Inferring Affordances Using Learning Techniques

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

[1] INRIA, Paris-Rocquencourt, France
[2] University of Trento, Italy
[3] University of L'Aquila, L'Aquila, Italy

Abstract. Interoperability among heterogeneous systems is a key challenge in today's networked environment, which is characterised by continual change in aspects such as mobility and availability. Automated solutions appear then to be the only way to achieve interoperability with the needed level of flexibility and scalability. While necessary, the techniques used to achieve interaction, working from the highest application level to the lowest protocol level, come at a substantial computational cost, especially when checks are performed indiscriminately between systems in unrelated domains. To overcome this, we propose to use machine learning to extract the high-level functionality of a system and thus restrict the scope of detailed analysis to systems likely to be able to interoperate.

1 Introduction

We live in a world populated by highly heterogeneous, networked, mobile and pervasive systems and services. Such heterogeneity may span the application layer, the middleware layer, and the underlying communication infrastructure. Interaction between these systems, where feasible, is customarily achieved through diverse ad hoc means for specific pairs of systems in a particular environment. *Principled* automatic composition can bring a labour-saving benefit–through generalisation over classes of systems–and can provide the flexibility needed to cope with rapidly changing contexts, dynamic service availability and user mobility.

Automatic service composition has three main phases: discovery of what services exist in the current scope; finding pairs or sets of services which are compatible, so as to make composition possible; and the actual process of connecting one system to another. The second step of finding matching pairs of systems can be a computationally costly procedure, both in terms of the number of combinations of systems which have been discovered, but also in terms of the deep behavioural (or protocol) analyses used to determine if a single pair is compatible.

Hence it is unreasonable to perform matching with all systems every time a new system is discovered. Indeed, detailed matching between heterogeneous systems working in wildly different application domains is nonsensical: the word processor on a traveller's laptop need not be compared against the air-traffic control infrastructure simply because he is situated inside the airport. On the other hand, matching against a document translation service may in fact be of some use.

What is required is a notion of *category of systems*; things that speak about the same domain. Then matching can be restricted to combinations falling within a given category. For this purpose, we define an *affordance* which represents the high-level functionality (capability) of a given system with reference to an ontology which specifies the domain of interest. A system may have several affordances, representing different facets of its functionality, each of which may even relate to a different domain.

In addition to restricting the scope of matching, affordances can further increase the efficiency of composition by exploiting a structured repository wherein system descriptions are stored according to the matching relation. Structuring the repository in this manner reduces the number of comparisons which need to be made when a new system is discovered, even within a given domain. Figure 1 illustrates the linear speed up of matching when affordances are used.

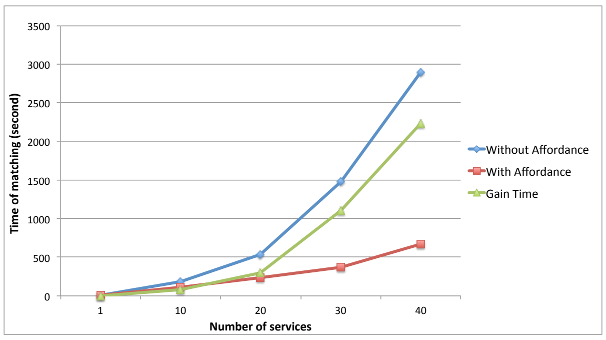

Fig. 1. Time of matching with and without using affordances

These benefits can only be reaped, however, when all systems are annotated with their respective affordances: a substantial effort for the great numbers of legacy systems, which provide only their interface description. However, it is worthwhile considering what process the programmer may go through when assigning an affordance. Given a set of "universally" agreed concepts in the ontology, the programmer can examine the interface and its documentation to determine which concepts best describe the broad category and functionality of the system. It goes without saying that to achieve this, the natural language descriptions and identifiers (such as method names) present in the interface will be used to make the classification.

We propose to use machine learning to automate the extraction of affordances from the interface description by classifying the natural-language text according to a pre-defined ontology of systems. Such an approach can fill the gap when a discovered system does not have a programmer-assigned affordance.

In the following, we set out in more detail the context of our problem, focussing on services, and discuss techniques that may be used to realise the approach.

2 Automatic service composition

To compose services automatically we can make use of a theory [5] for the automated synthesis of *mediating connectors* (also called *mediators*) that has been defined elsewhere [1]. That is, the service composition problem can be seen as an instance of the kind of problems the theory is able to model and solve.

More specifically, to compose services we need to: (i) *discover* the available ones, (ii) find *matching* pairs among them, and (iii) *synthesise* mediators that adapt the services behaviours allowing them to interoperate.

Our approach to dynamic service composition and interoperability is illustrated in Figure 2.

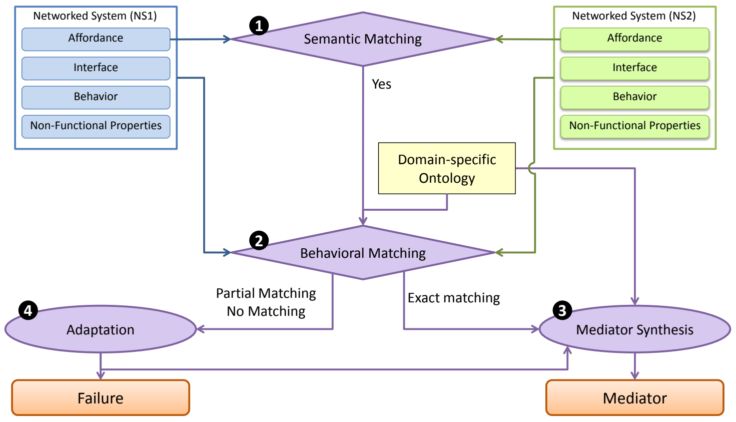

Fig. 2. Approach to dynamic interoperability

Two descriptions of networked systems (NSs) are given, including their interface, behaviour, non-functional properties and affordance descriptions. The first step consists of checking the compatibility of their *affordances*, high-level functionality, through the use of semantic matching (❶). Then, in the successful cases, a behavioural matching (❷) is performed by reasoning about both the NSs descriptions and the ontologies characterising their actions. In the case of exact behavioural matching, a mediator is synthesised (❸) based on the results of the reasoning in the previous step, while in the case of partial matching, a protocol adaptation (❹) is needed before the mediator synthesis. This process highlights the central role of the semantic matching of affordances in reducing the overall computation by acting as a kind of filter for the subsequent behavioural matching.

2.1 Affordances

An *affordance* denotes a high-level functionality provided to or required from the networked environment. Concretely, an affordance is specified as a tuple:

$$Aff = \langle Type, F, I, O \rangle$$

where:

- *Type* stands for a required (noted *Req*), provided (noted *Prov*) or required and provided (noted *Req_Prov*) affordance.
- F gives the semantics of the functionality associated with the affordance in terms of an ontology concept.
- I (resp. O) specifies the set of inputs (resp. outputs) of the affordance, which is defined as a tuple $\langle i_1, ..., i_n \rangle$ (resp. $\langle o_1, ..., o_m \rangle$) with each i_l (resp. o_k) being an ontology concept.

For example, $\langle Prov, AuctionHouse, \langle Goods \rangle, \langle Money \rangle \rangle$ is an affordance describing the provision of $AuctionHouse$ functionality with an input of *Goods* and an output of $Money$.

The first step in identifying the possible *compatibility* of two networked systems is to assess whether they respectively provide and require semantically matching affordances. For example, a $Procurement$ application, being a kind of $Buyer$, may match the above $AuctionHouse$, as a specific kind of $Seller$. Once a functional match is found at the affordance level, the more costly behavioural and non-functional matching can be performed.

2.2 Legacy applications

Unfortunately, legacy applications do not normally provide affordance descriptions. We must therefore rely upon an engineer to provide them manually, or find some automated means to extract the probable affordance from the interface description. Note that it is not strictly necessary to have a guaranteed correct affordance since falsely-identified matches will be caught in the subsequent detailed checks.

In this paper we focus on using machine learning to extract affordances from interface descriptions. Moreover we focus on the functional concept F of the affordance, rather than the inputs and outputs, though the overall approach would be notionally unchanged. Learning the inputs and outputs would require a straightforward division of the interface into parts which refer to data and those which refer to the functionality, and performing the learning procedure on each independently.

3 Affordance learning

This section provides an example interface description to bring the affordance learning problem into focus.

3.1 Typical interface

Listing 1.1 shows a small fragment of the WSDL interface description of the popular eBay [2] web service.

Listing 1.1. Ebay WSDL interface description

```
<!-- Call: AddItem -->
<xs:element name="AddItemRequest"
            type="ns:AddItemRequestType"/>
<xs:complexType name="AddItemRequestType">
  <xs:annotation>
    <xs:documentation>
      Defines a single new item and lists it  on a specified eBay site.
      &nbsp;<b>Also for Half.com</b>.
      Returns the item ID for the new listing, and returns fees
      the seller will incur for the listing (not including the Final
      Value Fee, which cannot be calculated until the item is sold).
    </xs:documentation>
    <xs:appinfo>
    <RelatedCalls>
      AddFixedPriceItem, AddItems, AddToItemDescription, GetItem,
      GetItemRecommendations, GetSellerList, RelistItem, ReviseItem,
      VerifyAddItem
    </RelatedCalls>
    <SeeLink>
      <Title>Listing an Item</Title>
      <URL>http://developer.ebay.com/...</URL>
    </SeeLink>
    <SeeLink>
      <Title>Listing Items</Title>
      <URL>http://developer.ebay.com/...</URL>
    </SeeLink>
    ...
```

This example provides extensive English text in both the documentation and the terms used in message and type names. Note that the complete description is approximately 130k lines long. In order to handle less verbose descriptions, documentation acquired from alternative sources such as `http://webservices.seekda.com/` can be used. It would not take an engineer, or indeed a layperson, long to determine the approximate purpose of the service, relying on key words such as ‘item’, ‘seller’ and ‘fee’. A concept from a pre-determined ontology, such as *AuctionHouse*, could then be assigned. Given such a description we propose to use machine learning to infer the appropriate affordance for the service.

3.2 Learning problem

The problem we are considering, then, is to find a function f which, given a parsed interface description with only the natural-language terms remaining, determines with some confidence the concept most appropriate for that service:

$$f : \mathit{Interface} \rightarrow (\mathit{Concept} \times \mathit{Confidence})$$

To achieve this, we provide a number of examples as training data relating interfaces to concepts: $\mathit{Interface} \times \mathit{Concept}$. These examples are acquired by searching for web service descriptions in online repositories, e.g., `webservicelist.com` and `xmethods.com`, and manually assigning to each a concept. The learning technique employed should then be able to generalise from the examples to produce an f to classify new examples. It is necessary to have a number of example interfaces for each concept we wish to assign to services.

Note that the problem could be tackled at (at least) two levels of granularity: the concepts could indicate the broad category of service within a "universal" ontology (taxonomy), or they could indicate a more specific service type within an ontology restricted to a specific domain. The learning problem is the same for both; all that changes is the breadth of automation we can achieve versus the depth of the domain. Arbitrarily increasing the breadth and depth of the ontology will impact confidence as it becomes increasingly likely that concepts are ambiguous.

4 Potential solution: machine learning of categorisers

We believe that the problem of affordance learning can draw many lessons from the long tradition of research in *text categorisation*: the problem of assigning a given document to one or more categories. The complexity of the system of categories may be low in some cases, such as a binary set {Positive, Negative} when classifying a customer review as positive or negative [12], and higher 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 systems is machine learning based on vector space document representations.

4.1 Introduction to machine learning

In general, we define machine learning as the problem of inducing a function (or system of functions) from a given data set. We may discern two main strands of machine learning methods: *supervised* and *unsupervised* methods.

The most archetypical problem setting in machine learning is the supervised setting. In supervised learning, the learning mechanism is provided with a (typically finite) set of labelled examples: a set of pairs $T = \{\langle x, y\rangle\}$. The goal is to make use of the example set T to induce a function f such that generally $f(x) = y$ for future, unseen instances of (x, y) pairs. Supervised learning methods in most cases learn much more accurate classifiers than their unsupervised counterparts, but require a human-annotated training set of significant size: the bigger the better. Examples of supervised learning methods commonly used include Support Vector Machines [3], which have been extensively studied for the problem of text categorisation [6]. For the problem of automatic association of WSDL interface descriptions with concepts, we thus need to gather a large set of interface descriptions and manually assign one or more concepts to every description.

As opposed to the supervised setting, the problem definition in unsupervised learning instead assumes the examples to be unlabelled, i.e. $T = \{x\}$. In order to be able to come up with anything useful when no supervision is provided, the learning mechanism needs a bias that guides the learning process. The most well-known example of unsupervised learning is probably k-means clustering [8], where the learner learns to categorise objects into broad categories even though the categories were not given a priori. More complex examples include grammar induction methods from raw text.

In addition to two main subfields of learning methods there are of course outliers and hybrids, such as *semisupervised* learning: Since it is costly to produce manually labelled training data, in some situations only a small labelled example set $T_s = \{\langle x, y\rangle\}$ is provided, while there is also available a larger unlabelled example set $T_u = \{x\}$. Semisupervised learning methods are able to make use of the labelled data T_s in combination with the unlabelled data T_u in order to improve over a plain supervised learner making use of T_s only. Another interesting learning paradigm is *active learning*, where the learning mechanism is able to select particularly informative unlabelled examples from an unlabelled dataset and ask an oracle (a human annotator or some sort of automatic mechanism) for a labelling. Typically, active learners are able to achieve a more efficient use of the training data than normal supervised learners, since their behaviour is more targeted towards distinguishing the difficult cases.

4.2 Representations for categorisation

In order to be able to apply standard supervised or unsupervised machine learning methods for building categorisers, we need to represent the objects we want to classify by extracting informative *features.* For categorisation of documents, the standard representation method maps every document into a vector space using the *bag-of-words* approach [13]. 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. The bag-of-words representation is considered the standard representation underlying most document classification approaches, and attempts to incorporate more complex structural information have mostly been unsuccessful for the task of categorisation of single documents [10] although more successful for complex relational classification tasks [9].

However, the task of classifying WSDL interface descriptions is different from classifying raw documents: 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 into regions representing the respective parts of the WSDL description. For instance, the description in Figure 1.1 contains a general documentation part in free text, as well as a number of textual descriptions of the methods defined by the interface.

In addition to the text, we believe that the various semi-structured identifiers should be included in the feature representation, most importantly the names of the methods defined by the interface but also the methods listed in the `RelatedCalls` section. The inclusion of identifiers will be important since

1) the textual content of the identifiers is often highly informative of the functionality provided by the respective methods; 2) the free text documentation is not mandatory and may not always be present. Extracting useful bag-of-words representations from the identifiers will likely have to use splitting heuristics relying on the presence of indicators such as underscores or CamelCase.

5 Conclusions

Principled automatic composition is the only means to overcome the manifold difficulties inherent in the problem of interoperability of diverse, heterogeneous systems. In contrast to incidental *ad hoc* solutions, automatic composition brings such benefits as scalability, self-adaptation, flexibility, resilience to faults, and tolerance of dynamic availability and user mobility. Affordances are the first weapon in attacking the problem, by categorising systems and so avoiding unnecessarily deep checks on systems whose high-level functionality is utterly different.

Affordances need not be especially precise—we are not looking for a surgical strike—since the detailed work is handled by behavioural and other compatibility checks. For this reason we are able to take advantage of machine learning to provide us with affordances when they have not been provided by the programmer. Techniques such as support vector machines can categorise free text according to a pre-defined ontology of systems, however it may be beneficial to treat the WSDL interface description as a semi-structured document, by, for example, separating method, input and output identifiers from pure documentation.

In addition to experimenting with different categorisers and the structure of the input, the provision and the generality of the ontology of systems poses a challenge. While we do not wish to limit the scope of the approach to a particular domain, having overly general concepts will again lead to unnecessary deep compatibility checks.

A number of similar approaches exist, particularly in the field of web services, such as [11, 7, 4], from which we can draw guidance. However, their aims and context often differ. In our case, the extraction of an affordance to categorise systems promises to bring such benefits as well-targeted compatibility checking, efficient storage of descriptions, and a potential for decentralisation.

Acknowledgments.

This work is done as part of the European FP7 ICT FET CONNECT project (`http://connect-forever.eu/`).

References

1. CONNECT Annex I: Description of Work. FET IP CONNECT EU project, FP7 grant agreement number 231167, `http://connect-forever.eu/`.
2. eBay WSDL. `http://developer.ebay.com/webservices/latest/ebaySvc.wsdl`

3. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory (1992)
4. Heß, A., Kushmerick, N.: Learning to attach semantic metadata to web services. In: ISWC. pp. 258–273 (2003)
5. Inverardi, P., Issarny, V., Spalazzese, R.: A theory of mediators for eternal connectors. In: ISoLA (2010)
6. Joachims, T.: Learning to Classify Text Using Support Vector Machines. Kluwer Academic Publishers (2002)
7. Klusch, M., Kapahnke, P., Zinnikus, I.: Sawsdl-mx2: A machine-learning approach for integrating semantic web service matchmaking variants. In: ICWS (2009)
8. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability (1967)
9. Moschitti, A.: Kernel methods, syntax and semantics for relational text categorization. In: Proc. of CIKM (2008)
10. Moschitti, A., Basili, R.: Complex linguistic features for text classification: A comprehensive study. In: Proc. of ECIR (2004)
11. Oldham, N., Thomas, C., Sheth, A.P., Verma, K.: Meteor-s web service annotation framework with machine learning classification. In: SWSWPC (2004)
12. 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 (2002)
13. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Tech. Rep. TR74-218, Cornell University (1974)