Automatic Service Categorisation through Machine Learning in Emergent Middleware

Amel Bennaceur[1], Valérie Issarny[1], Richard Johansson[4], Alessandro Moschitti[3], Romina Spalazzese[2], Daniel Sykes[1]

[1] INRIA, Paris-Rocquencourt, France first.last@inria.fr
[2] University of L'Aquila, Italy romina.spalazzese@di.univaq.it
[3] University of Trento, Italy moschitti@disi.unitn.it
[4] University of Gothenburg, Sweden richard.johansson@svenska.gu.se

Abstract. The modern environment of mobile, pervasive, evolving services presents a great challenge to traditional solutions for enabling interoperability. Automated solutions appear to be the only way to achieve interoperability with the needed level of flexibility and scalability. While necessary, the techniques used to determine compatibility, as a precursor to interaction, come at a substantial computational cost, especially when checks are performed between systems in unrelated domains. To overcome this, we apply machine learning to extract high-level functionality information through text categorisation of a system's interface description. This categorisation allows us to restrict the scope of compatibility checks, giving an overall performance gain when conducting matchmaking between systems. We have evaluated our approach on a corpus of web service descriptions, where even with moderate categorisation accuracy, a substantial performance benefit can be found. This in turn improves the applicability of our overall approach for achieving interoperability in the CONNECT project.

1 Introduction

The modern environment of mobile, pervasive, evolving services presents a great challenge to traditional solutions for enabling interoperability. The scale of complexity and heterogeneity of such devices and services, which adhere to many different standards and platforms, greatly increases the cost and difficulty of applying manual approaches. When mobility, dynamic availability, and the potential for evolution are additionally considered, the problem becomes insurmountable. Automatic approaches, termed *emergent middleware*, can overcome interoperability issues, provided that they are furnished with sufficient and relevant information, in a precise form, about the systems that should interact. This presents two sub-problems: how best to use the given information, and how to specify, extract, or discover such information. This paper addresses one case of the latter, namely, how to extract the high-level, abstract functionality information of a system, given only its detailed syntactic interface.

This high-level functionality, which we call an *affordance*, is expressed as a semantic concept from a domain ontology. Given such information we can efficiently check whether it is reasonable to attempt to make two systems interact. For example, there is little to be gained from attempting to overcome the differences between a system whose functionality is described as "Stock" and another whose functionality is described as "Weather". On the other hand there is no guarantee that two systems with the same affordance will be able to interact. To make a final assessment of compatibility, more in-depth analyses considering the interface and conversational protocol of the two systems are necessary. Avoiding such deep, time-consuming analyses motivates our use of affordances. When the affordances do not match, the detailed analyses can be omitted, providing an overall performance benefit for solving interoperability issues at runtime.

However, the requirement for affordance information places a burden on the system designer, and it is likely that legacy systems do not provide such detail. In this paper we describe an approach based on text categorisation, a machine learning technique that is able to categorise systems that have interface descriptions into affordances, based on the terms used in the interfaces. For example, an interface including many instances of the term "ticker" is likely to have the functionality corresponding to the "Stock" affordance. The assignment of affordances is thus completely automated and the full performance benefit of affordances can be reaped.

In Sections 2 and 3 we introduce the CONNECT project in which our work takes place, and outline the approach taken therein for discovering matching pairs of networked systems and synthesising *mediators* that enable the systems to communicate. In Section 4 we describe how text categorisation is applied to find each system's high-level functionality, and in Section 5 we show how this benefits the Discovery and Synthesis Enablers. Section 7 concludes.

2 Background

Our work takes place within the context of the CONNECT project[5]. The aim of the project is to overcome interoperability issues between protocols due to their heterogeneity at various levels by using an approach that dynamically generates the necessary interoperability solution that allows the systems to interact seamlessly. CONNECT hence promotes as a solution the dynamic synthesis of *emergent* CONNECT*ors* via which systems communicate. The emergent CONNECTors are concrete system entities synthesised according to the behavioural semantics of protocols executed by the interacting parties at application and middleware layers. The synthesis process is based on a formal foundation for CONNECTors, which allows learning, reasoning about and adapting the interaction behaviour of networked systems at runtime.

To reach these objectives the project undertakes interdisciplinary research, investigating the following issues and related challenges: (i) modelling and rea-

[5] http://connect-forever.eu/

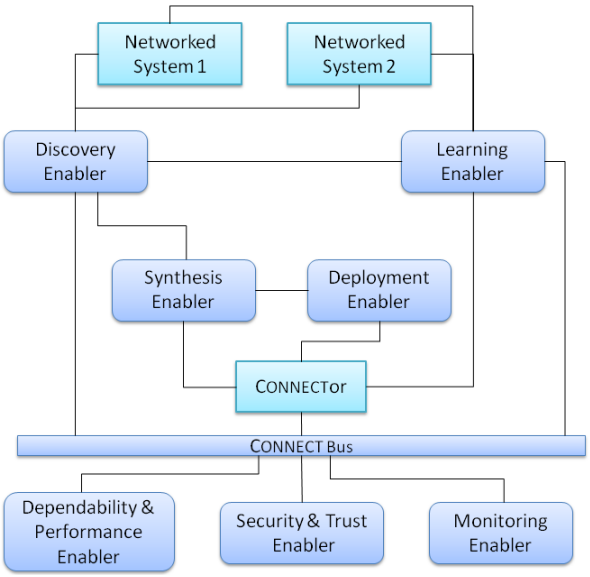

Fig. 1. CONNECT architecture.

soning about peer system functionality; (ii) modelling and reasoning about connector behaviour; (iii) runtime synthesis of CONNECTors; (iv) learning peer behaviour; (v) dependability assurance; and (vi) system architecture. The architecture to realise these objectives is illustrated in Figure 1.

We call the entities that implement the mechanisms which enable the required connections *enablers*. In summary, the enablers being developed as part of the architecture are:

- Discovery Enabler: it discovers the networked systems (our generic term for services and other systems) in the environment and collects their information, including interface description and ontological description. Ontological information, in particular, is used to perform a more efficient compatibility check with other systems, i.e., to identify whether, despite possible heterogeneity, one system provides the functionality that another requires.
- Learning Enabler: it infers models of the systems' interaction behaviour, i.e., models expressing how system services can be properly invoked. This enabler leverages active automata learning algorithms.
- Synthesis Enabler: it performs a compatibility check on the system models and, if compatible, automatically synthesises a CONNECTor that allows them to interact properly.
- Deployment Enabler: it deploys and manages the synthesised CONNECTors;
- Monitoring Enabler: it collects information from the CONNECTors, filters it, and passes it on to other requesting enablers;
- Dependability & Performance Enabler: it assesses dependability and performance properties at pre-deployment time and at runtime.

– Security and Trust Enabler: it collaborates with the Synthesis Enabler and with the Monitoring Enabler to check that possible security and trust requirements are met at runtime.

Within the described architecture, this paper focuses on the Discovery and Synthesis Enablers that benefit from the inference of high-level functionality through text categorisation.

3 Synthesising Emergent Middleware

Figure 2 outlines our overall approach to supporting emergent middleware by synthesising mediators dynamically. The key philosophy of this approach is to (i) *discover* available networked systems, (ii) *complete the descriptions* of networked systems, (iii) find *matching* pairs among them by analysing the descriptions of the networked systems, and (iv) *synthesise* mediators that allow them to interact by overcoming their incompatibilities.

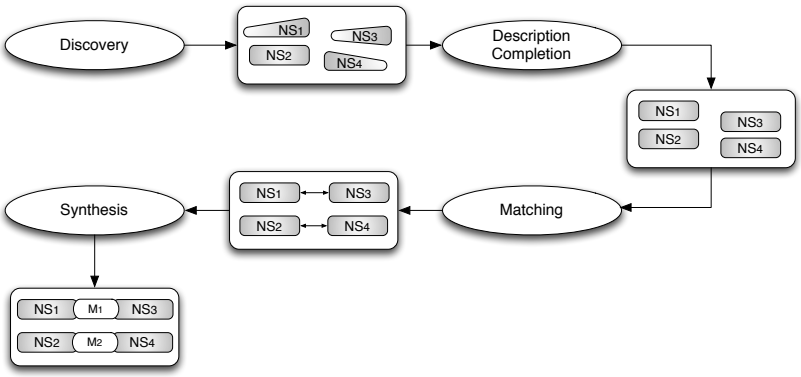

Fig. 2. Steps of creating emergent middleware.

Networked systems (NSs) are discovered by the Discovery Enabler. Their descriptions may be incomplete, leading the Discovery Enabler to invoke mechanisms that can infer the missing information. To infer the interaction behaviour, the Learning Enabler is invoked, while in this paper we introduce an additional *affordance classifier* that is able to infer the system's high-level functionality.

Given two complete networked system descriptions, the next step consists of checking their *functional* compatibility, i.e., whether at high level of abstraction, the functionality required by one system can be provided by the other (see Figure 3-❶). Functional matching is performed by checking the semantic compatibility of the networked systems' affordances using ontology reasoning. When a pair of NSs have compatible functionality, we verify that they can be made

interoperable so as to achieve this functionality through *behavioural matching* (see Figure 3-❷), which is performed by analysing the behaviour of both systems. Subsequently, we synthesise the appropriate mediator which allows the two systems to communicate (see Figure 3-❸).

Ontologies play a crucial role in supporting automatic service composition [1]. They formalise the domain-specific knowledge by describing the concepts of this domain, the functions, and relations between them [2]. Ontology reasoning is particularly important for inferring the relations between concepts in open environments [1], i.e., environments that consist of many interacting systems that are developed by different vendors and are either absolutely unaware of or have only partial knowledge about the global system.

In the remainder of this section, we describe the model of networked systems that allows us to reason about their ability to interoperate. Then we describe the different steps of the matching and synthesis process.

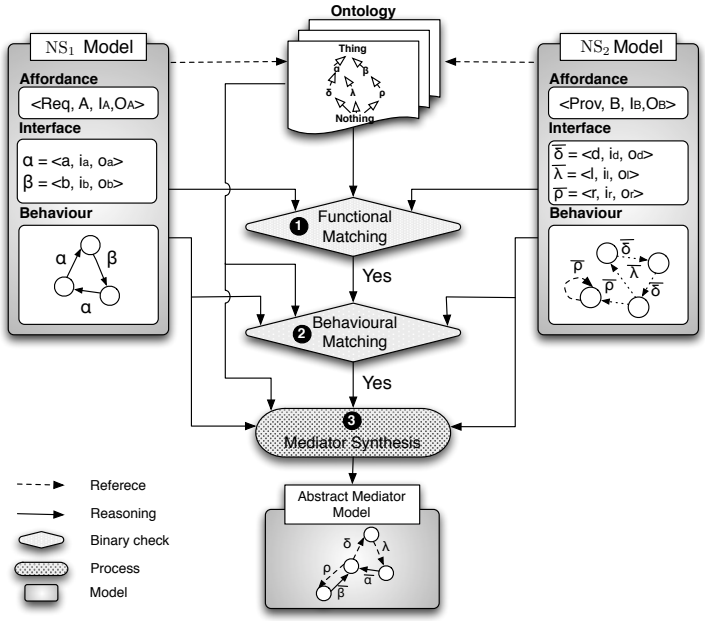

Fig. 3. Matching and synthesis.

Networked System Model

A networked system requires or provides an *affordance* to which it gives access via an explicit *interface*, and which it realises using a specific *behaviour*.

The affordance specifies the high-level functionality of a system and is defined as a tuple: $\mathcal{F}$=<t, c, i, o> where (i) t stands for provided (denoted **prov**) if the system is offering this functionality or required (denoted **req**) if it is consuming it; (ii) c gives the semantics of the functionality in terms of an ontology concept; (iii) i (resp. o) specifies the set of the high-level inputs (resp. outputs) of the functionality, which are defined as ontology concepts. All concepts belong to the same domain ontology $\mathcal{O}$ specifying the application-specific concepts and relations, i.e., $c, i, o \in \mathcal{O}$. Note that a **req** functionality produces the inputs I and consumes the corresponding outputs O. In a dual manner, a **prov** functionality consumes the inputs I and produces the corresponding outputs O. In the following we focus on the functionality concept c without considering data, overloading the term affordance where there is no ambiguity.

The interface defines the set of observable *actions* that the system requires from or provides to its execution environment, typically provided in the form of a WSDL[6] description. An *input action* α =<op, i, o> ($op, i, o \in \mathcal{O}$) requires an operation op for which it produces some input data i and consumes the output data o. Its dual *output action*[7] $\overline{\beta}$ =<$\overline{op}$, i, o> uses the inputs and produces the corresponding outputs. An interface $\mathcal{I}$ is then defined as: $\mathcal{I}$ ={<op_α, i_α, o_α>}∪{<$\overline{op_\beta}$, i_β, o_β>}.

The system behaviour describes its interaction with its environment and defines how the actions of its interface are co-ordinated to implement a specific affordance. We build upon state-of-the-art approaches to formalise system interaction using labelled transition systems (LTS) [3].

Ontology-based Functional Matching

Functional matching assesses whether the networked systems are functionally compatible using the following definition. A system requiring the functionality $\mathcal{F}_R$=<**req**,c_R, i_R, o_R> and a system providing the functionality $\mathcal{F}_P$=<**prov**, c_P, i_P, o_P>, are *functionally compatible*, written $\mathcal{F}_P \hookrightarrow \mathcal{F}_R$, *iff* in the associated ontology:

- c_P is a subtype of c_R,
- i_P is a supertype of i_R (contravariant), and
- o_P is a subtype of o_R (covariant).

following the Liskov substitution principle [4]. Intuitively, the $\mathcal{F}_P$ should provide at least the functionality required by $\mathcal{F}_R$ and may provide more.

Ontology-based Behavioural Matching

Behavioural matching assesses whether the networked systems are behaviourally compatible, i.e., whether there exists an intermediary system (a mediator) through

[6] http://www.w3.org/TR/wsdl
[7] Note the use of an overline to denote output actions.

which they can safely interact. Towards this end, we first infer the correspondence between the actions of the systems' interfaces so as to generate the mappings that perform the necessary translations between semantically-compatible actions. Various mappings relations may be defined, which primarily differ according to their complexity and inversely proportional flexibility. These mappings are generated according to the mediator capabilities, which includes receiving and sending messages, delaying the delivery of messages, and reasoning about the semantics of actions in order to generate actions by transforming and composing the original ones. We use an ontology-based model checking technique to explore the various possible mappings in order to produce a correct-by-construction mediator that guarantees that the two systems can successfully interact. Model checking is used to assess system correctness and automatically verify concurrent systems by exhaustively exploring the state space, which may be very large due to state space explosion. Although many solutions have been proposed to alleviate this issue at runtime [5], behavioural matching remains substantially more costly than functional matching.

Ontology-based Mediator Synthesis

The mediator enforces interoperation between functionally and behaviourally compatible systems despite their disparities. Mediator synthesis relies on the mappings computed during behavioural matching and refines them according to the characteristics of each networked systems and to the environment.

The specification of system functionality plays a valuable role in generating emergent middleware. It has also been acknowledged as crucial in open environments [6]. However, most legacy systems only exhibit their interface description. Hence, only partial knowledge about the system can be discovered. Given the central role of the functional matching of affordances in reducing the overall computation by acting as a filter for the subsequent behavioural matching, it is important to infer additional knowledge about the functional semantics of each networked system. Toward this goal, we use machine learning to extract the affordance of networked systems.

4 Affordance learning and categorisation

The problem consists in learning a *classifier* that is able to assign an affordance (specifically the functionality concept c) to a networked system automatically. The networked system has not been seen before, and its description includes an interface expressed in WSDL, but no affordance information. Note that it is not always necessary to have an absolutely correct affordance since falsely-identified matches may be caught in the subsequent detailed checks. Indeed, the pathological case with many false positives and no false negatives is equivalent to performing no affordance matching.

Since the interface is described by textual documentation, we can capitalise on the long tradition of research in *text categorisation* (TC). This studies approaches for automatically enriching text documents with semantic information

(metadata). The latter is typically expressed by topic categories: thus TC proposes methods to assign documents to one or more categories. In our case, the documents to categorise are interface descriptions, and the categories correspond to affordances. The size of the taxonomy may be small in some cases, such as a binary set, e.g., {POSITIVE, NEGATIVE} when classifying a customer review as positive or negative [7], and larger in other cases, such as the various structured classification systems used in library science. The main tool for implementing modern systems for automatic document classification is machine learning applied to documents represented with vector space models.

In order to be able to apply standard machine learning methods for building categorisers, we need to represent the objects we want to classify by extracting informative *features*. Such features are used as indications that an object belongs to a certain category. For categorisation of documents, the standard representation of features maps every document into a vector space using the *bag-of-words* approach [8]. In this method, every word in the vocabulary is associated with a dimension of the vector space, allowing the document to be mapped into the vector space simply by computing the occurrence frequencies of each word. For example, a document consisting of the string "get Weather, get Station" could be represented as the vector $(.., 2, .., 1, .., 1, ..)$ where, e.g., 2 is the frequency of the "get" token. The bag-of-words representation is considered the standard representation underlying most document classification approaches. In contrast, attempts to incorporate more complex structural information have mostly been unsuccessful for the task of categorisation of single documents [9] although they have been successful for complex relational classification tasks [10].

However, the task of classifying interface descriptions is different from classifying raw textual documents. Indeed, the interface descriptions are *semi-structured* rather than unstructured, and the representation method clearly needs to take this fact into account, for instance, by separating the vector space representation into regions for the respective parts of the interface description. In addition to the text, various semi-structured identifiers should be included in the feature representation, e.g., the names of the method and input parameters defined by the interface. The inclusion of identifiers is important since: (i) the textual content of the identifiers is often highly informative of the functionality provided by the respective methods; and (ii) the free text documentation is not mandatory and may not always be present. In a WSDL interface, we may have tags and structures as illustrated by the text fragment in Figure 4.

It is clear that splitting the CamelCase identifier `WeatherForecastSoap` into the tokens `soap`, `weather`, and `forecast` would provide more meaningful and generalised concepts, which the learning algorithm can use as features. Indeed, to extract useful word tokens from the identifiers, we split them into pieces based on the presence of underscores or CamelCase; all tokens are then normalised to lowercase.

Once the feature representation is available, we use it to learn several classifiers, each of them specialised to recognise if the WSDL expresses some target semantic properties. The latter can also be concepts of an ontology. Consequently,

```
<wsdl:portType name="WeatherForecastSoap">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeSoapIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameSoapIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="WeatherForecastHttpGet">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeHttpGetIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeHttpGetOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week  weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameHttpGetIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameHttpGetOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="WeatherForecastHttpPost">
  <wsdl:operation name="GetWeatherByZipCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a valid Zip Code (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByZipCodeHttpPostIn" />
    <wsdl:output message="tns:GetWeatherByZipCodeHttpPostOut" />
  </wsdl:operation>
  <wsdl:operation name="GetWeatherByPlaceName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Get one week weather forecast for a place name (USA)
    </wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameHttpPostIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameHttpPostOut" />
  </wsdl:operation>
</wsdl:portType>
```

Fig. 4. WSDL fragment for a weather service. Our approach treats such documents as unstructured text.

Category	Features
Stock	stock, list, symbol
Weather	weather, zip, forecast, code

Table 1. Most highly weighted features in the two-category experiment.

our algorithm may be used to learn classifiers that automatically assign ontology concepts to actions defined in NS interfaces. Of course, the additional use of domain (but at the same time general) ontologies facilitates the learning process by providing effective features for the interface representation. In other words, WSDL, domain ontologies and any other information contribute to defining the vector representation used for training the concept classifiers.

To demonstrate the validity of the approach empirically, we experimented with automatic classification of service topics. These can be used to characterise the affordance associated with an interface (i.e., using such concepts), from which it can be inferred if two NSs are implementing compatible affordances or not.

For this purpose, we collected a set of 14 WSDL descriptions to which we manually assigned two affordance labels (i.e., categories): 8 descriptions were classified as "Stock" category of Web services, i.e., dealing with stock-markets, and 6 descriptions in the "Weather" category, i.e., weather-related services. It is clear that knowing if the offered services belong to the categories above would help to determine the affordance.

The critical aspect is to find out if such categorisation can be automatically carried out by our machine learning approach. Thus we applied rigorous statistical methods for assessing its performance. In particular, we carried out a 3-fold cross-validation over the above-mentioned dataset. To train the models, we used linear support vector machines from LibLinear software [11]. In all three experiments (three folds), the achieved precision was 100%, i.e., the classifier was always able to choose the right category for the unknown interface.

Additionally, we analysed which were the most important features of the adopted interface representation. For this purpose, we recall that the support vector learning procedure results in a *weight vector* where each dimension corresponds to the dimensions used in the feature vectors. The magnitude of these weights can be interpreted as a measure of the importance of the respective features. Table 1 shows the most highly weighted features for the two categories. As we can see, the most prominent are perfect representatives of the classes: for the "Stock" category, the algorithms has decided that `stock` is the most important feature, and similarly for the "Weather" category.

However, testing on only two categories may not provide realistic findings as many more concepts are typically involved in NS interfaces. Therefore, to evaluate our approach in a more concrete application scenario, we used a collection of WSDL documents available on the Web[8]. Note that these introduce two sources of complexities: (i) a larger number of concepts and (ii) the WSDL files

[8] `http://www.andreas-hess.info/projects/annotator/ws2003.html`

Category	P	R	F	n
Mathematics	0.29	0.20	0.24	23
Business	0.17	0.08	0.11	46
Communication	0.71	0.80	0.75	49
Converter	0.57	0.63	0.61	65
CountryInfo	0.64	0.83	0.72	38
Developers	0.18	0.11	0.14	46
Finder	0.55	0.59	0.57	10
Money	0.72	0.72	0.72	56
News	0.70	0.63	0.67	30
Web	0.47	0.46	0.47	39

Table 2. Performance by category.

do not contain natural language descriptions, which clearly facilitate semantic extraction, i.e., semantic categorisation.

We selected the 10 most frequent categories for a total of 402 documents, and we trained and evaluated the classifiers using 8-fold cross-validation. In this experiment, the accuracy was 58%. Table 2 shows a detailed breakdown of the result. P indicates the *precision*, which is the number of documents correctly assigned to a category compared to the number that are correctly or incorrectly assigned to that category (a precision of 1 means there are no false positives). R indicates the *recall*, which is the number of documents correctly assigned to a category compared to the number that should be assigned to that category (a recall of 1 means there are no false negatives). For example, the "CountryInfo" category has a recall of 0.83, meaning that few documents of that category were falsely assigned to another. n indicates the number of documents manually assigned to each category while $\mathrm{F} = \frac{2PR}{P+R}$, i.e., the F-measure (harmonic means between P and R).

Again, we present the most highly weighted features for each category in Table 3. As we can see, these features are highly representative of the respective categories.

In summary, in the realistic scenarios our approach decreases its effectiveness, although preserving its applicability in tasks such as automatic affordance detection. The results are promising as we achieved good accuracy using basic TC techniques to train our classifiers, although we did not use structural information and background knowledge. Also the statistical learning theory suggests greater accuracy could be achieved by increasing the size of the training data. Finally, the meaningfulness of the features selected by the classifier demonstrate that it can easily derive the best properties, alleviating the designer from the burden of manual selection.

Category	Features
Mathematics	calculator, previous, at, value
Business	description, chart, parent, n
Communication	send, message, email, subject
Converter	to, translate, unit, my
CountryInfo	country, state, zip, postal
Developers	reverse, text, case, generate
Finder	whois, who, iwhois, results
Money	stock, amount, card, currency
News	news, quote, day, daily
Web	key, name, valid, d

Table 3. Most highly weighted features in the ten-category experiment.

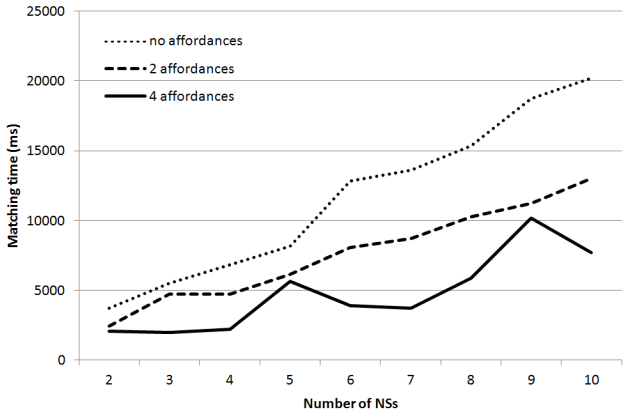

Fig. 5. Performance of matching with 0, 2, and 4 affordances.

5 Evaluation

Having shown that automatic service categorisation on the basis of interface descriptions is indeed feasible, we must now show that the affordances provided by the categorisation result in the expected benefit to discovery. The purpose of introducing affordances is to filter the number of service pairs for behavioural matching with a relatively efficient semantic check, and hence to reduce the overall time taken to conduct matchmaking when services are discovered.

After performing training offline, we integrated the trained classifier into the Discovery Enabler, which is responsible for matching pairs of networked systems. The Discovery Enabler invokes the classifier when it discovers a networked system that does not have an affordance. We then measured the time taken by the Discovery Enabler to perform matchmaking with and without the classifier.

Figure 5 shows the time taken to perform matchmaking after the sequential discovery of the given number of networked systems (up to 10). The results are

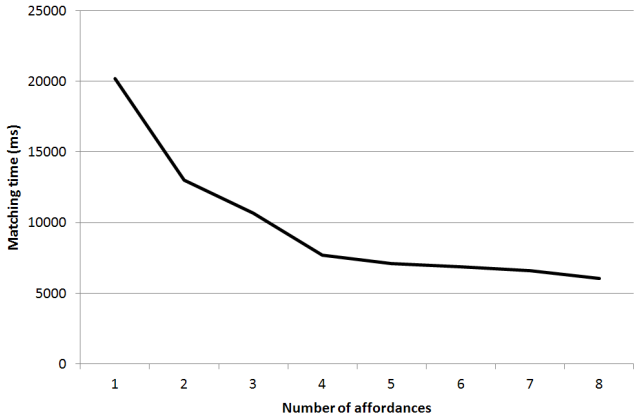

Fig. 6. Performance of matching after discovering 10 networked systems.

averaged over ten runs. The line with the steepest average gradient shows the time taken when no affordances are used, and so no categorisation takes place. Matchmaking in this case involves performing behavioural matching for every possible pair, i.e. n^2 checks for n NSs. The other lines show the time taken when the services are automatically categorised into two and four affordances respectively. Having just two affordances reduces the number of behavioural checks to $\frac{n^2}{2}$ and adds n^2 semantic checks. In the results, we find two affordances gives a 32% reduction in the matching time, and four affordances gives a further 37% reduction.

When two or three systems have been discovered, in the case with four affordances, we do not yet expect any matches. In fact the results show an almost constant time, around 2 seconds, for matching when no matches are found. This delay represents the overhead inherent in our prototype implementation of discovery resulting from parsing WSDL and BPEL and other steps internal to discovery.

Figure 6 shows the reduction in matching time as the number of affordances increases towards the number of systems. It can be observed that the worst case time involves one affordance or none, and the best case involves as many affordances as there are networked systems (no semantic matches will be found and so no behavioural checks will be required). This suggests that the domain ontology (taxonomy) in which the affordances are defined should be as detailed as possible. Note however that increasing the number of affordances can decrease the accuracy of categorisation as features (tokens in interface descriptions) become increasingly ambiguous. This effect can be seen to an extent in the second categorisation experiment with 10 categories compared to 2 categories in the first experiment.

6 Related work

Interoperability is a well known problem and its investigation has been done in many research contexts. For instance, in the form of supervisory control synthesis [16], discrete controller synthesis [17], component adaptors [18], protocol conversion [19,20,21], converter synthesis [22] to mention some. A work related to our mediator synthesis approach is the seminal paper by Yellin and Strom on protocol adaptor synthesis [23] that proposes an adaptor theory to characterise and solve the interoperability problem of augmented interfaces of applications.

In more recent years increasing attention has been paid in the Web Service area to business process integration and automatic mediation, e.g., [24,25,26,27], which are related to our synthesis of mediators in some aspects. Among them, it is worth mentioning the paper [28] on behavioural adaptation because it proposes a matching approach based on heuristic algorithms to match services for the adapter generation taking into account both the interfaces and the behavioural descriptions. Moreover, the Web services community has been also investigating how to support service substitution to enable interoperability with different implementations of a service (e.g., due to evolution or provision by different vendors). While early work has focused on semi-automated, design-time approaches [26,29], latest work concentrates on automated, run-time solutions [30,31]. This latter relates to our work because of the exploitation of ontologies to reason about interface mapping and the synthesis of mediators according to such mapping.

Despite the wide range of discovery protocols that heavily rely on semantic annotations to perform service matchmaking [12,13] there are few implementations that do not assume that these services advertise their semantically-annotated descriptions. The METEOR-S Framework [14] is able to assign semantic concepts to web services by considering their WSDL descriptions but without taking into account the unstructured data potentially available within the documentation tag that can give more information about the category the web service belongs to. Instead of attaching a category concept to a web service, SAWSDL-MX2 [15] evaluates the similarity between a pair of web services based on both structured and unstructured information included in their interfaces using support vector machines. This approach is the closest to ours but is clearly not scalable especially when considering environments where services may continuously be discovered.

Moreover, these approaches only consider the functionalities of the systems to perform discovery, which may result in a false positive matching either due to the imprecision of the learning process or because the behaviour has not been considered. In our approach, the affordance matching is complemented with behavioural matching so as to match pairs of systems more accurately. Indeed, when two systems match we are able to synthesise a mediator that ensures their interoperation.

7 Conclusions

The work we have described here aims to overcome a limitation of legacy discovery mechanisms, namely that they do not provide a high-level semantic description of a system's functionality (that we call an affordance). Through the application of support vector machines for text categorisation, we have shown that the burden of categorising systems, that is, determining their high-level functional semantics, can be lifted from the engineer and performed automatically with reasonable accuracy. The cases of inaccuracy can be divided into false positives, where two NSs have been assigned the same affordance when in fact they do not match, and false negatives, where two matching NSs are assigned different affordances and hence no attempt to connect them will be made. Minimising the number of false negatives (i.e. maximising recall) is hence critical for CONNECT. Greater accuracy may be achieved by finding more nuanced features, such as the structure of the document or token proximity, on which to base the categorisation.

Given such categorisation, affordance matching allows us to reduce the number of behavioural checks performed, and thus increase the performance of the matchmaking process as a whole. Our results show that the gain is relative to the number of affordances, with just two affordances providing a 32% performance increase. This performance increase benefits our overall aim in the CONNECT project, which is to provide solutions for interoperability at runtime, thus requiring efficient runtime mechanisms to identify compatibility and find solutions for overcoming incompatibilities.

In future work, we plan to investigate features that improve the accuracy of the categorisation, and apply categorisation in other specific areas of CONNECT. For example, it is desirable for each operation in an interface (as well as the interface as a whole) to give its semantics through an associated ontology concept. The approach taken in this paper may prove applicable to this problem.

Acknowledgements

This research has been supported by the EU FP7 projects: CONNECT – Emergent Connectors for Eternal Software Intensive Networking Systems (project number FP7 231167), EternalS – "Trustworthy Eternal Systems via Evolving Software, Data and Knowledge" (project number FP7 247758) and by the EC Project, LivingKnowledge – "Facts, Opinions and Bias" in Time (project number FP7 231126).

References

1. Blair, G.S., Bennaceur, A., Georgantas, N., Grace, P., Issarny, V., Nundloll, V., Paolucci, M.: The role of ontologies in emergent middleware: Supporting interoperability in complex distributed systems. In: Middleware'11. (2011)

2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook. Cambridge University Press (2003)
3. Keller, R.M.: Formal verification of parallel programs. Commun. ACM (1976)
4. Liskov, B.: Keynote address - data abstraction and hierarchy. In: Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum). OOPSLA '87, New York, NY, USA, ACM (1987) 17–34
5. Calinescu, R., Kikuchi, S.: Formal methods @ runtime. In: Monterey Workshop. (2010) 122–135
6. Baresi, L., Di Nitto, E., Ghezzi, C.: Toward open-world software: Issue and challenges. Computer (2006)
7. Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up? Sentiment classification using machine learning techniques. In: Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing, University of Pennsylvania, United States (2002) 79–86
8. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Technical Report TR74-218, Department of Computer Science, Cornell University, Ithaca, New York (1974)
9. Moschitti, A., Basili, R.: Complex linguistic features for text classification: A comprehensive study. In: Proceedings of the 26th European Conference on Information Retrieval Research (ECIR 2004), Sunderland, United Kingdom (2004) 181–196
10. Moschitti, A.: Kernel methods, syntax and semantics for relational text categorization. In: Proceedings of ACM 17th Conference on Information and Knowledge Management (CIKM), Napa Valley, United States (2008)
11. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research **9** (2008) 1871–1874
12. Li, H., Du, X., Tian, X.: A wsmo-based semantic web services discovery framework in heterogeneous ontologies environment. In: KSEM. (2007) 617–622
13. Pirrò, G., Trunfio, P., Talia, D., Missier, P., Goble, C.A.: Ergot: A semantic-based system for service discovery in distributed infrastructures. In: CCGRID. (2010) 263–272
14. Oldham, N., Thomas, C., Sheth, A.P., Verma, K.: Meteor-s web service annotation framework with machine learning classification. In: SWSWPC. (2004) 137–146
15. Klusch, M., Kapahnke, P., Zinnikus, I.: Sawsdl-mx2: A machine-learning approach for integrating semantic web service matchmaking variants. In: ICWS. (2009) 335–342
16. Brandin, B., Wonham, W.: Supervisory control of timed discrete-event systems. IEEE Transactions on Automatic Control **39**(2) (1994)
17. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event processes. Siam J. Control and Optimization **25**(1) (1987)
18. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptation. J. Syst. Softw. **74** (2005)
19. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE Journal on Selected Areas in Communications **8**(1) (1990) 127–142
20. Lam, S.S.: Correction to "protocol conversion". IEEE Trans. Software Eng. **14**(9) (1988) 1376
21. Okumura, K.: A formal protocol conversion method. In: SIGCOMM. (1986) 30–37
22. Passerone, R., de Alfaro, L., Henzinger, T.A., Sangiovanni-Vincentelli, A.L.: Convertibility verification and converter synthesis: two faces of the same coin. In: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. ICCAD '02 (2002) 132–139

23. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Program. Lang. Syst. **19** (1997)
24. Emilia Cimpian and Adrian Mocan: WSMX Process Mediation Based on Choreographies. In Bussler, C., Haller, A., eds.: Business Process Management Workshops. Volume 3812. (2005) 130–143
25. Vaculín, R., Neruda, R., Sycara, K.P.: An agent for asymmetric process mediation in open environments. In Kowalczyk, R., Huhns, M.N., Klusch, M., Maamar, Z., Vo, Q.B., eds.: SOCASE. Volume 5006 of Lecture Notes in Computer Science., Springer (2008) 104–117
26. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM (2007) 993–1002
27. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol mediation for adaptation in semantic web services. In: ESWC. (2006) 635–649
28. Motahari Nezhad, H.R., Xu, G.Y., Benatallah, B.: Protocol-aware matching of web service interfaces for adapter development. In: Proceedings of the 19th international conference on World wide web. WWW '10, New York, NY, USA, ACM (2010) 731–740
29. Ponnekanti, S., Fox, A.: Interoperability among independently evolving Web services. In: Proc. ACM/IFIP/USENIX Middleware Conference. (2004) 331–351
30. Denaro, G., Pezzé, M., Tosi, D.: Ensuring interoperable service-oriented systems through engineered self-healing. In: Proceedings of ESEC/FSE 2009, ACM Press (2009)
31. Cavallaro, L., Nitto, E.D., Pradella, M.: An automatic approach to enable replacement of conversational services. In: ICSOC/ServiceWave. (2009)
32. Heß, A., Kushmerick, N.: Learning to attach semantic metadata to web services. In: International Semantic Web Conference. (2003) 258–273