Comprehensive Service Matching with MatchBox

Paul Börding
Heinz Nixdorf Institute
University of Paderborn,
Germany
boerding@mail.upb.de

Melanie Bruns
Heinz Nixdorf Institute
University of Paderborn,
Germany
mbruns42@mail.upb.de

Marie C. Platenius
Heinz Nixdorf Institute
University of Paderborn,
Germany
m.platenius@upb.de

ABSTRACT

Nowadays, many service providers offer software components in the form of Software as a Service. Requesters that want to discover those services in order to use or to integrate them, need to find out which service satisfies their requirements best. For this purpose, service matching approaches determine how well the specifications of provided services satisfy their requirements (including structural, behavioral, and non-functional requirements). In this paper, we describe the tool-suite MatchBox that allows the integration of existing service matchers and their combination as part of flexibly configurable matching processes. Taking requirements and service specifications as an input, MatchBox is able to execute such matching processes and deliver rich matching results. In contrast to related tools, MatchBox allows users to take into account many different kinds of requirements, while it also provides the flexibility to control the matching process in many different ways.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.13 [**Software Engineering**]: Reusable Software; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

Keywords

Service Discovery, Service Matching, Matching Processes, Fuzzy Matching, Software Requirements, Framework

1. INTRODUCTION

Nowadays, more and more service providers offer software components in the form of deployed, ready-to-use services (Software as a Service) [11]. In order to benefit from these services (e.g., by using them standalone or integrating them as part of a service composition), service requesters need to discover those services that satisfy their requirements best. This service discovery is often performed by brokers that act as intermediaries between requesters and providers. In order to discover services, brokers apply *Service Matching* approaches that determine whether the specification of a provided service satisfies the requester's requirements.

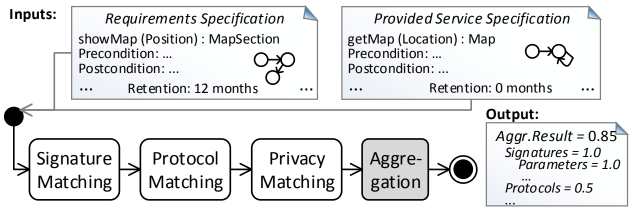


Figure 1: Matching Process Example

As such requirements can be both functional (e.g., "I want a service able to show maps.") and non-functional (e.g., "The service should be fast."), service matching approaches need to be comprehensive, i.e., be able to consider many different kinds of properties [3, 4, 5]. For example, matching approaches may focus on specifications of structural properties (e.g., signatures), behavior (e.g., pre- and postconditions or protocols), or non-functional properties (e.g., performance or privacy-related properties). Extracts of some simplified example specifications are shown in Fig. 1. In order to allow comprehensive matching of all these properties, in our earlier work, we proposed concepts for our framework MatchBox [13]. In particular, these concepts enable brokers to reuse and to combine matching approaches as part of comprehensive *matching processes*.

Figure 1 shows a simplified version of an exemplary matching process. In this example, the requirements for a Map Service are to be matched with the specification of a provided Map Service (e.g., Google Maps, OpenStreetMaps, ...). For this purpose, three exemplary matching steps have been combined into a process: Signature Matching, Protocol Matching, and Privacy Matching. A subsequent aggregation step aggregates the results of the single matching steps into a final result using an aggregation strategy, e.g., the weighted average. The final matching (here: 0.85) result is constructed hierarchically from child results and denotes how well the provided Map Service satisfies the given requirements: 0 means it does not match at all, 1 means it is a full match. Note that, matching processes may contain more steps, e.g., performance matching or price matching. Furthermore, the depicted matching process is simplified and several details are omitted for presentation purposes, e.g., the specific configurations per step and data flow.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2803181

Such matching processes can become complex but they offer many benefits. For example, we get more accurate matching results because we can consider many different kinds of requirements of a service. At the same time, matching process models provide us with an abstraction that is not only more flexible but also easier to handle compared to the amount of integration code we would normally need to integrate different matching steps. However, in order to make the concept of matching processes useful with low effort, we need appropriate tool support in order to create, configure, and execute matching processes.

In this paper, we describe a tool-suite that realizes these concepts: A requester's requirements and service specifications from multiple domains (e.g., Tourism, University Management, Image Processing,...) can be specified as an input, different instances of a matching process can be modeled, new matchers as well as aggregation strategies can be integrated. For a given matching process, the MatchBox tool-suite is able to validate and execute this process by interpretation of the underlying matching process model. The matching results are presented in a hierarchical form, providing rich feedback for service providers, service requesters, and brokers. As working with comprehensive input specifications raises the need for handling fuzzy requirements or incomplete specifications [2, 15], MatchBox also supports Fuzzy Matching [12, 14].

Existing matching approaches and their tools only support matching one or few kinds of requirements [14, 5], e.g., signatures. Also, these tools do not provide the flexibility to control the matching order and aggregation.

The benefit of our MatchBox tool-suite is its provision of a flexible environment that can be applied to various matching problems. The target applications are service discovery and component retrieval but also other kinds of software requirements can be matched using MatchBox.

This paper is structured as follows. In the next section, we give a brief overview of the architecture we used to realize MatchBox. In Section 3, we present how we applied MatchBox as part of a research project. Section 4 summarizes related tools and compares them to MatchBox and Section 5 draws conclusions. A screencast accompanying this paper can be found online [17].

2. OVERVIEW AND REALIZATION

Figure 2 shows the basic architecture of the MatchBox Framework including the components it consists of. It is build up using the Model View Controller paradigm. In the following, we briefly describe the constituent components and their purpose.

The `Model` contains three main components: the `MatchingProcess` model, the `InputSpecifications` model, and the `MatchingResult` model. The `MatchingProcess` consists of several settings and data that determine how the `MatchingProcess` will be executed. The `InputSpecifications` represent the requested requirements and provided service specifications that are matched to each other during the execution of the `MatchingProcess` (e.g., signatures or privacy preferences for the required map service). The `MatchingResult` is created during the execution of the `MatchingProcess` and represents final, aggregated matching results as well as the sub results that emerged from single steps that are part of the process.

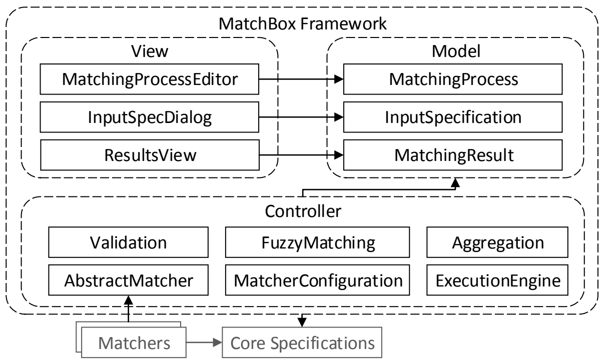


Figure 2: Simplified Architecture Overview

The `Controller` offers several features to manage, adapt, and execute a modeled `MatchingProcess`. Every part of the `Model` can be validated before the matching is performed using the `Validation` component. The `FuzzyMatching` component provides available Fuzzy Matching options in order to improve the matching process and possibly its results in the presence of uncertainties. For example, if the information on the provided service is incomplete or if the requirements are vague. The `Aggregation` component provides the possibility to add aggregation strategies that aggregate the results of the matching steps to the final matching result. For example, if it is more important that a service's signatures match the requirements, than that the privacy properties match, the signatures matching can receive a higher weight. The `AbstractMatcher` provides template methods which can be used to integrate `Matchers` into the framework. This leads to the advantage, that the framework can be extended with new matchers and configurations making it possible to combine advantages of multiple matchers. The `MatcherConfiguration` contains the configuration for every matcher that is used for the process. The `ExecutionEngine` is responsible for the interpretation of a modeled matching process in order to execute it.

The `View` provides components that allow users to display and edit all model parts. Using the `MatchingProcessEditor` (shown in the main part of Figure 3), the `MatchingProcess` is created and modified; the `InputSpecification` that a matching process matches at runtime is assembled within the `InputSpecDialog`; the corresponding `MatchingResults` are shown in the `ResultsView` (shown at the bottom Figure 3) after the execution of a process.

The MatchBox tool-suite is based on the core part of a predefined specification language (`Core Specifications`) used to describe the elements to be matched such as signatures or pre- and post-conditions. The `Matchers` that can be integrated into the framework need to be based on a specification language. MatchBox is independent from this specification language as soon as it provides specific core features.

MatchBox is part of SeSAME (Service Specification, Analysis, and Matching Environment) [1]. It is built as a number of Eclipse plugins using the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF). In particular, MatchBox utilizes the Eclipse extension point mechanism to enable the integration of new matchers and their configuration, input specification types, and aggregation strategies for the results. This makes MatchBox extensible and versatilely applicable. Download information and more details about the technical realization can be found on the MatchBox Website [19].

3. EXAMPLE APPLICATION

In the Collaborative Research Centre 901 "On-The-Fly Computing" [18], service matching is used to find suitable IT services for various kinds of customers. These services are composed from modular software components.

Within the scope of this research center, we integrated eight different matchers. For illustration, we introduce three of our matchers in the following. The **Ontological Signature Matcher** allows us to match the structure of service operations by means of their signatures. This matcher compares all signatures of the requested specification with all the signatures of the provided specification based on their input and output parameters. Protocol matching allows us to consider the interaction a service provides. Our **Path-based Protocol Matcher** compares protocols by calculating how many traces in the required protocol are covered by the provided protocol. We also added some matchers for non-functional properties. For example, privacy is an important issue when looking for external software in particular. Our **Privacy Matcher** compares privacy-related requirements to privacy policies that are part of the provided specifications. Examples for the privacy-related requirements this matcher considers are the allowed delegation depth or the retention time. In a case study [13], we showed that the integration effort for one matcher takes about half an hour, depending on the developer's experience with the Eclipse framework and MatchBox itself.

These three matchers have been combined to a matching process using the MatchBox tool-suite as shown in the screenshot in Figure 3. We also added an aggregation step that is used to apply a weighted averaging strategy. Using MatchBox, we are now able to validate and execute this process fully automatically. Alternatively, we can adapt it in various ways, e.g., by adding more matching steps, configuring matching steps, or changing the aggregation strategy.

As part of the research center, we performed several discoveries for services in several domains. In one example domain, services are composed to a university management service. These services include a room reservation service, a course registration service, and an exam management service. For this purpose, different kinds of requirements have been specified and matched to a collection of service specifications representing services in the market. Example matching results received after matching the specifications for the room reservation services using the depicted matching process can be viewed at the bottom of Figure 3. Here we can see MatchBox' hierarchical results, allowing the customer to trace how the results were calculated. In case of uncertainties, e.g., incomplete specifications, Fuzzy Matching has been applied and the view shows how much fuzziness was induced. Depending on the specifications' complexity, the shown matching process takes about 10 milliseconds to be finished for one service. This runtime increases linearly with the amount of services to be matched.

4. RELATED TOOLS

There exists a large body of work in the area of service matching and many matchers are already implemented. MatchBox applies at a meta level, providing ways to leverage and to combine existing matchers. Thus, it does not directly compete with these matchers. However, there are more benefits of MatchBox compared to related tools.

One striking difference between our work and the current state of the art in matching is the amount of detail in the matched service specifications. Among the aforementioned matchers, none of them can handle detailed specifications of privacy, reputation, and protocols. Furthermore, current matchers only return a list of services (e.g., [9], [8]), often ranked by their matching results. While some add annotations [6], only MatchBox offers hierarchical matching results, allowing users to understand, how results came into being. Users cannot only trace all results of the individual matching step, like signature matching, but they can even check which specific signatures do or do not match. Along with the result itself, the fuzziness grade and its source are stated.

While MatchBox's flexible matching processes offer the possibility to integrate multiple different aggregation strategies, many matchers use just one. Some matchers use adaptive aggregation, e.g., [7], [8]. Instead of letting brokers choose the aggregation, the matcher learns the optimum aggregation strategy. This adaptive aggregation might be a useful selectable aggregation strategy for future development. However, offering only one aggregation strategy does not allow brokers to specialize for separate domains or very demanding customers.

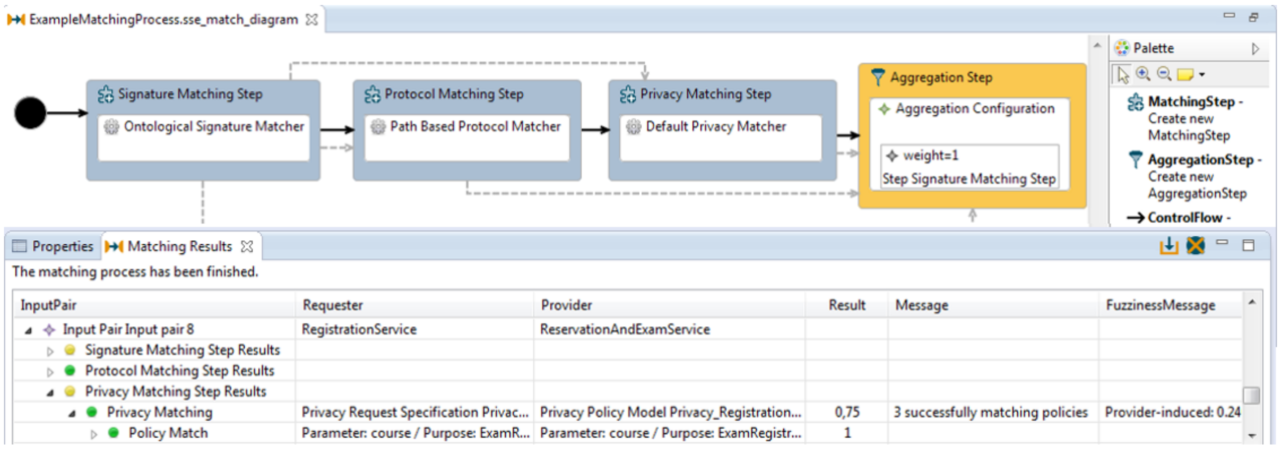
In addition to comprehensiveness and flexibility, in contrast to other tools, MatchBox also adds new features for Fuzzy Matching. For example, while quite a few matchers are able to deal with incomplete specifications (e.g., [10]) and some are capable of using approximations (e.g., [8]), they do not reflect this in their matching results. Both LARKS [16] and WSMO-MX [6] use fuzzy logic for matching of single aspects, but do not present any fuzziness values to the user. MatchBox enables fuzzy matching of different kinds of requirements and clearly shows the source and the amount of fuzziness in its results.

5. CONCLUSIONS

In this paper, we described the tool-suite MatchBox that allows the integration of existing service matchers and their combination as part of flexibly configurable matching processes. Taking requirements and service specifications as an input, MatchBox is able to execute modeled matching processes and deliver rich matching results that serve as feedback for service providers and service requesters. We applied MatchBox by integrating eight different matchers, creating different matching processes, and running them on specifications of requirements and services from different domains.

In contrast to related tools, MatchBox allows the user to take into account many different kinds of requirements and properties, while it also provides the flexibility to control the matching process in multiple ways. In general, the MatchBox tool-suite provides a flexible environment that can be applied to various matching problems. Even though the target applications are service discovery or retrieval of software components in general, in fact, also other kinds of software requirements can be matched using MatchBox due to the extensibility and substitutability of integrated matchers.

In the future, we want to extend MatchBox by introducing the idea of self-adaptive matching processes that optimize


ExampleMatchingProcess.sse_match_diagram
Signature Matching Step
Ontological Signature Matcher
Protocol Matching Step
Path Based Protocol Matcher
Privacy Matching Step
Default Privacy Matcher
Aggregation Step
Aggregation Configuration
weight=1
Step Signature Matching Step
Palette
MatchingStep - Create new MatchingStep
AggregationStep - Create new AggregationStep
ControlFlow -
Properties
Matching Results

The matching process has been finished.

InputPair	Requester	Provider	Result	Message	FuzzinessMessage
Input Pair Input pair 8	RegistrationService	ReservationAndExamService			
Signature Matching Step Results					
Protocol Matching Step Results					
Privacy Matching Step Results					
Privacy Matching	Privacy Request Specification Privac...	Privacy Policy Model Privacy_Registration...	0,75	3 successfully matching policies	Provider-induced: 0.24
Policy Match	Parameter: course / Purpose: ExamR...	Parameter: course / Purpose: ExamRegistr...	1		

Figure 3: Screenshot of the MatchBox tool-suite

themselves according to a given market situation, e.g., the size of the market.

6. ACKNOWLEDGMENTS

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (CRC 901).

7. REFERENCES

[1] S. Arifulina, M. Becker, M. C. Platenius, and S. Walther. SeSAME: Modeling and Analyzing High-Quality Service Compositions. In *Proc. of the 29th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*. ACM, 2014.

[2] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy goals for requirements-driven adaptation. In *18th IEEE Int. Requirements Engineering Conf. (RE)*, pages 125–134. IEEE, 2010.

[3] S. Becker, S. Overhage, and R. H. Reussner. Classifying Software Component Interoperability Errors to Support Component Adaption. In *7th Int. Symp. on Component-Based Software Engineering*, LNCS, pages 68–83. Springer, 2004.

[4] A. Beugnard, J. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 1999.

[5] C. Ghezzi and A. Mocci. Behavior model based component search: an initial assessment. In *Proc. of ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, pages 9–12. ACM, 2010.

[6] F. Kaufer and M. Klusch. Wsmo-mx: A logic programming based hybrid service matchmaker. In *4th European Conf. on Web Services*, pages 161–170. IEEE, 2006.

[7] M. Klusch and P. Kapahnke. OWLS-MX3: an adaptive hybrid semantic service matchmaker for OWL-S. In *Proc. of 3rd Int. Workshop on Semantic Matchmaking and Resource Retrieval (SMR2)*, 2009.

[8] M. Klusch and P. 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.

[9] J. Li. A fast semantic web services matchmaker for owl-s services. *Journal of Networks*, 8(5):1104–1111, 2013.

[10] N. Masuch, B. Hirsch, M. Burkhardt, A. Heßler, and S. Albayrak. Sema2: A hybrid semantic service matching approach. In *Semantic Web Services*, pages 35–47. Springer, 2012.

[11] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *Int. Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[12] M. C. Platenius. Fuzzy service matching in on-the-fly computing. In *Proc. of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 715–718, New York, NY, USA, 2013. ACM.

[13] M. C. Platenius, S. Arifulina, and W. Schäfer. MatchBox: A Framework for Dynamic Configuration of Service Matching Processes. In *Proc. of the 18th Int. ACM SIGSOFT Symposium on Component-Based Software Engineering*, pages 75–84. ACM, 2015.

[14] M. C. Platenius, M. von Detten, S. Becker, W. Schäfer, and G. Engels. A Survey of Fuzzy Service Matching Approaches in the Context of On-The-Fly Computing. In *Proc. of the 16th Int. ACM SIGSOFT Symposium on Component-Based Software Engineering*, pages 143–152, 2013.

[15] K. T. Stolee. Finding suitable programs: Semantic search with incomplete and lightweight specifications. In *34th Int. Conf. on Software Engineering (ICSE)*, pages 1571–1574. IEEE, 2012.

[16] K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous agents and multi-agent systems*, 5(2):173–203, 2002.

[17] University of Paderborn. MatchBox Screencast. https://goo.gl/MGJTcP.

[18] University of Paderborn. Website of CRC 901 "On-the-Fly Computing". http://sfb901.upb.de, Last Access: June 2015.

[19] University of Paderborn. Website of MatchBox. http://goo.gl/MMCxQT, Last Access: June 2015.