This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

Imprecise Matching of Requirements Specifications for Software Services using Fuzzy Logic

Marie C. Platenius, Wilhelm Schäfer
Software Engineering
Heinz Nixdorf Institute
Paderborn University, Germany
{m.platenius,wilhelm}@upb.de

Ammar Shaker, Eyke Hüllermeier
Intelligent Systems
Department of Computer Science
Paderborn University, Germany
{ammar.shaker,eyke}@upb.de

Matthias Becker
Software Engineering
Fraunhofer IEM
Paderborn, Germany
matthias.becker@iem.fraunhofer.de

***Abstract*—Today, software components are provided by global markets in the form of services. In order to optimally satisfy service requesters and service providers, adequate techniques for automatic service matching are needed. However, a requester's requirements may be vague and the information available about a provided service may be incomplete. As a consequence, fuzziness is induced into the matching procedure. The contribution of this paper is the development of a systematic matching procedure that leverages concepts and techniques from fuzzy logic and possibility theory based on our formal distinction between different sources and types of fuzziness in the context of service matching. In contrast to existing methods, our approach is able to deal with imprecision and incompleteness in service specifications and to inform users about the extent of induced fuzziness in order to improve the user's decision-making. We demonstrate our approach on the example of specifications for service reputation based on ratings given by previous users. Our evaluation based on real service ratings shows the utility and applicability of our approach.**

Index Terms: Service Selection, Service Matching, Requirements Specifications, Non-Functional Properties, Fuzzy Logic, Uncertainty, Decision Making

I. Introduction

The increasing popularity of paradigms like service-oriented computing and cloud computing is leading to a growing number of service providers offering software components in the form of deployed, ready-to-use services (Software as a Service, SaaS) [57]. In order to benefit from these services, service requesters need to discover the services that best satisfy their requirements. For this purpose, *service matching* approaches are used. These approaches determine whether the specification of a provided service satisfies the requester's requirements specification [56]. For each provided service, a matching approach delivers a matching result that indicates to what extent the service specification satisfies the given requirements specification. A number of matching approaches already exist. The majority of them focuses either on structural properties (e.g., signatures), behavioral properties (e.g., pre- and post-conditions or protocols), or non-functional properties (e.g., performance or reputation) (see [19] or PvDB+13 for an overview).

Nevertheless, finding the "perfect match" remains a challenge, due to the imperfect nature of the information involved in the matching procedure [61]. To begin with, a requester's requirements for a service are often specified vaguely, for example in terms of natural language expressions (e.g., the service should be "fast"). Moreover, the available information on a provided service is typically incomplete and/or imprecise. For example, for a service that newly enters the market, determination of its reputation becomes difficult. Thus, various sources of uncertainty or *fuzziness* make exact matching no longer feasible.

We present a fuzzy matching approach that overcomes these limitations. Our main contribution is twofold:

- We distinguish between multiple types and sources of fuzziness: This is accomplished with the help of formal concepts from fuzzy logic, which allows capturing and modeling various types of incomplete, imprecise, and uncertain information within a unified, coherent mathematical framework. Moreover, such information accounts for the graded nature of a requester's satisfaction towards the requirements in an adequate manner.
- Our approach delivers an expressive matching result that informs users about the extent of induced fuzziness. In detail, the matching results consist of a degree of necessity and a degree of possibility with respect to the fulfillment of the requirements, i.e., a lower and an upper bound on the degree of user satisfaction. Thus, our method provides important extra information about uncertainty in service matching.

As opposed to this, existing approaches coping with imprecise matching proposed in the literature (see [64] for an overview) do not distinguish between different types of fuzziness. For example, some approaches (e.g., [1], [45]) only consider fuzziness in terms of graded user satisfaction, but they ignore uncertainty. Furthermore, they only produce a single matching degree, often chosen arbitrarily and lacking a clear semantics. In particular, they do not deliver any feedback on how uncertain these matching results are. In this way, requesters cannot assess their risks regarding to what extent the matching result might have been adulterated due to the imperfection of the information involved.

For the purpose of illustration, we mainly adopt the example of reputation data, i.e., a set of values representing the reputation of a service based on ratings given by previous

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

users. Reputation is a service property of special importance for customers and providers as it reflects how satisfied previous users were with a service. In addition, we show how our approach can be combined with other matching approaches like signature matching or matching of other quality properties on the basis of our earlier work [62]. Furthermore, we demonstrate how our approach can be applied to other service properties such as performance. The usefulness of our approach is demonstrated in a case study including four experiments. In particular, we collected real service ratings from the website TrustRadius.com [79] and applied our approach to this data.

Our approach extends traditional approaches from the areas of service discovery and component retrieval with extra benefits for the targeted end users. Service requesters specify their requirements in a simple way and are allowed to remain vague. Due to the awareness of present uncertainty within the matching results, requesters are able to make better decisions when selecting between services. Corresponding to their risk aversion, they become able to choose either services that are certainly a good match or services that may potentially (but not necessarily) be perfect matches. In addition, the service providers profit from additional information about 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. All in all, we enable service matching under more realistic circumstances compared to the related work. We are able to explicitly deal with imprecision and incompleteness in service specifications, which makes our approach well applicable in practice.

Our paper is structured as follows. In Section II, we explain the addressed problem of fuzziness in service matching by using an example. Furthermore, we derive requirements for the solution. On the basis of these requirements, we discuss the related work in fuzzy service matching in Section III. Section IV introduces foundations of fuzzy logic. The fuzzy matching approach proposed in Section V is based on these foundations and addresses the requirements explained earlier. In Section VI, we describe how the proposed fuzzy matching approach can be integrated into a matching process. We give some details about the technical realization of our fuzzy matching approach in Section VII. Experiments building on this realization are presented in Section VIII. Section IX concludes the paper.

II. Problem Definition and Requirements

Service matching is the process of comparing a specification of requirements for a service to specifications of the services provided in a service market [56]. Thus, the matching approach is tightly connected to the representation of the specifications the approach operates on. Different representations are possible as many different properties of a service have to be matched, including functional properties (e.g., signatures or protocols) as well as non-functional properties (e.g., performance or reputation). In the following, we take the reputation of a service as an example.

A. Specifying Reputation Requirements

The *reputation* of a service is measured on the basis of ratings stated by previous users, to express their satisfaction with a certain service. Thus, reputation indicates not only the popularity of a service but also its trustworthiness based on human judgement [70]. Technically, the reputation of a service is an *aggregation* of a set of *ratings*. Each rating has different properties, like value (e.g., 4 stars within a 5-star scale), age (e.g., two months), and context (e.g., a rating about the performance or about the availability of a service). In the following, we explain the representation we propose to specify reputation requirements. The detailed language definition in the form of a metamodel can be found in [2].

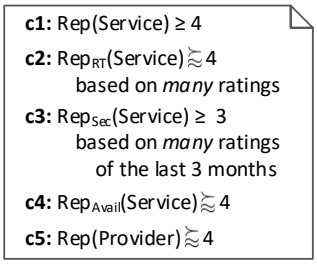
Reputation requirements can be modeled as a list of conditions. As an example, consider the requirements specification in Figure 1(a). This requirements specification consists of five conditions, `c1` – `c5`. For a full match, all conditions have to be satisfied. If not all conditions are satisfied, the matching approach returns a result that denotes to what extent the conditions are satisfied. Based on this result, the requester can compare different services easily in order to select one. The more details a requester specifies, the more accurately can the matching approach determine results that actually fit the requester's interests. However, with an increasing complexity, also the set of services matching the requirements to a high extent becomes smaller.

Each of the conditions in the requirements specification refers to properties related to service reputation. For example, `c1` checks whether the overall reputation of a service is greater or equal to 4 (based on a 5-star range as it is common in today's app stores). The conditions `c2`, `c3`, and `c4` check context-specific reputation values, i.e., the perceived response time of the service (`c2`), the perceived security of a service (`c3`), and the perceived availability of a service (`c4`). As a further restriction to the reputation values requested by `c2` and `c3`, the value must have been aggregated on the basis of at least a specific number of ratings. This is specified using the fuzzy term "many". Such restrictions are useful as a reputation value's reliability increases with the number of ratings it has been calculated from. Furthermore, `c2` and `c4` use a soft threshold, suggesting that the lower bound should be approximately 4 with a toleration of slightly smaller values; for example, a reputation of 3.95 would still be considered acceptable to some degree. In `c3`, there is a restriction with respect to time. Here, the reputation value should have been created on the basis of 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 in a specific domain increased due to some incident). `C5` is about the reputation of the service's provider.

B. Reputation Matching

Figure 1(b) shows an example extract of the contents of a reputation system in a tabular notation. These contents are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering



(a) Requirements Specification

Entity	Context	# Ratings	
Service: ImagePro1	Overall	300	Rep(300 ratings) = 4.5
Service: ImagePro1	ResponseTime	80	Rep (80 ratings) = 3.5
Service: ImagePro1	Security	30	Rep (30 ratings, 3 months) = 2.75
Service: ImagePro1	Availability	10	Rep (10 ratings) = 4
Service: PicProcessor	Overall	90	Rep (90 ratings) = 3.35
Service: PicProcessor	ResponseTime	20	Rep (20 ratings) = 1.5
Service: PicProcessor	Security	20	Rep (20 ratings, 3 months) = 4.5
Service: PicProcessor	Availability	10	Rep (10 ratings) = 3
...	...	...	...
Provider: IPServices	Overall	130	Rep (130 ratings) = 3.5

(b) Available Ratings & Example Reputation Values

Fig. 1. Example requirements and available service information

used to evaluate the conditions of the requirements specification explained above. For example, reputation values based on different contexts for the services `ImagePro1` and `PicProcessor` are depicted. We also see the reputation of the service provider `IPServices`, which is the provider of both the `ImagePro1` and the `PicProcessor` services. The third column depicts how many ratings are available per service in total. The rightmost column depicts some example reputation values calculated on the basis of these ratings. Please note that these are dynamic values not stored in the reputation system but derived from the ratings stored in the system on the basis of a selected aggregation function during the matching procedure. There are several possibilities to aggregate ratings to reputation values. For convenience, the ones depicted in the figure are exactly those that are needed to evaluate the given requirements specification.

After the required reputation values considering all requested restrictions have been determined, a simple exact matching of each condition is a numerical comparison. We refer to this approach as a *Traditional Matching Approach*. For example, for `ImagePro1`, `c1` evaluates to `true` because the overall reputation value can be calculated on the basis of 300 ratings and turns out to be 4.5 (cf. the first row of the reputation system depicted in Figure 1). In contrast, `PicProcessor` can already be discarded as the overall reputation is only 3.35. As shown in this example, an exact matching approach requires perfect information (e.g., a high number of ratings) as well as precise conditions given by the requester. In the remainder of this paper, we explain why this is an unrealistic assumption and how we can deal with incomplete information as well as imprecise requirements using *fuzzy matching* techniques.

C. Fuzziness in Service Matching

As already mentioned in the introduction, service matching is affected by various types of uncertainty, imprecision, and incompleteness—subsequently subsumed under the notion of "fuzziness" in a broad sense—of the information being involved and the data being processed. In [61], we distinguished three possible sources of fuzziness in service matching: the requester, the provider, and the matching algorithm. Requester-induced fuzziness is brought by the requirements specification and, moreover, is caused by the graded nature of the requester's satisfaction. In contrast to this, provider-induced fuzziness is due to a lack of information about service properties. Finally, algorithm-induced fuzziness (which is less relevant for this paper) might be caused by approximate computations and heuristics used in (computationally intractable) matching algorithms.

Interestingly, the distinction between different *sources* of fuzziness goes hand in hand with a distinction between different *types* of fuzziness:

- *Vagueness:* Requirements on services are often specified *vaguely* (e.g., using natural language expressions such as "close to 4 stars"). In our examples from Section II-A such a soft threshold is denoted by $\gtrsim$. Such vague requirements are not immediately amenable to computational processing. The concrete meaning of such expressions can be captured by fuzzy sets. Modeling a linguistic expression in terms of a so-called membership function (i.e., a precise mathematical object) is sometimes referred to as a process of "precisiation" [92].
- *Gradedness:* The vagueness of a 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, since a service can match the requirements *to some degree*, the requester can be *partially* satisfied with a service. For example, requesters are often tolerant with regard to slight deviations from the target. A requester demanding an average reputation of 4 stars might also be satisfied with a service having a reputation of 3.95, at least to some extent. Again, 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). In the context of the matching problem, the fuzzy set will then serve as a soft constraint.
- *Uncertainty, partial ignorance:* In many cases, service providers do not offer precise information about a service, either because of business interests or because details are difficult to determine. For example, when matching data with respect to the reputation of a service, provider-induced fuzziness occurs particularly for new services that got only few ratings so far. Alternatively, ratings may be lacking because of the unrateability of a service

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

that has only been used as part of a composition and is not directly visible to the rater [17]. All in all, there are various reasons for why properties of a service can be affected by uncertainty or, more specifically, partial ignorance. As a consequence of missing information about a service, the matching procedure becomes uncertain. However, the user may not be aware of the fact that the returned matching result is unreliable because it has been calculated on the basis of missing information.

- *Approximations:* As already noted, algorithm-induced fuzziness might be caused by matching algorithms that apply approximations or other kinds of relaxations or heuristics. The purpose is to keep the matching process efficient. Like incomplete information, approximations can lead to *uncertainty* regarding the computed matching results.

As a consequence of induced fuzziness, matching approaches may deliver adulterated matching results that do not notify the users about the induced fuzziness. This 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 may have very different meanings. In particular, it could mean (a) the service fully matches half of the requirements specification and fully mismatches the other half, (b) the service matches all parts of the specifications moderately, or (c) the degree of matching is completely uncertain because detailed information is missing. Thus, while the "fuzziness" of the match refers to graded user satisfaction in Cases (a) and (b), it is due to uncertainty and incomplete information in Case (c). As explained in the following sections, our approach carefully distinguishes between these sources and makes them transparent to the user.

D. Requirements

From the above mentioned foundations, we can derive four requirements for an appropriate fuzzy matching approach in order to cope with induced fuzziness:

(*R1*) *Deliver Unadulterated Matching Results:* Even though fuzziness might have been introduced into the matching procedure, an appropriate matching approach should deliver matching results that are not misleading. This means, the delivered matching results should not lead to false positives and false negatives.

(*R2*) *Inform about Extent of Fuzziness:* In order to assess whether a user can trust a returned matching result, the user needs to know how much fuzziness has been induced. For example, if the induced fuzziness is very high, a user may decide to not take the risk and consider other services that may be worse matches. Information about the extent of induced fuzziness could be represented numerically (e.g., "Fuzziness of ImagePro1 is `0.1`; Fuzziness of PicProcessor is `0.3`") or using qualitative terms (e.g., "Fuzziness within the result for ImagePro1 is `low`; Fuzziness within the result for PicProcessor is `high`"). How much fuzziness can be accepted depends on the specific user's risk aversion. Fulfilling *R2* supports the fulfillment of *R1*.

(*R3*) *Inform about Fuzziness Sources:* A fuzzy matching approach has to distinguish between different fuzziness sources (e.g., requester-induced fuzziness and provider-induced fuzziness). Based on this, the user can be informed about fuzziness sources. Knowing the source of fuzziness that has been induced in a matching result, may enable the user 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 bout the provided service. An adequate output could be: "The result for ImagePro1 contains `provider-induced fuzziness`".

(*R4*) *Handle Different Fuzziness Types:* In order to handle fuzziness during service matching appropriately, different fuzziness types need to be distinguished during the matching procedure (e.g., vagueness, gradedness, and uncertainty). This distinction is important as different types may have different consequences. In particular, in contrast to gradedness, uncertainty may become a problem to the requester when interpreting the delivered matching result. Furthermore, not every calculus is able to model all these fuzziness types. For example, probability theory does not distinguish between uncertainty and gradedness, as we will discuss in more detail in Section IV.

In general, information on fuzziness sources influencing the matching result and the extent of this influence gives the requesters and providers more insights about the matching results. Based on informative matching results that reflect fuzziness instead of concealing it, users can make more informed decisions. In some cases, the users may even be able to overcome some fuzziness occurrences by different means (e.g., adapting the requirements specification or delivering more information at the provider's side).

III. Related Work

We surveyed the related work of fuzzy service matching according to the guidelines for systematic literature reviews by Kitchenham et al. [12], [35] due to the high number of papers related to service matching that has been published [64]. All details about the procedure for selecting and performing reviews can be found on our paper's website [63]. Based on this, we selected the 31 most related approaches and discuss them in the following.

In particular, we reviewed the selected publications with respect to three comparison criteria that reflect the requirements described in Section II-D:

- `Matching Result Format` (see *R1* and *R2*) indicates how comprehensive the matching results returned by the described matching approach are and to which extent they reflect induced fuzziness. Possible values

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

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).

- `Considered Fuzziness Types` (see *R4*) answers the question, which of the introduced fuzziness types have been considered in the described matching approach. Possible values are Gradedness ("`Gra`"), Vagueness ("`Vag`"), Uncertainty ("`Unc`"), and Approximations ("`App`").
- `Considered Fuzziness Sources` (see *R3*) answers the question, which of the introduced fuzziness sources have been considered in the described matching approach. Possible values are Requester ("`Req`"), Provider ("`Prov`"), and Algorithm ("`Algo`").

Table I shows the comparison. From this comparison, we can conclude that related approaches are limited with respect to multiple issues that we discuss in the following.

a) Differentiation of Several Fuzziness Types and Sources: During our reviews, we noticed that the many 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., [45]). By introducing gradual matching results, they want to get rid 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.

The same holds for approaches working with approximations during matching. These approaches potentially lead to uncertainty, but this fact is not visible to the user (e.g., [5]). All in all, no approach explicitly addressed uncertainty.

Furthermore, existing approaches do not explicitly distinguish between fuzziness sources. In many approaches, provider-induced as well as requester-induced fuzziness emerges, however, most of the approaches do not treat them in distinct ways, which again leads to adulterated matching results.

Only Bacciu et al. [4], [5] mention that fuzziness on the requester side is more about preferences, while fuzziness on the provider 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. [4], [5] 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 [45], 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.

b) Expressiveness of Matching Results: As can be seen in the table, most of the related approaches deliver gradual result scores. 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, [93] treats missing specifications as a mismatch, mixing mismatches with uncertainty.

In [66], 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's implementation. Adaptations are only possible as non-invasive extensions (e.g., [55]). Sora and Todinca's approach [72], [71] provides either "select" or "reject" combined with either "weak" or "strong" as an output. 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.

We also found that approaches based on fuzzy logic usually transform into a crisp matching result after having matched based on fuzzy sets (the approach by Bacciu et al. [4], [5] discussed above being one exception). This transformation leads to a loss of information. In particular, the crisp matching result does not reflect fuzziness anymore.

Within our selected set of publications, we did not find any approach that returns interval matching results, or other result formats of a similar expressiveness as our possibility-necessity intervals that will be introduced later. In general,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

TABLE I
COMPARISON OF FUZZY MATCHING APPROACHES

Approach	***R1***, ***R2*** (Matching Result Format)	***R4*** (Considered Fuzziness Types)	***R3*** (Considered Fuzziness Sources)
Almulla et al. [1]	– (score)	∼ (Gra, Vag)	– (Req)
Bacciu et al. [4], [5]	– (score)	∼ (Gra, Vag, App)	+ (Req, Prov, Algo)
Bai et al. [6], [7], [51], [52]	– (score)	– (App)	– (Algo)
Chao et al. [13], [31], [32], [33], [49], [48]	– (score)	– (Vag)	– (Req)
Constantinescu et al. [15]	– (score)	– (App)	– (Algo)
Cock et al. [18]	– (score)	∼ (Vag, Gra)	– (Req)
Cooper et al. [16]	– (score)	– (Vag)	∼ (Req, Prov)
Fenza et al. [27]	∼ (multiple scores)	– (App)	∼ (Req, Algo)
Jeong et al. [34]	– (score)	– (App)	– (Algo)
Klusch et al. [38], [39], [42]	– (degree)	– (App)	– (Algo)
Klusch, Kapahnke [41], [40]	– (degree)	– (App)	– (Algo)
Küster et al. [43], [44], [45], [46]	– (score)	∼ (Vag, Gra)	∼ (Req, Prov)
Liu et al. [53]	– (score)	∼ (Vag, Gra)	– (Req)
Liang et al. [47]	– (score)	– (App)	– (Algo)
Ma et al. [54]	– (score)	– (App)	– (Algo)
Pantiniotakis et al. [58]	– (score)	– (Vag)	∼ (Req, Prov)
Peng et al. [59]	– – (boolean)	– (Vag)	∼ (Req, Prov)
Qu et al. [65]	– (score)	– (Vag)	– (Prov)
Rao, Sarma [66]	∼ (reuse decisions)	– (Vag)	– (Prov)
Schönfisch et al. [68]	– – (boolean)	– (App)	– (Algo)
Sangers et al. [67]	– (score)	– (App)	– (Algo)
Şora, Todinca [72], [71]	– (weak/strong select/reject)	– (Vag)	∼ (Req, Prov)
Sycara et al. [73]	– (score)	– (App)	– (Algo)
Torres et al. [77]	– (score)	– (Vag)	– (Req)
Toninelli et al. [76]	– (score)	– (App)	– (Algo)
Toch et al. [75]	– (score)	– (App)	– (Algo)
Thakker et al. [74]	– (score)	– (App)	– (Algo)
Wang et al. [84]	– (score)	– (Vag)	∼ (Req, Prov)
Wang et al. [85], [86]	– (score)	– (Vag)	∼ (Req, Prov)
Xiong, Fan [88]	– (score)	– (Vag)	∼ (Req, Prov)
Zilci et al. [93]	– (score)	– (Vag)	∼ (Req, Prov)
Approach proposed in this paper	+ (n-p-interval)	∼/+ (Vag, Gra, Unc)	∼ (Req, Prov)

most of the approaches hardly mentioned the result format and its properties at all.

c) Fuzziness in Related Software Engineering Disciplines: Since only few service matching approaches explicitly deal with fuzziness in terms of uncertainty, we also took into account literature from related software engineering disciplines.

For example, in his position paper [29], David Garlan lists several sources of uncertainty in software engineering. These sources fit well to our service matching problem. For example, we also have (a) the human involved in both specifying services and requirements as well as in making the final decisions before acquiring a service based on the matching results, (b) different platforms where services can be deployed, 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 we 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 more expressive matching results.

Esfahani et al. [25], [26] take some of these sources up again and extend the list with further sources. In their approach POISED [25], [26], estimates of uncertainty in self-adaptation problems are integrated into a possibilistic analysis in order to make better adaptation decisions within the scope of self-adaptive systems. Similar to our approach, they work with lower and upper bounds to estimate the range of the uncertainty. In general, their goals are in line with ours: identifying the sources of uncertainty and estimating the extent of uncertainty. However, in contrast to the domain of making adaptation decision, the domain of service matching, on the one hand, requires a more fine-grained classification of sources (e.g., requester-induced fuzziness vs. provider-induced fuzziness as a refinement of "uncertainty due to human in the loop"), while, on the other hand, other sources are less relevant (e.g., "uncertainty due to noise"). Furthermore, our approach incorporates different kinds of uncertain information, i.e., statistical information on the provider's side and vague information on the requester side.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

Based on [83], Perez-Palacin and Mirandola [60] constructed a classification of uncertainty in software models. This is related to the requester-induced and provider induced fuzziness as our specifications are also software models to a certain extent. Three of the five orders of uncertainty listed there can be directly 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.
- 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. We claim that existing (fuzzy) service matching approaches lead to the 2nd order of uncertainty as 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 reduce the order of induced uncertainty from the 2nd order to the 1st. This enables the user to cope with the known uncertainty and to assess its risks. Furthermore, the user may even get to the 0th order by eliminating the known uncertainty. Reaching the 1st order is essential to enable strategies to completely eliminate uncertainty. As a consequence, this also means eliminating uncertainty of Order 3 (uncertainty about the lack of a process to find out the lack of awareness of uncertainty), while Order 4 (the lack of knowledge about the orders of uncertainty) is of no concern for the user of our approach.

Furthermore, approaches related to requirements specification for self-adaptive systems under consideration of uncertainty (e.g., RELAX [87] and FLAGS [8]) are related to our approach. 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, these approaches focus on internal behavioral details (e.g., real-time constraints), that can be specified with temporal logic. Thus, it requires deeper knowledge of the system's internal behavior, which we typically do not have when working with service-based systems, where only the interface descriptions are publicly visible.

In reliability analysis (e.g., [14], [89]), 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 [89]. Yin et al. [89] discuss three ways to present this uncertainty: bounds, confidence intervals, and probability distributions. The confidence intervals are comparable to the n-p-intervals we will introduce in the following sections. However, the presented analytical approaches are limited to special kinds of models and parameter distributions; they are hard to generalize. For example, they are not applicable to our reputation specifications. Again, the considered 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.

d) Summary of Shortcomings: To sum up, related work incorporates fuzziness to a limited extent. In particular, related approaches (a) neither distinguish between fuzziness types like gradedness and uncertainty, nor between fuzziness sources like requester and provider. In addition, related approaches (b) do not address the presentation and the semantics of fuzzy matching results. These shortcomings lead to the fact that the matching results returned to users of these approaches are deceptive. They do not allow to assess risks that are caused by matching procedures being processed on imperfect inputs and algorithms. Furthermore, the user does not get feedback that the results are deceptive and, thus, has no opportunity to improve this situation.

As opposed to this, the last line of Table I shows that the approach presented in this paper addresses these issues. On the one hand, it delivers informative matching results in the form of n-p-intervals (explained in Section V). On the other hand, it considers both vagueness and gradedness on the requester's side as well as uncertainty on the provider's side. An extension of our approach to deal with algorithm-induced fuzziness in the form of approximations is planned for future work. This extension will extend the already existing approaches once more, with a focus on an informative matching result that does not conceal fuzziness, but communicates it to the user for supporting the decision to be made based on matching results.

IV. Foundations of Fuzzy Modeling

In the following, we discuss foundations of *fuzzy sets* and *possibility theory* our approach is based on.

A. The Notion of a Fuzzy Set

A fuzzy subset A of a reference set U is identified by a so-called *membership function*, often denoted μ_A, which is a generalization of the characteristic function of an ordinary subset [90]. For each element $x \in U$, this function specifies the degree of membership of x in the fuzzy set. Usually, membership degrees $\mu_A(x)$ are taken from the unit interval $[0, 1]$, i.e., a membership function is a mapping $U \longrightarrow [0, 1]$. We denote the set of fuzzy subsets of U by $\mathbb{F}(U)$.

Fuzzy sets formalize the idea of *graded membership*, i.e., the idea that an element can belong "more or less" to a set. Consequently, a fuzzy set allows for modeling concepts

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

with non-sharp boundaries. Consider the set of services with good reputation as an example. Is it reasonable to say that an average rating of at least 3.7 is good and 3.6 is not good? In fact, 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, it becomes possible to express grades such as: a service with rating 4.5 is completely fulfilling the requirements ($\mu_A(4.5) = 1$), a rating of 3.7 is "more or less" good ($\mu_A(3.7) = 1/2$, say), and 2.5 is definitely not good ($\mu_A(2.5) = 0$).

Fuzzy sets are often associated with natural language expressions and, as already mentioned, are used to capture the meaning of such expressions in a precise, mathematical form. Thus, they conveniently act as a human-machine interface between a symbolic and a numeric level of modeling. Moreover, a fuzzy set can have different semantic interpretations [21]. In particular, the membership degrees of a fuzzy set can be interpreted in terms of *preference* and *uncertainty*. For example, consider the fuzzy set A of services with "high reputation", formalized in terms of a membership function μ_A on $U = [0, 5]$. Being provided as a requirement by a requester, A can be interpreted in terms of preference: $\mu_A(x)$ is the degree to which the requester is satisfied with a service having an average rating of x. As opposed to this, if "high" is given by the provider as (imprecise) information about the reputation, A is interpreted in terms of uncertainty: $\mu_A(x)$ is the degree to which x is considered *possible* as the true value of the average rating. This interpretation will be discussed in more detail in Section IV-C below.

B. Fuzzy Logic

In conjunction with generalized logical (set-theoretical) operators, the concept of a fuzzy set can be developed into a generalized set theory, which in turn provides the basis for generalizing theories in different branches of (pure and applied) mathematics as well as fuzzy set-based approaches to intelligent systems design. The term "fuzzy logic" is commonly used as an umbrella term for a collection of methods, tools, and techniques for constructing systems of that kind.

To operate with fuzzy sets in a formal way, fuzzy logic offers generalized set-theoretical or logical connectives (like in the classical case, there is a close correspondence between set theory and logic). Especially important in this regard is a class of operators called *triangular norms* or *t-norms* for short [37]. A t-norm $\top$ is a $[0,1] \times [0,1] \longrightarrow [0,1]$ mapping which is associative, commutative, monotonically increasing (in both arguments), and which satisfies the boundary conditions $\top(\alpha, 0) = 0$ and $\top(\alpha, 1) = \alpha$ for all $0 \leq \alpha \leq 1$. Well-known examples of t-norms include the minimum $(\alpha, \beta) \mapsto \min(\alpha, \beta)$ and the product $(\alpha, \beta) \mapsto \alpha\beta$. A t-norm naturally qualifies as a generalized logical *conjunction*. Moreover, it can be used to define the *intersection* of fuzzy subsets $A, B \in \mathbb{F}(U)$ as follows: $\mu_{A\cap B}(x) = \top(\mu_A(x), \mu_B(x))$ for all $x \in U$. Likewise, the standard *negation* operator $\alpha \mapsto 1-\alpha$ can be used to model the set-theoretical complementation, i.e., the membership function of the complement $\bar{A} = U \setminus A$ of A in U: $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$ for all $x \in U$.

C. Possibility Theory

Possibility theory is a general uncertainty calculus. Although it could in principle be studied independently of fuzzy logic, there is a close connection between both theories, because possibility distributions are often derived from fuzzy sets by interpreting membership degrees in terms of degrees of possibility [91]. Formally, a *possibility distribution* π is again a mapping $U \longrightarrow [0, 1]$. A distribution of that kind induces a *possibility measure* $\Pi : 2^U \longrightarrow [0, 1]$, which is defined by the supremum, i.e., the least upper bound, of $\pi(u)$:

$$\Pi(A) = \sup_{u\in A} \pi(u) \tag{1}$$

for all $A \subseteq U$. The measure Π captures uncertain information about a true (but unknown) value $u_0 \in U$. For each subset $A \subseteq U$, $\Pi(A)$ is the degree of plausibility that $u_0 \in A$.

A possibility distribution Π can be interpreted as a compact representation of a family of probability measures, namely all measures P that are upper-bounded by Π (in the sense that $P(A) \leq \Pi(A)$ for all measurable $A \subseteq U$). Thus, a lack of information (ignorance) can arguably 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, then each value $x \in U = [0, 5]$ appears to be fully plausible. In probability theory, this situation could be modeled by the uniform distribution with density $p(x) \equiv 1/5$. This is the same distribution, however, that also models perfect knowledge about the equal probability of each value (e.g., derived from a very large number of service ratings). Thus, in standard probability, complete ignorance cannot be distinguished from perfect (probabilistic) knowledge in this situation.

The limited expressivity of probability measures is due to their self-reflexivity: $P(A) = 1 - P(U \setminus A)$ for all $A \subseteq U$. Therefore, it is impossible to express that A is plausible (i.e., probable) without saying that the complement of A is implausible. In possibility theory, implausibility is captured by a second measure, a so-called *necessity measure* N, which is dual to Π in the sense that $N(A) = 1 - \Pi(U \setminus A)$. Put in words, a subset A is considered necessary to the same extent to which the complement of A is considered implausible. One easily verifies that $N(A) \leq \Pi(A)$ for all $A \subseteq U$.

The domain of possibility/necessity measures can be extended from subsets $A \subseteq U$ to fuzzy subsets $A \in \mathbb{F}(U)$:

$$\Pi(A) = \sup_{x\in U} \min\big(\pi(x), \mu_A(x)\big) \ , \tag{2}$$

$$N(A) = 1 - \sup_{x\in U} \min\big(\pi(x), 1 - \mu_A(x)\big) \ . \tag{3}$$

V. Fuzzy Reputation Matching

Our main idea is to enable matching despite fuzziness induced into the matching process by imprecise or incomplete specifications, as explained in Section II-C. In particular, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

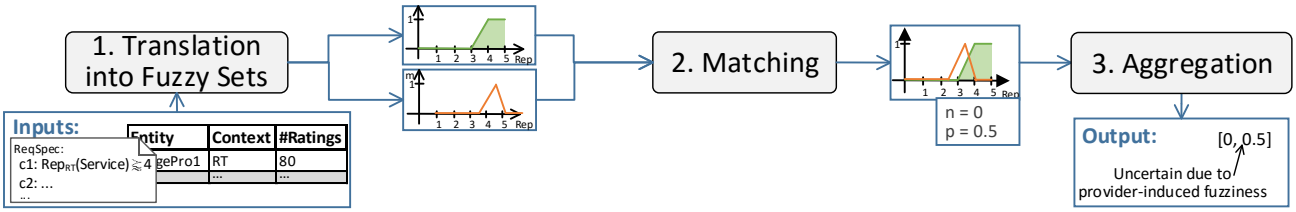


Fig. 2. Fuzzy Reputation Matching Procedure

matching approach should return not only the matching result as an objective measure of how much a service specification satisfies given requirements, but also reflect how much fuzziness has been induced. As a benefit, the requester can make a more informed decision. We can expect that, in most cases, requesters will prefer services with good matching results that come with a low amount of fuzziness. However, the amount of fuzziness and the degree of the matching result may form a trade-off, as discussed later.

We explain our approach on the basis of the example requirements depicted in Figure 1(a). These requirements cover requester-induced fuzziness as well as provider-induced fuzziness. Requester-induced fuzziness occurs in `c2`, `c4`, and `c5`, indicated by the soft threshold. Provider-induced fuzziness occurs due to the frequentist nature of the information, i.e., the reputation data. The more ratings available, the lower is the fuzziness. We discuss this issue in detail in the following sections.

A. Overview

Our fuzzy reputation matching approach is based on fuzzy sets and possibility theory. Figure 2 visualizes an overview of the procedure for a concrete example. Please note that this example shows only extracts—the steps iterate over all conditions that are part of the requirements specification. In general, the procedure is performed for each provided service to be considered (e.g., each provided service that matches the functional requirements). The procedure takes the requirements specification and all required ratings that are available for a provided service to be matched as an input. There are three main steps that are performed: 1. `Translation into Fuzzy Sets`, 2. calculation of `Matching` results, and 3. `Aggregation`. A matching result informing not only about the quality of the match but also about the induced fuzziness is returned as an output.

B. Step 1: Translation into Fuzzy Sets

In order to perform the main matching algorithm, the inputs need to be transformed into fuzzy sets. Thus, both the requirements as well as the reputation value based on the available ratings of a service need to be modeled as membership functions.

The benefit of a translation into fuzzy sets is that this enables us to use a coherent mathematical framework that is able to cope with fuzziness and uncertainty. Furthermore, this approach can be leveraged to address QoS-drift, i.e., perception of numerical values for non-functional properties due to a change in the domain knowledge [78].

1) Creation of fuzzy sets from requirements: Fuzzy requirements specifications, i.e., conditions containing a soft threshold, are transformed into membership functions denoting fuzzy sets. All other requirements are transformed into conventional sets (which are special cases of fuzzy sets with $\{0,1\}$-valued membership functions).

Figure 3 depicts the sets created from Conditions `c1` – `c5` from Figure 1(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 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. For the other hard constraint, `c3`, the threshold is 3.

Requirements `c2`, `c4`, and `c5` are transformed into membership functions denoting fuzzy sets as there are no hard, but soft thresholds. This is a process of *precisiation*, in which membership functions are assigned to symbolic expressions. To this end, we make use of modeling techniques that have been developed in the field of fuzzy logic, including mathematical tools such as modifier functions, linguistic hedges, and generalized quantifiers. Thus, we specify a set of rules that define how to turn expressions into membership functions. This includes the rule to turn an expression $\gtrsim x_0$ into a fuzzy set with membership function $x \mapsto \min(\max(x - x_0 + 1, 0), 1)$, as shown by the transformation results depicted in Figure 3. For example, the function for transforming the expression $Rep_{RT}(Service) \gtrsim 4$ from `c2` is $x \mapsto \min(\max(x-3,0),1)$ because $x_0 = 4$.

Refering to the fuzziness types introduced in Section II-C, *vagueness* is reflected within the fuzzy sets by the size of the set's gradient, i.e., the interval defining "fuzzy part". Conversely, *gradedness* rather refers to the location of this gradient (e.g., if it is closer to the right or closer to the left), depending on the threshold given with the requirements specification.

2) Creation of fuzzy sets from ratings: The case where requirements are specified on reputation values derived from ratings is interesting for several reasons, notably as it involves both vagueness and (frequentist) statistical information. In general, the requester will be interested in a characteristic

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

value like the expectation, i.e., the average rating of a service, and specify a soft constraint in the form of a fuzzy set A on this value; for example, the mean rating should be $\gtrapprox 4$ in `c2`.

In practice, the true expectation m is unknown; instead, only an estimation $\hat{m}$ of this value is available, namely the arithmetic mean of a finite set of ratings. There are two problems to be considered. First, we need to characterize the uncertainty about m or, stated differently, the reliability of the average $\hat{m}$, in a proper way. To this end, we make use of *bootstrapping*, a statistical resampling technique that allows estimating a probability distribution for the expectation [22]. 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 will provide an idea of the variance of the estimation.

Second, the information should be represented within our formal framework, so as to make it amenable to further processing. Thus, the probability function $p(\cdot)$ should actually be represented in the form of a possibility distribution. To this end, we adopt the interpretation of possibility as upper probability and apply an established probability-possibility transformation [20]:

$$\pi_m(x) = \sup\left\{\alpha \in [0,1] \,|\, x \in C_{1-\alpha}\right\} \ , \tag{4}$$

where $C_{1-\alpha}$ is the $1-\alpha$ confidence interval derived from the distribution $p(\cdot)$, i.e., the shortest interval around $\hat{m}$ covering a probability mass of $1-\alpha$.

a) Special cases: There are two special cases of the above problem that allow for a simplified specification of π_m: a very small and a very large sample of ratings. If only a few ratings are available (e.g., only 2 or 3, in which case bootstrapping will hardly make sense), or perhaps even no rating at all, it might be best to define $\pi_m \equiv 1$, thereby reflecting complete ignorance about m. In contrast, if the number of ratings is very large, $\hat{m}$ will approximate m extremely well, so that this value can be assumed to be known with sufficient certainty; for example, if there are more than 1,000 ratings for the service `ImagePro1` in `c1`, with an average of $\hat{m} = 4.5$. This value will then be extremely close to the true expectation. Then, we pretend to be completely certain by modeling information about m in terms of the possibility distribution π_m with $\pi_m(x) = 1$ if $x = \hat{m}$ and $\pi_m(x) = 0$ if $x \neq \hat{m}$.

b) Additional restrictions: In general, the sample size will directly influence the specificity of the possibility distribution π_m. The more ratings are available, the more peaked π_m will be around the average $\hat{m}$. Thus, the reliability of the information about m is reflected by π_m in a proper way. The requester may nevertheless be interested in specifying an extra restriction on the sample size using a fuzzy term (like "many" in `c2` and `c3`). Again, requirements of that kind can be formalized in terms of a soft constraint A with the membership function μ_A. This constraint is aggregated with the constraint about the reputation value using a conjunctive operator based on a t-norm (e.g., the minimum). The sample size itself is typically known precisely and, therefore, can be represented in terms of a possibility distribution π_s that yields a value of 1 for the sample size and 0 otherwise.

C. Step 2: Matching

In the previous section, we have explained how each of the requester's conditions c is translated into a corresponding fuzzy set A_c with the membership function μ_{A_c}, effectively representing a soft constraint that needs to be fulfilled. Likewise, we have seen how information B_c about a service can be formalized in terms of a possibility distribution π_{B_c}. The latter might be a single-point distribution, in the case where precise information is offered by the provider, but may also reflect a certain level of uncertainty about the true value. In our running example, it is derived from user ratings.

Now, matching the requirements with information about a service comes down to comparing A_c and B_c. What can we say about the degree of satisfaction of the requester? As explained in Section IV-C, this question can be answered by computing the possibility and the necessity of the requirement A_c under the distribution π_{B_c} according to Equation 2 and Equation 3, respectively:

$$\Pi_{B_c}(A_c) = \sup_{x \in U} \min\left(\pi_{B_c}(x), \mu_{A_c}(x)\right) \ , \tag{5}$$

$$N_{B_c}(A_c) = 1 - \sup_{x \in U} \min\left(\pi_{B_c}(x), 1 - \mu_{A_c}(x)\right) \ . \tag{6}$$

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

$$n_c = N_{B_c}(A_c) \leq \Pi_{B_c}(A_c) = p_c \ . \tag{7}$$

As an illustration, suppose A_c and B_c to be standard subsets. In that case, $[n_c, p_c] = [0, 0]$ if $A_c \cap B_c = \emptyset$ (the requester is certainly not satisfied), $[n_c, p_c] = [1, 1]$ if $B_c \subseteq A_c$ (the requester is certainly satisfied), and $[n_c, p_c] = [0, 1]$ otherwise (the requester is possibly but not necessarily satisfied). In the more general case where either A_c or B_c are fuzzy, n_c and p_c may also assume intermediate values.

The necessity and possibility degree in Equation 7 can be interpreted as a lower and an upper bound on the satisfaction of the requester, respectively: n_c is the degree to which the requester is *necessarily* satisfied, and p_c the degree to which she is *possibly* satisfied. Moreover, the length of the interval reflects the level of ignorance of the service properties. Thus, 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 (reflecting certainty about the satisfaction of the requester).

Illustration: In Figure 3, the (soft) constraints A_c specified by the requester are depicted with a shadowed background indicating the region where a requirement is satisfied. Furthermore, the possibility distributions informing about properties B_c of the services `ImagePro1` and `PicProcessor` are depicted.

As for the matching of requirements and properties, some of the results are easy to recognize. For example, we have a full match ($B_c \subseteq A_c$), and therefore $n_c = p_c = 1$, for `ImagePro1` in the cases of `c1`, and `c4`, and for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

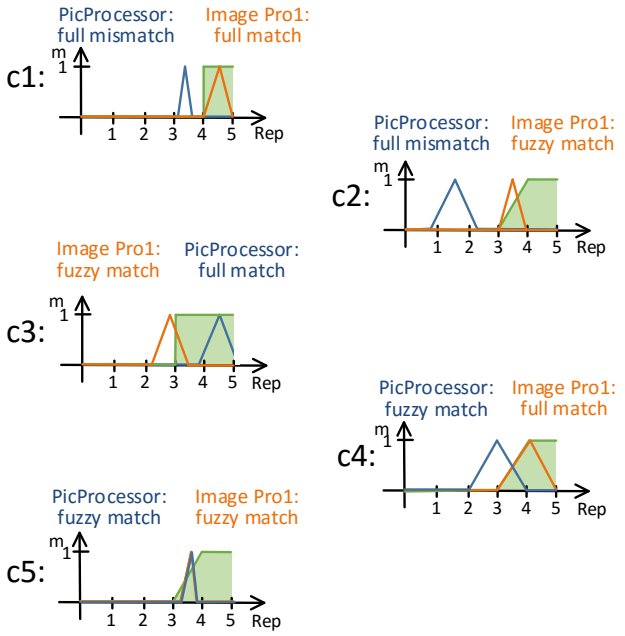


Fig. 3. Example Matching Results

`PicProcessor` in the case of `c3`. Likewise, we have a full mismatch ($B_c \cap A_c = \emptyset$), and therefore $n_c = p_c = 0$, for `PicProcessor` in the cases of `c1` and `c2`.

The other cases are partial or fuzzy matches. More specifically, the necessity-possibility intervals $[n_c, p_c]$ are obtained according to Equations 5–6. The complete results in the form of these intervals are depicted in Table 1.

condition	ImagePro1	PicProcessor
c1	$[1, 1]$	$[0, 0]$
c2	$[0.5, 0.5]$	$[0, 0]$
c3	$[0, 1]$	$[1, 1]$
c4	$[1, 1]$	$[0, 0.5]$
c5	$[0.286, 0.615]$	$[0.286, 0.615]$

TABLE II
RESULTS FOR THE SETS IN FIGURE 3

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

- First, there might be uncertainty about the requester's satisfaction, i.e., the interval $[n, p]$ will get wider.
- Second, the result can be more ambiguous, in the sense that the interval moves closer to the middle point $1/2$; in that case, we neither have a clear match nor a clear mismatch, but rather something in-between.

As already mentioned, uncertainty about the true satisfaction, which is naturally quantified in terms of the difference $p - n$, is closely connected with uncertainty on the side of the provider. In fact, the latter is a necessary requirement. If

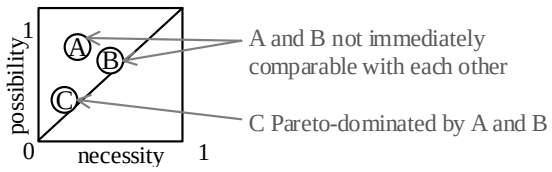


Fig. 4. Services associated with degrees n/p to which they necessarily/possibly meet the requirements specification

the information about a service is precise, then $n = p$. As for the second type of fuzziness in the result, the location of the interval $[n, p]$ is mainly influenced by the requester: The more "fuzzily" she specifies her requirements, i.e., the less clearly she states what she likes and what she dislikes, the more "mediocre" the result will be, i.e., the closer $[n, p]$ will be located around the middle point $1/2$.

D. Step 3: Aggregation

So far, a service has been evaluated on each of the conditions $c_1, \ldots, c_q$ separately, resulting in corresponding intervals $[n_{c_i}, p_{c_i}]$. In order to obtain an overall result, these intervals need to be aggregated. In the literature, three classes of aggregation operators are distinguished: conjunctive, disjunctive, and generalized averaging operators [30]. Conjunctive operators include the class t-norms (cf. Section IV-B); using an operator of this kind, the aggregated result is high only if the individual result are high on *all* conditions. Disjunctive operators, on the other hand, are fully compensative. There, a high result on a single requirement is enough to have a high overall result. Averaging operators, which include the (weighted) arithmetic mean, are in-between conjunctive and disjunctive operators.

Recall that n_c and p_c are interpreted, respectively, as a lower and upper bound of the requester's satisfaction on condition c. Since all aggregation operators AGG are monotonic, the set of possible degrees of overall satisfaction can be obtained by aggregating, respectively, the lower bounds and the upper bounds:

$$\begin{aligned}[n, p] &= \{\, \mathrm{AGG}(s_1, \ldots, s_q) \mid s_i \in [n_{c_i}, p_{c_i}] \,\} \\ &= \big[\, \mathrm{AGG}(n_{c_1}, \ldots, n_{c_q}),\ \mathrm{AGG}(p_{c_1}, \ldots, p_{c_q}) \,\big]\end{aligned} \tag{8}$$

In this paper, we apply the arithmetic mean for aggregation, leaving a deeper study of this choice for future work. The arithmetic mean is a reasonable choice because, on the one hand, it does not deliver as extreme results as alternative operators like the minimum or the maximum would. On the other hand, we do not need additional information like weights for the weighted mean that would need to be evaluated, as well. Nevertheless, we still expect the overall tendencies of the results to be robust against the choice of the aggregation operator to a certain degree, i.e., that the resulting data is that unambiguous such that only slight variations occur due to the choice of the aggregation operator.

It is important to note that, since the overall evaluation of a service is given in the form of an interval (Equation 8),

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

two services are not necessarily comparable to each other; instead, one service can *dominate* another one only in a Pareto sense (cf. Figure 4). Nevertheless, a total order on services, if required for futher processing, can be enforced by reducing intervals to scalars. For example, a simple approach is to map an interval $[n, p]$ to its risk-aversion-specific reduction r as follows:

$$r = \alpha\, n + (1 - \alpha)\, p \in [0, 1] \ , \tag{9}$$

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 a service evaluation. For $\alpha = 0$, the user is risk-affine and completely concentrates on the upper bound. In other words, for a risk-averse user, n gets the higher weight and for a risk-affine user, p gets the higher weight. Requesters can decide for a value on the basis of different factors, including not only their own character but also the context such as the domain of the service that they are searching (e.g., if a banking service is searched, the requester wants not to be very risky), or the size of the queried market (e.g., if a requester knows that there are many services provided on the current market that probably fit her interests, so that she does not need to take risks).

Nevertheless, as depicted in Figure 2, the fuzzy matching procedure returns the aggregated matching results based on the interval built by the n and p values. The general extent of the fuzziness is denoted by the range between n and p. The n-p-intervals show the important aspect of uncertainty that is not shown by a scalar because the size of the interval reflects the extent of induced uncertainty. Thus, these results can now be used to compare several services with each other in order to select the most fitting service under consideration of uncertainty and risk aversion. More concrete examples for situations where bad decision-making can be prevented due to the intervals are given in Section VIII.

The design of our matching algorithm leads to the fact that the size of the n-p-intervals define the extent of *provider-induced fuzziness* in terms of *uncertainty*. This can also be seen when taking into account the result for `c3` depicted in Figure 3 and in Table V-C. There is no requester-induced fuzziness but—due to the frequentist nature of the reputation data—the highest amount of uncertainty is induced (the interval is `[0,1]`). The *requester-induced fuzziness* induced within the requirements specification leads to *gradedness* and *vaguess* that has been reflected within the fuzzy sets as discussed in Section V-B1. Gradedness is also reflected within the matching result whenever either n or p take a value inbetween 0 and 1. By modifying her requirements specification, the requester is able to move the interval and, thereby, potentially decrease its size, thus, increase the occurring uncertainty. Thus, for easier usage, the matching results delivered by our approach can be annotated with a note on the fuzziness source and type to be taken into account by the requester. An example is "This result is affected by a *high* amount of *uncertainty* due to *provider-induced fuzziness*," with "high" being an interpretation based on the interval.

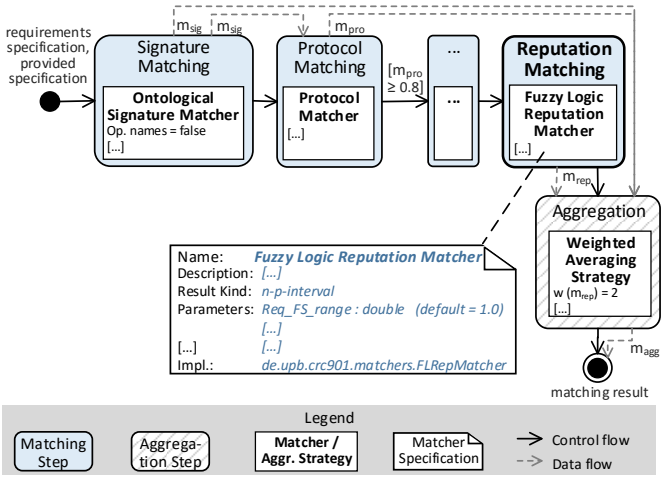


Fig. 5. Example Matching Process

VI. Integration into a Fuzzy Matching Process

In the following, we describe how the fuzzy reputation matching approach proposed in Section V can be combined with other matching approaches as part of a *matching process*. Moreover, we adopt the example of fuzzy performance matching in order to show how our fuzzy-logic-based approach can be applied to other service properties.

A. Matching Process Integration

In our earlier work [62], we presented MatchBox, a framework that supports a user in combining service matching approaches and reusing them within configurable matching processes. As soon as matchers have been integrated, MatchBox can be used to design and redesign matching processes arbitrarily often at model level, i.e., without having to reimplement any source code. This enables a dynamic adaption to different markets' and different requesters' requirements. Additionally, the possibility to work at model level provides the user with an abstraction that leads to a better overview than the source code itself and, thereby, also increases maintainability of matching processes.

Up to now, the MatchBox framework was designed with regard to traditional matching approaches that deliver traditional matching results, ignoring all kinds of fuzziness. In the following, we show how we can extend MatchBox by integrating fuzzy matching steps such that we gain a novel concept of *fuzzy matching processes*.

Figure 5 depicts a simplified visualization of a matching process model with an integrated `Reputation Matching` step. The reputation matching step executes a `Fuzzy Logic Reputation Matcher`, representing a matcher implemented on the basis of our approach proposed in Section V. In addition to the reputation matching step, there is a `Signature Matching` step, and a `Protocol Matching` step and potentially many more matching steps considering further functional and non-functional service properties. The signature matching step executes an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

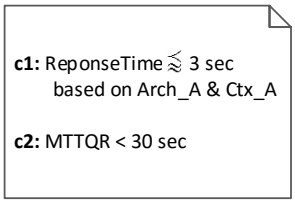
`Ontological Signature Matcher` with the configuration possibility to decide whether operation names (`Op. names`) should be considered during matching. The protocol matching step processes the matching result delivered by the signature matcher (m_{sig}) in order to deliver a protocol matching result (m_{pro}).

The control flow is a design decision made by the matching process designer. A good heuristic to choose an order of matching steps is to begin with matching steps that deliver results that serve as hard constraints and end with optional matching results. For example, in small markets with a low number of suitable services, users will put a higher focus on good matches regarding functional properties and view non-functional properties as a nice-to-have. In these cases, matching steps like protocol matching have to deliver good results, otherwise, the process terminates (indicated by the guard $[m_{pro} \leq 0.8]$). For this reason, we integrated the reputation matching step as the last matching step. On the other hand, if execution time is an issue (e.g., in very large service markets), an alternative solution would be to put fast matching steps first in order to filter out as many services as possible before executing the more time-consuming matching steps. In such cases, the reputation matching step should be one of the first steps within the matching process.

At the end of the example process, there is an aggregation step that aggregates the results from all matching steps using a weighted averaging strategy. If the user is very interested in the reputation data, then the weight for the reputation matching step should be higher than the weights of other steps (here: $w(m_{rep}) = 2$, with 1 being the (default) weight for all other steps, normalization is automatically performed by the framework). For fuzzy matching results, the aggregation requires special aggregation strategies that either work directly on fuzzy matching results and also deliver matching results. Alternatively, traditional aggregation strategies could be used if the fuzzy matching results are two scalars, as described in Section V-D, beforehand.

In addition, Figure 5 shows a simplified specification of the `Fuzzy Logic Reputation Matcher`. Each matcher to be used within a MatchBox matching process needs to be provided with such a matcher specification in order to enable the framework to support the user's modeling activities and to automatically validate and execute matching processes. For example, the `Result Kind` has to be known to the framework in order to delegate it to the aggregation strategy so that it can automatically process this kind of result when aggregating all matching results. Furthermore, the configuration possibilities for each matcher can be specified here. For example, the reputation matcher's parameter `Req_FS_range` refers to the fuzzy sets derived from soft constraints during the translation from the requirements specification into fuzzy sets. The parameter defines how "steep" the gradient will be, i.e., the possibility range. As a default (and also in the example shown in Section V-B1), this parameter is set to 1.0.

In particular, such matching processes enable us to combine matching of structural and behavioral service properties



(a) Requirements Specification

Entity	Metric	Architecture	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
...	...	...	...	...

(b) Service Information

Fig. 6. Performance Matching Example

with non-functional properties. For example, we could design a matching process where matching steps with respect to structural and behavioral properties act as filters (see example discussed above), such that only services matching these properties will be considered within the matching steps for non-functional properties. The first could then be viewed as hard constraints, and the latter as soft constraints.

More detailed information on MatchBox, without concepts on fuzzy matching, can be found in our earlier work [62].

B. Fuzzy Performance Matching

The reputation matching approach presented in this paper is only an example our fuzzy matching concepts can be applied to. In this section, we want to briefly illustrate how to apply our concepts to other matching approaches. As a second example, we selected performance matching based on performance prediction [11] and quality metrics [9]. Figure 6 shows an example explained in the following.

The representation of requirements and information about a service in this figure is similar to the representation in Figure 1(a). Here again, requirements consist of several conditions, as depicted in the left part of the figure. In this example, the conditions target response time (RT) and mean time to quality repair (MTTQR) [9]. RT is the time that a service requires to serve a response for a request. MTTQR is a metric to measure the elasticity of a software service, i.e., the mean time which a service needs to adapt to increasing or decreasing workload by, for example, scaling its execution environment out or in.

Information about how a market's services perform with respect to these metrics is depicted in Figure 6(b). However, performance metrics are always specific to a service's (a) internal architecture and (b) external context [11]. The internal architecture is determined by the service's software architecture, its implementation, and its execution environment. The external context is determined by the service's input (work) and request rate (load). For example, an image processing service's response time is probably much lower if the service is executed in a high-performance compute center compared to the service being executed on a standard mobile phone. Furthermore, the response time depends on the number of concurrent service requests. Thus, the table in Figure 6 contains different rows for different architectures and different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

contexts. The representation of execution environments and usage scenarios is heavily simplified in the table. Actually all abbreviations (e.g., "Arch_A", "Ctx_A") refer to more complex models as known from performance engineering (e.g., [11]).

Fuzziness may occur in different manners. For example, Condition `c1` contains an approximation operator which may lead to requester-induced fuzziness. Furthermore, the requester did not specify the internal architecture for `c2`. The internal architecture may not even be fixed but vary in case of elastic services such as 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. In addition, there may be provider-induced fuzziness if a value is not available for a certain environment or for a certain usage scenario. Thus, similar as in our reputation example, it is recommendable to model performance specifications as fuzzy sets, as described in [10], and to apply our approach as described in Section V.

Again, a fuzzy performance matching approach like this can be integrated as part of a matching process as explained in Section VI-A.

C. Generalizability

Performance is only one further example of how to apply our approach to other service properties than reputation. In general, all specifications from which we can derive numerical data can be modeled using fuzzy sets. This is most common for non-functional properties of a service (in addition to reputation and performance, common examples are reliability or privacy-related properties). However, also functional properties can be modeled accordingly.

As an example, consider structural service specifications in the form of operation signatures. Service matching approaches based on signatures often focus on (ontological) parameter types that a service takes as an input or returns as an output (e.g., [6]). Incompleteness and imprecisions could come into play for both involved roles again. They occur on the requester side whenever she specifies requests where she leaves certain parameters open ("I want a room reservation service but I don't care about the detailed input format"). Likewise, it appears on the provider side whenever the domain knowledge the ontology of data types is based on lacked certain knowledge about the relation between different terms. In order to apply our approach, both the number of parameters as well as the distance between different types (e.g., based on ontological relations) can be transformed into numerical values. As another example, consider behavioral specifications in the form of protocols modeled with finite automata. Here again, the requester side could leave room for variations (e.g., if the protocol was derived from informal requirements specifications), while the provider side could be incomplete (e.g., if the protocol was derived based on reverse engineering techniques). Examples for quantifiable properties based on such models are the number of iterations or the length of traces in general. In both examples, our approach can be applied.

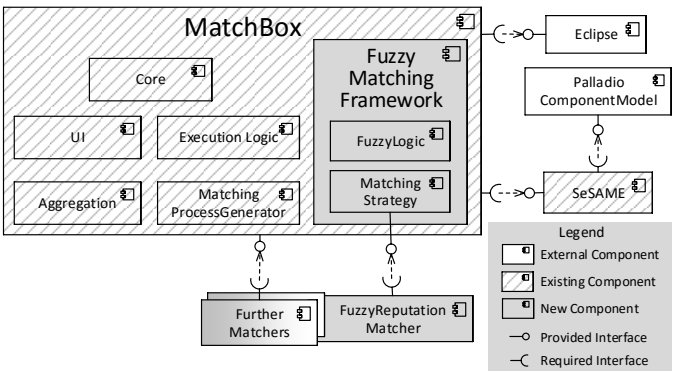


Fig. 7. Architecture of the Implementation based on MatchBox

VII. Implementation

Our approach has been implemented as a part of the MatchBox implementation [81] in the form of an Eclipse-based framework for comprehensive, configurable matching processes. The implementation is independent of the particular matcher (e.g., the framework can be used for reputation matching but also for other approaches such as performance matching). This has been achieved by designing an architecture (see Figure 7) based on three substitutable layers: fuzzy logic layer, matching strategy layer, and the concrete matcher. In order to evaluate the approach presented in this paper, we used this framework in combination with a reputation matcher. This matcher accepts input models in the form of the specifications described in Sections II and V, transforms them into fuzzy sets, and returns a matching result to the user.

Input specifications (i.e., requirements specifications and service specifications or other service data) are specified within our SeSAME framework [3], [82] which is based on the Palladio Component Model [11]. Accordingly, all metamodels are specified using the Eclipse Modeling Framework (EMF).

All matching results produced by MatchBox are stored and presented in a hierarchical format collecting everything a user may want to know about the performed matching process. The MatchBox UI displays matching processes and matching results in a graphical way.

Furthermore, MatchBox matching process implementations are highly configurable. This also holds for the fuzzy matching framework. For example, membership functions of fuzzy sets can be derived in a more soft or a more strict way, depending on the specified matcher configuration (cf. Figure VI-A).

For more information on the tool including screenshots, a screencast, and installation instructions, we refer to the MatchBox website [81].

VIII. Evaluation

In the following, we report about four experiments we performed using the implementation presented in Section VII within the scope of our collaborative research center "On-The-Fly Computing" [80]. The data used for the evaluation as well as the complete results can be found online [63].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

A. Goals and Method

Our evaluation goal was to investigate whether our fuzzy matching approach supports the service requester's decision-making by (a) explicitly incorporating fuzziness during service matching and by (b) reflecting this fuzziness within the matching result. Following the guidelines of evidence-based software engineering by Kitchenham et al. [36], we formulated our evaluation question as follows: "Do interval matching results computed on the basis of fuzzy logic improve a service requester's decision-making in the presence of fuzziness?"

In order to answer this question, we applied our approach as described in this paper to a set of example specifications in four different experiments:

- *Experiment 1*: The first experiment is a toy example, where we tested our evaluation setting in general and the metrics we use to measure improved decision-making in particular. These metrics are *distinguishability* and the *uncertainty* represented within a matching result. We consider these two criteria as the most important in the requester's decision-making when selecting a service.
- *Experiment 2*: This experiment is the main focus of our case study. It uses the setting of Experiment 1 but in a larger scope and based on real data. This experiment is supposed to answer the main evaluation question.
- *Experiment 3*: In this experiment, we investigate scalability as one property being a basic for the applicability of our approach. Here, we focused on scalability in terms of the amount of matched data.
- *Experiment 4*: The last experiment investigates both the transferability of our approach to other service properties and, in addition, the correctness of matching results compared to results reported in the literature. By correctness, we mean that the computed matching results highly correspond to the reported results.

Addressing on *distinguishability* and the *uncertainty representation* as the main focus of our evaluation is in line with acknowledged decision-making literature. For example, Ellsbergs studies [23], [24] discussed the importance of uncertainty in decision-making by showing that people prefer taking a known risk rather than taking an unknown risk even though the probability for a better outcome may be low. Others agree that coping with uncertainty "lies at the heart of making a decision" because uncertainty constitutes a "major obstacle to effective decision-making" [50] and that human choices depend on how much relevant information is missing [28]. In all these studies, distinguishability is a foundation to decision-making, too.

In the following, we give more detailed information about each experiment and its results.

B. Experiment 1: Toy Example

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 (see Figure 1(a)). All specifications in this experiment have been constructed manually.

Following the example by Esfahani et al. [25], we compared our approach to an alternative approach that does not handle fuzziness. The alternative approach is the *traditional matching approach* introduced in Section II-B that uses exact matching. The traditional approach ignores all induced 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 function as used for aggregating the results in our new approach.

Recall that, in our new approach, the evaluation of a service is represented in the form of a necessity/possibility interval $[n, p]$ or, given a level of risk aversion, a score r as defined in Equation 9. Our experiments are intended to show some advantages of this representation in comparison to the traditional results, notably the following:

- *Distinguishability*: Comparing requirements and services in a fine-granular way using fuzzy sets will often allow for distinguishing services that match equally well under the traditional approach. Given the evaluation of a set of services, we define the degree of distinguishability as the probability that a pair of two services selected (uniformly) at random from all pairs (and a level of risk aversion randomly in $[0, 1]$) can be compared, i.e., do not have the same score.
- *Uncertainty representation*: The width of the interval $[n, p]$ informs about the uncertainty of the evaluation. In particular, the upper bound p informs about the potential of a service. In principle, the suboptimality of a service with evaluation $[n, p]$ is not proven unless there is another service with evaluation $[n', p']$ such that $p < n'$; in this case, we say that the former is *dominated* by the latter (note that the term "dominated" here does not refer to Pareto-dominance but to a weaker concept of domination).

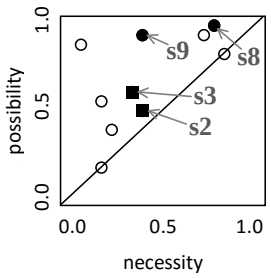
Figure 8(a) shows the matching results for each service. The traditional results (TR) are gradual to a specific extent because of the aggregation of results per condition into an overall result. Nevertheless, as can be seen, we cannot differentiate between some of the services. For example, `s2` and `s3` both have a matching result of 0.4, and `s8` and `s9` both have a score of 0.8. Thus, selecting one of the services on the basis of the traditional results becomes a difficult task for the requester. The degree of distinguishability for the traditional results – calculated based on the metric explained above – is 0.8.

For comparison, the degree of distinguishability for the fuzzy results (FR) from Figure 8(a) is 1. This means that all services can be distinguished from each other on the basis of the new matching results which are represented by $[n, p]$ intervals. Figure 8(a) shows the $[n, p]$ intervals as well as the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

Service	TR	FR	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) Traditional results (TR) compared to new, fuzzy results (FR)



(b) Fuzzy results depicted in a necessity/possibility diagram

Fig. 8. Results of Experiment 1

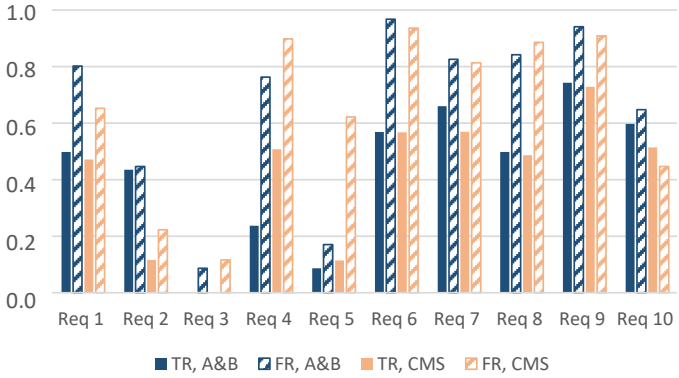


Fig. 9. Degrees of Distinguishability (Exp. 2)

best and worst rank possible for each service. The best rank is obtained if the service assumes its upper bound p while all others assume their lower bounds n, and vice versa for the worst rank. For example, `s10` could be on rank one, if full information would lead to a matching degree equal to the upper bound of 0.85. However, full information could also have negative consequences for this services, i.e., the final matching degree of `s10` could develop towards the lower bound of 0.1 and the other services develop towards higher degrees. In this situation, `s10` would get the very worst rank. In contrast, Service `s8` is a good choice in any way as it could be ranked first, but it will never be worse than Rank 4. This information is valuable not only for the requesters but also for service providers because—since we deal with provider-induced fuzziness here—it allows them to decide whether they should spend effort in providing more information. In the case of reputation data, the providers could, for example, invest into special test licenses for their services so that they motivate more users to publish ratings.

Moreover, Figure 8(b) depicts the results in a necessity/possibility diagram, where each service is represented as a point. The more certain the evaluation $[n, p]$ of a service, the closer it lies to the diagonal (because $n \approx p$). 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. These matching results contain a high amount of fuzziness reflected by large intervals. Obviously, the representation based on $[n, p]$ intervals provides more information than the traditional results in the table and, thereby, supports decision-making much better.

For example, consider again the services `s2` and `s3` (represented by a black filled square) and `s8` and `s9` (represented by a black filled circle) within the necessity/possibility diagram. `s8` being located closer to the right upper corner than `s9` shows that `s8` is the better choice than `s9`, in any case. `s2` and `s3` do not dominate each other and are both rather average results.

C. Experiment 2: TrustRadius Ratings

The setting for Experiment 2 is similar to the first one except for the use of real input data and real requesters instead of simulated data. We collected ratings from the software rating website *TrustRadius.com* [79]. At TrustRadius.com, software is classified into different categories regarding the functionality and rated by people according to different properties (e.g., usability, availability, performance, and overall recommendation). For example, at the time of evaluation, TrustRadius listed 34 services with in total 520 ratings in the category "Content Management Systems" (CMS).

We matched these services against 25 requirements specifications (including one to five conditions) that have been specified by computer science students and graduate students that had no knowledge about the internals of the matching algorithm. Figure 10 shows two examples specified by two different test persons. The properties allowed within the conditions in these requirements specifications were limited to the kinds of ratings TrustRadius contained most data for: Overall Rating, Usability Rating, Availability Rating, and Performance Rating. In addition to the Content Management Systems, we repeated this experiment for the category "Accounting and Budgeting Services" (A&B), which comprised 23 services with a total of 615 ratings. In total, this leads to 850 matching results for CMS and 575 matching pairs for A&B.

Figure 9 provides a summary of the degrees of distinguishability of the matching results for the first ten requirements specifications within the two data sets (A&B and CMS). The results are shown both for the traditional results (TR) and for the new fuzzy results (FR). The higher the bar, the better is the distinguishability among the results. For example, the distinguishability of 0 for the traditional results for `Req 3` shows that all services are associated with the same result in this case. In contrast, the fuzzy results at least allows to distinguish between some different results. This trend is also shown for the other requirements. As can be seen, the fuzzy approach consistently improves the distinguishability across almost all requirements specifications. This fact is also confirmed by the Wilcoxon signed-rank test. We conducted this test over the pairs of distinguishabilities of the tradi-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

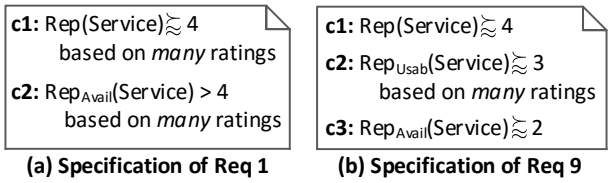


Fig. 10. Two of the example requirements specifications used within Experiments 2 and 3

tional and the fuzzy approach. The null hypothesis that the median difference between the pairs of distinguishabilities is zero, can be rejected as the obtained p-value is smaller than 1×10^{-4} ($p = 9.556 \times 10^{-5}$). This finding is in line with our expectations. In the following, we discuss a selection of the results in more detail. For the complete lists of results, refer to the website [63].

As an illustration, Figure 11 shows the diagrams for Requester 1, 6, 8 and 9 in the A&B and CMS setting, respectively. As can be seen, the specification of Requester 1 lead to more precise evaluations of the services (points are lying close to the diagonal) than those of the three others. In line with Figure 10(a), the diagram also reflects that `Req 1` is quite demanding, since even the highest possibility degrees do not exceed 0.5. Nevertheless, such ratings are of course more useful than scores of 0 that are likely to be obtained in the traditional approach. As opposed to this, the requirements of `Req 9` (see Figure 10(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, suggesting some obvious choices for this requester. Alternatively, she may consider tightening her requirements.

When inspecting some selected results in even more detail, we 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 only distinguished between three groups of services: services that do not match at all, 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 are the well known CMS services Joomla! and Drupal. In contrast, considering the interval results (see Figure 11, Req 6, right diagram), Joomla! ([0.4,0.65]) dominates Drupal ([0.217,0.55]). Joomla! is potentially the better match and more certainly a good match. Still, for both services, the possibility value, i.e., the upper threshold, is rather low. 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, according to the interval results ([0.375, 0.75]), potentially a much better match, even though Joomla! is more certainly a good match.

On the basis of these results, when choosing among these two services, a risk-averse requester would choose WordPress, and a risk-affine requester would rather choose Joomla!.

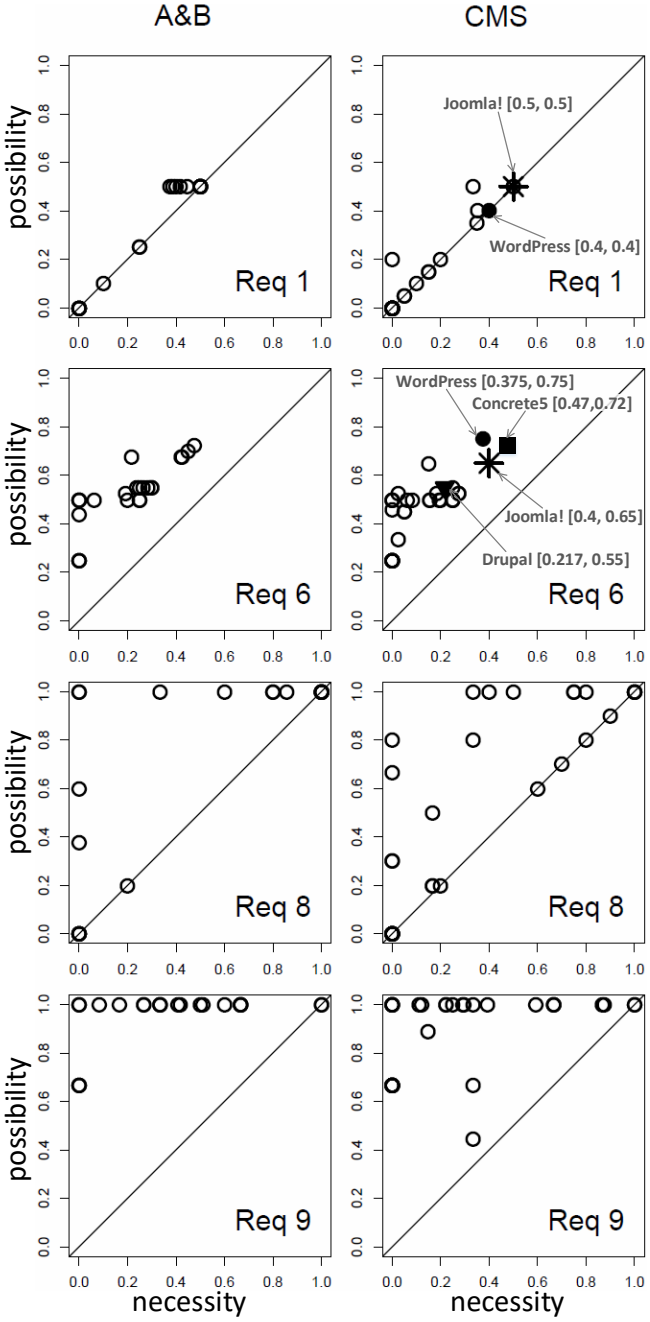


Fig. 11. Example necessity/possibility diagrams of Experiment 2

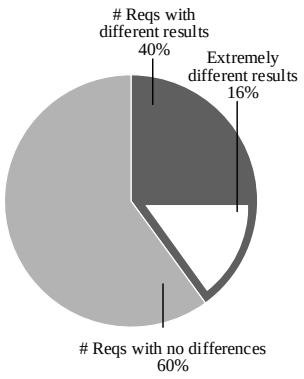
Thereby, the interval results help requesters in finding their best choice depending on their own characteristics much better than the traditional results. In fact, there is another service, Concrete5, dominating Joomla! and building a small Pareto front with WordPress. So, typically, the real decision would probably be between these two services.

Furthermore, the result again strongly depends on the requirements specification. In the results for most of the other requirements specifications, Joomla! clearly dominates WordPress. As an example, consider Figure 11 again (Req 1, right diagram), where Joomla! dominates all other services

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

	# Differences in Top1	Top3	Top5
Req 1	0	0	2
Req 2	1	2	2
Req 3	0	0	0
Req 4	0	0	0
Req 5	1	3	5
Req 6	1	3	5
Req 7	0	0	2
Req 8	0	0	1
Req 9	0	0	0
Req 10	0	0	0
Req 11	0	0	0
Req 12	0	0	0
Req 13	0	0	0
Req 14	0	0	1
Req 15	0	0	0
Req 16	1	3	5
Req 17	0	0	0
Req 18	0	0	0
Req 19	0	0	0
Req 20	0	0	0
Req 21	0	1	3
Req 22	0	1	3
Req 23	0	0	0
Req 24	0	0	0
Req 25	0	0	0
# (>0)	4	6	10

(a) List of differences



(b) Proportions of differences

Rank	Ranked by n	Ranked by p
1	Concrete5 [**0.47**,0.72]	WordPress [0.375,**0.75**]
2	Joomla! [**0.4**,0.65]	Concrete5 [0.47,**0.72**]
3	WordPress [**0.375**,0.75]	Joomla! [0.4,**0.65**]
4	Cascade S. [**0.275**, 0.525]	ExpressionE. [0.15, **0.65**]
5	Kentico8 [**0.275**,0.525]	Sitecore [0.25, **0.55**]

(c) Ranking for Req 6

Fig. 12. Results of Experiment 2 with ranks within CMS case

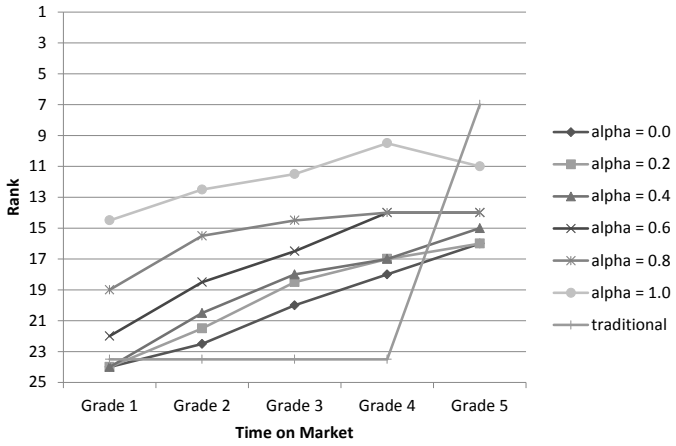


Fig. 13. Results of Experiment 2 with varying completeness grades

with a result of [0.5,0.5], while WordPress got a result of [0.4,0.4]. These differences among the results for the same services but set in relation with different requirements specifications indicate that it makes sense to work with more complex requirements specifications allowing the requesters more freedom than related works do.

In general, when considering the traditional results, some services often "hide" within a group of equal matching results, while the interval results more clearly identify them as good candidates depending on the risk a requester is willing to take.

This fact can also be observed when comparing different rankings. We can either rank the services by their necessity value in descending order, or we can rank them by their possibility value in descending order. In the cases where uncertainty is induced, the necessity-ranking and the possibility-ranking can be expected to be different from each other. Figure 12(a) lists the differences in these rankings with respect to the top most promising CMS-services (top 1, top 3, and top 5) for each requirements specification. The numbers in the single lines are the numbers of considered ranks that were allocated with different services. For example, a 5 in the top-5-column means that 5 out of 5 positions differ between the two rankings. As can be seen, in 4 out of 25 cases, the top 1 service differs in the two rankings. These are exactly those cases, where the necessity/possibility diagrams depict (at least small) Pareto fronts. Considering the top 3 results, the rankings differ in 6 out of 15 cases and in the top 5 results, the ranking differs in even 10 out of 15 cases. These numbers are also visualized in the pie chart in Figure 12(b). As shown there, in 40% of the cases, we get different results due to uncertainty. Thus, in 40% of the cases, the decisions made by the requesters are improved when considering our interval results. In 16%, we even get extremely different results, meaning that even the services at the top position of the rankings differ. One example is the ranking for `Req 6`, displayed in Figure 12(c). On the one hand, when ranking the results based on the necessity values, i.e., the ranking a risk-averse user would prefer, `Concrete5` gets the first rank, `Joomla!` gets the second, and `WordPress` gets the third. On the other hand, when ranking the results based on the possibility values, i.e., the ranking a risk-affine user would prefer, `WordPress` gets the first rank, `Concrete5` gets the second, and `Joomla!` gets the third. These findings are also in line with the corresponding necessity/possibility diagram in Figure 11. We can find even more extreme differences. For example, when the necessity-ranking for `Req 5`, ranks services into the top 5 that, however, receive the very worst ranks in the possibility-ranking. Such examples show how important the consideration of lower and upper bounds, and, consequently, the consideration of uncertainty is. Here, the consideration of uncertainty clearly prevents potentially bad decisions made by the requester.

With a final experiment, we like to show that, compared to the traditional approach, our fuzzy approach tends to be more robust in situations where information is scarce, and hence to yield a more stable and reliable assessments of services. To this end, we look at how the rank of a service changes depending on how often it has been rated. More specifically, simulating a scenario in which a new service enters the market, we start with the first quarter of the ratings (Grade 1) and successively add the other quarters till reaching the total number of ratings available in the data set (Grade 5). These grades represent the x-axis of Figure 13. The y-axis represents the rank within the group of services matched to `Req 8`. The graph shows the ranks for service `OU Campus` at different grades of completeness and for different levels of risk-aversion of the requester. The risk-aversion is calculated with Equation 9.

As an example, Figure 13 shows results for the service `OU`

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

Campus and Req 8. The graph depicts the mean rank of this service at different grades of completeness, and for different levels of risk-aversion of the requester (calculated with Equation 9). Overall, OU Campus appears to be a mediocre service, though its ranking gets better with an increasing number of ratings. Moreover, as expected, the ranking drops when increasing the level of risk-aversion, although the ranks for all levels of risk-aversion converge as the grade of completeness increases (recall that, in the limit, the interval $[n, p]$ reduces to a single point). In summary, the ranking of the service depends on the amount of information available and the risk-aversion of the requester, but the changes are moderate.

In contrast to this, the traditional matching approach shows a much more drastic behavior. Due to its binary nature, the request is either matched or not, hence the service is either ranked high or low. In this case, OU Campus does not reach the top 20 until Grade 5 of completeness, when it suddenly jumps to Rank 7 (the ranks plotted here are mean ranks; as discussed earlier, the possibly large number of services with equal rating is another problem of the traditional approach). Thus, while the fuzzy approach enables us to identify and reliably assess (partially) matching services even in situations when only limited data is available, the traditional matching is much more prone toward the incompleteness of the underlying data.

D. Experiment 3: Scalability

In a third experiment, we tested how our approach scales to a broad range of input data. For this purpose, we generated 1000 random requirements specifications and matched them against all services in all categories from TrustRadius. These were, at evaluation time, in total 924 services with altogether 10907 ratings. As a consequence, we got 924000 matching results. The mean total runtime for all 924000 matching runs amounts to 642 seconds (10.7 minutes). We measure this on a system with Windows 8.1, 64-bit with 8GB RAM and an Intel(R) Core(TM) i7-2640M CPU @2.80GHz. More information about the data are given on the website [63].

In light of the high number of input data, we find this result very satisfying. In reality, we do not expect that so many services need to be matched at the same time on the basis of reputation data, especially since many services can be filtered out by earlier matching steps that check simpler properties. Furthermore, manual inspection showed that much of the measured time is spent on saving matching results in the form of complex object trees that are constructed in order to provide the user with as much information about the matching as possible.

E. Experiment 4: Performance Matching

Our fourth experiment is based on data from a performance simulation of a self-adaptive service, as described in [10]. In [10], three different service architectures of a cloud-based web service have been simulated. The three architectures vary from each other in terms of their autonomous resource provisioning. Service S_a can provision or deprovision resources like virtual machines every 5s, Service S_b every 10s, and Service S_c every 20s. The provisioning and deprovisioning times are limited by the rate each system architecture monitors its own mean response time, i.e., the frequency of its self-adaptation feedback loop. The performance simulation provides predicted response times and the predicted MTTQR for each self-adaptive service architecture in a specified workload context. For all three services, service requests with an exponentially distributed interarrival rate $\lambda = 4\frac{1}{s}$ for 100 seconds have been simulated. Within these 100 seconds, the authors obtained 415 response time values for S_a, 393 values for S_b, and 376 values for S_c. Furthermore, one MTTQR value for each of the services S_a, S_b, and S_c has been obtained. According to the results reported in [10], S_a is the best performing service with respect to response time and MTTQR. It dominates S_b while S_b again dominates S_c.

For our experiment, we used the same simulation data as an input for our fuzzy performance matcher in order to compare our matching results to the reported results. Table III shows the matching results for four manually specified requirements specifications Req 1′, Req 2′, Req 3′, and Req 4′ matched to the service specifications of S_a, S_b, and S_c. The requirements specifications increase in their strictness starting with a requested response time lower than 3s and a requested MTTQR lower than 40s in Req 1′ down to a requested response time lower than 1.5s and a requested MTTQR lower than 10s in Req 4′.

TABLE III
MATCHING RESULTS FOR EXPERIMENT 4

	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 [10]	1	2	3

As we can see in the results, our matching results are in line with the results reported in [10]. 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 not very strict.

These results show that our matching results fit to the reference results. However, in addition, our results also provide the extra information of fuzziness. For example, for Req 3′, S_a is not necessarily a full match. Due to imperfect information, we can only determine that the matching result is higher than 0.9097. This information is especially helpful for a very risk-averse requester. As another example, the provider of S_b can learn that providing more information can potentially improve her ranking in general, however, providing more information definitely is not sufficient to rank her service best for most of the example requirements specifications. She would also need to adapt the service itself (e.g., its implementation) in order to improve her revenue.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2016.2632115, IEEE Transactions on Software Engineering

F. Discussion and Limitations

Addressing the two metrics *distinguishability* and the representation of *uncertainty* as discussed above, allows us to conclude from our evaluation results that a user can make a more informed decision based on the matching results delivered by our new approach compared to traditional ones. Especially the results of Experiment 2, based on real service ratings given by real users, show that (a) the distinguishability that comes with the interval results as well as (b) the consideration of uncertainty can prevent bad decisions when selecting a service.

Furthermore, we showed in Experiment 3 that our approach is also able to handle a big amount of data, which makes it applicable in practice. In addition, Experiment 4 showed the correctness of our matching results compared to reference results from the literature. Experiment 4 also showed that the matching approach can be used not only for reputation specifications, but also for specifications of other properties, like performance.

Nevertheless, there are still threats to the validity of our evaluation. For example, although we were able to apply our reputation matching approach to a lot of real data, we still got relatively few ratings per service. This leads to a number of large intervals (almost [0,1]). However, compared to the traditional results, the user is at least provided with the information that the matching result is almost unknown, instead of leading the user to trust an unreliable result which may lead to a bad choice. Using the interval results, the user has the possibility to decide on her own how much risk she is willing to take.

Furthermore, the traditional matching approach we used as a baseline could also vary in its functionality. For example, a matching approach leading to more fine-grained results would also make an interesting comparison. The same holds for the performance simulations used for comparison. In the future, we need to find more possibilities to compare our matching results to other matching approaches. However, we still need to take into account challenges like the availability of example specifications and the comparability of fuzzy matching results to results in simpler formats.

Another problem when evaluating fuzzy matching results is that there is no "ground truth". On the basis of reputation data, we cannot determine which service is truly the best selection or what is truly the correct degree of induced fuzziness. Experiment 4 addresses this problem to some extent as we have reported values for other service properties, like performance, that we can use for evaluating whether the tendencies of the fuzzy matching results correspond to the reported values. But still, we cannot identify false positives or false negatives, as it is often done for traditional matching results that only distinguish between "match" and "mismatch". Furthermore, the "best result" also varies from user to user, so it is hard to incorporate such knowledge into an evaluation.

One limitation to our current approach is the transformation of specifications into fuzzy sets. At the moment, we are using the same transformation for all requesters, although different requesters may have different interpretations of soft thresholds. It should be investigated whether requesters should be able to model the fuzzy sets used for matching on their own under consideration of the extra effort for requesters. In general, the appropriate level of complexity for specification languages used in this context is subject to research. The question is not only how much complexity requesters can handle but also how much existing service markets support. For example, many of today's app stores support rather simple reputation models (e.g., no differentiation between multiple contexts).

IX. Conclusion

In this paper, we have introduced an approach for fuzzy service matching on the basis of imperfect information. We presented a systematic procedure based on fuzzy logic and possibility theory that extends traditional approaches from the areas of service discovery and component retrieval. In particular, our approach goes beyond related work in terms of informativeness of the returned matching result. Moreover, unlike our approach, related approaches do not distinguish between fuzziness in the sense of graded user satisfaction and fuzziness in the sense of incomplete information about service properties. Using our approach, the user can make a more informed decision when choosing among provided services. By explicitly considering imprecise and incomplete service specifications, our approach makes service matching applicable in practice as it considers more realistic assumptions than traditional approaches.

There are still open issues in this area. First of all, we need to apply our approach to other kinds of service descriptions (e.g., security requirements or behavioral specifications) and perform corresponding case studies. Moreover, we plan to address other fuzziness types and sources, e.g., approximations and algorithm-induced fuzziness. Along with these concepts, the user can be provided with matching results that indicate whether the input specifications or the matching algorithm induced fuzziness. Based on this, the user could also be provided with proposals for how to reduce the occurred fuzziness, if possible. Furthermore, our approach could benefit from more advanced techniques for aggregating evaluations on different conditions into an overall evaluation. The weighted average we are currently using for that purpose is not very flexible; in particular, one could imagine that a requester will not only average but also adopt conjunctive or disjunctive modes of aggregation [69].

Acknowledgment

This work was supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901). We also would like to thank our student assistants Ludovic Larger, Paul Börding, and Melanie Bruns for contributing to the implementation and evaluation of our approach. Last but not least, special thanks go to TrustRadius for allowing us to use their software rating data for the evaluation of our approach.

REFERENCES

[1] M. Almulla, K. Almatori, and H. Yahyaoui. A QoS-based Fuzzy Model for Ranking Real World Web Services. In *Proc. of the 18th IEEE Int. Conf. on Web Services*, ICWS'11, pages 203–210. IEEE, 2011.

[2] S. Arifulina, M. C. Platenius, M. Becker, G. Engels, and W. Schäfer. An Overview of Service Specification Language and Matching in On-The-Fly Computing, Technical Report. Technical Report tr-ri-15-347, Heinz Nixdorf Institute, University of Paderborn, July 2015.

[3] S. Arifulina, S. Walther, M. Becker, and M. C. Platenius. SeSAME: Modeling and Analyzing High-quality Service Compositions. In *Proc. of the 29th ACM/IEEE Int. Conf. on Automated Software Engineering*, ASE'14, pages 839–842. ACM, 2014.

[4] D. Bacciu, M. G. Buscemi, and L. Mkrtchyan. Adaptive Service Selection–A Fuzzy-valued Matchmaking Approach. Technical report, Università di Pisa, 2009.

[5] D. Bacciu, M. G. Buscemi, and L. Mkrtchyan. Adaptive Fuzzy-valued Service Selection. In *Proc. of the ACM Symposium on Applied Computing*, SAC'10, pages 2467–2471. ACM, 2010.

[6] L. Bai and M. Liu. A Fuzzy-set based Semantic Similarity Matching Algorithm for Web Service. In *Proc. of the 5th Int. Conf. on Services Computing*, SCC'08, pages 529–532. IEEE, 2008.

[7] L. Bai and M. Liu. Fuzzy Sets and Similarity Relations for Semantic Web Service Matching. *Computers & Mathematics with Applications*, 61(8):2281–2286, 2011.

[8] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy Goals for Requirements-Driven Adaptation. In *Proc. of the 18th IEEE Int. Requirements Engineering Conf.*, RE'10, pages 125–134. IEEE, 2010.

[9] M. Becker, S. Lehrig, and S. Becker. Systematically Deriving Quality Metrics for Cloud Computing Systems. In *Proc. of the 6th ACM/SPEC Int. Conf. on Performance Engineering*, ICPE'15, pages 169–174, New York, NY, USA, 2015. ACM.

[10] M. Becker, M. Luckey, and S. Becker. Performance Analysis of Self-Adaptive Systems for Requirements Validation at Design-Time. In *Proc. of the 9th ACM SigSoft Int. Conf. on Quality of Software Architectures*, QoSA'13, pages 43–52. ACM, 2013.

[11] S. Becker, H. Koziolek, and R. Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[12] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from Applying the Systematic Literature Review Process Within the Software Engineering Domain. *Journal of Systems and Software*, 80(4):571–583, 2007.

[13] K.-M. Chao, M. Younas, C.-C. Lo, and T.-H. Tan. Fuzzy Matchmaking for Web Services. In *Proc. of the 19th Int. Conf. on Advanced Information Networking and Applications*, volume 2 of *AINA'15*, pages 721–726. IEEE, 2005.

[14] L. Cheung, L. Golubchik, N. Medvidovic, and G. Sukhatme. Identifying and Addressing Uncertainty in Architecture-Level Software Reliability Modeling. In *Proc. of the Int. Parallel and Distributed Processing Symposium*, pages 1–6. IEEE, 2007.

[15] I. Constantinescu, W. Binder, and B. Faltings. Flexible and Efficient Matchmaking and Ranking in Service Directories. In *Proc. of the 12th IEEE Int. Conf. on Web Services*, ICWS'05, pages 5–12. IEEE, 2005.

[16] K. Cooper, J. W. Cangussu, R. Lin, G. Sankaranarayanan, R. Soundararadjane, and E. Wong. An Empirical Study on the Specification and Selection of Components Using Fuzzy Logic. In *Proc. of the Int. Symposium on Component-Based Software Engineering*, CBSE'05, pages 155–170. Springer, 2005.

[17] I. da Silva and A. Zisman. Decomposing Ratings in Service Compositions. In *Proc. of the 11th Int. Conf. on Service-Oriented Computing*, volume 8274 of *ICSOC'13*, pages 474–482. Springer Berlin Heidelberg, 2013.

[18] M. De Cock, S. Chung, and O. Hafeez. Selection of Web Services with Imprecise QoS Constraints. In *Proc. of the IEEE/WIC/ACM Int. Conf. on Web Intelligence*, pages 535–541. IEEE Computer Society, 2007.

[19] H. Dong, F. Hussain, and E. Chang. Semantic Web Service Matchmakers: State of the Art and Challenges. *Concurrency and Computation: Practice and Experience*, 25(7), 2012.

[20] D. Dubois, L. Foulloy, G. Mauris, and H. Prade. Probability-Possibility Transformations, Triangular Fuzzy Sets, and Probabilistic Inequalities. *Reliable computing*, 10(4):273–297, 2004.

[21] D. Dubois and H. Prade. The Three Semantics of Fuzzy Sets. *Fuzzy Sets and Systems*, 90(2):141–150, 1997.

[22] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, 1993.

[23] D. Ellsberg. Risk, Ambiguity, and the Savage axioms. *The Quarterly Journal of Economics*, 75(4):643–669, 1961.

[24] D. Ellsberg. *Risk, Ambiguity and Decision*. Garland Publishing, Inc., 2015.

[25] N. Esfahani, E. Kouroshfar, and S. Malek. Taming Uncertainty in Self-Adaptive Software. In *Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conf. on Foundations of Software Engineering*, ESEC/FSE'11, pages 234–244. ACM, 2011.

[26] N. Esfahani and S. Malek. Uncertainty in Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013.

[27] G. Fenza, V. Loia, and S. Senatore. A Hybrid Approach to Semantic Web Services Matchmaking. *Int. Journal of Approximate Reasoning*, 48:808–828, 2008.

[28] C. R. Fox and A. Tversky. Ambiguity Aversion and Comparative Ignorance. *The Quarterly Journal of Economics*, 110(3):585–603, 1995.

[29] D. Garlan. Software Engineering in an Uncertain World. In *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research*, pages 125–128. ACM, 2010.

[30] M. Grabisch, J. Marichal, R. Mesiar, and E. Pap. *Aggregation Functions*. Cambridge University Press, 2009.

[31] C.-L. Huang, K.-M. Chao, and C.-C. Lo. A Moderated Fuzzy Matchmaking for Web Services. In *Proc. of the Fifth Int. Conf. on Computer and Information Technology*, pages 1116–1122. IEEE, 2005.

[32] C.-L. Huang, C.-C. Lo, K.-M. Chao, and M. Younas. Reaching Consensus: A Moderated Fuzzy Web Services Discovery Method. *Information and Software Technology*, 48(6):410–423, 2006.

[33] C.-L. Huang, C.-C. Lo, Y. Li, K.-M. Chao, J.-Y. Chung, and Y. Huang. Service Discovery Through Multi-Agent Consensus. In *Proc. of the Int. Workshop Service-Oriented System Engineering*, pages 37–44. IEEE, 2005.

[34] B. Jeong, H. Cho, and C. Lee. On the Functional Quality of Service (FQoS) to Discover and Compose Interoperable Web Services. *Expert Systems with Applications*, 36(3):5411–5418, Apr. 2009.

[35] B. Kitchenham and S. Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering. *Technical Report EBSE 2007-001*, 2007.

[36] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based Software Engineering. In *Proc. of the 26th Int. Conf. on Software Engineering*, ICSE'04, pages 273–281. IEEE Computer Society, 2004.

[37] E. Klement, R. Mesiar, and E. Pap. *Triangular Norms*. Kluwer Academic Publishers, 2002.

[38] M. Klusch, B. Fries, and K. Sycara. Automated Semantic Web Service Discovery with OWLS-MX. In *Proc. of the Fifth Int. joint Conf. on Autonomous agents and multiagent systems*, pages 915–922. ACM, 2006.

[39] M. Klusch, B. Fries, and K. 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.

[40] M. Klusch and P. Kapahnke. iSeM: Approximated Reasoning for Adaptive Hybrid Selection of Semantic Services. *The Semantic Web: Research and Applications*, pages 30–44, 2010.

[41] M. Klusch and P. Kapahnke. The iSeM Matchmaker: A Flexible Approach for Adaptive Hybrid Semantic Service Selection. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012.

[42] M. Klusch, P. Kapahnke, and B. Fries. Hybrid Semantic Web Service Retrieval: A Case Study With OWLS-MX. In *Proc. of the 2nd IEEE Int. Conf. on Semantic Computing*, ICSC'08, pages 323–330. IEEE, 2008.

[43] U. Küster and B. König-Ries. Semantic Service Discovery with DIANE Service Descriptions. In *Proc. of the Int. Conferences on Web Intelligence and Intelligent Agent Technology-Workshops*, pages 152–156. IEEE Computer Society, 2007.

[44] U. Küster and B. König-Ries. Semantic Service Discovery with DIANE Service Descriptions. *Semantic Web Services Challenge*, pages 152–156, 2009.

[45] U. Küster, B. König-Ries, M. Klein, and M. Stern. DIANE: A Matchmaking-Centered Framework for Automated Service Discovery, Composition, Binding, and Invocation on the Web. *Int. Journal of Electronic Commerce*, 12(2):41–68, 2007.

[46] U. Küster, B. König-Ries, M. Stern, and M. Klein. DIANE: An Integrated Approach to Automated Service Discovery, Matchmaking and Composition. In *Proc. of the 16th Int. Conf. on World Wide Web*, WWW'07, pages 1033–1042. ACM, 2007.

[47] Q. A. Liang and H. Lam. Web Service Matching by Ontology Instance Categorization. In *Proc. of the 5th Int. Conf. on Services Computing*, volume 1 of *SCC'08*, pages 202–209. IEEE, 2008.

[48] W. Lin, C. Lo, K. Chao, and M. Younas. Consumer-Centric QoS-aware Selection of Web Services. *Journal of Computer and System Sciences*, 74(2):211–231, 2008.

[49] W.-L. Lin, C.-C. Lo, K.-M. Chao, and M. Younas. Fuzzy Consensus on QoS in Web Services Discovery. In *Proc. of the 20th Int. Conf. on Advanced Information Networking and Applications*, AINA'06, pages 791–798. IEEE, 2006.

[50] R. Lipshitz and O. Strauss. Coping with Uncertainty: A Naturalistic Decision-Making Analysis. *Organizational Behavior and Human Decision Processes*, 69(2):149–163, 1997.

[51] M. Liu, W. Shen, Q. Hao, and J. Yan. An Weighted Ontology-Based Semantic Similarity Algorithm for Web Service. *Expert Systems with Applications*, 36(10):12480–12490, 2009.

[52] M. Liu, W. Shen, Q. Hao, J. Yan, and L. Bai. A Fuzzy Matchmaking Approach for Semantic Web Services with Application to Collaborative Material Selection. *Computers in Industry*, 63(3):193–209, Apr. 2012.

[53] X. Liu, K. K. Fletcher, and M. Tang. Service Selection Based on Personalized Preference and Trade-Offs Among QoS Factors and Price. In *Proc. of the First Int. Conf. on Services Economics*, SE'12, pages 32–39. IEEE, 2012.

[54] J. Ma, Y. Zhang, and J. He. Efficiently Finding Web Services Using a Clustering Semantic Approach. In *Proc. of the Int. Workshop on Context enabled source and service selection, integration and adaptation: organized with the 17th Int. World Wide Web Conf.*, WWW'08, page 5. ACM, 2008.

[55] H. R. Motahari-Nezhad, G. Y. Xu, and B. Benatallah. Protocol-Aware Matching of Web Service Interfaces for Adapter Development. In *Proc. of the 19th Int. Conf. on World Wide Web*, WWW'10, page 731, New York, New York, USA, 2010. ACM.

[56] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[57] 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.

[58] I. Patiniotakis, S. Rizou, Y. Verginadis, and G. Mentzas. Managing Imprecise Criteria in Cloud Service Ranking with a Fuzzy Multi-Criteria Decision Making Method. In *Proc. of the Int. Conf. on Service-Oriented and Cloud Computing*, ESOCC'13, pages 34–48. Springer, 2013.

[59] Z. Peng, W. He, and D. Chen. Research on Fuzzy Matching Model for Semantic Web Services. In *Proc. of the 3rd Int. Conf. on Intelligent System and Knowledge Engineering*, ISKE'08, pages 440–445. IEEE, 2008.

[60] D. Perez-Palacin and R. Mirandola. Uncertainties in the Modeling of Self-adaptive Systems: A Taxonomy and an Example of Availability Evaluation. In *Proc. of the 5th ACM/SPEC Int. Conf. on Performance Engineering*, ICPE'14, pages 3–14. ACM, 2014.

[61] M. C. Platenius. Fuzzy Service Matching in On-The-Fly Computing. In *Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE'13, pages 715–718. ACM, 2013.

[62] M. C. Platenius, W. Schäfer, and S. Arifulina. MatchBox: A Framework for Dynamic Configuration of Service Matching Processes. In *Proc. of the 18th Int. ACM SIGSOFT Symposium on Component-Based Software Engineering*, CBSE'15, pages 75–84, New York, NY, USA, 2015. ACM.

[63] M. C. Platenius, A. Shaker, M. Becker, E. Hüllermeier, and W. Schäfer. Imprecise Matching of Requirements Specifications for Software Services using Fuzzy Logic – Website. http://goo.gl/OkfNrJ, Last Access: Jan. 2016.

[64] 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*, CBSE'13, pages 143–152. ACM, 2013.

[65] L. Qu, Y. Wang, and M. Orgun. Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment. In *Proc. of the Int. Conf. on Services Computing*, SCC'13, pages 152–159. IEEE, 2013.

[66] D. V. Rao and V. Sarma. A Rough–Fuzzy Approach for Retrieval of Candidate Components for Software Reuse. *Pattern recognition letters*, 24(6):875–886, 2003.

[67] J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin. Semantic Web Service Discovery Using Natural Language Processing Techniques. *Expert Systems with Applications*, 40(11):4660–4671, 2013.

[68] J. Schönfisch, W. Chen, and H. Stuckenschmidt. A Purely Logic-Based Approach to Approximate Matching of Semantic Web Services. In *CEUR Workshop Proceedings*, volume 667, pages 37–52. RWTH, 2010.

[69] R. Senge and E. Hüllermeier. Top-Down Induction of Fuzzy Pattern Trees. *IEEE Transactions on Fuzzy Systems*, 19(2):241–252, 2011.

[70] C. Shapiro. Premiums for High Quality Products as Returns to Reputations. *The Quarterly Journal of Economics*, 98(4):659–680, 1983.

[71] I. Şora and D. Todinca. Specification-Based Retrieval of Software Components Through Fuzzy Inference. *Acta Polytechnica Hungarica*, 3(3):123–135, 2006.

[72] I. Şora and D. Todinca. Using Fuzzy Logic for the Specification and Retrieval of Software Components. Technical report, Universitatea Politehnica Timisoara, 2009.

[73] 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.

[74] D. Thakker, T. Osman, and D. Al-Dabass. Semantic-driven matchmaking and composition of web services using case-based reasoning. In *Proc. of the Fifth European Conf. on Web Services*, ECOWS'07, pages 67–76. IEEE, 2007.

[75] E. Toch, A. Gal, I. Reinhartz-Berger, and D. Dori. A Semantic Approach to Approximate Service Retrieval. *ACM Transactions on Internet Technology (TOIT)*, 8(1):2, 2007.

[76] A. Toninelli, A. Corradi, and R. Montanari. Semantic-Based Discovery to Support Mobile Context-Aware Service Access. *Computer Communications*, 31(5):935–949, 2008.

[77] R. Torres, H. Astudillo, and R. Salas. Self-Adaptive Fuzzy QoS-Driven Web Service Discovery. In *Proc. of the IEEE Int. Conf. on Services Computing*, SCC'11, pages 64–71. IEEE, 2011.

[78] R. Torres, N. Bencomo, and H. Astudillo. Addressing the QoS Drift in Specification Models of Self-Adaptive Service-Based Systems. In *Proc. of the 2nd Int. Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, RAISE'13, pages 28–34. IEEE, 2013.

[79] TrustRadius. TrustRadius – Software Reviews by Real Users – website. https://www.trustradius.com/, Last Access: Jan. 2016.

[80] University of Paderborn. Collaborative Research Center "On-The-Fly Computing" (CRC 901). http://sfb901.uni-paderborn.de, Last Access: Aug. 2016.

[81] University of Paderborn. Website of MatchBox. http://goo.gl/MMCxQT, Last Access: Jan. 2016.

[82] University of Paderborn. Website of SeSAME - Service Specification, Analysis, and Matching Environment. http://sfb901.uni-paderborn.de/sfb-901/projects/project-area-b/tools/sesame.html, Last Access: Jan. 2016.

[83] W. E. Walker, P. Harremoës, J. Rotmans, J. P. van der Sluijs, M. B. van Asselt, P. Janssen, and M. P. Krayer von Krauss. Defining Uncertainty: A Conceptual Basis for Uncertainty Management in Model-Based Decision Support. *Integrated assessment*, 4(1):5–17, 2003.

[84] H.-C. Wang, C.-S. Lee, and T.-H. Ho. Combining Subjective and Objective QoS Factors for Personalized Web Service Selection. *Expert Systems with Applications*, 32(2):571–584, 2007.

[85] P. Wang. QoS-Aware Web Services Selection with Intuitionistic Fuzzy Set Under Consumer's Vague Perception. *Expert Systems with Applications*, 36(3):4460–4466, 2009.

[86] P. Wang, K.-M. Chao, C.-C. Lo, C.-L. Huang, and Y. Li. A Fuzzy Model for Selection of QoS-Aware Web Services. In *IEEE Int. Conf. on e-Business Engineering*, pages 585–593. IEEE, 2006.

[87] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *Proc. of the 17th IEEE Int. Requirements Engineering Conf.*, RE'09, pages 79–88. IEEE, 2009.

[88] P. Xiong and Y. Fan. QoS-Aware Web Service Selection by a Synthetic Weight. In *Proc. of the Fourth Int. Conf. on Fuzzy Systems and Knowledge Discovery*, volume 3, pages 632–637. IEEE, 2007.

[89] L. Yin, M. A. Smith, and K. S. Trivedi. Uncertainty Analysis in Reliability Modeling. In *Proc. of the Annual Reliability and Maintainability Symposium*, pages 229–234. IEEE, 2001.

[90] L. Zadeh. Fuzzy Sets. *Information and control*, 8(3):338–353, 1965.

[91] L. Zadeh. Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems*, 1(1), 1978.
[92] L. Zadeh. Is There a Need for Fuzzy Logic? *Information Sciences*, 178(13):2751–2779, 2008.
[93] B. I. Zilci, M. Slawik, and A. Kupper. Cloud Service Matchmaking Using Constraint Programming. In *Proc. of the 24th Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 63–68. IEEE, 2015.

Marie Christin Platenius is a postdoctoral researcher in the Software Engineering Group at the Heinz Nixdorf Institute, Paderborn University, Germany. She leads the software architecture team at the Collaborative Research Center 901 On-The-Fly Computing. She received the PhD degree in computer science from Paderborn University in 2016. Her thesis is titled Fuzzy Matching of Comprehensive Service Specifications and was supervised by Prof. Wilhelm Schäfer. Her research interests include model-driven, component based, and service-oriented software engineering, software architecture and design, as well as fuzziness and uncertainty in software engineering.

Ammar Shaker received his Bachelor degree in Informatics Engineering from the University of Damascus (Syria) in 2005, and his Master degree in Data and Knowledge Engineering (DKE) from the Otto-von-Guericke University Magdeburg in 2010. He is a research and teaching assistant in the group of his PhD advisor, Prof. Eyke Hüllermeier, first at Philipps-University Marburg and meanwhile at the University of Paderborn. His main research interests include machine learning and data mining, in particular evolving fuzzy systems and learning from data streams.

Matthias Becker manages the Software Quality Group at the Fraunhofer Research Institution for Mechatronic Systems Design IEM in Paderborn, Germany. He holds a M.Sc., in computer science and currently writes a PhD at the Paderborn University. At Fraunhofer IEM, he leads projects centered around software quality assurance, security in cyber-physical systems, and software architecture for software-intensive systems. His main research interests include software performance engineering, self-adaptive system engineering, model-driven engineering, as well as software architecture and design.

Eyke Hüllermeier is with the Department of Computer Science at the University of Paderborn (Germany), where he holds an appointment as a full professor and heads the Intelligent Systems Group. He holds M.Sc. degrees in mathematics and business computing, a Ph.D. in computer science, and a Habilitation degree. His research interests are centered around methods and theoretical foundations of intelligent systems, with a specific focus on machine learning and reasoning under uncertainty. He has published more than 200 articles on these topics in top-tier journals and major international conferences, and several of his contributions have been recognized with scientific awards. Professor Hüllermeier is Co-Editor-in-Chief of Fuzzy Sets and Systems, one of the leading journals in the field of Computational Intelligence, and serves on the editorial board of several other journals, including Machine Learning, Data Mining and Knowledge Discovery, and the International Journal on Approximate Reasoning.

Wilhelm Schäfer was full professor and head of the Software Engineering Group at the University of Paderborn from 1994 to 2015. He was and is member of many national and international program committees in software engineering. He was a member of the IEEE Transactions on Software Engineering Editorial Board (2007–2009), he was PC-Chair of the 5th European Software Engineering Conference (ESEC), Barcelona in 1995, PC co-chair of the 23rd International Conference on Software Engineering in Toronto in 2001 and General Chair of the 30th International Conference on Software Engineering held in Leipzig in 2008. His research interests include model-driven development of mechatronic systems, re-engineering and software processes.