whether such methods are applicable and beneficial for service matching, is an open issue.

Algorithm-Induced Fuzziness

The fuzzy matching approaches presented in this thesis focus on provider-induced, requester-induced, and transformation-induced fuzziness. In the future, it would be interesting to also investigate algorithm-induced fuzziness more deeply. Unlike the other three fuzziness sources, we expect algorithm-induced fuzziness to occur especially in matching steps considering functional properties, e.g., matching pre- and post-conditions or protocols, as these often require algorithms of high computational complexity. For example, in Boerding's thesis [Boe15], algorithm-induced fuzziness in combination with a matching approach based on SMT solving is discussed.

Improving Matching Results by Adaptation

In the literature, there are various approaches (e.g. [BR04, GMS12, HA10, IT13, IMTA05, MPS12]) for adaption of software components and services in cases of mismatches, e.g., by the automated generation of adapters. Considering comprehensive as well as fuzzy service matching, new research questions rise up. For example, current adaption approaches focus on either signatures or protocols. But how can we adapt services with respect to comprehensive specifications also describing multiple functional as well as non-functional properties? Existing approaches only target one or two kinds of specifications. Furthermore, we should address coping with fuzziness using adaption approaches. Concerning this subject, the question is: How can we leverage adaption approaches in cases where the matching results are not necessarily bad but uncertain? In order to investigate questions like these, our comprehensive and fuzzy matching concepts could be combined with an extension of adaption approaches like Service Automata [GMS12] which enforce service properties by monitoring and suppressing method calls at runtime.

Service Composition based on Fuzzy Matching Results

Service matching is the foundation for both service discovery and service composition as both require service selection based on some matching result. Up to now, we discussed

how to select a service based on the matching result and additional information about the fuzziness. However, it is unclear, how the information about fuzziness should be used within the composition. For example, one approach to service composition are one-to-many matching approaches (e.g., [HGEJ12b, BCP08]). Unlike the one-to-one-matching approaches addressed in this thesis, one-to-many matching approaches already consider situations where a partial match requires to select further services on order to achieve a full match. We could leverage these approaches in order to cope with uncertain matches, however, this could lead to oversized services with redundant capabilities. This is especially a problem as it is unclear how non-functional properties should be taken into account in these approaches: For functional properties, one-to-many matching approaches usually assume a "the more the better" approach. However, when considering properties like price or performance, this solution fails because of the arising trade-offs. Hence, there is research potential in this area, too.

Interdisciplinary Extensions

The research subjects addressed in this thesis could also be investigated from another point of view taking into account further research areas. For example, the question how to deal with uncertain decisions based on fuzzy matching results could be investigated more deeply by consulting psychological theories about human decision-making (e.g., [RCS97, Sve90, MS76]). Similarly, a more extensive cooperation with economics regarding market theories could contribute to bringing comprehensive and fuzzy matching into practice. For example, research regarding "two-sided matching markets" (e.g., [DG85]) could be leveraged.

User Studies

The incorporation of interdisciplinary collaborations is beneficial for the subject of the evaluation of service matching concepts, too. For example, psychologists could collaborate in planning and executing user studies regarding human decision-making based on (fuzzy) matching results. Exemplary research questions are: "Do human requesters find extra information on fuzziness helpful when selecting a service based on a matching result?", or "What is an appropriate level of abstraction for matching results that are understandable for human requesters?". The applied methods should also be compared with suggestions from the literature about empirical software engineering (e.g., [Kit96, KPP+02, KDJ04, Pre00]).

Moreover, user studies would not only be highly interesting in order to investigate how potential service requesters and providers deal with (fuzzy) matching results, but also how potential matching designers work with matching process configurations for different application domains using MatchBox.

In general, user studies are needed for a level-II-validation [BR08] of our concepts. One idea for a validation strategy incorporating human subjects has already been proposed by Bruns [Bru15].

Service Matching Evaluation Framework

As we pointed out in the validation sections of the previous chapters, evaluating service matching approaches is a complex task coming with many difficulties. Nevertheless, evaluation of service matching approaches needs more attention and tools to support this task also going beyond user studies. Like, SME2 [KP], MatchBox also already provides an environment to measure simple metrics like precision and recall and runtime for matching

runs. However, this functionality does not solve questions like "where to get example specifications?" or "how to get expert results to compare to?". The same problems also hold for pure testing purposes. One idea for addressing the first question could be to improve generative approaches. For example, PerOpterix [KKR11] could be leveraged to systematically generate a variety of service specifications. PerOpterix applies multi-objective evolutionary algorithms in order to generate alternative software architectures for the purpose of performance analysis. Similarly, such a meta-heuristic learning mechanism could contribute to generating exactly those service specification variants that are most interesting when comparing their evaluation results.

Service Matching meets Reengineering

Concepts from the area of reverse engineering, have already been taken into account in service matching in our earlier work [PvDS+13]. Here, we applied concepts used for rating detected design pattern occurrences for the quantification of partial matches based on the graph-based condition language Visual Contracts [HHL05, NHOH10].

In our reengineering approach Archimetrix [PvDB12, vDPB13], we presented a process to detect and remove design deficiencies during architecture reconstructions from monolithic software systems. There are two ways to combine this work with fuzzy matching of comprehensive service specifications:

On the one hand (1), Archimetrix' detection of design deficiencies could benefit from our fuzzy matching results because the detection of design deficiencies can be inflicted with fuzziness occurrences from various fuzziness sources. For example, deficiencies could be specified imprecisely, or the graph matching algorithm used for detection could be speed up by introducing faster approaches based on approximations and heuristics. In these cases, a detected design deficiency could be assigned with a fuzzy matching result indicating how certain this deficiency really is.

On the other hand (2), our matching concepts could leverage Archimetrix' concepts for a deficiency ranking and the previewed deficiency removal when presenting and applying fuzziness reduction strategies. For example, just like in Archimetrix, the reduction of fuzziness occurrences could also have side-effects, like the aggravation of other fuzziness occurrences. Such side-effects could be foreseen in a similar way than Archimetrix foresees the impact of architectural changes.

BIBLIOGRAPHY

The bibliography is structured into four parts: my own publication, the theses I supervised, foreign literature, and referenced websites and standards.

Own Publications

[PSB+] Marie Christin Platenius, Ammar Shaker, Matthias Becker, Eyke Hüllermeier, and Wilhelm Schäfer. Imprecise Matching of Requirements Specifications for Software Services using Fuzzy Logic. *IEEE Transactions on Software Engineering*. Submitted Feb. 2016, Revised Aug. 2016, Current Status: Under Review. 16, 83, 88, 100, 102, 105, 106, 108, 110, 111, 112, 115, 116, 125, 126, 127, 128, 129, 130, 131, 137, 228, 231, 249

[APM+15] Svetlana Arifulina, Marie Christin Platenius, Felix Mohr, Gregor Engels, and Wilhelm Schäfer. Market-Specific Service Compositions: Specification and Matching. In *Proceedings of the IEEE 11th World Congress on Services (SERVICES), Visionary Track: Service Composition for the Future Internet*, pages 333–340. IEEE Computer Society, June 2015. ISBN:978-1-4673-7275-6. doi:10.1109/SERVICES.2015.58. 31, 49

[BBP15] Paul Börding, Melanie Bruns, and Marie Christin Platenius. Comprehensive Service Matching with MatchBox. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*, pages 974–977, New York, NY, USA, September 2015. ACM. ISBN:978-1-4503-3675-8. URL: `http://doi.acm.org/10.1145/2786805.2803181`, doi:10.1145/2786805.2803181. 31, 57

[BPB15] Matthias Becker, Marie Christin Platenius, and Steffen Becker. Cloud Computing Reduces Uncertainties in Quality-of-Service Matching! In Guadalupe Ortiz and Cuong Tran, editors, *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC'14, Revised Selected Papers*, volume 508 of *CCIS*, pages 153–159. Springer International Publishing, September 2015. ISBN:978-3-319-14886-1. URL: `http://dx.doi.org/10.1007/978-3-319-14886-1_15`, doi:10.1007/978-3-319-14886-1_15. 2, 11

[PAPS15] Marie Christin Platenius, Svetlana Arifulina, Ronald Petrlic, and Wilhelm Schäfer. Matching of Incomplete Service Specifications Exemplified by Privacy Policy Matching. In Guadalupe Ortiz and Cuong Tran, editors, *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC'14, Revised Selected Papers*, volume 508 of *CCIS*, pages 6–17. Springer International Publishing, September 2015. ISBN:978-3-319-14885-4. URL: http://dx.doi.org/10.1007/978-3-319-14886-1_2, doi:10.1007/978-3-319-14886-1_2. 14, 15, 16, 22, 23, 24, 83, 98, 117, 118, 119, 133, 134, 227, 238

[PSA15] Marie Christin Platenius, Wilhelm Schäfer, and Svetlana Arifulina. MatchBox: A Framework for Dynamic Configuration of Service Matching Processes. In *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE'15)*, CBSE '15, pages 75–84, New York, NY, USA, May 2015. ACM. ISBN:978-1-4503-3471-6. URL: http://doi.acm.org/10.1145/2737166.2737174, doi:10.1145/2737166.2737174. 29, 30, 31, 33, 34, 35, 39, 43, 47, 48, 53, 54, 56, 57, 58, 66, 68, 71, 231

[APB+14] Svetlana Arifulina, Marie Christin Platenius, Steffen Becker, Christian Gerth, Gregor Engels, and Wilhelm Schäfer. Market-Optimized Service Specification and Matching. In Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri, editors, *Proceedings of the 12th International Conference on Service-Oriented Computing (ICSOC'14)*, volume 8831 of *Lecture Notes in Computer Science*, pages 543–550. Springer Berlin Heidelberg, November 2014. ISBN:978-3-662-45391-9. URL: http://dx.doi.org/10.1007/978-3-662-45391-9_47, doi:10.1007/978-3-662-45391-9_47. 31, 49

[AWBP14] Svetlana Arifulina, Sven Walther, Matthias Becker, and Marie Christin Platenius. SeSAME: Modeling and Analyzing High-quality Service Compositions. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*, pages 839–842, New York, NY, USA, September 2014. ACM. ISBN:978-1-4503-3013-8. URL: http://doi.acm.org/10.1145/2642937.2648621, doi:10.1145/2642937.2648621. 57

[JBPP14a] Alexander Jungmann, Sonja Brangewitz, Ronald Petrlic, and Marie Christin Platenius. Incorporating Reputation Information into Decision-Making Processes in Markets of Composed Services. *International Journal on Advances in Intelligent Systems, issn 1942-2679*, 7(3&4):572–594, December 2014. URL: http://www.iariajournals.org/intelligent_systems/intsys_v7_n34_2014_paged.pdf. 17, 24, 105, 178, 180

[JBPP14b] Alexander Jungmann, Sonja Brangewitz, Ronald Petrlic, and Marie Christin Platenius. Towards a Flexible and Privacy-Preserving Reputation System for Markets of Composed Services. In *Proceedings of the 6th International Conferences on Advanced Service Computing (SERVICE COMPUTATION'14)*,

pages 49–57. IARIA XPS Press, May 2014. ISBN:978-1-61208-337-7. 17, 24, 178, 180

[PBS14] Marie Christin Platenius, Steffen Becker, and Wilhelm Schäfer. Integrating Service Matchers into a Service Market Architecture. In Paris Avgeriou and Uwe Zdun, editors, *Proceedings of the 8th European Conference on Software Architecture (ECSA '14)*, pages 210–217. Springer International Publishing, August 2014. ISBN:978-3-319-09970-5. URL: http://dx.doi.org/10.1007/978-3-319-09970-5_19, doi:10.1007/978-3-319-09970-5_19. 12, 57, 168

[PJP+14] Ronald Petrlic, Alexander Jungmann, Marie Christin Platenius, Wilhelm Schäfer, and Christoph Sorge. Security and Privacy Challenges in On-The-Fly Computing. In *Tagungsband zur 4. Konferenz Software-Technologien und -Prozesse (STeP 2014)*, pages 131–142. Berlin, Boston: De Gruyter Oldenbourg Wissenschaftsverlag, May 2014. ISBN:978-3-11-035865-0. URL: http://www.degruyter.com/view/product/428427. 14, 173

[Pla13] Marie Christin Platenius. Fuzzy Service Matching in On-The-Fly Computing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*, pages 715–718, New York, NY, USA, August 2013. ACM. ISBN:978-1-4503-2237-9. URL: http://doi.acm.org/10.1145/2491411.2492405, doi:10.1145/2491411.2492405. 81, 83, 91, 92, 93, 94, 172

[PvDB+13] Marie Christin Platenius, Markus von Detten, Steffen Becker, Wilhelm Schäfer, and Gregor Engels. A Survey of Fuzzy Service Matching Approaches in the Context of On-The-Fly Computing. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering (CBSE '13)*, pages 143–152, New York, NY, USA, June 2013. ACM. ISBN:978-1-4503-2122-8. URL: http://doi.acm.org/10.1145/2465449.2465454, doi:10.1145/2465449.2465454. 2, 5, 29, 57, 74, 81, 82, 83, 87, 100, 116, 139, 172, 245

[PvDS+13] Marie Christin Platenius, Markus von Detten, Wilhelm Schäfer, Christian Gerth, and Gregor Engels. Service Matching under Consideration of Explicitly Specified Service Variants. In *Proceedings of the 20th International Conference on Web Services (ICWS'13)*, pages 613–614. IEEE Computer Society, June 2013. doi:10.1109/ICWS.2013.98. 188

[vDPB13] Markus von Detten, Marie Christin Platenius, and Steffen Becker. Reengineering Component-Based Software Systems with Archimetrix. *Journal on Software & Systems Modeling*, 13(4):1239–1268, 2013. ISSN:1619-1374. URL: http://dx.doi.org/10.1007/s10270-013-0341-9, doi:10.1007/s10270-013-0341-9. 188

[PvDB12] Marie Christin Platenius, Markus von Detten, and Steffen Becker. Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR'12)*, pages 255–264. IEEE Computer Society, March 2012. doi:10.1109/CSMR.2012.33. 188

[PvDT11] Marie Christin Platenius, Markus von Detten, and Dietrich Travkin. Visualization of Pattern Detection Results in Reclipse. In *Proceedings of the 8 International Fujaba Days*, pages 33–37. Kasseler Informatikschriften (KIS), May 2011. URL: `https://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2012053041248`.

[vDP09] Markus von Detten and Marie Christin Platenius. Improving Dynamic Design Pattern Detection in Reclipse with Set Objects. *Proceedings of the 7th International Fujaba Days*, pages 15–19, November 2009.

Supervised Theses

[Ban14] Dorina Bano. Modeling and Matching of Reputation of Services in On-The-Fly Computing. Master's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, March 2014. 17, 24, 238

[Boe15] Paul Börding. Fuzzy Matching von Vor- und Nachbedingungen in Servicespezifikationen. Bachelor's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, March 2015. In German. 66, 83, 121, 186, 241

[Bru15] Melanie Bruns. Design of an Evaluation Strategy for Fuzzy Service Matching. Bachelor's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, December 2015. 26, 27, 62, 83, 124, 187

[Bun14] Mirko Bunse. Measuring Transformation-induced Uncertainty in Service Matching: A Feasibility Study. Bachelor's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, October 2014. 66, 83, 94, 95, 121

[Gao14] Yuan Gao. Combination of Service Matching Steps in Consideration of Efficiency and Fuzziness. Master's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, May 2014. 31, 48

[Mer14] Sven Merschjohann. Fuzzy Matching of Service Price Specifications. Master's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, September 2014. 67, 83, 121

[Neu15] Conrad Neumann. Ein Framework für Fuzzy Service Matching basierend auf Fuzzy Sets. Bachelor's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, February 2015. In German. 24, 67, 83

[Pet13] Vanessa Petrausch. Klassifizierung unterschiedlicher Ansätze zum Matching von Services. Bachelor's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, May 2013. In German. 74, 139

[Vij14] Shafi Vijapurwala. Handling Incomplete Service Specifications using Fuzzy Matching. Master's thesis, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, June 2014. 83, 121

Literature

[AAY11] Mohamed Almulla, Kawthar Almatori, and Hamdi Yahyaoui. A QoS-Based Fuzzy Model for Ranking Real World Web Services. In *Proceedings of the 18th International Conference on Web Services (ICWS'13)*, pages 203–210. IEEE Computer Society, July 2011. doi:10.1109/ICWS.2011.43. 141, 142, 144, 145

[Aba12] Daniel J. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2):37–42, 2012. ISSN:0018-9162. doi:10.1109/MC.2012.33. 172

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, pages 121–135. Springer Berlin Heidelberg, 2010. ISBN:978-3-642-16145-2. URL: `http://dx.doi.org/10.1007/978-3-642-16145-2_9`, doi:10.1007/978-3-642-16145-2_9. 67

[ACC+02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, October 2002. ISSN:0098-5589. doi:10.1109/TSE.2002.1041053. 155, 161

[AL05] Sudhir Agarwal and Steffen Lamparter. User preference based automated selection of web service compositions. In *Proceedings of the ICSOC Workshop on Dynamic Web Processes (DWP'05) at the 3rd International Conference on Service-Oriented Computing (ICSOC'05)*, pages 1–12. IBM TechReport RC 23822, December 2005. 79

[ALR01] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001. 171

[AMM10] Eyhab Al-Masri and Qusay H. Mahmoud. WSB: A Broker-centric Framework for Quality-driven Web Service Discovery. *Software: Practice and Experience*, 40(10):917–941, September 2010. ISSN:0038-0644. doi:10.1002/spe.v40:10. 78, 79

[ANW09] Nabil I. Al-Najjar and Jonathan Weinstein. The Ambiguity Aversion Literature: A Critical Assessment. *Economics and Philosophy*, 25:249–284, 11 2009. ISSN:1474-0028. doi:10.1017/S026626710999023X. 82

[APB+15] Svetlana Arifulina, Marie Christin Platenius, Matthias Becker, Gregor Engels, and Wilhelm Schäfer. An Overview of Service Specification Language and Matching in On-The-Fly Computing (v0.2). Technical Report tr-ri-15-347, Heinz Nixdorf Institute, University of Paderborn, July 2015. 20, 24, 31, 49, 50

[APG+14] Svetlana Arifulina, Marie Christin Platenius, Christian Gerth, Steffen Becker, Gregor Engels, and Wilhelm Schäfer. Configuration of Specification Language and Matching for Services in On-The-Fly Computing (v0.1). Technical Report tr-ri-14-342, Heinz Nixdorf Institute, University of Paderborn, July 2014. 31, 50, 51, 69, 72

[Ari] Svetlana Arifulina. Solving Heterogeneity for a Successful Service Market. Phd Thesis, Paderborn University; to appear. 6, 13, 14, 31, 50, 51, 72, 81, 93

[Arm00] Phillip G. Armour. The Five Orders of Ignorance. *Communications of the ACM*, 43(10):17–20, October 2000. ISSN:0001-0782. URL: `http://doi.acm.org/10.1145/352183.352194`, doi:10.1145/352183.352194. 90

[BBK03] Felix Bachmann, Len Bass, and Mark Klein. Deriving architectural tactics: A step toward methodical architectural design. Technical report, Software Engineering Institute, Carnegie Mellon University, 2003. CMU/SEI-2003-TR-004. 173

[BBM09] Davide Bacciu, Maria Grazia Buscemi, and Lusine Mkrtchyan. Adaptive Service Selection–A Fuzzy-valued Matchmaking Approach. 2009. Technical Report, Università di Pisa. 141, 142, 143, 146

[BBM10] Davide Bacciu, Maria Grazia Buscemi, and Lusine Mkrtchyan. Adaptive Fuzzy-valued Service Selection. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10)*, pages 2467–2471, New York, NY, USA, 2010. ACM. ISBN:978-1-60558-639-7. URL: `http://doi.acm.org/10.1145/1774088.1774598`, doi:10.1145/1774088.1774598. 2, 91, 141, 142, 143, 144, 145, 146

[BCP08] Antonio Brogi, Sara Corfini, and Razvan Popescu. Semantics-based Composition-oriented Discovery of Web Services. *ACM Transactions on Internet Technology (TOIT)*, 8(4):19:1–19:39, October 2008. ISSN:1533-5399. URL: `http://doi.acm.org/10.1145/1391949.1391953`, doi:10.1145/1391949.1391953. 187

[BG94] Michael Buckland and Fredric Gey. The Relationship between Recall and Precision. *Journal of the American Society for Information Science*, 45(1):12–19, January 1994. ISSN:0002-8231. 27

[BHKMT13] Steffen Becker, Lucia Happe (Kapova), Raffaela Mirandola, and Catia Trubiani. Towards a Methodology Driven by Relationships of Quality Attributes for QoS-based Analysis. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13)*, pages 311–314, New York, NY, USA, 2013. ACM. ISBN:978-1-4503-1636-1. URL: `http://doi.acm.org/10.1145/2479871.2479914`, doi:10.1145/2479871.2479914. 172

[BHS09] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*,

International Handbooks on Information Systems, pages 21–43. Springer Berlin Heidelberg, 2009. ISBN:978-3-540-70999-2. URL: http://dx.doi.org/10.1007/978-3-540-92673-3_1, doi:10.1007/978-3-540-92673-3_1. 21

[BIPT09] Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Automatic Synthesis of Behavior Protocols for Composable Web-Services. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE'09)*, pages 141–150, New York, NY, USA, 2009. ACM. ISBN:978-1-60558-001-2. URL: http://doi.acm.org/10.1145/1595696.1595719, doi:10.1145/1595696.1595719. 5, 72

[BJPW99] Antoine Beugnard, Jean-Marc Jézéquel, Noel Plouzeau, and Damien Watkins. Making components contract aware. *Computer*, 32(7):38–45, July 1999. ISSN:0018-9162. doi:10.1109/2.774917. 21, 29, 33

[BK12] Rahul C. Basole and Jürgen Karla. Value Transformation in the Mobile Service Ecosystem: A Study of App Store Emergence and Growth. *Service Science*, 4(1):24–41, March 2012. ISSN:2164-3962. URL: http://dx.doi.org/10.1287/serv.1120.0004, doi:10.1287/serv.1120.0004. 8

[BKB+07] Pearl Brereton, Barbara Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007. ISSN:0164-1212. doi:10.1016/j.jss.2006.07.009. 73, 139, 146

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, 2009. ISSN:0164-1212. Special Issue: Software Performance – Modeling and Analysis. doi:http://dx.doi.org/10.1016/j.jss.2008.03.066. 59, 92, 115, 116, 146, 237

[BL08] Li Bai and Min Liu. A Fuzzy-set based Semantic Similarity Matching Algorithm for Web Service. In *Proceedings of the 5th International Conference on Services Computing (SCC'08)*, volume 2, pages 529–532. IEEE Computer Society, July 2008. doi:10.1109/SCC.2008.147. 141, 142, 145, 146

[BL11] Li Bai and Min Liu. Fuzzy sets and similarity relations for semantic web service matching. *Computers & Mathematics with Applications*, 61(8):2281–2286, 2011. ISSN:0898-1221. URL: http://www.sciencedirect.com/science/article/pii/S0898122110007467, doi:http://dx.doi.org/10.1016/j.camwa.2010.09.049. 141, 142, 143, 146

[BLB13] Matthias Becker, Markus Luckey, and Steffen Becker. Performance Analysis of Self-adaptive Systems for Requirements Validation at Design-time. In *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA'13)*, pages 43–52, New York, NY, USA, 2013. ACM. ISBN:978-1-4503-2126-6. URL: http://doi.acm.org/10.1145/2465478.2465489, doi:10.1145/2465478.2465489. 104, 116, 130, 131

[BLB15] Matthias Becker, Sebastian Lehrig, and Steffen Becker. Systematically Deriving Quality Metrics for Cloud Computing Systems. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE'15)*, pages 169–174, New York, NY, USA, 2015. ACM. ISBN:978-1-4503-3248-4. URL: http://doi.acm.org/10.1145/2668930.2688043, doi:10.1145/2668930.2688043. 115

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns*. John Wiley and Sons, 1996. ISBN:978-0471958697. 173

[BOR04] Steffen Becker, Sven Overhage, and Ralf Reussner. Classifying Software Component Interoperability Errors to Support Component Adaption. In Ivica Crnkovic, Judith A. Stafford, Heinz W. Schmidt, and Kurt Wallnau, editors, *Proceedings of the 7th International Symposium on Component-Based Software Engineering (CBSE'04)*, volume 3054 of *Lecture Notes in Computer Science*, pages 68–83. Springer Berlin Heidelberg, May 2004. ISBN:978-3-540-24774-6. URL: http://dx.doi.org/10.1007/978-3-540-24774-6_8, doi:10.1007/978-3-540-24774-6_8. 29

[BPDRL07] Ayomi Bandara, Terry Payne, David De Roure, and Tim Lewis. A Semantic Approach for Service Matching in Pervasive Environments. September 2007. URL: http://eprints.soton.ac.uk/id/eprint/264551. 79

[BPS10] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. Fuzzy Goals for Requirements-Driven Adaptation. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10)*, pages 125–134. IEEE Computer Society, September 2010. doi:10.1109/RE.2010.25. 147

[BR04] Steffen Becker and Ralf Reussner. The Impact of Software Component Adaptors on Quality of Service Properties. In Carlos Canal, Juan Manuel Murillo, and Pascal Poizat, editors, *Proceedings of the First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT 04)*, pages 25–30, 2004. URL: http://sdqweb.ipd.uka.de/publications/pdfs/becker2004b.pdf. 186

[BR08] Rainer Böhme and Ralf Reussner. Validation of Predictions with Measurements, 2008. URL: http://dx.doi.org/10.1007/978-3-540-68947-8_3, doi:10.1007/978-3-540-68947-8_3. 72, 124, 137, 187

[Bru09] Jake Brutlag. Speed matters for google web search. June 2009. URL: http://venturebeat.com/wp-content/uploads/2009/11/delayexp.pdf. 169

[Bry00] Barrett R. Bryant. Object-oriented natural language requirements specification. In *Proceedings of the 23rd Australasian Computer Science Conference. (ACSC'00)*, pages 24–30. IEEE Computer Society, 2000. ISBN:0-7695-0518-X. doi:10.1109/ACSC.2000.824376. 165

[BV08] Umesh Bellur and Harin Vadodaria. On Extending Semantic Matchmaking to Include Preconditions and Effects. In *Proceedings of the 15th International Conference on Web Services (ICWS'08)*, pages 120–128. IEEE Computer Society, Sept 2008. doi:10.1109/ICWS.2008.18. 33

[BVG08] Umesh Bellur, Harin Vadodaria, and Amit Gupta. Semantic Matchmaking Algorithms. *Advances in Greedy Algorithms*, pages 481–502, 2008. URL: http://cdn.intechweb.org/pdfs/5832.pdf. 19

[BW03] Wolf-Tilo Balke and Matthias Wagner. Towards Personalized Selection of Web Services. In *Proceedings of the 12th International World Wide Web Conference (WWW'03, Alternate Paper Tracks)*, pages 20–24, 2003. ISBN:963-311-355-5. 78

[CBF05] Ion Constantinescu, Walter Binder, and Boi Faltings. Flexible and efficient matchmaking and ranking in service directories. In *Proceedings of the 12th International Conference on Web Services (ICWS'05)*, pages 5–12. IEEE Computer Society, July 2005. doi:10.1109/ICWS.2005.62. 141, 142, 144, 145

[CCC+08] Alessio Carenini, Dario Cerizza, Marco Comerio, Emanuele Della Valle, Flavio De Paoli, Andrea Maurino, Matteo Palmonari, and Andrea Turati. GLUE2: A Web Service Discovery Engine with Non-Functional Properties. In *Proceedings of the 6th European Conference on Web Services (ECOWS'08)*, pages 21–30. IEEE Computer Society, November 2008. ISBN:978-0-7695339905. doi:10.1109/ECOWS.2008.12. 76

[CCL+05] Kendra Cooper, João W Cangussu, Rong Lin, Ganesan Sankaranarayanan, Ragouramane Soundararadjane, and Eric Wong. An Empirical Study on the Specification and Selection of Components Using Fuzzy Logic. In George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, Clemens Szyperski, and Kurt Wallnau, editors, *Proceedings on the 8th International Symposium on Component-Based Software Engineering (CBSE'05)*, pages 155–170. Springer Berlin Heidelberg, May 2005. ISBN:978-3-540-32049-4. URL: http://dx.doi.org/10.1007/11424529_11, doi:10.1007/11424529_11. 141, 142, 145

[CDG+06] Liliana Cabral, John Domingue, Stefania Galizia, Alessio Gugliotta, Vlad Tanasescu, Carlos Pedrinaci, and Barry Norton. IRS-III: A Broker for Semantic Web Services based Applications. In *Proceedings of the 5th*

International Semantic Web Conference (ISWC'06), pages 201–214. Springer Berlin Heidelberg, 2006. ISBN:978-3-540-49055-5. URL: http://dx.doi.org/10.1007/11926078_15, doi:10.1007/11926078_15. 79

[CF11] Alfredo Cuzzocrea and Marco Fisichella. Discovering Semantic Web Services via Advanced Graph-based Matching. In *Proceedings of the International Conference on Systems, Man, and Cybernetics (SMC'11)*, pages 608–615. IEEE Computer Society, 2011. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6083778&isnumber=6083622, doi:10.1109/ICSMC.2011.6083778. 93

[CGMS07] Leslie Cheung, Leana Golubchik, Nenad Medvidovic, and Gaurav Sukhatme. Identifying and addressing uncertainty in architecture-level software reliability modeling. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–6. IEEE Computer Society, March 2007. doi:10.1109/IPDPS.2007.370524. 147

[CIW99] Daniele Compare, Paola Inverardi, and Alexander L. Wolf. Uncovering Architectural Mismatch in Component Behavior. *Science of Computer Programming*, 33(2):101–131, 1999. ISSN:0167-6423. doi:10.1016/S0167-6423(98)00006-9. 2

[CPZ13a] Elisa Costante, Federica Paci, and Nicola Zannone. Privacy-Aware Web Service Composition and Ranking. In *Proceedings of the 20th International Conference on Web Services (ICWS'13)*, pages 131–138. IEEE Computer Society, June 2013. doi:10.1109/ICWS.2013.27. 14, 15, 16, 22, 237

[CPZ13b] Elisa Costante, Federica Paci, and Nicola Zannone. Privacy-Aware Web Service Composition and Ranking. *International Journal of Web Services Research (IJWSR)*, 10(3):1–23, 2013. ISSN:1545-7362. doi:10.4018/ijwsr.2013070101. 14, 22

[CSH06] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A Survey on Ontology Mapping. *ACM Sigmod Record*, 35(3):34–41, September 2006. ISSN:0163-5808. URL: http://doi.acm.org/10.1145/1168092.1168097, doi:10.1145/1168092.1168097. 3, 13

[CTO10] Yassin Chabeb, Samir Tata, and Alain Ozanne. YASA-M: A Semantic Web Service Matchmaker. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10)*, pages 966–973. IEEE Computer Society, April 2010. doi:10.1109/AINA.2010.122. 78, 79

[CYLT05] Kuo-Ming Chao, Muhammad Younas, Chi-Chun Lo, and Tao-Hsin Tan. Fuzzy Matchmaking for Web Services. In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, volume 2, pages 721–726. IEEE Computer Society, March 2005. doi:10.1109/AINA.2005.189. 141, 142, 143, 144, 146

[CZC08] Marco Crasso, Alejandro Zunino, and Marcelo Campo. Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, 71(2):144–164, 2008. ISSN:0167-6423. URL: http://www.sciencedirect.com/science/article/pii/S0167642308000282, doi:10.1016/j.scico.2008.02.002. 155

[DA09] Demian Antony D'Mello and V.S. Ananthanarayana. Semantic Web Service Selection Based on Service Provider's Business Offerings. *International Journal of Simulation: Systems, Science and Technology (IJSSST)*, 10(2):25–37, 2009. ISSN:1473-804x. 77, 78, 79

[DAS08b] Demian Antony D'Mello, V.S. Ananthanarayana, and T. Santhi. A QoS Broker Based Architecture for Dynamic Web Service Selection. In *Proceedings of the Second Asia International Conference on Modeling & Simulation (AICMS'08)*, pages 101–106. IEEE Computer Society, May 2008. doi:10.1109/AMS.2008.94. 78

[DCCH07] Martine De Cock, Sam Chung, and Omar Hafeez. Selection of Web Services with Imprecise QoS Constraints. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*, pages 535–541. IEEE Computer Society, 2007. ISBN:0-7695-3026-5. URL: http://dx.doi.org/10.1109/WI.2007.92, doi:10.1109/WI.2007.92. 141, 142, 144

[Dea07] Alan Dearle. Software Deployment, Past, Present and Future. In *2007 Future of Software Engineering (FOSE'07)*, pages 269–284. IEEE Computer Society, 2007. ISBN:0-7695-2829-5. URL: http://dx.doi.org/10.1109/FOSE.2007.20, doi:10.1109/FOSE.2007.20. 173

[DFMP04] Didier Dubois, Laurent Foulloy, Gilles Mauris, and Henri Prade. Probability-Possibility Transformations, Triangular Fuzzy Sets, and Probabilistic Inequalities. *Reliable Computing*, 10(4):273–297, 2004. ISSN:1573-1340. URL: http://dx.doi.org/10.1023/B:REOM.0000032115.22510.b5, doi:10.1023/B:REOM.0000032115.22510.b5. 85, 86, 106

[DG85] Gabrielle Demange and David Gale. The strategy structure of two-sided matching markets. *Econometrica: Journal of the Econometric Society*, pages 873–888, 1985. 187

[DHC12] Hai Dong, Farookh Khadeer Hussain, and Elizabeth Chang. Semantic Web Service matchmakers: State of the Art and Challenges. *Concurrency and Computation: Practice and Experience*, 25(7), 2012. doi:10.1002/cpe.2886. 2, 5, 19, 29, 182

[DKRA08] Demian Antony D'Mello, Ivneet Kaur, Namratha Ram, and V S Ananthanarayana. Semantic Web Service Selection Based on Business Offering. In *Proceedings of the 2nd European Symposium on Computer Modeling and Simulation (EMS)*, pages 476–481. IEEE Computer Society, September 2008. doi:10.1109/EMS.2008.24. 78

[DMES14] Christian Diedrich, Matthias Meyer, Lars Evertz, and Wilhelm Schäfer. Dienste in der Automatisierungstechnik. *atp edition - Automatisierungstechnische Praxis*, pages 24–35, December 2014. In German. doi:10.17560/atp.v56i12.467. 9

[Dor79] Robert Dorfman. A Formula for the Gini Coefficient. *The Review of Economics and Statistics*, 61(1):146–149, 1979. ISSN:00346535, 15309142. URL: `http://www.jstor.org/stable/1924845`. 26

[DP97] Didier Dubois and Henry Prade. The Three Semantics of Fuzzy Sets. *Fuzzy Sets and Systems*, 90(2):141–150, 1997. ISSN:0165-0114. doi:http://dx.doi.org/10.1016/S0165-0114(97)00080-8. 84

[DS05] Schahram Dustdar and Wolfgang Schreiner. A Survey on Web Services Composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005. doi:10.1504/IJWGS.2005.007545. 1, 21

[DS12a] Corinna Dohle and Leena Suhl. An Optimization Model for the optimal Usage of Water Tanks in Water Supply Systems. In *Proceedings of the International Conference on Applied Mathematical Optimization and Modelling (APMOD'12)*, pages 404–408. Books on Demand, 2012. 178

[dSZ13] Icamaan da Silva and Andrea Zisman. Decomposing Ratings in Service Compositions. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'13)*, volume 8274 of *Lecture Notes in Computer Science*, pages 474–482. Springer Berlin Heidelberg, 2013. ISBN:978-3-642-45005-1. URL: `http://dx.doi.org/10.1007/978-3-642-45005-1_36`, doi:10.1007/978-3-642-45005-1_36. 92

[EHH+] Gregor Engels, Andreas Hess Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, et al. *Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten*. ISBN:978-3-86491-387-7. 9

[EKM11] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. Taming Uncertainty in Self-Adaptive Software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE'11)*, pages 234–244, New York, NY, USA, 2011. ACM. ISBN:978-1-4503-0443-6. URL: `http://doi.acm.org/10.1145/2025113.2025147`, doi:10.1145/2025113.2025147. 92, 124, 131, 147

[Ell61] Daniel Ellsberg. Risk, Ambiguity, and the Savage axioms. *The Quarterly Journal of Economics*, 75(4):643–669, 1961. ISSN:00335533, 15314650. URL: `http://www.jstor.org/stable/1884324`. 82

[Ell15] Daniel Ellsberg. *Risk, Ambiguity and Decision*. Garland Publishing, Inc., 2015. ISBN:0-8153-4022-2. 82

[ESAEA04] Fatih Emekci, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. A Peer-to-Peer Framework for Web Service Discovery with Ranking. In *Proceedings of the 11th IEEE International Conference on Web Services (ICWS'04)*, pages 192–199. IEEE Computer Society, 2004. doi:10.1109/ICWS.2004.1314739. 78, 79

[ET93] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, 1993. ISBN:9780412042317. 105

[FB88] Deborah Frisch and Jonathan Baron. Ambiguity and Rationality. *Journal of Behavioral Decision Making*, 1(3):149–157, 1988. ISSN:1099-0771. URL: `http://dx.doi.org/10.1002/bdm.3960010303`, doi:10.1002/bdm.3960010303. 83, 120

[FCKK11] Davide Falessi, Giovanni Cantone, Rick Kazman, and Philippe Kruchten. Decision-making Techniques for Software Architecture Design: A Comparative Survey. *ACM Computing Surveys (CSUR)*, 43(4):1–28, October 2011. ISSN:0360-0300. URL: `http://doi.acm.org/10.1145/1978802.1978812`, doi:10.1145/1978802.1978812. 182

[FGT12] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. A Formal Approach to Adaptive Software: Continuous Assurance of Non-Functional Requirements. *Formal Aspects of Computing*, 24(2):163–186, 2012. ISSN:1433-299X. URL: `http://dx.doi.org/10.1007/s00165-011-0207-2`, doi:10.1007/s00165-011-0207-2. 146

[FH14] Markus Fockel and Jörg Holtmann. A requirements engineering methodology combining models and controlled natural language. In *Proceedings of the IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE'14)*, pages 67–76. IEEE Computer Society, Aug 2014. doi:10.1109/MoDRE.2014.6890827. 165

[FLS08] Giuseppe Fenza, Vincenzo Loia, and Sabrina Senatore. A Hybrid Approach to Semantic Web Services Matchmaking. *International Journal of Approximate Reasoning*, 48(3):808–828, August 2008. ISSN:0888-613X. URL: `http://dx.doi.org/10.1016/j.ijar.2008.01.005`, doi:10.1016/j.ijar.2008.01.005. 141, 142, 144, 145

[FT95] Craig R. Fox and Amos Tversky. Ambiguity Aversion and Comparative Ignorance. *The Quarterly Journal of Economics*, 110(3):585–603, August 1995. ISSN:00335533, 15314650. URL: `http://www.jstor.org/stable/2946693`. 83, 120

[Fuc10] Norbert E. Fuchs. Controlled natural language. *Lecture Notes in Computer Science*, 2010. doi:10.1007/978-3-642-14418-9. 158

[Gar10] David Garlan. Software engineering in an uncertain world. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*

(FoSER '10), pages 125–128, New York, NY, USA, 2010. ACM. ISBN:978-1-4503-0427-6. URL: http://doi.acm.org/10.1145/1882362.1882389, doi:10.1145/1882362.1882389. 90, 147

[GCB08] Daniela Grigori, Juan Carlos Corrales, and Mokrane Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Information Systems*, 33(7):681–698, 2008. ISSN:0306-4379. URL: http://www.sciencedirect.com/science/article/pii/S0306437908000161, doi:http://dx.doi.org/10.1016/j.is.2008.02.004. 21

[GDHS15] Christopher Gerking, Stefan Dziwok, Christian Heinzemann, and Wilhelm Schäfer. Domain-Specific Model Checking for Cyber-Physical Systems. In Michalis Famelis, Daniel Ratiu, Martina Seidl, and Gehan Selim, editors, *Proceedings of the 12th Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa'15) co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS'15)*, Ottawa, Canada, September 2015. 161, 165

[GF94] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proceedings of the First International Conference on Requirements Engineering (RE'94)*, pages 94–101. IEEE Computer Society, April 1994. ISBN:0-8186-5480-5. doi:10.1109/ICRE.1994.292398. 161

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN:978-0-201-63361-2. 11, 43, 46, 56

[GHM+08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008. ISSN:1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826808000413, doi:http://dx.doi.org/10.1016/j.websem.2008.05.001. 14, 79

[GMMP09] Michel Grabisch, Jean-Luc Marichal, Radko Mesiar, and Endre Pap. *Aggregation Functions*, volume 127 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2009. ISBN:9781139643221. 25

[GMS12] Richard Gay, Heiko Mantel, and Barbara Sprick. Service Automata. In *Proceedings of the 8th International Workshop on Formal Aspects of Security and Trust (FAST'12)*, Lecture Notes in Computer Science 7140, pages 148–163. Springer Berlin Heidelberg, 2012. ISBN:978-3-642-29420-4. URL: http://dx.doi.org/10.1007/978-3-642-29420-4_10, doi:10.1007/978-3-642-29420-4_10. 11, 180, 186

[GNM+96] Joseph Goguen, Doan Nguyen, José Meseguer, Du Zhang, and Valdis Berzins. Software component search. *Journal of Systems Integration*, 6(1-2):93–

134, 1996. ISSN:1573-8787. URL: http://dx.doi.org/10.1007/BF02262753, doi:10.1007/BF02262753. 78

[GPK04] Katerina Goševa-Popstojanova and Sunil Kamavaram. Software Reliability Estimation under Uncertainty: Generalization of the Method of Moments. In *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, pages 209–218. IEEE Computer Society, March 2004. doi:10.1109/HASE.2004.1281745. 147

[GRRC+07] José Marıa Garcıa, David Ruiz, Antonio Ruiz-Cortés, Octavio Martın-Dıaz, and Manuel Resinas. An Hybrid, QoS-Aware Discovery of Semantic Web Services Using Constraint Programming. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'07)*, volume 4749 of *Lecture Notes in Computer Science*, pages 69–80. Springer Berlin Heidelberg, 2007. ISBN:978-3-540-74973-8. URL: http://dx.doi.org/10.1007/978-3-540-74974-5_6, doi:10.1007/978-3-540-74974-5_6. 78

[GTRRC08] José Marıa Garcıa, Ioan Toma, David Ruiz, and Antonio Ruiz-Cortés. A Service Ranker based on Logic Rules Evaluation and Constraint Programming. In *Proceedings of the 2nd ECOWS Non-Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, volume 411. Citeseer, 2008. 78, 79

[GW06] Holger Giese and Robert Wagner. Incremental Model Synchronization with Triple Graph Grammars. In *Proceedings of the Model Driven Engineering Languages and Systems: 9th International Conference, (MoDELS'06)*, pages 543–557. Springer Berlin Heidelberg, 2006. ISBN:978-3-540-45773-2. URL: http://dx.doi.org/10.1007/11880240_38, doi:10.1007/11880240_38. 164

[HA10] Oliver Hummel and Colin Atkinson. Automated creation and assessment of component adapters with test cases. In *Proceedings of the 13th International Symposium on Component-Based Software Engineering (CBSE'10)*, pages 166–181. Springer Berlin Heidelberg, June 2010. ISBN:978-3-642-13238-4. URL: http://dx.doi.org/10.1007/978-3-642-13238-4_10, doi:10.1007/978-3-642-13238-4_10. 186

[Haz89] Gordon B. Hazen. Ambiguity Aversion and Ambiguity Content in Decision Making Under Uncertainty. *Annals of Operations Research*, 19(1):415–433, 1989. ISSN:1572-9338. URL: http://dx.doi.org/10.1007/BF02283532, doi:10.1007/BF02283532. 83

[HCL05] Chun-Lung Huang, Kuo-Ming Chao, and Chi-Chun Lo. A Moderated Fuzzy Matchmaking for Web Services. In *Fifth International Conference on Computer and Information Technology (CIT'05)*, pages 1116–1122. IEEE Computer Society, Sept 2005. doi:10.1109/CIT.2005.21. 141, 142, 143, 146

[HGEJ12a] Zille Huma, Christian Gerth, Gregor Engels, and Oliver Juwig. A UML-based Approach for Automatic Service Discovery in Heterogeneous Domains. In *Proceedings of the Forum at the Conference on Advanced Information Systems Engineering (CAiSE'12), CEUR Workshop Proceedings*, volume 855 of *Lecture Notes in Computer Science*, pages 90–97. CEUR-WS.org, 2012. URL: `http://ceur-ws.org/Vol-855/paper11.pdf`. 14, 30, 77, 78, 79

[HGEJ12b] Zille Huma, Christian Gerth, Gregor Engels, and Oliver Juwig. Towards an Automatic Service Discovery for UML-Based Rich Service Descriptions. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS'12)*, volume 7590 of *Lecture Notes in Computer Science*, pages 709–725. Springer Berlin Heidelberg, 2012. ISBN:978-3-642-33665-2. URL: `http://dx.doi.org/10.1007/978-3-642-33666-9_45`, doi:10.1007/978-3-642-33666-9_45. 30, 77, 78, 187

[HHL05] Jan Hendrik Hausmann, Reiko Heckel, and Marc Lohmann. Model-Based Development of Web Service Descriptions Enabling a Precise Matching Concept. *International Journal of Web Services Research*, 2(2):67–84, 2005. 188

[HKR13] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity: What it is, and What it is Not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC'13)*. USENIX, June 2013. 170

[HLCY06] Chun-Lung Huang, Chi-Chun Lo, Kuo-Ming Chao, and Muhammad Younas. Reaching consensus: A moderated fuzzy web services discovery method. *Information and Software Technology*, 48(6):410–423, 2006. ISSN:0950-5849. URL: `http://www.sciencedirect.com/science/article/pii/S0950584905001977`, doi:http://dx.doi.org/10.1016/j.infsof.2005.12.011. 72, 141, 142, 144

[HLL+05] Chun-Lung Huang, Chi-Chun Lo, Yinsheng Li, Kuo-Ming Chao, Jen-Yao Chung, and Ying Huang. Service Discovery through Multi-Agent Consensus. In *IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, pages 37–44. IEEE Computer Society, 2005. doi:10.1109/SOSE.2005.32. 72, 94, 113, 141, 142, 144

[HS07] Willem-Jan van den Heuvel and Martin Smits. Transformation of the Software Components and Web Services Market. *Proceedings of the BLED 2007*, pages 246–258, 2007. 5, 81

[IMTA05] Paola Inverardi, Leonardo Mostarda, Massimo Tivoli, and Marco Autili. Synthesis of Correct and Distributed Adaptors for Component-based Systems: An Automatic Approach. In *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering (ASE'05)*,

pages 405–409, New York, NY, USA, 2005. ACM. ISBN:1-58113-993-4. URL: http://doi.acm.org/10.1145/1101908.1101981, doi:10.1145/1101908.1101981. 11, 186

[IO05] M.G. Ilieva and Olga Ormandjieva. Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation. In *10th International Conference on Applications of Natural Language to Information Systems (NLDB'05)*, pages 392–397. Springer Berlin Heidenberg, June 2005. ISBN:978-3-540-32110-1. URL: http://dx.doi.org/10.1007/11428817_45, doi:10.1007/11428817_45. 165

[IR00] Masahiro Inuiguchi and Jaroslav Ramık. Possibilistic linear programming: a brief review of fuzzy mathematical programming and a comparison with stochastic programming in portfolio selection problem. *Fuzzy Sets and Systems*, 111(1):3–28, 2000. ISSN:0165-0114. doi:10.1016/S0165-0114(98)00449-7. 85

[IT13] Paola Inverardi and Massimo Tivoli. Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 3–12, Piscataway, NJ, USA, 2013. IEEE Press. ISBN:978-1-4673-3076-3. URL: http://dl.acm.org/citation.cfm?id=2486788.2486790. 11, 186

[IV98] Nancy Ide and Jean Véronis. Introduction to the Special Issue on Word Sense Disambiguation: The State of the Art. *Computational Linguistics – Special issue on word sense disambiguation*, 24(1):2–40, 1998. ISSN:0891-2017. URL: http://dl.acm.org/citation.cfm?id=972719.972721. 164

[Jak15] Marie-Christine Jakobs. Speed Up Configurable Certificate Validation by Certificate Reduction and Partitioning. In *Proceedings of the 13th International Conference on Software Engineering and Formal Methods (SEFM'15)*, pages 159–174. Springer International Publishing, 2015. ISBN:978-3-319-22969-0. doi:10.1007/978-3-319-22969-0_12. 172

[JCL09] Buhwan Jeong, Hyunbo Cho, and Choonghyun Lee. On the functional quality of service (FQoS) to discover and compose interoperable web services. *Expert Systems with Applications*, 36(3):5411–5418, April 2009. ISSN:0957-4174. URL: http://www.sciencedirect.com/science/article/pii/S095741740800359X, doi:http://dx.doi.org/10.1016/j.eswa.2008.06.087. 141, 142

[JKK13] Alexander Jungmann, Bernd Kleinjohann, and Lisa Kleinjohann. Learning service recommendations. *International Journal of Business Process Integration and Management*, 6(4):284–297, January 2013. 73

[Joh97] Ralph E. Johnson. Frameworks = (Components + Patterns). *Communications of the ACM*, 40(10):39–42, 1997. ISSN:0001-0782. URL: http://doi.acm.org/10.1145/262793.262799, doi:10.1145/262793.262799. 38

[Jos07] Nicolai Josuttis. *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007. ISBN:9780596529550. 11

[JRGL+05] Michael C. Jaeger, Gregor Rojec-Goldmann, Christoph Liebetruth, Gero Mühl, and Kurt Geihs. Ranked Matching for Service Descriptions using OWL-S. In *Kommunikation in Verteilten Systemen (KiVS'05)*, pages 91–102. Springer Berlin Heidenberg, March 2005. ISBN:978-3-540-27301-1. URL: http://dx.doi.org/10.1007/3-540-27301-8_8, doi:10.1007/3-540-27301-8_8. 29, 78, 79, 182

[JT04] Michael C. Jaeger and Stefan Tang. Ranked Matching for Service Descriptions using DAML-S. In *Proceedings of CAiSE'04 Workshops*, volume 228, pages 217–128, 2004. 78

[Kap13] Georgia M. Kapitsaki. Reflecting User Privacy Preferences in Context-Aware Web Services. In *Proceedings of the 20th International Conference on Web Services (ICWS'13)*, pages 123–130. IEEE Computer Society, 2013. doi:10.1109/ICWS.2013.26. 14, 22

[KBB+09] Barbara Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen G. Linkman. Systematic Literature Reviews in Software Engineering – A Systematic Literature Review. *Information & Software Technology*, 51(1):7–15, 2009. ISSN:0950-5849. Special Section - Most Cited Articles in 2002 and Regular Research Papers. doi:http://dx.doi.org/10.1016/j.infsof.2008.09.009. 73, 139

[KC07] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. *Technical Report v2.3, EBSE 2007-001*, 2007. 73, 139

[KCL+95] Jyrki Kontio, Show-Fune Chen, Kevin Limperos, Roseanne Tesoriero, Gianluigi Caldiera, and Mike Deutsch. A COTS Selection Method and Experiences of Its Use. In *Proceedings of the 20th Annual Software Engineering Workshop*, pages 189–215. NASA, November 1995. 78

[KDJ04] Barbara Kitchenham, Tore Dyba, and Magne Jorgensen. Evidence-based Software Engineering. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pages 273–281. IEEE Computer Society, 2004. ISBN:0-7695-2163-0. URL: http://dl.acm.org/citation.cfm?id=998675.999432. 124, 187

[KF07] Matthias Klusch and Benedikt Fries. Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls. In *Proceedings of the SMR2 2007 Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR'07)*, volume 243. CEUR-WS.org, November 2007. URL: http://ceur-ws.org/Vol-243/SMR2-2007-paper4.pdf. 141, 142

[KFS06] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated Semantic Web Service Discovery with OWLS-MX. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, pages 915–922, New York, NY, USA, 2006. ACM. ISBN:1-59593-303-4. URL: http://doi.acm.org/10.1145/1160633.1160796, doi:10.1145/1160633.1160796. 68, 141, 142

[KFS09] Matthias Klusch, Benedikt Fries, and Katia Sycara. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):121–133, 2009. ISSN:1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826808000838, doi:http://dx.doi.org/10.1016/j.websem.2008.10.001. 78, 141, 142

[Kit96] Babara Kitchenham. DESMET: A method for evaluating Software Engineering methods and tools, Technical Report TR96-09. *Department of Computer Science, University of Keele, Staffordshire*, 1996. ISSN:1353-7776. 187

[KK06] Frank Kaufer and Matthias Klusch. WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In *Proceedings of the 4th European Conference on Web Services (ECOWS'06)*, pages 161–170. IEEE Computer Society, 2006. doi:10.1109/ECOWS.2006.39. 78

[KK08] Matthias Klusch and Patrick Kapahnke. Semantic Web Service Selection with SAWSDL-MX. In *Proceedings of the Second International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR'08)*, volume 416, pages 3–17. CEUR-WS.org, October 2008. URL: http://ceur-ws.org/Vol-416/paper1.pdf. 78, 79

[KK09a] Matthias Klusch and Patrick Kapahnke. OWLS-MX3: an adaptive hybrid semantic service matchmaker for OWL-S. In *Proceedings of 3rd International Workshop on Semantic Matchmaking and Resource Retrieval (SMR2), USA*, 2009. 68, 78, 79

[KK09b] Matthias Klusch and Frank Kaufer. WSMO-MX: A Hybrid Semantic Web Service Matchmaker. *Web Intelligence and Agent Systems: An International Journal*, 7(1):23–42, 2009. doi:10.3233/WIA-2009-0153. 78, 79

[KK10] Matthias Klusch and Patrick Kapahnke. iSeM: Approximated Reasoning for Adaptive Hybrid Selection of Semantic Services. *Proceedings of the 7th Extended Semantic Web Conference (ESWC'10)*, pages 30–44, May 2010. URL: http://dx.doi.org/10.1007/978-3-642-13489-0_3, doi:10.1007/978-3-642-13489-0_3. 78, 141, 142, 143

[KK12a] Matthias Klusch and Patrick Kapahnke. Adaptive Signature-Based Semantic Selection of Services with OWLS-MX3. *Multiagent and Grid Systems*, 8(1):69–82, 2012. doi:10.3233/MGS-2012-0184. 68, 78, 79

[KK12b] Matthias Klusch and Patrick Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012. ISSN:1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826812000777, doi:http://dx.doi.org/10.1016/j.websem.2012.07.003. 4, 76, 78, 79, 141, 142

[KKF08] Matthias Klusch, Patrick Kapahnke, and Benedikt Fries. Hybrid Semantic Web Service Retrieval: A Case Study With OWLS-MX. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC'08)*, pages 323–330. IEEE Computer Society, August 2008. ISBN:978-0-7695-3279-0. doi:10.1109/ICSC.2008.20. 78, 79, 141, 142, 143

[KKR07] Ulrich Küster and Birgitta König-Ries. Semantic Service Discovery with DIANE Service Descriptions. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology – Workshops*, pages 152–156. IEEE Computer Society, 2007. ISBN:0-7695-3028-1. URL: http://dl.acm.org/citation.cfm?id=1339264.1339680. 141, 142

[KKR09] Ulrich Küster and Birgitta König-Ries. Semantic Service Discovery with DIANE Service Descriptions. *Semantic Web Services Challenge*, pages 199–216, 2009. URL: http://dx.doi.org/10.1007/978-0-387-72496-6_12, doi:10.1007/978-0-387-72496-6_12. 141, 142

[KKR11] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization. In *7th International Conference on the Quality of Software Architectures (QoSA'11) and 2nd International Symposium on Architecting Critical Systems (ISARCS'11)*, pages 33–42, New York, NY, USA, 2011. ACM. ISBN:978-1-4503-0724-6. URL: http://doi.acm.org/10.1145/2000259.2000267, doi:10.1145/2000259.2000267. 182, 188

[KKRKS07] Ulrich Küster, Birgitta König-Ries, Michael Klein, and Mirco Stern. DIANE: A Matchmaking-Centered Framework for Automated Service Discovery, Composition, Binding, and Invocation on the Web. *International Journal of Electronic Commerce*, 12(2):41–68, 2007. doi:10.2753/JEC1086-4415120202. 91, 140, 141, 142, 143

[KKRSK07] Ulrich Küster, Birgitta König-Ries, Mirco Stern, and Michael Klein. DIANE: An Integrated Approach to Automated Service Discovery, Matchmaking and Composition. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*, pages 1033–1042, New York, NY, USA, 2007. ACM. ISBN:978-1-59593-654-7. URL: http://doi.acm.org/10.1145/1242572.1242711, doi:10.1145/1242572.1242711. 141, 142

[KKZ09b] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. SAWSDL-MX2: A Machine-learning Approach for Integrating Semantic Web Service

Matchmaking Variants. In *Proceedings of the 16th International Conference on Web Services (ICWS'09)*, pages 335–342. IEEE Computer Society, 2009. doi:10.1109/ICWS.2009.76. 78

[KKZ12] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. Adaptive Hybrid Semantic Selection of SAWSDL Services with SAWSDL-MX2, 2012. doi:10.4018/978-1-4666-0185-7.ch007. 78

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. doi:10.1613/jair.301. 160

[Klu08] Matthias Klusch. Semantic Web Service Coordination. In *CASCOM: Intelligent Service Coordination in the Semantic Web*, pages 59–104. Springer, 2008. ISBN:978-3-7643-8575-0. URL: `http://dx.doi.org/10.1007/978-3-7643-8575-0_4`, doi:10.1007/978-3-7643-8575-0_4. 19, 182

[Klu12] Matthias Klusch. *Overview of the S3 Contest: Performance Evaluation of Semantic Service Matchmakers*, chapter 1, pages 17–34. Springer Berlin Heidelberg, 2012. ISBN:978-3-642-28735-0. URL: `http://dx.doi.org/10.1007/978-3-642-28735-0_2`, doi:10.1007/978-3-642-28735-0_2. 68, 76

[KMP02] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Springer Science+Business Media Dordrecht, 2002. ISBN:978-90-481-5507-1. doi:10.1007/978-94-015-9540-7. 85

[Kon95] Jyrki Kontio. OTSO: A Systematic Process for Reusable Software Component Selection. 1995. Technical Report CS-TR-3478. 78, 79, 247

[Kon96] Jyrki Kontio. A Case Study in Applying a Systematic Method for COTS Selection. In *Proceedings of the 18th International Conference on Software Engineering (ICSE'96)*, pages 201–209. IEEE Computer Society, Mar 1996. doi:10.1109/ICSE.1996.493416. 78

[KPP+02] Barbara Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, August 2002. ISSN:0098-5589. doi:10.1109/TSE.2002.1027796. 187

[KvdHD06] Natallia Kokash, Willem-Jan van den Heuvel, and Vincenzo D'Andrea. Leveraging Web Services Discovery with Customizable Hybrid Matching. In *Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC'06)*, pages 522–528. Springer Berlin Heidelberg, 2006. ISBN:978-3-540-68148-9. URL: `http://dx.doi.org/10.1007/11948148_50`, doi:10.1007/11948148_50. 74, 75, 76

[LA07] Steffen Lamparter and Anupriya Ankolekar. Automated Selection of Configurable Web Services. *Wirtschaftinformatik Proceedings 2007*, 2007. URL: http://aisel.aisnet.org/wi2007/28. 78

[LAS+14] Erhan Leblebici, Anthony Anjorin, Andy Schürr, Stephan Hildebrandt, Jan Rieke, and Joel Greenyer. A Comparison of Incremental Triple Graph Grammar Tools. *Electronic Communications of the EASST*, 67, 2014. doi:10.14279/tuj.eceasst.67.939.928. 164

[LASG07] Steffen Lamparter, Anupriya Ankolekar, Rudi Studer, and Stephan Grimm. Preference-based Selection of Highly Configurable Web Services. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 1013–1022, New York, NY, USA, 2007. ACM. ISBN:978-1-59593-654-7. URL: http://doi.acm.org/10.1145/1242572.1242709, doi:10.1145/1242572.1242709. 78

[Les86] Michael Lesk. Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation (SIGDOC'86)*, pages 24–26, New York, NY, USA, 1986. ACM. ISBN:0-89791-224-1. URL: http://doi.acm.org/10.1145/318723.318728, doi:10.1145/318723.318728. 164

[LFT12] Xiaoqing Liu, Kenneth Kofi Fletcher, and Mingdong Tang. Service Selection Based on Personalized Preference and Trade-Offs among QoS Factors and Price. In *Proceedings of the First International Conference on Services Economics (SE'12)*, pages 32–39. IEEE Computer Society, June 2012. doi:10.1109/SE.2012.5. 141, 142, 144

[Li99] Liwu Li. A Semi-Automatic Approach to Translating Use Cases to Sequence Diagrams. In *Proceedings of Technology of Object-Oriented Languages and Systems*, pages 184–193. IEEE Computer Society, July 1999. doi:10.1109/TOOLS.1999.779011. 165

[LL08] Qianhui Althea Liang and Herman Lam. Web Service Matching by Ontology Instance Categorization. In *Proceedings of the 5th IEEE International Conference on Services Computing (SCC'08)*, volume 1, pages 202–209. IEEE Computer Society, July 2008. doi:10.1109/SCC.2008.133. 141, 142

[LLCY06] Wei-Li Lin, Chi-Chun Lo, Kuo-Ming Chao, and M. Younas. Fuzzy Consensus on QoS in Web Services Discovery. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, volume 1, pages 791–798. IEEE Computer Society, April 2006. doi:10.1109/AINA.2006.186. 141, 142

[LLCY08] Wei-Li Lin, Chi-Chun Lo, Kuo-Ming Chao, and M. Younas. Consumer-Centric QoS-aware Selection of Web Services. *Journal of Computer and System Sciences*, 74(2):211–231, 2008. ISSN:0022-0000. Web and Mobile Information Systems 2006. URL: http://www.sciencedirect.com/

science/article/pii/S0022000007000505, doi:http://dx.doi.org/10.1016/j.jcss.2007.04.009. 141, 142

[LPDA04] Daniel Lucredio, Antonio Francisco do Prado, and Eduardo Santana De Almeida. A Survey on Software Components Search and Retrieval. In *Proceedings of the 30th Euromicro Conference*, pages 152–159. IEEE Computer Society, September 2004. ISBN:0-7695-2199-1. doi:10.1109/EURMIC.2004.1333367. 2, 19

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997. ISSN:1433-2779. doi:10.1007/s100090050010. 67

[LS97] Raanan Lipshitz and Orna Strauss. Coping with Uncertainty: A Naturalistic Decision-Making Analysis. *Organizational Behavior and Human Decision Processes*, 69(2):149–163, 1997. ISSN:0749-5978. URL: http://www.sciencedirect.com/science/article/pii/S0749597897926790, doi:http://dx.doi.org/10.1006/obhd.1997.2679. 83, 89, 90

[LSB14] Emmanuel Letier, David Stefan, and Earl T. Barr. Uncertainty, Risk, and Information Value in Software Requirements and Architecture. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, pages 883–894, New York, NY, USA, 2014. ACM. URL: http://doi.acm.org/10.1145/2568225.2568239, doi:10.1145/2568225.2568239. 124

[LSH+12] Min Liu, Weiming Shen, Qi Hao, Junwei Yan, and Li Bai. A fuzzy matchmaking approach for Semantic Web Services with application to collaborative material selection. *Computers in Industry*, 63(3):193–209, April 2012. ISSN:0166-3615. URL: http://www.sciencedirect.com/science/article/pii/S0166361511001151, doi:http://dx.doi.org/10.1016/j.compind.2011.10.001. 141, 142

[LSHY09] Min Liu, Weiming Shen, Qi Hao, and Junwei Yan. An weighted ontology-based semantic similarity algorithm for web service. *Expert Systems with Applications*, 36(10):12480–12490, December 2009. ISSN:0957-4174. URL: http://www.sciencedirect.com/science/article/pii/S0957417409003741, doi:http://dx.doi.org/10.1016/j.eswa.2009.04.034. 141, 142, 145

[Man06] Keith S. Manson. Computer system with natural language to machine language translator, August 2006. US Patent 7085708 B2. URL: https://www.google.com/patents/US7085708. 165

[Men04] Vladimir Mencl. Deriving Behavior Specifications from Textual Use Cases. In *Proceedings of the International Workshop on Intelligent Technologies for Software Engineering (WITSE'04)*. Oesterreichische Computer Gesellschaft, September 2004. 165

[Mey88] Bertrand Meyer. *Object-Oriented Software Construction*, volume 2. Prentice Hall New York, 1988. ISBN:9780136291558. 21

[MHB+12] N. Masuch, B. Hirsch, M. Burkhardt, A. Heßler, and S. Albayrak. SeMa2: A Hybrid Semantic Service Matching Approach. In *Semantic Web Services*, chapter 3, pages 35–47. Springer Berlin Heidelberg, 2012. ISBN:978-3-642-28735-0. URL: http://dx.doi.org/10.1007/978-3-642-28735-0_3, doi:10.1007/978-3-642-28735-0_3. 78, 79

[ML08] Ralph Mietzner and Frank Leymann. Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In *Proceedings of the IEEE International Conference on Services Computing (SCC'08)*, volume 2, pages 359–366. IEEE Computer Society, July 2008. doi:10.1109/SCC.2008.85. 79

[MM07] Nenad Medvidovic and Sam Malek. Software Deployment Architecture and Quality-of-Service in Pervasive Environments. In *International Workshop on Engineering of Software Services for Pervasive Environments (ESSPE'07)*, pages 47–51, New York, NY, USA, 2007. ACM. ISBN:978-1-59593-798-8. URL: http://doi.acm.org/10.1145/1294904.1294911, doi:10.1145/1294904.1294911. 146

[MMAG11] Indika Meedeniya, Irene Moser, Aldeida Aleti, and Lars Grunske. Architecture-Based Reliability Evaluation under Uncertainty. In *Proceedings of the joint ACM SIGSOFT Conference and ACM SIGSOFT Symposium on Quality of Software Architectures and Architecting Critical Systems (QoSA/ISARCS'11)*, pages 85–94, New York, NY, USA, 2011. ACM. ISBN:978-1-4503-0724-6. URL: http://doi.acm.org/10.1145/2000259.2000275, doi:10.1145/2000259.2000275. 147, 148

[Mor97] Ana M. Moreno. Object-Oriented Analysis from Textual Specifications. In *Proceedings of the Ninth International Conference on Software Engineering and Knowledge Engineering (SEKE'97)*, June 1997. 165

[MPMM98] Roland T. Mittermeir, Heinz Pozewaunig, Ali Mili, and Rym Mili. Uncertainty Aspects in Component Retrieval. In *Proceedings of the 7th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 564–571. EDK, July 1998. ISBN:2-84254-013-1. 19, 81, 82, 167

[MPS12] Radu Mateescu, Pascal Poizat, and Gwen Salaün. Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. *IEEE Transactions on Software Engineering*, 38(4):755–777, July 2012. ISSN:0098-5589. doi:10.1109/TSE.2011.62. 186

[MRE07] Abdallah Mohamed, Guenther Ruhe, and Armin Eberlein. COTS Selection: Past, Present, and Future. In *Proceedings of the 14th Annual IEEE*

International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), pages 103–114. IEEE Computer Society, March 2007. doi:10.1109/ECBS.2007.28. 11, 29, 247

[MS76] Henry Montgomery and Ola Svenson. On decision rules and information processing strategies for choices among multiattribute alternatives. *Scandinavian Journal of Psychology*, 17(1):283–291, September 1976. doi:10.1111/j.1467-9450.1976.tb00241.x. 187

[MS04] E. Michael Maximilien and Munindar P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, Sept 2004. ISSN:1089-7801. doi:10.1109/MIC.2004.27. 77

[MSZ11] Khaled Mahbub, George Spanoudakis, and Andrea Zisman. A monitoring approach for runtime service discovery. *Automated Software Engineering – An International Journal*, 18(2):117–161, 2011. ISSN:1573-7535. URL: `http://dx.doi.org/10.1007/s10515-010-0077-5`, doi:10.1007/s10515-010-0077-5. 78, 79

[Mur04] Richard Murch. *Autonomic Computing*. IBM Press, 2004. ISBN:013144025X. 153

[MXB10] Hamid Reza Motahari-Nezhad, Guang Yuan Xu, and Boualem Benatallah. Protocol-Aware Matching of Web Service Interfaces for Adapter Development. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, page 731, New York, New York, USA, 2010. ACM. ISBN:978-1-60558-799-8. doi:10.1145/1772690.1772765. 2, 77, 78, 79, 145

[MZH08] Jiangang Ma, Yanchun Zhang, and Jing He. Efficiently Finding Web Services Using a Clustering Semantic Approach. In *Proceedings of the 2008 International Workshop on Context enabled source and service selection, integration and adaptation: organized with the 17th International World Wide Web Conference (WWW'08)*, pages 1–8, New York, NY, USA, 2008. ACM. ISBN:978-1-60558-107-1. URL: `http://doi.acm.org/10.1145/1361482.1361487`, doi:10.1145/1361482.1361487. 141, 142

[NHOH10] Muhammad Naeem, Reiko Heckel, Fernando Orejas, and Frank Hermann. Incremental Service Composition Based on Partial Matching of Visual Contracts. In *Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering (FASE'10)*, pages 123–138. Springer Berlin Heidelberg, 2010. ISBN:978-3-642-12029-9. doi:10.1007/978-3-642-12029-9_9. 188

[NK13] Le Duy Ngan and Rajaraman Kanagasabai. Semantic Web service discovery: state-of-the-art and research challenges. *Personal and Ubiquitous Computing*, 17(8):1741–1752, 2013. URL: `http://dx.doi.org/10.1007/s00779-012-0609-z`, doi:10.1007/s00779-012-0609-z. 1

[PBS14] Marie Christin Platenius, Steffen Becker, and Wilhelm Schäfer. Integrating Service Matchers into a Service Market Architecture. Technical Report tr-ri-14-340, Heinz Nixdorf Institute, University of Paderborn, June 2014. 12, 167, 168, 169, 171, 172, 174, 175, 178, 179, 182, 183, 228, 231

[PCP09] Matteo Palmonari, Marco Comerio, and Flavio De Paoli. Effective and Flexible NFP-Based Ranking of Web Services. In *Proceedings of the 7th IEEE International Conference on Service-Oriented Computing (ICSOC'09)*, pages 546–560. Springer Berlin Heidelberg, November 2009. ISBN:978-3-642-10383-4. URL: `http://dx.doi.org/10.1007/978-3-642-10383-4_40`, doi:10.1007/978-3-642-10383-4_40. 141, 142, 144, 146

[PHC08] Zhichao Peng, Wenhua He, and Daiwu Chen. Research on Fuzzy Matching Model for Semantic Web Services. In *Proceedings of the International Conference on Intelligent System and Knowledge Engineering (ISKE'08)*, volume 1, pages 440–445. IEEE Computer Society, Nov 2008. doi:10.1109/ISKE.2008.4730970. 141, 142

[PJvR+16] Marie Christin Platenius, Klementina Josifovska, Lorijn van Rooijen, Svetlana Arifulina, Matthias Becker, Gregor Engels, and Wilhelm Schäfer. An Overview of Service Specification Language and Matching in On-The-Fly Computing (v0.3). Technical Report tr-ri-16-349, Heinz Nixdorf Institute, University of Paderborn, January 2016. 13, 14, 22, 31, 228, 229, 236, 237, 238, 239

[PKCH05] Jyotishman Pathak, Neeraj Koul, Doina Caragea, and Vasant G. Honavar. A Framework for Semantic Web Services Discovery. In *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management (WIDM '05)*, pages 45–50, New York, NY, USA, 2005. ACM. ISBN:1-59593-194-5. URL: `http://doi.acm.org/10.1145/1097047.1097057`, doi:10.1145/1097047.1097057. 79

[PPM14] Diego Perez-Palacin and Raffaela Mirandola. Uncertainties in the Modeling of Self-adaptive Systems: a Taxonomy and an Example of Availability Evaluation. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE'14)*, pages 3–14, New York, NY, USA, 2014. ACM. ISBN:978-1-4503-2733-6. URL: `http://doi.acm.org/10.1145/2568088.2568095`, doi:10.1145/2568088.2568095. 89, 90, 147

[PR11] Klaus Pohl and Chris Rupp. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level – IREB compliant*. Rocky Nook, Inc., 1st edition, 2011. ISBN:1933952814, 9781933952819. 155

[Pre00] Lutz Prechelt. *Rolle und Methodik kontrollierter Experimente in der Softwaretechnik*. PhD thesis, University of Karlsruhe, 2000. 27, 62, 71, 187

[PRVM13] Ioannis Patiniotakis, Stamatia Rizou, Yiannis Verginadis, and Gregoris Mentzas. Managing Imprecise Criteria in Cloud Service Ranking with a Fuzzy Multi-criteria Decision Making Method. In *Proceedings of the Second European Conference on Service-Oriented and Cloud Computing (ESOCC'13)*, pages 34–48. Springer Berlin Heidelberg, September 2013. ISBN:978-3-642-40651-5. URL: `http://dx.doi.org/10.1007/978-3-642-40651-5_4`, doi:10.1007/978-3-642-40651-5_4. 93, 100, 141, 142, 144

[PTDL08] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: A Research Roadmap. *International Journal on Cooperative Information Systems*, 17(2):223–255, 2008. ISSN:1793-6365. doi:10.1142/S0218843008001816. 1

[PvdH03] Michael P. Papazoglou and Willem-Jan van den Heuvel. Service-Oriented Computing: State of the Art and Open Research Issues. *IEEE Computer*, 40(11):38–45, 2003. ISSN:0018-9162. doi:10.1109/MC.2007.400. 1

[QWO13] Lie Qu, Yan Wang, and Mehmet Orgun. Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment. In *Proceedings of the International Conference on Services Computing (SCC'13)*, pages 152–159. IEEE Computer Society, June 2013. doi:10.1109/SCC.2013.92. 141, 142

[RBH+07] Ralf Reussner, Steffen Becker, Jens Happe, Heiko Koziolek, Klaus Krogmann, and Michael Kuperberg. The Palladio Component Model. *Technical Report*, 21(5), 2007. 237

[RCS97] Rob Ranyard, W. Ray Crozier, and Ola Svenson. *Decision Making: Cognitive models and explanations*, volume 1. Routledge, 1997. ISBN:0-415-15818-4. 187

[RKLB09] Michael Rambold, Holger Kasinger, Florian Lautenbacher, and Bernhard Bauer. Towards Autonomic Service Discovery – A Survey and Comparison. In *Proceedings of the 6th IEEE International Conference on Services Computing (SCC'09)*, pages 192–201. IEEE Computer Society, Sept 2009. doi:10.1109/SCC.2009.59. 1

[RS03] D. Vijay Rao and V.V.S. Sarma. A rough-fuzzy approach for retrieval of candidate components for software reuse. *Pattern Recognition Letters*, 24(6):875–886, March 2003. ISSN:0167-8655. doi:10.1016/S0167-8655(02)00199-X. 82, 141, 142, 143, 145

[RS05] Jinghai Rao and Xiaomeng Su. A Survey of Automated Web Service Composition Methods. pages 43–54, July 2005. URL: `http://dx.doi.org/10.1007/978-3-540-30581-1_5`, doi:10.1007/978-3-540-30581-1_5. 21

[SB98] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998. ISSN:1045-9227. URL: http://dx.doi.org/10.1109/TNN.1998.712192, doi:10.1109/TNN.1998.712192. 160

[SCS10] Jörg Schönfisch, Willy Chen, and Heiner Stuckenschmidt. A purely logic-based approach to approximate matching of Semantic Web Services. In *Proceedings of the International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*, volume 667, pages 37–52. CEUR-WS.org, 2010. 141, 142, 143, 145

[SDMIS04] Balsamo Simonetta, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004. ISSN:0098-5589. doi:10.1109/TSE.2004.9. 92, 115

[SFHC13] Jordy Sangers, Flavius Frasincar, Frederik Hogenboom, and Vadim Chepegin. Semantic Web service discovery using natural language processing techniques. *Expert Systems with Applications*, 40(11):4660–4671, 2013. ISSN:0957-4174. URL: http://www.sciencedirect.com/science/article/pii/S0957417413001279, doi:10.1016/j.eswa.2013.02.011. 141, 142

[SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley New York, 2nd edition, 2002. ISBN:0201745720. 1

[SH11] Robin Senge and Eyke Hüllermeier. Top-Down Induction of Fuzzy Pattern Trees. *IEEE Transactions on Fuzzy Systems*, 19(2):241–252, April 2011. ISSN:1063-6706. doi:10.1109/TFUZZ.2010.2093532. 26

[Sha83] Carl Shapiro. Premiums for High Quality Products as Returns to Reputations. *The Quarterly Journal of Economics*, 98(4):659–680, 1983. URL: http://www.jstor.org/stable/1881782. 16

[SJ15] Unnati S. Shah and Devesh C. Jinwala. Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey. *ACM SIGSOFT Software Engineering Notes*, 40(5):1–7, September 2015. ISSN:0163-5948. URL: http://doi.acm.org/10.1145/2815021.2815032, doi:10.1145/2815021.2815032. 186

[SM86a] Gerard Salton and Michael J McGill. *Introduction to modern information retrieval*. McGraw-Hill, Inc., 1986. 26

[SMSA+05] Tanveer Syeda-Mahmood, Gauri Shah, Rama Akkiraju, Anca-Andrea Ivan, and Richard Goodwin. Searching Service Repositories by Combining Semantic and Ontological Matching. In *Proceedings of the 12th International Conference on Web Services (ICWS'05)*, pages 13–20. IEEE Computer Society, July 2005. doi:10.1109/ICWS.2005.102. 78

[SO11] Sebastian Schlauderer and Sven Overhage. How Perfect are Markets for Software Services? An Economic Perspective on Market Deficiencies and Desirable Market Features. In *Proceedings of the 19th European Conference on Information Systems (ECIS'11)*. AIS Electronic Library, 2011. URL: http://aisel.aisnet.org/ecis2011/110. 1, 5, 12, 81, 100, 168, 169, 171, 172

[SS08] Christian Sanchez and Leonid Sheremetov. A Model for Semantic Service Matching with Leftover and Missing Information. In *Proceedings of the Eighth International Conference on Hybrid Intelligent Systems (HIS'08)*, pages 198–203. IEEE Computer Society, September 2008. doi:10.1109/HIS.2008.70. 141, 142

[ŞT06] Ioana Şora and Doru Todinca. Specification-based Retrieval of Software Components through Fuzzy Inference. *Acta Polytechnica Hungarica*, 3(3):123–135, 2006. 81, 141, 142, 145

[ŞT09] Ioana Şora and Doru Todinca. Using Fuzzy Logic for the Specification and Retrieval of Software Components. 2009. 141, 142, 145

[SV06] Thomas Stahl and Markus Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. ISBN:0470025700. 38, 54

[Sve90] Ola Svenson. Some propositions for the classification of decision situations. *Contemporary Issues in Decision-making*, pages 17–31, 1990. 187

[SW05] Eleni Stroulia and Yiqiao Wang. Structural and Semantic Matching for Assessing Web-Service Similarity. *International Journal of Cooperative Information Systems*, 14(04):407–437, 2005. ISSN:1793-6365. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218843005001213, doi:10.1142/S0218843005001213. 2, 4, 78

[SWKL02] Katia Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002. ISSN:1573-7454. URL: http://dx.doi.org/10.1023/A:1014897210525, doi:10.1023/A:1014897210525. 78, 79, 141, 142, 144

[SZ11] George Spanoudakis and Andrea Zisman. Designing and Adapting Service-based Systems: A Service Discovery Framework. In *Service Engineering*, pages 261–297. Springer Vienna, 2011. ISBN:978-3-7091-0414-9. URL: http://dx.doi.org/10.1007/978-3-7091-0415-6_10, doi:10.1007/978-3-7091-0415-6_10. 78, 79

[TAS11] Romina Torres, Hernán Astudillo, and Rodrigo Salas. Self-Adaptive Fuzzy QoS-Driven Web Service Discovery. In *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC'11)*, pages 64–71. IEEE Computer Society, July 2011. doi:10.1109/SCC.2011.87. 141, 142, 143, 144, 146

[TCM08] Alessandra Toninelli, Antonio Corradi, and Rebecca Montanari. Semantic-based discovery to support mobile context-aware service access. *Computer Communications*, 31(5):935–949, 2008. ISSN:0140-3664. doi:http://dx.doi.org/10.1016/j.comcom.2007.12.026. 141, 142, 144

[TGRBD07] Eran Toch, Avigdor Gal, Iris Reinhartz-Berger, and Dov Dori. A Semantic Approach to Approximate Service Retrieval. *ACM Transactions on Internet Technology (TOIT)*, 8(1), November 2007. ISSN:1533-5399. URL: `http://doi.acm.org/10.1145/1294148.1294150`, doi:10.1145/1294148.1294150. 141, 142

[TMG+13] Salah-Eddine Tbahriti, Brahim Medjahed, Chirine Ghedira, Djamal Benslimane, and Michael Mrissa. Respecting Privacy in Web Service Composition. In *Proceedings of the 20th International Conference on Web Services (ICWS'13)*, pages 139–146. IEEE Computer Society, June 2013. doi:10.1109/ICWS.2013.28. 14

[TN07] Vicenç Torra and Yasuo Narukawa. *Modeling Decisions: Information Fusion and Aggregation Operators*. Cognitive Technologies. Springer Berlin Heidelberg, 2007. ISBN:9783540687894. URL: `https://books.google.de/books?id=pfqxjwEACAAJ`. 25, 26, 227

[TOAD07] Dhavalkumar Thakker, Taha Osman, and David Al-Dabass. Semantic-Driven Matchmaking and Composition of Web Services Using Case-Based Reasoning. In *Proceedings of the Fifth European Conference on Web Services (ECOWS'07)*, pages 67–76. IEEE Computer Society, Nov 2007. doi:10.1109/ECOWS.2007.11. 141, 142, 145

[TT08] Vuong Xuan Tran and Hidekazu Tsuji. QoS based ranking for web services: Fuzzy approaches. In *Proceedings of the 4th International Conference on Next Generation Web Services Practices*, pages 77–82. IEEE Computer Society, 2008. 100

[vdHYP01] Willem-Jan van den Heuvel, Jian Yang, and Michael P. Papazoglou. Service Representation, Discovery, and Composition for E-marketplaces. In *Proceedings of the 9th International Conference on Cooperative Information Systems (CoopIS'01)*, pages 270–284. Springer Berlin Heidelberg, 2001. ISBN:978-3-540-44751-1. URL: `http://dx.doi.org/10.1007/3-540-44751-2_21`, doi:10.1007/3-540-44751-2_21. 78

[VHA06] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer. Towards P2P-Based Semantic Web Service Discovery with QoS Support. In *Proceedings of the International Business Process Management Workshops, Revised Selected Papers*, pages 18–31. Springer Berlin Heidelberg, September 2006. ISBN:978-3-540-32596-3. URL: `http://dx.doi.org/10.1007/11678564_3`, doi:10.1007/11678564_3. 79

[VPAH07] Le-Hung Vu, Fabio Porto, Karl Aberer, and Manfred Hauswirth. An Extensible and Personalized Approach to QoS-enabled Service Discovery. In

Proceedings of the 11th International Conference on Database Engineering and Applications Symposium (IDEAS'07), pages 37–45. IEEE Computer Society, Sept 2007. doi:10.1109/IDEAS.2007.4318087. 76, 77

[vSBCR02] Rini van Solingen, Vic Basili, Gianluigi Caldiera, and H. Dieter Rombach. *Goal Question Metric (GQM) Approach*. John Wiley & Sons, Inc., 2002. ISBN:9780471028956. URL: http://dx.doi.org/10.1002/0471028959.sof142, doi:10.1002/0471028959.sof142. 62

[WAB+10] Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damljanovic, Brian Davis, Norbert Fuchs, et al. On Controlled Natural Languages: Properties and Prospects. In *Proceedings of the International Workshop on Controlled Natural Language (CNL'09), Revised Papers*, pages 281–289. Springer Berlin Heidelberg, 2010. ISBN:978-3-642-14418-9. URL: http://dx.doi.org/10.1007/978-3-642-14418-9_17, doi:10.1007/978-3-642-14418-9_17. 158

[Wan09] Ping Wang. QoS-aware web services selection with intuitionistic fuzzy set under consumer's vague perception. *Expert Systems with Applications*, 36(3):4460–4466, 2009. ISSN:0957-4174. URL: http://www.sciencedirect.com/science/article/pii/S0957417408002339, doi:http://dx.doi.org/10.1016/j.eswa.2008.05.007. 141, 142, 143, 144, 146

[WCL+06] Ping Wang, Kuo-Ming Chao, Chi-Chun Lo, Chun-Lung Huang, and Yinsheng Li. A Fuzzy Model for Selection of QoS-Aware Web Services. In *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE'06)*, pages 585–593. IEEE Computer Society, Oct 2006. doi:10.1109/ICEBE.2006.3. 141, 142, 144

[WCR09] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. A Framework for Constructing Semantically Composable Feature Models from Natural Language Requirements. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*, pages 211–220. Carnegie Mellon University, 2009. URL: http://dl.acm.org/citation.cfm?id=1753235.1753265. 165, 166

[WHdM09] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. A Concurrent Portfolio Approach to SMT Solving. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 715–720. Springer Berlin Heidelberg, 2009. ISBN:978-3-642-02657-7. URL: http://dx.doi.org/10.1007/978-3-642-02658-4_60, doi:10.1007/978-3-642-02658-4_60. 4, 93

[WHR+03] Warren E. Walker, Poul Harremoës, Jan Rotmans, Jeroen P. van der Sluijs, Marjolein B. A. van Asselt, Peter Janssen, and Martin P. Krayer von Krauss. Defining Uncertainty: A Conceptual Basis for Uncertainty Management in

Model-Based Decision Support. *Integrated Assessment*, 4(1):5–17, 2003. doi:10.1076/iaij.4.1.5.16466. 89, 147

[WLH07] Hei-Chia Wang, Chang-Shing Lee, and Tsung-Hsien Ho. Combining subjective and objective QoS factors for personalized web service selection. *Expert Systems with Applications*, 32(2):571–584, 2007. ISSN:0957-4174. URL: `http://www.sciencedirect.com/science/article/pii/S0957417406000546`, doi:http://dx.doi.org/10.1016/j.eswa.2006.01.034. 141, 142

[WS03a] Yiqiao Wang and Eleni Stroulia. Flexible Interface Matching for Web-Service Discovery. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)*, pages 147–156. IEEE Computer Society, Dec 2003. ISBN:0-7695-1999-7. doi:10.1109/WISE.2003.1254478. 78

[WS03b] Yiqiao Wang and Eleni Stroulia. Semantic Structure Matching for Assessing Web-Service Similarity. In *Proceedings of the First International Conference on Service-Oriented Computing (ICSOC'03)*, pages 194–207. Springer, December 2003. ISBN:978-3-540-24593-3. URL: `http://dx.doi.org/10.1007/978-3-540-24593-3_14`, doi:10.1007/978-3-540-24593-3_14. 78

[WSB+09] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Jean-Michel Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE'09)*, pages 79–88. IEEE Computer Society, Aug 2009. doi:10.1109/RE.2009.36. 147

[WV07] Yao Wang and Julita Vassileva. A Review on Trust and Reputation for Web Service Selection. In *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pages 25–25. IEEE Computer Society, June 2007. doi:10.1109/ICDCSW.2007.16. 16

[WW05] Jian Wu and Zhaohui Wu. Similarity-based Web service matchmaking. In *Proceedings of the International Conference on Services Computing (SCC'05)*, volume 1, pages 287–294. IEEE Computer Society, July 2005. doi:10.1109/SCC.2005.93. 77, 78, 79

[WWWB11] Dengping Wei, Ting Wang, Ji Wang, and Abraham Bernstein. SAWSDL-iMatcher: A customizable and effective Semantic Web Service matchmaker. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):402–417, December 2011. ISSN:1570-8268. {JWS} special issue on Semantic Search. URL: `http://www.sciencedirect.com/science/article/pii/S1570826811000722`, doi:http://dx.doi.org/10.1016/j.websem.2011.08.001. 78, 79

[XF07] Pengcheng Xiong and Yushun Fan. QoS-Aware Web Service Selection by a Synthetic Weight. In *Proceedings of the Fourth International Conference on*

Fuzzy Systems and Knowledge Discovery (FSKD'07), volume 3, pages 632–637. IEEE Computer Society, August 2007. doi:10.1109/FSKD.2007.462. 141, 142

[YFH09] Yu Yi, Thomas Fober, and Eyke Hüllermeier. Fuzzy operator trees for modeling rating functions. *International Journal of Computational Intelligence and Applications*, 8(04):413–428, 2009. 46

[YST01] Liang Yin, Marcel A. J. Smith, and Kishor S. Trivedi. Uncertainty Analysis in Reliability Modeling. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 229–234. IEEE Computer Society, 2001. doi:10.1109/RAMS.2001.902472. 147, 148

[YT97] John Yen and W. Amos Tiao. A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 87–96. IEEE Computer Society, January 1997. doi:10.1109/ISRE.1997.566845. 81, 88, 146

[YWK+11] Edwin Yaqub, Philipp Wieder, Constantinos Kotsokalis, Valentina Mazza, Liliana Pasquale, Juan Lambea Rueda, Sergio García Gómez, and Augustín Escámez Chimeno. A generic platform for conducting sla negotiations. In *Service Level Agreements for Cloud Computing*, pages 187–206. Springer, 2011. 73

[Zad65] Lotfi A. Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965. ISSN:0019-9958. URL: `http://www.sciencedirect.com/science/article/pii/S001999586590241X`, doi:http://dx.doi.org/10.1016/S0019-9958(65)90241-X. 4, 84

[Zad78] Lotfi A. Zadeh. Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems*, 1(1), 1978. ISSN:0165-0114. URL: `http://www.sciencedirect.com/science/article/pii/0165011478900295`, doi:http://dx.doi.org/10.1016/0165-0114(78)90029-5. 85, 86

[Zad96] Lotfi A. Zadeh. Fuzzy logic = Computing With Words. *IEEE Transactions on Fuzzy Systems*, 4(2):103–111, 1996. 98

[Zad08] Lotfi A. Zadeh. Is there a need for fuzzy logic? *Information Sciences*, 178(13):2751–2779, 2008. ISSN:0020-0255. doi:10.1016/j.ins.2008.02.012. 100

[Zha98] Qiao Zhang. Fuzziness – vagueness – generality – ambiguity. *Journal of Pragmatics*, 29(1):13–31, 1998. ISSN:0378-2166. doi:http://dx.doi.org/10.1016/S0378-2166(97)00014-3. 88, 186

[ZSDS13] Andrea Zisman, George Spanoudakis, John Dooley, and Igor Siveroni. Proactive and Reactive Runtime Service Discovery: A Framework and Its Evaluation. *IEEE Transactions on Software Engineering (TSE)*, 39(7):954–974, July 2013. ISSN:0098-5589. doi:10.1109/TSE.2012.84. 78

[ZSK15] Begüm Ilke Zilci, Mathias Slawik, and Axel Küpper. Cloud Service Matchmaking Using Constraint Programming. In *Proceedings of the IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'15)*, pages 63–68. IEEE Computer Society, June 2015. doi:10.1109/WETICE.2015.44. 57, 82, 141, 142, 143, 145

[ZW95] Amy Moormann Zaremski and Jeannette M. Wing. Signature Matching: A Tool for Using Software Libraries. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 4(2):146–170, April 1995. ISSN:1049-331X. URL: `http://doi.acm.org/10.1145/210134.210179`, doi:10.1145/210134.210179. 21, 45, 247

[ZW97] Amy Moormann Zaremski and Jeannette M. Wing. Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(4):333–369, October 1997. ISSN:1049-331X. URL: `http://doi.acm.org/10.1145/261640.261641`, doi:10.1145/261640.261641. 2

Referenced Websites and Standards

[AF] appFigures, Inc. appFigures app tracking platform. Last Access: April 2016. URL: https://appfigures.com/. 8

[Amaa] Amazon. Amazon App Shop – Website. Last Access: April 2016. URL: www.amazon.de/app-shop. 12

[Amab] Amazon Web Services. Website. Last Access: April 2016. URL: aws.amazon.com. 1, 11

[App] Apple. AppStore – Website. Last Access: April 2016. URL: https://itunes.apple.com. 12

[BMWI] Bundesministerium für Wirtschaft und Energie. Industrie 4.0: Digitalisierung der Wirtschaft. Last Access: April 2016. URL: http://www.bmwi.de/DE/Themen/Industrie/industrie-4-0.html. 9

[DBBD+05] Jos De Bruijn, Christoph Bussler, John Domingue, et al. Web Service Modeling Ontology (WSMO), 2005. Last access: April 2016. URL: https://www.w3.org/Submission/WSMO/. 76

[Ecl] The Eclipse Foundation. Eclipse Marketplace. Last Access: April 2016. URL: http://marketplace.eclipse.org/. 1, 12

[Gooa] Google. Google Maps. Last Access: April 2016. URL: https://maps.google.com. 1

[Goob] Google. Google Play – Website. play.google.com/, Last Access: April 2016. URL: https://play.google.com/. 1, 12

[Gooc] Google. Google scholar. Last Access: April 2016. URL: http://scholar.google.de. 73

[IBMCM] IBM. IBM Cloud Marketplace – Website. Last Access: April 2016. URL: http://www.ibm.com/marketplace/cloud/us/en-us/. 1

[Ins] Instagram. Instagram. Last Access: April 2016. URL: https://instagram.com/. 1

[KP] Matthias Klusch and Kapahnke Patrick. Semantic Web Service Matchmaker Evaluation Environment (SME2). Last access: April 2016. URL: http://projects.semwebcentral.org/projects/sme2. 68, 76, 187

[Mica] Microsoft. Windows Phone Store – Website. Last Access: April 2016. URL: http://www.windowsphone.com/de-de/store. 1

[Micb] Microsoft. Windows Store – Website. Last Access: April 2016. URL: http://www.windowsstore.com/. 12

[Obj10] Object Management Group. UML 2.3 Superstructure, 2010. URL: http://www.omg.org/spec/UML/2.3. 48

[OMG11] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1, January 2011. URL: http://www.omg.org/spec/QVT/1.1/. 164

[OMG13] OMG. OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1, June 2013. URL: http://www.omg.org/spec/MOF/2.4.1. 37

[OSGI] OSGi Alliance. *Osgi service platform, release 3*. IOS Press, Inc., 2003. 59

[PSB+b] Marie Christin Platenius, Ammar Shaker, Matthias Becker, Eyke Hüllermeier, and Wilhelm Schäfer. Imprecise Matching of Requirements Specifications for Software Services using Fuzzy Logic – Website. Last Access: April 2016. URL: http://goo.gl/OkfNrJ. 249

[S3C] S3 Contest Organisation Committee. Annual international contest s3 on semantic service selection retrieval performance evaluation of matchmakers for semantic web services. Last Access: April 2016. URL: http://www-ags.dfki.uni-sb.de/~klusch/s3/. 76

[Sal] Salesforce.com, Inc. Salesforce AppExchange. Last Access: April 2016. URL: https://appexchange.salesforce.com/. 1, 12

[Ser] HRS Hotel Reservation Service. HRS – Hotel Reservation Service. Last Access: April 2016. URL: http://www.hrs.de/. 1

[SFB901] Paderborn University. Collaborative Research Center “On-The-Fly Computing" (CRC 901). Last Access: April 2016. URL: http://sfb901.uni-paderborn.de. 8, 12, 19, 24, 61, 121, 123, 155, 167, 178

[SI] StrikeIron. StrikeIron – Website. Last Access: April 2016. URL: http://www.strikeiron.com/. 12

[Sta] Statista, Inc. Statista Infographics. Last Access: April 2016. URL: http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. 8

[Tru] TrustRadius. TrustRadius – Software Reviews by Real Users – website. Last Access: April 2016. URL: https://www.trustradius.com/. 127, 239

[UPBa] Paderborn University. PAUL – Paderborner Assistenzsystem für Universität und Lehre. Last Access: April 2016. URL: https://paul.uni-paderborn.de. 1

[UPBb] Paderborn University. Website of the Market Demonstrator. Last Access: April 2016. URL: http://goo.gl/3tkQSi. 62, 67, 71

[UPBc] Paderborn University. Website of MatchBox. Last Access: April 2016. URL: http://goo.gl/MMCxQT. 57, 249

[UPBd] Paderborn University. Website of SSE – Service Specification Environment. Last Access: April 2016. URL: http://goo.gl/E7QjPN. 59

[UPBe] Paderborn University. Website of SeSAME – Service Specification, Analysis, and Matching Environment. ast Access: April 2016. URL: http://http://goo.gl/UBgQEb. 57

[W3C] W3C. Web services architecture. Last Access: April 2016. URL: http://www.w3.org/TR/ws-arch/. 12, 182

List of Figures

1.1 The Most Important Roles and Artifacts Involved in Service Matching 2
1.2 University Management Example Services 3
1.3 Example Requirements, Service Specification, and Ontology 4
1.4 Overview of the Solution 6
1.5 OTF Service Market and Interaction Between its Roles 8

2.1 Parts of a Service 12
2.2 Signature Specification Example 13
2.3 Example Specifications of Privacy Preferences and Properties 15
2.4 Examples for Reputation Specifications 17
2.5 Example Matching Scenario as a Sequence Diagram 18
2.6 Signature Matching Example 20
2.7 Privacy Matching Algorithm (based on [PAPS15]) 22
2.8 Overview of Aggregation Operators (based on [TN07]) 25

3.1 Abstract overviews of exemplary matching implementations 32
3.2 Exemplary simplified matching processes 36
3.3 MatchBox's metamodel for matching processes 37
3.4 Overview of MatchBox' Workflow 38
3.5 Overview of Models and Languages used during the MatchBox Workflow . . 40
3.6 Exemplary matcher definitions 41
3.7 Exemplary Template Method application within the MatchBox Framework . 43
3.8 Complete Example Process 45
3.9 Exemplary definition of an aggregation strategy 46
3.10 Process Configuration Phase 47
3.11 Matching Process Generation Overview 49
3.12 Process Execution Phase 52
3.13 Metamodel for Input Assignment 54
3.14 Overview of Interactions during the Matching Process Execution 55
3.15 Exemplary Matching Results 56
3.16 Matching Results Metamodel 57
3.17 Architecture of the MatchBox Prototype 58
3.18 Screenshot Matcher Integration 59
3.19 Screenshot Process Configuration: Matchers and Aggregation Strategies . . . 60
3.20 Screenshot Hierarchical Matching Results 61

4.1 Fuzziness Types and their Interrelations 87
4.2 Classification of Fuzziness Sources 91
4.3 Example Specifications with Annotated Fuzziness Sources 95
4.4 Temporal Order of the Origins of Potential Fuzziness Occurrences 96
4.5 Overview of the Fuzzy Matching Procedure 97
4.6 Procedure for Fuzzy Matching Based on Fuzzy Logic 101
4.7 Exemplary Fuzzy Reputation Requirements and Reputation Values 101
4.8 Example Fuzzy Sets for the Requirements Specification from Figure 4.7 . . . 102
4.9 Transformation into the Membership Function in Function 4.3 103
4.10 Example Transformation Results for Condition c3 104
4.11 Example for a Simple Bootstrapping Procedure 106
4.12 Example Fuzzy Sets for the Ratings for the Provided Services from Figure 4.7 107
4.13 Matching the Example Fuzzy Sets from Figures 4.8 and 4.12 108
4.14 Example for Calculation of p and n for Condition c4 109
4.15 Matching Results for the Ratings Constraints in c2 and c3 111
4.16 Services associated with degrees n/p to which they necessarily/possibly meet the requirements specification (based on [PSB$^+$]) 112
4.17 Examples for Varying n-p Intervals 113
4.18 Variables used to Estimate Transformation-induced Fuzziness 113
4.19 Performance Matching Example [PSB$^+$] 116
4.20 Procedure for Fuzziness Quantification as an Extension 117
4.21 Example Specifications for Fuzzy Privacy Matching 118
4.22 Exemplary Mapping from Retention Period Values to Severity Fuzziness Scores 120
4.23 Architecture of the Fuzzy Matching Prototypes 122
4.24 Screenshot of Some Fuzzy Matching Results for the Reputation Matcher . . . 123
4.25 Results of Experiment 2.1.1 (based on [PSB$^+$]) 126
4.26 Example Requirements Specifications for Experiments 2.1.1–2.1.3 [PSB$^+$] . 128
4.27 Results of Experiment 2.1.2 [PSB$^+$] 129

5.1 Matching Results Metamodel Extended with Fuzzy Matching Results 151
5.2 Matching Process Metamodel Extended with Fuzzy Guards 152
5.3 Architecture of FuzzyMatchBox 154
5.4 Problem of Understanding Matching Results and Proposed Solution 156
5.5 Transformation into an Input for Matching 159
5.6 Object Diagram View of Transformed and Matched Model Elements 161
5.7 Extended Architecture to include Back-Transformations 165
5.8 Overview of the requirements for integrating a matcher into a market [PBS14] 169
5.9 Architectural alternatives for matcher integration [PBS14] 174
5.10 Interaction with the matcher on OTF water network optimization service market 181

A.1 Main MatchBox Metamodel 234
A.2 Metamodel for Definition of Matchers and Aggregation Strategies 235
A.3 Market Specification Metamodel 236
A.4 Rule Set Metamodel 236
A.5 Metamodel for Privacy Specifications (based on [PJvR$^+$16]) 237
A.6 Metamodel for Ratings (based on [PJvR$^+$16]) 238

A.7 Metamodel for Reputation Requirements Specifications (based on [PJvR+16]) 239

B.1 Ontological Signature Matcher Definition 242
B.2 Fuzzy Condition Matcher Definition 242
B.3 Trace-Includion-based Protocol Matcher Definition 242
B.4 Privacy Matcher Definition 243
B.5 Reputation Matcher Definition 243
B.6 Weighted Averaging Aggregation Strategy 243
B.7 Minimum Aggregation Strategy 243
B.8 Operation2Interface Aggregation Strategy 244

D.1 Example Interaction between Market Components in the Waternet Example . 250

LIST OF TABLES

3.1 Exemplary Generation Rules 51
3.2 GQM Goal Definition 62
3.3 GQM Questions and Metrics 63
3.4 Exemplary Input Pairs from the University Management Example Domain . . 64
3.5 Results for Matching Process Variations for Inputs from the University Management Example 64
3.6 Results for Matching Process Variations with Input Specifications from the Image Processing Example Domain 65
3.7 GQM Goal Definition 66
3.8 GQM Questions and Metrics 66
3.9 Measured Metric Values [PSA15] 68
3.10 Comparison of Meta Matching Approaches 76
3.11 Comparison of Matching Processes in Further Matching Approaches 78

4.1 Exemplary Recommended Fuzziness Coping Strategies 99
4.2 Results for the Sets in Figure 4.13 109
4.3 Different Cases of Transformation-induced Fuzziness 114
4.4 Estimated Fuzziness Scores for the Running Example 120
4.5 Degrees of Distinguishability (Exp. 2) 127
4.6 Matching Results for Experiment 2.1.4 [PSB$^+$] 131
4.7 Statistics for Fuzziness Scores in Reputation Matching 132
4.8 Matching Results and Fuzziness Scores from Experiment 2.2 134
4.9 Matching Results and Fuzziness Scores from Experiment 2.3 135
4.10 Comparison of Fuzzy Matching Approaches: Fuzziness and Result 141
4.11 Comparison of Fuzzy Matching Approaches: Specifications and Complexity . 142

5.1 Where to integrate the matcher? [PBS14] 175
5.2 Adapted version of Table 5.1: Where to integrate the matcher in OTF markets? [PBS14] 179

A

METAMODELS

In the following, we describe the most important metamodels defining the abstract syntax of the specifications used throughout this thesis.

A.1 MatchBox Main Metamodels

Figure A.1 shows the metamodel that serves as the main data model of MatchBox. Using an instance of this model, matching processes can be created, edited, saved, and loaded. In particular, they can also be executed via interpretation at runtime. The metamodel is structured in three parts: the model elements in the green part on the left are involved during process configuration (see Section 3.7); the model elements in the blue part in the upper right corner are created when assembling input specifications (see Section 3.8.1), and the model elements in the red part in the lower right corner are created during the execution of a matching process (see Section 3.8). The blue classes and references are added when integrating fuzzy matching concepts (see Chapter 4 and Section 5.2) into MatchBox.

The left part of the model specifies the elements that are instantiated during the "Configure process" phase. In particular, there are different kinds of ProcessSteps (MatchingStep and AggregationStep) and Transitions (DataFlow and ControlFlow). MatchingStep elements refer to concrete matchers via the matcherID given in the matcher definition. In the same way, AggregationStep elements refer to aggregation strategies via the aggregationID. A MatcherConfiguration as well as a AggregationConfiguration can be specified by adding Parameter elements. In addition, AggregationConfigurations may refer to weight elements. ControlFlow transitions can contain different kinds of Guards, which refer to DataFlow transitions. The distinction between MatchingResultGuards, IntervalResultGuards, and FuzzinessScoreGuards has been introduced with fuzzy matching processes.

The classes in the upper right corner specify the InputPairs that serve as an input for the matching process at runtime. They are instantiated during the "Configure inputs" phase. InputPair elements can be enabled or not, and they hold an ontologyPath referring to the ontologies the contained specifications are associated with. An InputPair has exactly one requester and exactly one provider, both of type SpecificationCollection. A SpecificationCollection is a collection of references to external models of any type. All instantiated InputPairs and all SpecificationCollections form an InputSpecification.

For each input, a MatchingResult will be set during the execution of the matching process. A MatchingResult can consist of several children. There are different kinds of MatchingResult subclasses specifying the result value in different ways. Furthermore, a MatchingResult can contain a list of FuzzinessAnnotation elements, each of which refers to

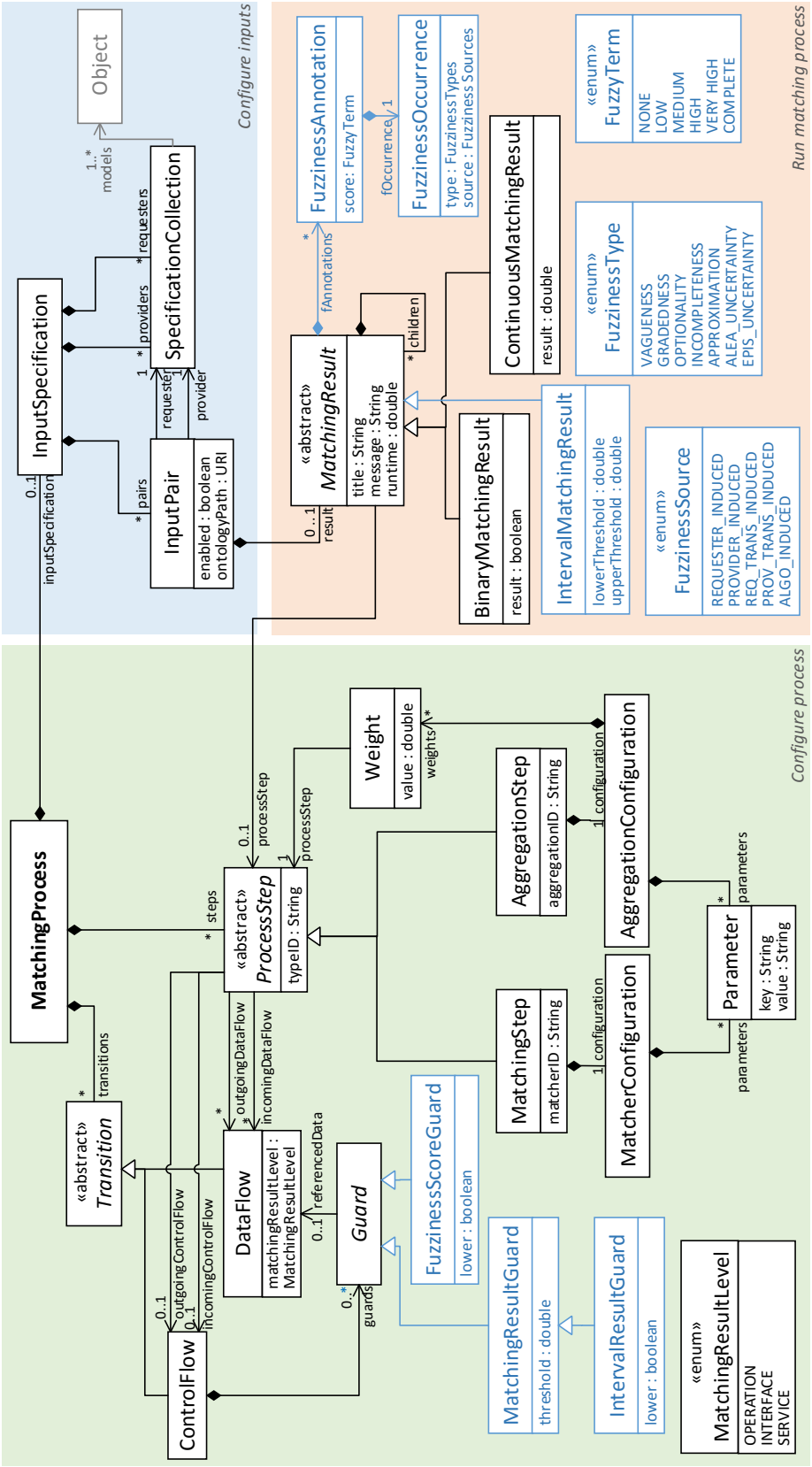


Figure A.1: Main MatchBox Metamodel

a FuzzinessOccurrence. The type and the source of a FuzzinessOccurrence as well as the score of a FuzzinessAnnotation can be selected in predefined enumerations. More subclasses can be added easily.

Figure A.2 shows the abstract syntax for matcher and aggregation strategy definitions in form of a metamodel. As this metamodel shows, both definitions are pretty similar.

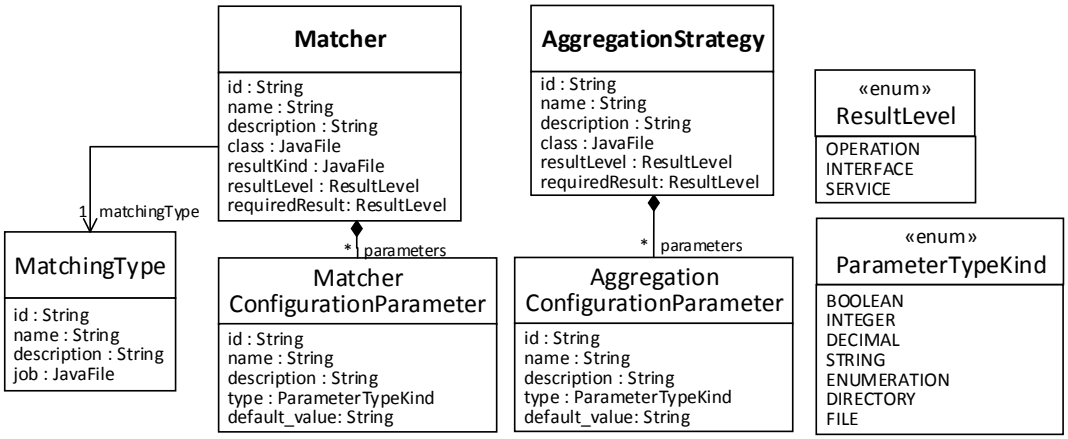


Figure A.2: Metamodel for Definition of Matchers and Aggregation Strategies

A.2 Matching Process Generator Metamodels

Figure A.3 shows the market specification metamodel. A MarketSpecification consists of an arbitrary number of MarketProperty objects. For each considered market property, there is one subclass of MarketProperty. For example, there is the class Standardization. This property can have a property within the range defined by the enumeration StandardizationValues. Each property has its own range, most often defined by an enumeration. New properties can easily be added by creating another subclass of MarketProperty. The subclass solution also allows instances, where not all properties are set. This is intended because not every user may have knowledge about every property in every market. The generation process (see Section 3.7.2) is designed to be also able to deal with such incomplete market specifications.

Figure A.4 shows the metamodel for the rules user for the matching process generation procedure. The root class is GeneratorRuleSet. It contains an arbitrary number of rules of type GeneratorRule. There are four different subclasses for different rule types: SelectionRule, MatcherConfigurationRule, ControlFlowConfigurationRule and AggregationConfigurationRule. A rule's type is important because it determines at which point in time within the generation procedure a rule is applied. Furthermore, a rule refers to an instance of the MarketProperty class from Figure A.3 and to an Operator. Depending on the rule's type, an operator can be a SelectionOperator, a ConfigurationOperator, a ControlFlowConfigurationOperator, or an AggregationConfiguationOperator. Each operator has a matcherID in order to determine which matcher is selected or configured. Here, we assume that each matcher appears only once in a matching step. We decided to identify a matcher via its ID instead of providing an enumeration of selectable matchers in order to

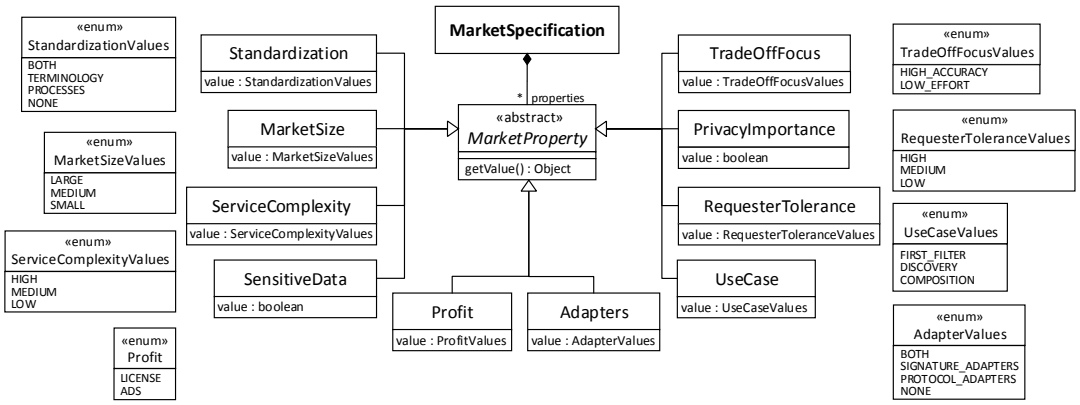


Figure A.3: Market Specification Metamodel

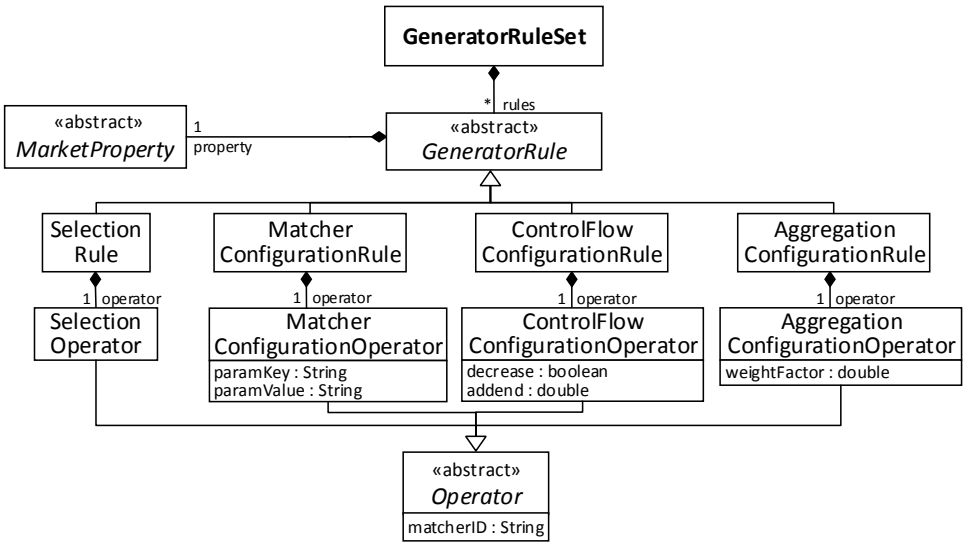


Figure A.4: Rule Set Metamodel

make the rule set metamodel applicable to different sets of integrated matchers so that it is independent of the setup phase (in contrast to its instance). For the same reason, we refer to configuration parameters within a matcher using the paramKey.

A.3 Service Specification Metamodels

In the following, we briefly describe the metamodels that define the syntax of the specification languages used for the privacy and reputation specifications from our running example throughout this thesis. All other metamodels that are mentioned can be found in our technical report [PJvR+16].

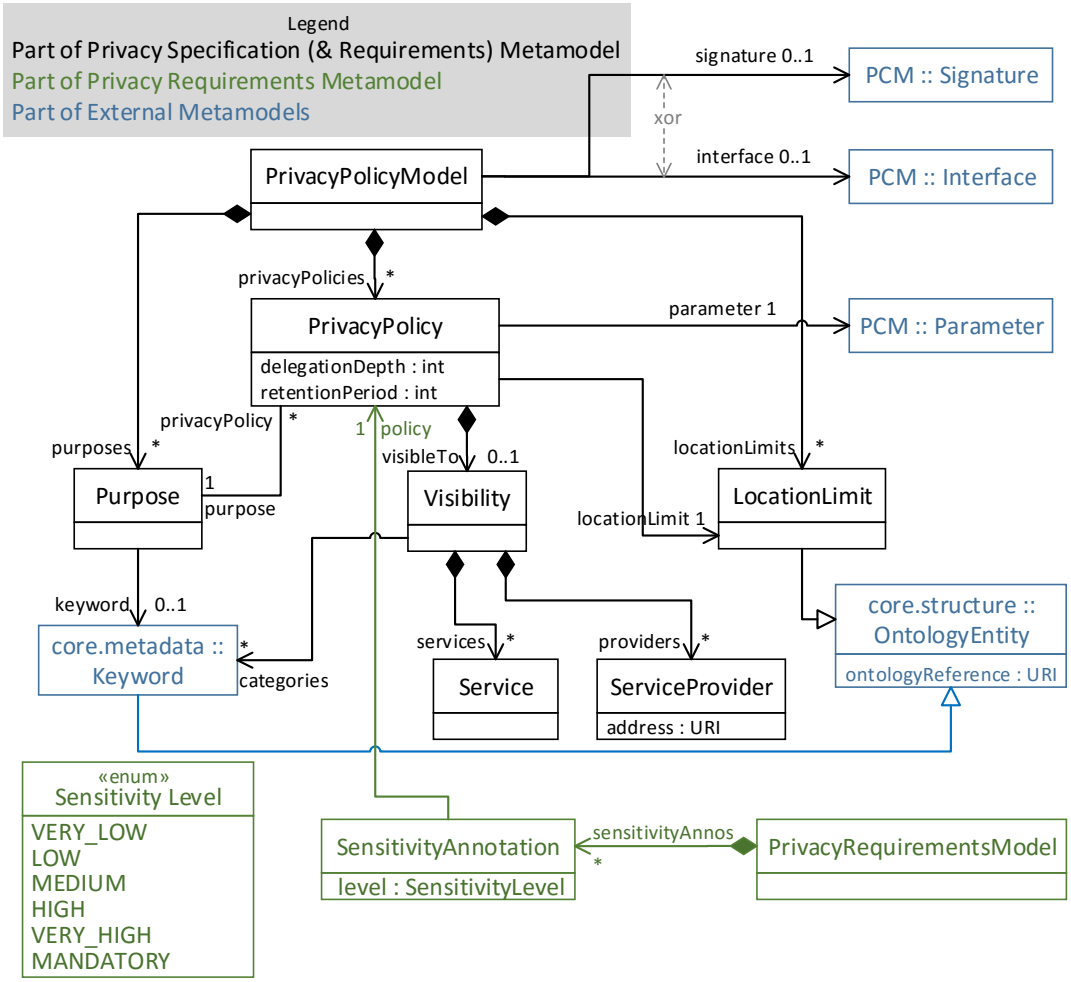


Figure A.5: Metamodel for Privacy Specifications (based on [PJvR+16])

A.3.1 Privacy Specification

Figure A.5 shows a metamodel for the privacy specifications used as examples in Chapters 2 and 4. Conceptually, the privacy metamodel is partially based on the language defined by Costante et al. [CPZ13a].

A PrivacyPolicyModel contains an arbitrary number of PrivacyPolicy objects. There may be one or more elements of type PrivacyPolicy per Parameter of the Interface or the Signature the privacy specification is associated with. Here, the interfaces and all its constituent parts are specified using the palladio component model (PCM) [BKR09, RBH+07]. Alternatively, also other specification languages like WSDL (Web Service Description Language) could be used for this purpose. For a clearer view, Figure A.5 abstracts from some technical details. For example, all model elements implement a name and an ID attribute from PCM's Entity interface.

Each PrivacyPolicy specifies the two integers delegationDepth and retentionPeriod. Furthermore, it can refer to a Purpose element, which refers to a Keyword. A Keyword is an OntologyEntity and, thus, references an ontological concepts via an ontologyReference.

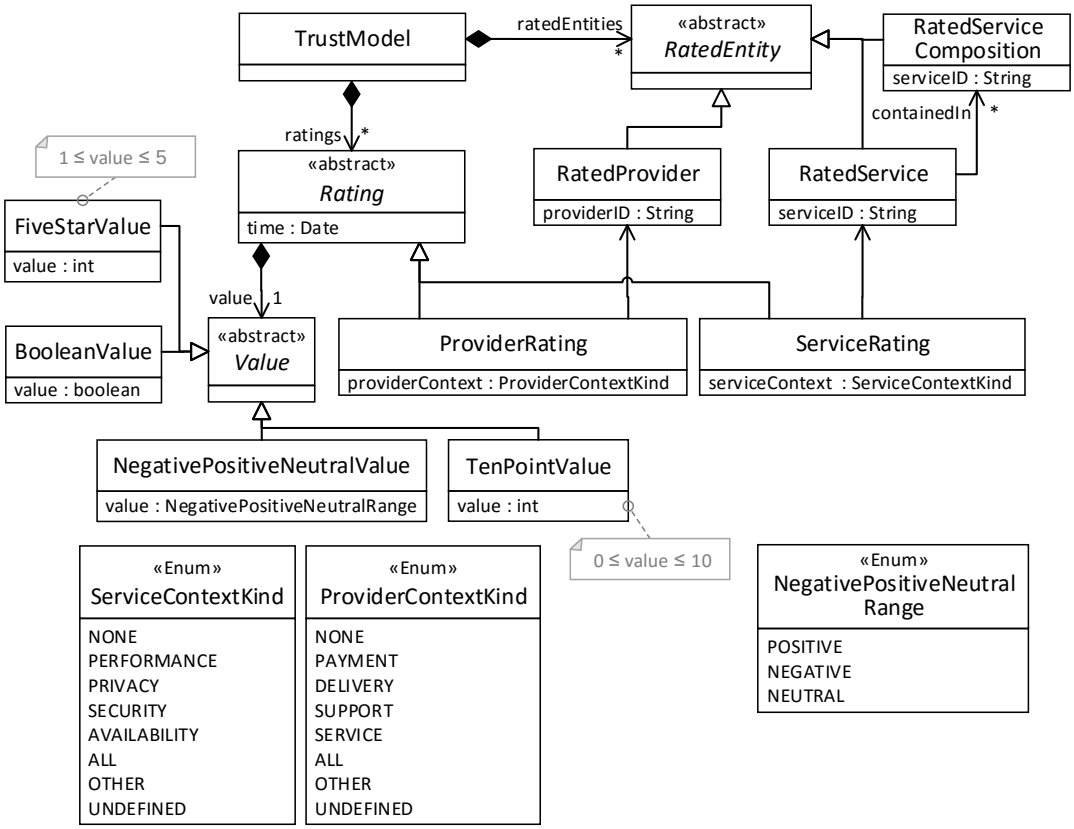


Figure A.6: Metamodel for Ratings (based on [PJvR+16])

Furthermore, a PrivacyPolicy refers to the class Visibility. Visibility an be specified using either (service) categories based on Keyword elements, using Service elements, or using ServiceProvider elements. Last but not least, a PrivacyPolicy refers to a LocationLimit, which is also an OntologyEntity.

The same metamodel can also be used to model privacy requirements. For this purpose, a PrivacyRequirementsModel is created. Such a model contains a number of SensitivityAnnotation elements, each of one can be associated with a PrivacyPolicy.

Another description of our privacy specification language including a more technical view and more examples can be found in our paper [PAPS15] and in our technical report [PJvR+16].

A.3.2 Reputation Specification

The reputation of a service describes how a service has been rated by previous users in the past. Usually reputation data is stored in a a reputation system. Following related work, we realized service's reputation as an aggregated value based on a set of user ratings stored in a "trust model" [Ban14]. This model captures trust towards different entities in form of ratings. Figure A.6 shows the metamodel we use to model and store ratings for our examples in Chapters 2 and 4.

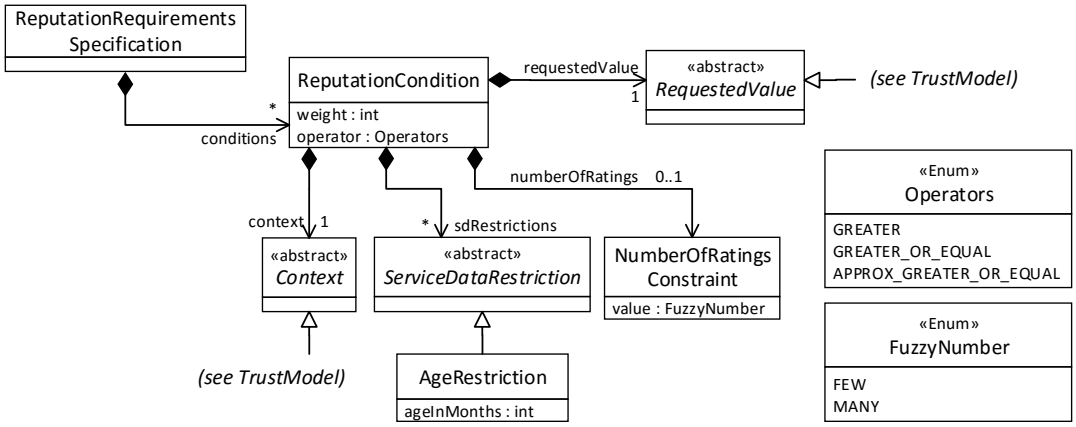


Figure A.7: Metamodel for Reputation Requirements Specifications (based on [PJvR+16])

A Trustmodel consists of an arbitrary number of Ratings. Each Rating is either a ProviderRating, i.e., a rating of a service provider, or a ServiceRating, i.e., a rating of a service itself. A ProviderRating always refers to a RatedProvider identified by a providerID. A ServiceRating, in contrast, refers to a RatedService identified by a serviceID. For future extensions regarding reputation of service compositions, we already prepared the class RatedServiceComposition which contains an arbitrary number of RatedService elements. This class is not used for our current approaches, though. RatedEntity

Both, ProviderRating and ServiceRating refer to enumerations of different context elements. This way, we can model ratings for a certain characteristic of a service or a provider, e.g., a service's performance, or a provider's support.

Each Rating has a time, defining the exact time the rating has been given. Furthermore, it has a value which can be specified by one of four different ranges: A FiveStarValue is an integer between 1 and 5 as common in today's app stores, a BooleanValue represents a "like" or "did not like" statement, aNegativePositiveNeutralValue comes with an additional neutral value, and a TenPointValue is an integer between 0 and 10 like the ratings given by TrustRadius [Tru]. Internally, all values can be transformed into the same range before matching.

The whole model is designed to be extensible. For example, new ranges, new rated entities, or new context kinds can easily be added.

Requirements specifications in our examples are based on the metamodel depicted in Figure A.7. A ReputationRequirementsSpecification consists of an arbitrary number of ReputationCondition elements. A ReputationCondition contains a RequestedValue. Like in Figure A.6, different ranges can be supported using inheritance. For example, we could use the same or similar subclasses as in the trustmodel. The same holds for a ReputationCondition's Context. Furthermore, each ReputationCondition has a weight and an operator. Optionally, the condition refers to elements of type ServiceDataRestriction and to a NumberOfRatingsConstraint. ServiceDataRestriction are specified using subclasses, e.g., the AgeRestriction, which specifies the maximum age in months a rating may have to be considered.

B

Integrated Matchers and Aggregation Strategies

In the following, we show the matcher definitions of some of the matchers we exemplarily integrated into the MatchBox framework as well as aggregation strategies.

B.1 Matcher Definitions

Figure B.1 shows the matcher definition of the ontological signature matcher. This definition has already been described in Section 3.6.1. In addition, we added some more matcher parameters: The parameter Parameter_Types enables the matching designer to switch on and off the functionality to match types. Input_Parameters and Output_Parameters allow the matching designer to determine whether only input or also output parameters or both should be considered during matching. Exceptions determines whether an operation's exception is taken into account.

Figure B.2 shows the matcher definition of an exemplary condition matcher based on the Bachelor's Thesis by Boerding [Boe15]. The depicted matcher delivers continuous operation level results. The matcher internally calls an SMT solver for matching conditions, if it cannot manage the condition on its own, due to the condition's complexity. Using the parameter SMTSolvingThreshold, the matching designer can determine whether the SMT solver should always be called (=0.0) or only for intermediate matching results above the given threshold. Furthermore, this matcher comes with multiple matching strategies that can be switched using the Strategy parameter. For functioning, the matcher needs continuous operation level results.

The protocol matcher's definition depicted in Figure B.3 has already been explained in Section 3.6.1.

Figure B.4 defines a privacy matcher, like the one described in Section 2.4.3, and Figure B.5 describes a reputation matcher, like the one described in Section 2.4.4.

B.2 Aggregation Strategy Definitions

The Weighted Averaging Aggregation Strategy and its definition depicted in Figure B.6 has already been explained in Section 3.6.2. Figure B.7 and B.8 show two additional aggregation strategies: a minimum aggregation strategy and the Operation2Interface Aggregation Strategy addressed in Section 3.6.2.

Name:	***Ontological Signature Matcher***
Id:	*matcher.signatures.ont*
Description:	*The ontological signature matcher leverages the relations specified in a domain ontology. Using these relations, the matcher checks for covariance and contravariance in order to decide whether two signatures comply wrt. parameter types. The matcher is highly configurable such that also other parts of the signature can be considered, e.g., operation names or exceptions.*
Type:	*matching.signatures*
Result Format:	*continuous [0,1]*
Result Level:	*operation*
Parameters:	*Operation_Names : boolean (default = false)* *Parameter_Names : boolean (default = false)* *Parameter_Types : boolean (default = true)* *Input_Parameters : boolean (default = true)* *Output_Parameters : boolean (default = true)* *Exceptions : boolean (default = false)*
Required Result:	-
Class:	*de.upb.crc901.matchers.OntologicalSigMatcher*

Figure B.1: Ontological Signature Matcher Definition

Name:	***Quick Condition Matcher***
Id:	*matcher.condition.quick*
Description:	*Matches pre- and postconditions specified in a FOL-based language and returns a fuzzy matching result. Needs signature matching results to run.*
Type:	*matching.condition*
Result Format:	*continuous*
Result Level:	*operation*
Parameters:	*SMTSolvingThreshold : double (default = 0.0)* *Strategy : enum {SIMPLE, COMPLEX, DC} (default = SIMPLE)*
Required Result:	*Level: operation; Format: continuous*
Class:	*de.upb.crc901.matchers.QuickConditionsMatcher*

Figure B.2: Fuzzy Condition Matcher Definition

Name:	***Trace-Inclusion-based Protocol Matcher***
Id:	*matcher.protocols.ti*
Description:	*This matching approach checks if all the traces of the requested service find a matching trace in the provided protocol.*
Type:	*matching.protocols*
Result Format:	*continuous*
Result Level:	*interface*
Parameters:	*RollOut_Threshold : integer (default = 5)*
Required Result:	*Level: operation; Format: continuous*
Class:	*de.upb.crc901.matchers.TIProtocolMatcher*

Figure B.3: Trace-Includion-based Protocol Matcher Definition

Name:	***Privacy Matcher***
Id:	*matcher.privacy*
Description:	*Checks whether the privacy specification of the provided service is at least as strict as the given privacy preferences based on several properties, e.g., retention time, delegation depth.*
Type:	*matching.privacy*
Result Format:	*continuous*
Result Level:	*interface*
Parameters:	*GradualResult : boolean (default = true)*
Required Result:	*Level: operation, Format: continuous*
Class:	*de.upb.crc901.matchers.DefaultPrivacyMatcher*

Figure B.4: Privacy Matcher Definition

Name:	***Traditional Reputation Matcher***
Id:	*matcher.reputation.trad*
Description:	*Matches reputation requirements to ratings stored in a TrustModel in a strict way: Each required condition can either be a full match or a mismatch; results per condition are aggregated for the final result.*
Type:	*matching.reputation*
Result Format:	*continuous*
Result Level:	*service*
Parameters:	-
Required Result:	-
Class:	*de.upb.crc901.matchers.reputation.TraditionalReputationMatcher*

Figure B.5: Reputation Matcher Definition

Name:	***Weighted Averaging Aggregation Strategy***
Id:	*aggregation.weighted_average*
Description:	*This aggregation strategy computes the mean value of all incoming matching results under consideration of weights assigned to each matching step.*
Result Format:	*continuous [0,1]*
Result Level:	*interface*
Parameters:	*"weight" : double (default = 1.0)*
Required Result:	*Level: interface; Format: continuous [0,1]*
Class:	*de.upb.crc901.aggregation.strategies.WeightedAggregationStrategy*

Figure B.6: Weighted Averaging Aggregation Strategy

Name:	***Minimum Aggregation Strategy***
Id:	*aggregation.minimum*
Description:	*This strategy computes the minimum value of all incoming matching results.*
Result Format:	*continuous [0,1]*
Result Level:	*interface*
Parameters:	-
Required Result:	*Level: interface; Format: continuous*
Class:	*de.upb.crc901.aggregation.strategies.MinimumAggregationStrategy*

Figure B.7: Minimum Aggregation Strategy

Name:	***OperationResultLevel2InterfaceResultLevel Aggregation***
Id:	*aggregation.operationlevel2interfacelevel*
Description:	*This aggregation strategy takes an arbitrary number of matching results on operation level and aggregates them into a matching result on interface level. For this purpose, it tries to create an injective mapping.*
Result Format:	*continuous [0,1]*
Result Level:	*operation*
Parameters:	*-*
Required Result:	*Level: interface; Format: continuous*
Class:	*de.upb.crc901.aggregation.strategies.OperationResult2InterfaceResultAggregationStrategy*

Figure B.8: Operation2Interface Aggregation Strategy

C

LITERATURE SURVEYS

In the following, we give some more information on the conduction of our literature surveys presented in Sections 3.12 and 4.10.

C.1 Keywords

In the following, we list the complete keywords used to select literature as explained in Section 3.12 and Section 4.10. These keywords have been identified during our primary survey [PvDB+13] and refined during later reviews. These refinements were due to our refined definitions of terms like matching and fuzziness and they lead to the fact that we came to another number of publications to be reviewed than in our primary survey.

Service/Component Keywords
The following set of keywords has been used to identify literature within the most relevant research areas for matching of software entities, e.g., Service-Oriented Computing and Component-Based Software Engineering. At least one of the following keywords had to appear in the publication's title in order to potentially included for further review.

- Service (including Web Service and Cloud Service)
- Component
- COTS (Commercial-off-the-Shelf Components)
- Interface
- Software (including Software agents)

Matching Keywords
The following set of keywords has been used to identify literature about matching of software entities. At least one of the following keywords had to appear in the publication's title in order to potentially included for further review. Furthermore, the keyword had to refer to one of the Service/Component keywords listed above.

- Matcher / Matching / Match
- Matchmaker / Matchmaking
- Selection / Select
- Discovery / Discover
- Ranking / Rank
- Retrieval / Retrieve
- Search / Searching

- Find / Finding

MatchBox Keywords
The following set of keywords has been used to identify literature related to MatchBpx. At least one of the following keywords had to appear in the publication's abstract in order to be potentially included for further review. Furthermore, the keyword had to refer to one of the Matching keywords listed above. As explained in Section 3.12, many publications included because of such keywords were often excluded after a fulltext review as they turned out to be not related to MatchBox for reasons like only considering a single matching approach.

- Framework / Engine
- Combines / Combined
- Process
- Multiple Criteria / Different Types of Requirements
- Personalized
- Custom / Customizable
- Discovery Components
- Steps
- Variants
- Hybrid
- Filtering / Filters
- Rich Descriptions
- Flexible
- Extensible
- Strategies / Assessment Methods

Fuzziness Keywords
The following set of keywords has been used to identify literature within the area of fuzzy matching. At least one of the following keywords had to appear in the publication's abstract in order to be potentially included for further review. Furthermore, the keyword had to refer to one of the Matching keywords listed above. The keywords' relevance decreases from the first to the last items in the list. As explained in Section 4.10, publications included because of the keywords "Relaxed / Relaxing", "Partial Match", and "Flexible / Flexibility" were often excluded after a fulltext review as they turned out to be not relevant for our definition of fuzzy matching.

- Fuzzy / Fuzziness
- Uncertain / Uncertainty
- Vague / Vagueness
- Imprecise / Imprecision
- Approximate /Approximation
- Probabilistic
- Incompleteness
- Imperfect Information

- Missing Information
- Estimate / Estimation
- Relaxed / Relaxing
- Flexible / Flexibility
- Soft

C.2 Exclusion Criteria

- Service composition: A couple of publications address both the service matching and the service composition problem within one step by applying one-to-many or many-to-many matching. These approaches most often do not work with matching results but directly insert a service into a service composition. Furthermore, service composition is future work of this thesis and we focus on one-to-one matching for now. Thus, we excluded these approaches.
- Literature written in other languages than English: in order to achieve high repeatability, only papers written in English are considered.
- Publications with the length of less than 6 pages: in order to achieve a high validity, literature sources which have less than six pages are excluded as they cannot present enough details to be interesting for our survey.
- Literature unavailable via common digital libraries: in order to achieve high repeatability, only papers that are available in common digital libraries are considered. This should also ensure a certain degree of quality.
- Old literature: Only literature published within the last 20 years (year 1995 and newer) was considered. We chose such a long time span because we also wanted to consider approaches that developed before the "semantic web wave" in order to get very different viewpoints influenced by different technologies and research trends. Furthermore, 1995 seemed to be a good threshold because the most cited specification-based matching approaches including Zaremski's and Wing's approach [ZW95] and OTSO [Kon95] which are considered important milestones in COTS selection [MRE07] were published in 1995. Moreover, we assume that valuable research results from earlier publications are also reflected in the more recent approaches.
- (Only for matching processes survey:) In order to find approaches that are related to MatchBox, we excluded all approaches that only consider one single matching approach.
- (Only for fuzzy matching survey:) Ambiguous fuzziness keywords: A few papers have been excluded after manual inspection revealed that they do not contain any of the fuzziness types we defined. Examples for fuzziness keywords that lead to such papers are "partial match(ing)", "relaxed match(ing)", and "flexible match(ing)"/"flexibility".

D

EVALUATION DATA

In the following, we describe how to access all data that is utilized for the case studies within this thesis.

D.1 Validation of Matching Processes

The MatchBox prototype used for the case studies within this thesis can be downloaded via our Eclipse UpdateSite as describe on the MatchBox Website [UPBc]. The specifications used within the scope of the case studies are accessible within a MatchBox installation as Eclipse Example Projects (with one exception, see Section D.2). The specifications include requirements specifications, service specifications, and matching processes with matching results. Furthermore, all mentioned matchers have been integrated into the available MatchBox prototype.

D.2 Validation of Fuzzy Matching

Case Study 2.1 is part of our paper [PSB$^+$]. For more detailed data, refer to the paper's website [PSB+b]. The specifications used within the scope if Case Study 2.2 are accessible within a MatchBox installation as described in Section D.1.

D.3 Validation of Market Integration

An example interaction for the market integration application described in 5.4.3 is depicted in Figure D.1.

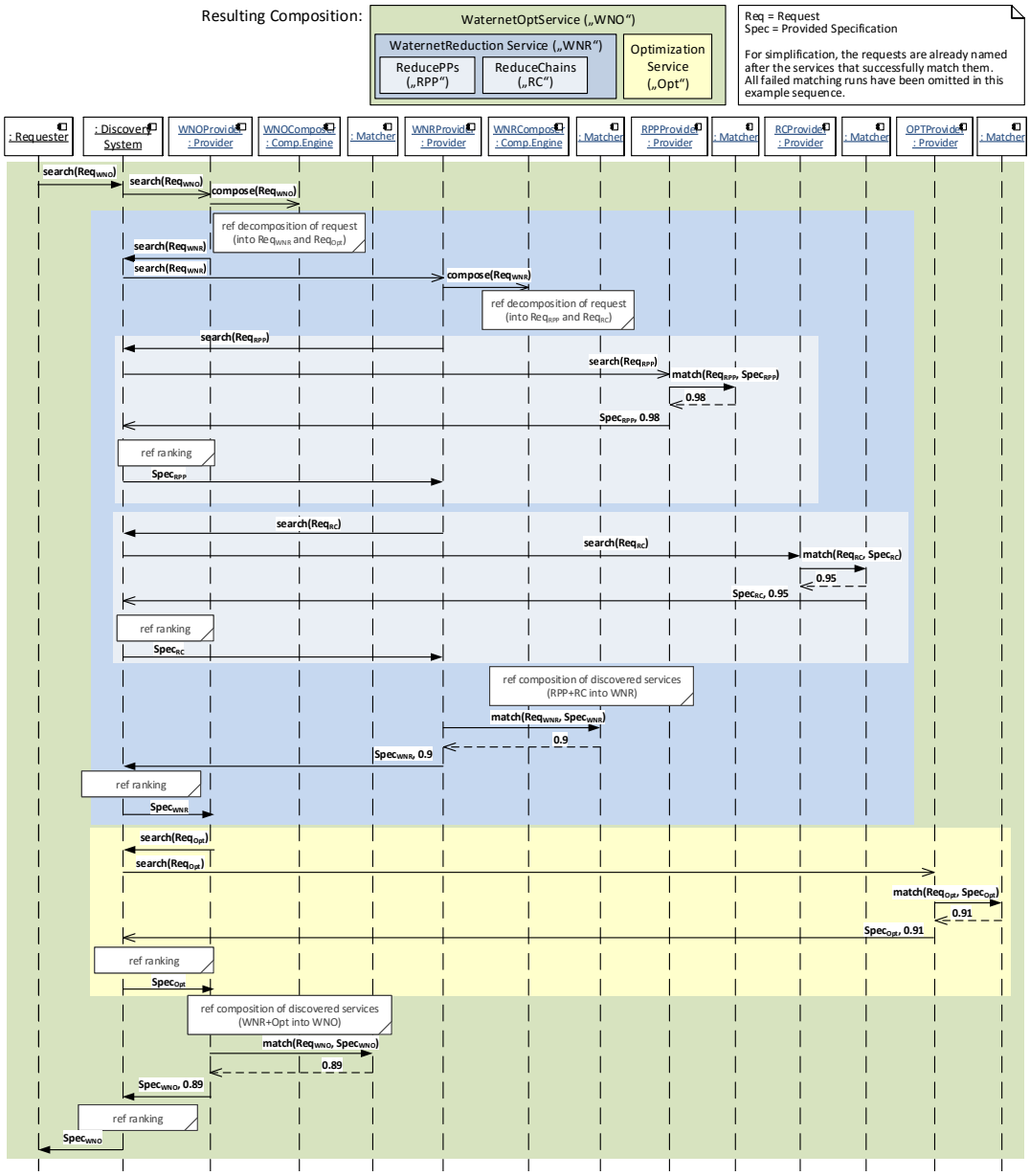


Figure D.1: Example Interaction between Market Components in the Waternet Example