J Intell Inf Syst (2016) 47:375–401
DOI 10.1007/s10844-015-0369-0

Characterization and search of web services through intensional knowledge

Devis Bianchini[1] · **Paolo Garza**[2] · **Elisa Quintarelli**[3]

Received: 26 July 2013 / Accepted: 8 June 2015 /
Published online: 26 June 2015

Abstract Web service technologies are widely adopted to access services and compose new applications starting from software components available from the shelf. Consequently, more and more service descriptions are becoming available on the network to designers, who often filter them according to keyword-based search, thus obtaining huge amounts of matching results, that, if not properly controlled, lead to an information overload that might cause confusion rather than knowledge. In this paper, we propose to apply data mining techniques to SOAP-based service descriptions in order to infer patterns providing a summarized and integrated representation of service functionalities. These patterns provide succinct (intensional) knowledge that can be directly queried or used to drive exploratory searches. Specifically, we propose W-DREAM (Web services DiscoveRy via intEnsionAl knowledge Mining), an infrastructure to perform intensional service representation and querying to support application designers to select the web services that best suit their needs.

Keywords Intensional knowledge representation · Intensional knowledge querying · Web service search engine · Data mining

✉ Paolo Garza
paolo.garza@polito.it

Devis Bianchini
devis.bianchini@ing.unibs.it

Elisa Quintarelli
elisa.quintarelli@polimi.it

[1] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Via Branze 38, 25123 Brescia, Italy

[2] Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

[3] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

1 Introduction

SOAP-based Web services enable interoperability between software components implemented in a distributed, possibly heterogeneous, environment, by providing a platform-independent way to expose their software interface (specified using the Web Service Description Language, WSDL) and by activating the operations included in the interface through messages expressed according to the Simple Object Access Protocol (SOAP). Platform independency makes Web services a powerful tool to develop new web applications starting from existing components, although implemented and deployed using different languages. Service providers make available their own functionalities as Web services, by providing WSDL documents that are used by web designers to develop new, service-oriented applications, which invoke component Web services in an orchestrated way. The *trait d'union* for the interaction between providers and service-oriented application designers is a repository of Web services where they can be searched and selected for aggregation. In this paper, we focus on Web service search, with the aim of supporting service-oriented application composition.

We rely on a paradigm for Web service search that is *iterative* and *explorative*. It is *iterative* since the web designer selects and aggregates available Web services step-by-step, with the aim of building a composite service-oriented application. It is *explorative* since the web designer has not in mind a complete list of useful services for reaching her goal, when she starts aggregating available Web services; indeed, at each step she chooses among proposed Web services, adaptively evolving the goal according to the contents of the repository, the suggestions of the system, and the current status of the application under development (i.e., the set of already selected Web services). This differentiates our approach from the ones targeted to Web service composition, that are based on AI planning and automated synthesis techniques (see Hull (2005) and Rao and Su (2005) for a survey). These approaches take as input a well-defined goal, expressed as a designer request (e.g., book a trip to Rome), often using formal languages, and try to generate a consistent plan (e.g., a sequence of service invocations) such that the goal is satisfied. These approaches treat the goal as a black-box and might lead to higher performance complexity (e.g., NP-hard (Hull 2005)), without enabling web designer to have any control on the intermediate service selection steps to obtain the final application.

Unfortunately, existing Web service repositories do not have the pre-requisites for satisfying an iterative and explorative search. They do not enable an iterative search, since they have not been designed to serve service aggregation, but only atomic service search and selection. Furthermore, they present several limitations that have been highlighted in Sabou and Pan (2005). In particular: (i) they only provide basic keyword-driven search, where specified keywords are searched within the WSDL description of the Web service (e.g., BindingPoint, .NET XML Web Services Repertory); (ii) they provide browsing through few, coarse-grained categories which are not able to express the distinguishing features of all Web services they gather (WebServiceX.NET, Web Service List, `Seekda`, Xmethods); (iii) some of them enable multi-facet based browsing, where facets suffer from the same limitations of categories and it is not possible to formulate queries which combine more facets together. These repositories allow for specifying very general queries, but the identification of the best Web service for the application under development among the large set of returned ones is not provided. Semantic-based Web service discovery approaches (e.g., SWoogle (Dong et al. 2004)) have been proposed to face such limitations and improve search precision and recall, but they have not been designed for explorative purposes, since

they force web designers to pose very specific queries, that is, they are suitable when the designer exactly knows the service she is looking for.

As often happens, large amount of information leads to an information overload that may cause confusion rather than knowledge. Therefore, the increased number of available services (`Seekda` counts over 28,600 Web services, Xmethods several hundreds of services, numbers which are ever growing) motivated the design of a new system to: (i) provide the visualization of a compact, summarized representation of Web service repository contents, in particular when they are highly populated; (ii) provide a query language on such a compact representation, in order to enable an explorative search within the repository; (iii) take into account the current status of the application under development while providing such a representation, thus supporting an iterative Web service selection and aggregation.

For instance, consider a designer who is in charge of implementing an application for the participants of an international academic conference, also including a hotel booking service. Suppose that the designer has to select one of the available hotel booking services. She has not in mind what are exactly the inputs/outputs of the Web service to include in the application under development. Therefore, she performs a generic search using `hotel` and `booking` as keywords for the `service category`. If the `Seekda` search engine is used, the designer obtains 36 Web services among search results, split on four pages[1]. The designer will probably select randomly one service among the returned ones in the first or the second page, because the in-depth analysis of all Web service details, by inspecting their WSDL documents, would be a time-consuming task. Now consider the following intensional answer:

90 % of the hotel booking services have the credit card number as input parameter

An *intensional answer* (Pirotte et al. 1991; Motro 1994) gives a summarized representation of the query results (in our case, Web services), which represent the so-called *extensional answer*. In our example, the intensional answer suggests that the most of the available hotel booking Web services is characterized by the credit card number as input parameter. This knowledge can be exploited by the designer to focus her choice among Web services which present this parameter. Since intensional answers are very compact, they are more easily manageable by human beings, therefore might be useful in an exploratory search system. Moreover, such compact answer can suggest the designer to look for Secure Payment Web services to be included in the web application she is developing, to manage payment requests via credit card. Hence, the provided intensional answer suggests the next needed service and the potential query to submit, thus enabling iterative query refinement.

The idea is that Web services whose interface contains the most popular parameters (i) are the best candidates to satisfy the requirements of many users and (ii) can be more easily substituted by other Web services if needed. Intensional answers help the designer to identify these sets of Web services.

The very simple example reported above shows only one possible use of intensional knowledge in the proposed Web service searching system. In the following sections, we will also describe how the intensional knowledge can be used to mine interesting information from the log files recording Web service invocations. In particular, the use of log files allows selecting a Web service not only by considering its interface, but also the data (i.e., the content) that can be accessed through it.

[1]Numbers in this example refer to the status of the `Seekda` search engine on November 11th, 2012.

In this paper, we propose W-DREAM (**W**eb services **D**iscove**R**y via int**E**nsion**A**l knowledge **M**ining), an infrastructure to perform intensional service representation and querying to support service-oriented application design. Our contributions are the following:

- the definition of a set of patterns used to provide an intensional representation of Web service interfaces and data accessible through them; patterns can be used to provide intensional answers to designer's queries when extensional answers are composed of a large set of services;
- the definition of a data mining algorithm for the extraction of the patterns defined above;
- an intensional querying system that supports designers to query the patterns in order to explore the Web service repository for iterative development of new service-oriented applications.

The structure of the paper is the following: in Section 2 we provide preliminary definitions and we further motivate the paper with a running example; Sections 3 and 4 describe in detail patterns representation, mining and querying; the W-DREAM architecture, which implements the patterns repository and the intensional querying system, is described in Section 5; experiments are presented in Section 6; finally, related and future work are discussed in Sections 7 and 8, respectively.

2 Background

In this section we introduce the running example used throughout the paper and some background definitions about data mining.

2.1 Running example

Consider the example of the service-oriented application for the international academic conference to be organized in Italy, introduced above. The application must support both the preliminary activities (e.g., participants' on line registration and hotel reservations) and the on site activities (e.g., searching for a good ethnic restaurant for dinner when attending the conference). Rather than implementing the whole application from scratch, the designer aims at reusing available hotel booking services, secure on-line payment services or search services to enable conference participants to find out hotels, restaurants and other facilities nearby the conference venue.

Performing a keyword-based search on `Seekda`, the designer may find: (i) about 36 hotel booking services; (ii) about 3139 search services; (iii) about 40 on-line payment services. Less populated sets of search results can be obtained only if the designer is able to issue a more precise Web service request, but this is not possible in an exploratory search scenario, such as the one we are considering here, where the designer aims at inspecting the repository of services to decide which ones could be selected and inserted within the application under development. Among the obtained results, the designer has to inspect the WSDL of each service to find out details about input/output parameters: for instance, the designer might select an on-line payment service to be included in the final application only if the hotel booking services enable payment by credit card. Moreover, the designer should be able to check if services enable to book a hotel in the city where the conference is located or what are the credit card types accepted by on-line payment services.

All this information is of paramount importance to perform focused service selection and aggregation.

Our aim within W-DREAM is to use data mining techniques to extract useful intensional representation of Web service repository contents, in order to help the designer to learn some new and previously unknown succinct knowledge about the available services and make a more focused choice.

Consider our running example and suppose the designer is looking for the first Web service to be included in the service-oriented application under development. The designer starts the search process by specifying a service category (e.g., service category=hotel booking). We denote this kind of query as *category-driven*. The search engine of the system provides the following *intensional answers*:

60 % of the hotel booking services have the credit card and the city as input parameters

35 % of the hotel booking services have the hotel category as output parameters

The percentage can be used by the designer to choose the best subset of services for her application. Since 60 % of hotel booking services present the credit card and the city as input parameters, the designer might suppose that these two parameters are important for the hotel room booking providers. The same consideration holds for the hotel category as output parameter. If available, a hotel booking Web service with at least the three parameters highlighted by the intensional answer (credit card, city and hotel category) should be selected.

Now W-DREAM can suggest other useful services that can be combined with the previous designer's choice, by automatically applying a *parameter-driven query* that looks for services having some parameters in common with the one related with previously selected services. This new query, which can be applied iteratively, can be exploited to suggest additional Web services to be included in the service-oriented application under development. As an example, suppose that the designer selected a hotel booking service that presents an address among its output parameters. The system will automatically create and execute a query searching for all the Web services characterized by an (input) parameter of type address. A potential useful intensional answer proposed to the designer is the following one:

65 % of services which have an address among its parameters are mapping services

This is an example of even more explorative search, since the designer was not originally thinking on mapping services to be included within the application under development, but she starts considering them after the system visualizes the intensional answer.

The intensional answers shown here only consider the signature of the available services (e.g., the type of input and output parameters) or the categories they belong to and provide a functional classification of services. However, if the values assumed by the input and output parameters are also known (for instance, extracting them from SOAP messages exchanged with Web services), then better choices can be performed. Consider the following intensional answers:

100 % of on-line payment services accept VISA as credit card type

80 % of hotel booking services which have been invoked with Rome as the city input parameter present the IBAN code as input parameter

5 % of hotel booking services which have been invoked with Rome as the city input parameter present the credit card number as input parameter

The designer might use this kind of patterns, that combines information coming from the service description repository and the invocation log files, to better focus the attention on a subset of services. For instance, the first pattern could be used to focus on hotel booking services which enable the use of VISA as credit card type, since this choice ensures the availability of on-line payment services which accept VISA credit cards, if required. The other two patterns suggest that, since in Italy credit cards are frequently not used, the majority of the services that allow booking hotels in Rome are characterized by the IBAN code (International Bank Account Number). Since the application under development is related to an Italian conference located in Rome, the designer may decide to include within the service-oriented application, a hotel booking service which accepts bank transfer as payment type. Note that by using additional information about the data (i.e., content) associated with the available services, the designer might choose a Web service that is not the same picked before. However, the use of the intensional answers mined from the combination of log files and web service descriptions can be applied with success only when a significant number of invocations have been logged. If small log files are considered the mined patterns could be not statistically significant and hence they could be not representative of all the content accessible through the indexed web services. We provide this additional facility within the WDREAM system; the constraints and pre-requisites of this facility will be detailed in the next sections.

2.2 Itemset and association rule mining problem

Given the running example described above, the problem of extracting intensional answers from a Web service repository can be faced by applying data mining techniques to extract *association rules* from a set of data items. The general problem can be described as follows. We will adapt it to the Web service context in Section 2.3, for which we will provide examples.

Given a dataset D, composed of sets of items, *itemset mining* consists of extracting those subsets of items, called itemsets in the following, that are frequent in D (Agrawal and Srikant 1994). Suppose that D is the dataset of a supermarket, where each record represents the set of items bought by a customer. By applying an itemset mining algorithm on D, enforcing a minimum frequency threshold *minsup*, all the itemsets composed of items that appear together in at least *minsup* records are extracted. Each of the extracted itemsets is called *frequent itemset*.

Definition 1 (Transaction) More formally, let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a set of items. A transaction t is defined as a subset of $\mathcal{I}$ (i.e., $t \subseteq \mathcal{I}$), while a dataset D is a set of transactions.

Let X be an itemset in $\mathcal{I}$ (i.e., $X \subseteq \mathcal{I}$). Given a transaction $t \in D$, we say that X matches t if $X \subseteq t$. A *support* value is usually associated to each itemset X, where the support of X (denoted as $sup(X)$) is defined as the fraction of transactions in D matched by X. Given a minimum frequency threshold *minsup*, the itemset mining problem consists in extracting from D all the itemsets X such that $sup(X) \geq minsup$. By exploiting frequent itemsets, association rules can be extracted.

Definition 2 (Association rule) An association rule is an implication in the form $X \Rightarrow Y$, where both X and Y are itemsets. An association rule is used to quantify the likelihood of finding the itemset Y within the dataset D given the presence of the itemset X in the same dataset. The quality of an association rule is usually measured by means of its support, that is, $sup(X \cup Y)$, and its confidence. Confidence, denoted with $conf(X \Rightarrow Y)$, corresponds to the conditional probability of finding Y, having found X and is given by

$$conf(X \Rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)} \tag{1}$$

The association rule mining task consists of extracting from D all the rules with a support at least equal to a minimum support threshold (*minsup*) and a confidence at least equal to a minimum confidence threshold (*minconf*).

2.3 Transactional representation of web service repository

We model a SOAP-based Web service repository as a set of Web service descriptors, defined as follows.

Definition 3 (Web service descriptor) A Web service descriptor WSD^i is a pair

$$WSD^i = \langle C^i, WSDL^i \rangle \tag{2}$$

where: C^i represents the set of categories where the Web service has been classified (to be as more general as possible, we admit that each Web service can be classified in more than one category); $WSDL^i$ is the WSDL document that describes the Web service interface (*abstract part*) and how to invoke its operations through the SOAP protocol (*concrete part*).

Although the only standard that has been defined for the structure of a Web service repository, namely the UDDI (Universal Description, Discovery and Integration[2]), did not received the diffusion that has been foreseen, nevertheless contents of existing repositories can be abstracted using the structure formalized in (2). They may also contain additional information (e.g., on the Web service providers) which we do not consider because they are out of the scope of our approach. Like traditional SOA design, in our approach we require the providers to expose only the WSDL of their Web services to show how to properly invoke Web service functionalities. Consider for example the WSDL document of the hotel booking service shown in Fig. 1.

The document describes the Web service operation `bookRoom`, that has five inputs and three outputs, each of them described through a name and a type. Paramater types are qualified names uniquely identified through their name and the namespace where they are defined. Some input/output paramater types belong to the set of XML Schema built-in types (e.g., `string` or `integer`, qualified through the `xsd` suffix), other ones are domain-specific parameter types (e.g., `NumberOfStars`, `CreditCardType` or `Price`), qualified through the `travel` suffix. The concrete part (within the `wsdl:binding` tags) describes SOAP binding for exchanging messages with the `bookRoom` operation. The distinguishing feature is the value of `style` attribute of the `soap:binding` tag. For example, if `style="rpc"`, request/response SOAP messages exchanged with the operation are serialized as shown in Fig. 2, while if `style="document"` they are serialized

[2]See http://uddi.xml.org/.

```
<wsdl:definitions
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:travel="http://localhost:8080/travelSchema.xsd">

  <wsdl:message name="bookRoomRequest">
    <wsdl:part name="City" type="xsd:string"/>
    <wsdl:part name="Country" type="xsd:string"/>
    <wsdl:part name="NumberOfPersons" type="xsd:integer"/>
    <wsdl:part name="CreditCard" type="travel:CreditCardType"/>
    <wsdl:part name="CreditCardNumber" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="bookRoomResponse">
    <wsdl:part name="HotelName" type="xsd:string"/>
    <wsdl:part name="NumberOfStars" type="travel:NumberOfStars"/>
    <wsdl:part name="Price" type="travel:Price"/>
  </wsdl:message>
  <wsdl:portType name="FindHotel">
    <wsdl:operation name="bookRoom">
      <wsd:input message="bookRoomRequest" name="bookRoomIn"/>
      <wsd:output message="bookRoomResponse" name="bookRoomOut"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="FindHotelBinding" type="FindHotel">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="bookRoom">
      <soap:operation soapAction=""/>
      <wsdl:input name="bookRoomRequest">
        <soap:body use="literal">
      </wsdl:input>
      <wsdl:output name="bookRoomResponse">
        <soap:body use="literal">
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="FindHotelService">
    <wsdl:port binding="FindHotelBinding" name="FindHotelService">
      <soap:address location="http://..."/>
    </wsdl:port>
  </wsdl:service/>
</wsdl:definitions>
```

Fig. 1 An example of WSDL document of an hotel booking service

as shown in the same figure, but the tags `bookRoom` which denote the operation are omitted. The way `Price` is expressed in the response (that is, with the addition of an attribute `currency` to denote the adopted currency) is defined in the `travel` namespace[3].

Given the WSDL specification $WSDL^i$ of the Web service descriptor WSD^i, we extract association rules both from the abstract part of the WSDL, concerning the description of Web service interface, and from the log files of Web service invocations, concerning instantiations of SOAP messages exchanged with the service. To extract association rules, the descriptions of Web service interfaces within the repository, as well as the

[3]More details on SOAP binding and encoding rules can be found at http://www.w3.org/TR/soap/.

Fig. 2 Example of SOAP messages exchanged with the hotel booking service

```
# SOAP request
<soap:Envelope>
  ...
  <soap:Body>
    <bookRoom>
      <City>Rome</City>
      <Country>Italy</Country>
      <NumberOfPersons>2</NumberOfPersons>
      <CreditCard>VISA</CreditCard>
      <CreditCardNumber>***</CreditCardNumber>
    </bookRoom>
  </soap:Body>
</soap:Envelope>

#SOAP response
<soap:Envelope>
  ...
  <soap:Body>
    <bookRoomResponse>
      <HotelName>RomanEmpireHotel</HotelName>
      <NumberOfStars>Five</NumberOfStars>
      <Price currency="euro">125</Price>
    </bookRoomResponse>
  </soap:Body>
</soap:Envelope>
```

contents of log files, must be expressed as sets of transactions as explained in the following of this section. We denote the former with *transactional service dataset* (*TSD*) and the latter with *transactional log dataset* (*TLD*). Specifically, in the transactional service dataset each operation is represented as a transaction, composed of the following items:

- an item `(serviceID=value)`, representing the identifier (within W-DREAM system) of the service which the operation belongs to;
- one or more items `(serviceCategory=value)`, representing the categories associated with the service which the operation belongs to;
- an item `(operationName=value)`;
- one or more items `(input=inputName)`, representing the names of the input parameters;
- one or more items `(inputParameterType=inputType)`, where each `inputType` is a qualified name representing the type of an input parameter;
- one or more items `(output=outputName)`, representing the names of the output parameters;
- one or more items `(outputParameterType=outputType)`, where each `outputType` is a qualified name representing the type of an output parameter.

Consider the abstract part of the WSDL document reported in Fig. 1. The `bookRoom` operation of the service is represented through the transaction reported in Fig. 3.

Association rules which are extracted from the contents within exchanged SOAP messages may provide additional information about the Web service to select, as shown in the running example. To extract these rules, the available service invocations, obtained from Web service invocation logs, are expressed as a set of transactions in the transactional log dataset, where each transaction represents an invocation of a Web service operation. Each transaction is composed of the following items:

- an item `(serviceID=value)`, representing the identifier of the invoked service;

```
{(serviceID=1),(serviceCategory="Travel"),(operationName="bookRoom"),
 (inputName="City"),(inputParameterType="xsd:string"),
 (inputName="Country"),(inputParameterType="xsd:string"),
 (inputName="NumberOfPersons"),(inputParameterType="xsd:integer"),
 (inputName="CreditCard"),(inputParameterType="travel:CreditCardType"),
 (inputName="CreditCardNumber"),(inputParameterType="xsd:string"),
 (outputName="HotelName"),(outputParameterType="xsd:string"),
 (outputName="NumberOfStars"),(outputParameterType="travel:NumberOfStars"),
 (outputName="Price"),(outputParameterType="xsd:decimal") }
```

Fig. 3 Transactional representation of Web service descriptor operations

- one or more items `(inputParameterName_$IN=value)`, where `inputParameterName` is the name of an input parameter and `value` is the parameter value, as extracted from SOAP messages;
- one or more items `(outputParameterName_$OUT=value)`, where `outputParameterName` is the name of an output parameter and `value` is the parameter value, as extracted from SOAP messages.

Consider the invocation reported in Fig. 2; it is represented through the transaction reported in Fig. 4.

3 Intensional knowledge management

The proposed system includes an intensional knowledge manager module that mines a compact, summarized representation of the most interesting (frequent) association rules from both the transactional service dataset and the transactional log dataset. We propose three different types of association rules that we consider relevant for supporting iterative and explorative Web service search. We denote them as *patterns*.

Signature pattern A signature pattern is a type of association rule that represents frequent correlations between the information available in the descriptions of Web services (e.g., Web service category, type and name of parameters).

Content pattern A content pattern is a type of association rule that represents frequent correlations within data extracted from the log files of the invoked Web services.

Signature&content pattern A signature&content pattern is a type of association rule that represents frequent correlations obtained by combing both the descriptions of Web services and data extracted from log files.

In the following we formally define patterns. Then, we show how data mining techniques can be used to extract the patterns. Finally, the physical representation of patterns is described.

```
{(serviceID=1),(City_$IN="Rome"),(Country_$IN="Italy"),
 (NumberOfPersons_$IN="2"),(CreditCard_$IN="VISA"),
 (CreditCardNumber_$IN="***"),(HotelName_$OUT="RomanEmpireHotel"),
 (NumberOfStars_$OUT=‘‘Five’’),(Price_$OUT="125")}
```

Fig. 4 Transactional representation of a Web service invocation

Definition 4 (Signature pattern) Formally, a signature pattern is represented as follows. Let be $\mathcal{I}^s$ the set of items such that $TSD \subseteq \mathcal{P}(\mathcal{I}^s)$, where $\mathcal{P}(\mathcal{I}^s)$ denotes the power set of $\mathcal{I}^s$. This means that $\mathcal{I}^s$ contains only items that compose transactions in *TSD*. A signature pattern is an association rule $X \Rightarrow Y$ such that $X, Y \subseteq \mathcal{I}^s$.

This pattern allows for representing the main characteristics of the indexed web services in terms of input and output parameters and categories. The following table provides some examples of association rules that are expressed according to the signature pattern.

	X	⇒	Y	sup	conf
P1.1	(ServiceCategory= "HotelBooking")	⇒	(inputParameterType= "City"), (inputParameterType= "CreditCardType")	5 %	90 %
P1.2	(ServiceCategory= "Booking")	⇒	(outputParameterType= "Hotel")	4%	20%
P1.3	(ServiceCategory= "RestaurantSearch")	⇒	(inputParameterType= "foodType")	5 %	60 %
P1.4	(ServiceCategory= "HotelBooking")	⇒	(outputParameterType= "hotelCategory")	4 %	70 %

The first rule specifies that 90 % of the hotel booking services are characterized by at least two input parameters of type `City` and `CreditCardType`, respectively, by considering the confidence value of the rule. Hence, 90 % of the hotel booking services can be invoked only providing at least the city where the hotel to book should be located and by specifying a credit card for booking. Since the support of the rule is equal to 5 %, we can learn that 5 % of the Web services available in the registry are hotel booking services and are characterized by at least two input parameters of type `City` and `CreditCardType`, respectively. The second rule provides information regarding the output parameters of the services of the macro category `Booking`. According to the rule, 4 % of the services in the registry are booking services with a hotel as output. Among booking services, 20 % are characterized by an output parameter of type `Hotel` (hence, 20 % of the booking services are probably hotel booking services). The third rule specifies that in the 60 % of cases it is possible to specify the food type while searching for a restaurant (i.e., 60 % of the `restaurantSearch` services are characterized by an input parameter of type `FoodType`). The last rule specifies that 70 % of the available hotel booking services are characterized by at least an output parameter of type `hotelCategory`.

Definition 5 (Content pattern) Formally, a content pattern is represented as follows. Let be $\mathcal{I}^c$ the set of items such that $TLD \subseteq \mathcal{P}(\mathcal{I}^c)$, where $\mathcal{P}(\mathcal{I}^c)$ denotes the power set of $\mathcal{I}^c$. This means that $\mathcal{I}^c$ contains only items that compose transactions in *TLD*. A content pattern is an association rule $X \Rightarrow Y$ such that $X, Y \subseteq \mathcal{I}^c$.

The content pattern represents correlation at the data level, i.e., correlation among the values assumed by the parameters of the services extracted from log files. The following table provides some examples of association rules that are expressed according to the content pattern.

	X	⇒	Y	sup	conf
P2.1	(City_$IN="Rome")	⇒	(NumberOfStars_$OUT="Five")	9 %	90 %
P2.2	(City_$IN="Rome")	⇒	(NumberOfStars_$OUT="Four")	1 %	10 %
P2.3	(City_$IN="Ostia")	⇒	(NumberOfStars_$OUT="Three")	5 %	80 %
P2.4	(City_$IN="Florence")	⇒	(NumberOfStars_$IN="Three")	5 %	80 %

The first two rules highlight a strong correlation between the city of Rome (Rome is the value of the input parameter `City`) and the stars of its hotels (in 90 % of the cases when the city input parameter of a service is set to Rome then the number of stars of the returned hotel is a five star hotel, while in the other 10 % of the cases the returned number of stars is four). The third rule shows that in 80 % of the analyzed data, when the city of the hotel is Ostia (a city near Rome), the number of stars is three. The first three rules, which summarize the correlations between cities and number of stars of the hotels, can be used when developing an application for a conference located in Rome. Since it is well known than many of the conference participants, in particular PhD students, cannot book expensive hotels, the application designer would probably focus her attention on Web services that allow for booking low and medium level hotels located near Rome. By using the intensional answer about the city of Ostia, which is close to Rome, the designer may decide to choose a Web service that allows for booking also hotels in Ostia for the conference located in Rome.

Definition 6 (Signature&content pattern) A signature&content pattern is represented as follows. Let be $\mathcal{I}^s$ and $\mathcal{I}^c$ the sets of items defined above for signature and content patterns, respectively. A signature&content pattern is an association rule $X \Rightarrow Y$ such that $X, Y \subseteq \mathcal{I}^s \cup \mathcal{I}^c$.

The signature&content pattern combines both signature of Web services and content data accessible through their interfaces. Examples of association rules that are expressed according to the signature&content pattern are shown in the following table.

	X	⇒	Y	sup	conf
P3.1	(ServiceCategory= "HotelBooking"), (City_$IN="Rome")	⇒	(inputParameterType= "CreditCardType")	2 %	100 %
P3.2	(ServiceCategory= "HotelBooking"), (City_$IN="Florence")	⇒	(CreditCardType_$IN= "Visa")	5%	75 %

The first rule indicates that 100 % of the hotel booking services to reserve a hotel located in Rome (content information) require the credit card to complete the reservation (service description information). Hence, a credit card is always needed to book a hotel in Rome.

3.1 Interestingness measures

Traditional measures, such as support and confidence, can be exploited to select the most interesting rules from the Web service registry to summarize Web service descriptions and invocation records. However, we define a new measure to characterize content and signature&content patterns for Web service analysis, because we argue that some features of these patterns can not be represented by using traditional measures. The

Table 1 An example (simplified) of transactional log dataset

{ (serviceId=1),($City\_\$IN$ =Rome),($Number of Stars\_\$OUT$ =Five) }
{ (serviceId=1),($City\_\$IN$ =Rome),($Number of Stars\_\$OUT$ =Five) }
{ (serviceId=1),($City\_\$IN$ =Rome),($Number of Stars\_\$OUT$ =Five) }
{ (serviceId=2),($City\_\$IN$ =Rome),($Number of Stars\_\$OUT$ =Five) }
{ (serviceId=2),($City\_\$IN$ =Florence),($Number of Stars\_\$OUT$ =Five) }
{ (serviceId=2),($City\_\$IN$ =Florence),($Number of Stars\_\$OUT$ =Four) }

new measure we propose, called $distinctServices$, counts for each association rule p expressed according to the content pattern the number of distinct services associated with p. Consider the example of transactional log dataset *TLD* reported in Table 1 and the rule $p_1 : \{(City\_\$IN = '' Rome'') \Rightarrow (Number of Stars\_\$OUT = '' Five'')\}$. The *TLD* contains the invocations of two hotel booking services, identified by two distinct `serviceID`. The support of p_1 is 4 (absolute support) because it matches 4 transactions of *TLD*. However, the number of distinct services matching the rule is 2. We define the *distinctService* measure as this number, formally defined as follows.

Definition 7 (Distinct service measure) Let *TLD* be a transactional log dataset, *X* a pattern mined from *TLD*, and *M* the set of transactions in *TLD* matched by *X*. Each transaction in *TLD* is characterized by a service identifier. The distinct service measure, denoted as $distinctServices(X)$, is defined as the number of distinct service identifiers in *M*.

Hence, given a rule expressed according to the content or signature&content patterns, its support represents the number of invocations of services that match the rule, while the value of $distinctService$ represents the number of distinct Web services in the repository whose invocations match the rule. The $distinctServices$ and the *support* measures serve different targets. The $distinctService$ measure can be used, for example, to understand how easily a Web service can be substituted by another one. In fact, if the value of $distinctServices$ is high it means that many different services have invocation logs that match a given rule. Hence, a significant set of exchangeable services are probably available if $distinctServices$ is high.

3.2 Rule mining algorithm

Given the transactional representation of Web service descriptors and their invocations stored within *TSD* and *TLD*, traditional rule mining algorithms can be applied to extract the rules, according to the patterns described above. However, traditional rule mining algorithms cannot be used to compute straightforwardly the $distinctService$ measure for content and signature&content patterns. Two possible approaches can be used to compute this measure for each pattern:

1. the introduction of proper data structures in the current itemset and rule mining algorithms in order to store for each rule expressed according to content or signature&content patterns the set of services whose invocation match the rule, and consequently the number of distinct services for each rule;

```
Algorithm: Computation of the distinctService measure

Input: Transactional log dataset TLD, frequent pattern set R
Output: The value of distinctService for each pattern in R

 1: for each r ∈ R do
 2:   serviceIDs = ∅;
 3:   for each l ∈ TLD do
 4:     if (r ⊆ l) then
 5:       serviceIDs = serviceIDs ∪ {l.serviceID};
 6:     end if
 7:   done
 8:   r.distinctService = cardinality of serviceIDs;
 9: done
```

Fig. 5 Computation of the $distinctService$ measure

2. the application of a post-processing step, after the application of traditional rule mining techniques.

We decided to propose a solution based on the post-processing approach. We currently discarded the first approach because the integration of an ad-hoc data structure, able to store for each rule the set of related services, is not efficient for what concerns memory space consumption. The use of a post-processing step allows for efficient mining of the set of frequent rules. In fact, only the limited set of selected rules, used to synthetically represent Web service descriptions and their invocations, are analyzed in the post-processing step in order to compute the $distinctService$ measure.

The pseudo code of the post-processing step is reported in Fig. 5. The post-processing step is based on two loops. An outer loop is used to scan the set of selected frequent rules (*R*) (lines 1-9), while an inner loop scans, for each rule, the transactional log dataset *TLD* from which the rules have been extracted (lines 3-7). In the inner loop, for each rule *r* the set of service identifiers associated with the transactions matched by *r* is generated. The value of the $distinctService$ measure for *r* is set to the cardinality of the set $serviceIDs$ (line 8). As it is well known in the database query optimization area, if data is properly sorted, the computation of counting operations, such as `count(distinct serviceID)`, is improved. In this particular case the set $serviceIDs$ and its cardinality can be computed more efficiently if the transactions in *TLD* are sorted by `serviceID`. Rule mining algorithms have been implemented in the W-DREAM architecture as described in details in Section 5.

Moreover, the proposed algorithm is executed periodically. Hence, it is a periodic off-line operation used to refresh the content of the intensional knowledge repository exploited to provide intensional answers to web designers. The periodicity of the mining algorithm execution depends on the dynamics of the Web service repository (e.g., how often new Web service descriptions are added, how often Web services are invoked and invocation data are stored in the *TLD*).

3.3 Physical representation of association rules

The described rules can be stored on disk by exploiting different approaches. We decided to use XML to store the extracted patterns since XML is platform-independent and Web service standards (namely, WSDL and SOAP) are XML-based as well. Hence, a common data model and query engine can be used to query both Web service descriptions

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT RuleSet (AssociationRule+)>
<!ELEMENT AssociationRule (RuleBody, RuleHead)>
<!ATTLIST AssociationRule NumberItemHead CDATA #REQUIRED
                          NumberItemBody CDATA #REQUIRED
                          support CDATA #REQUIRED
                          distinctServices CDATA #REQUIRED
                          confidence CDATA #REQUIRED>
<!ELEMENT RuleBody (item+)>
<!ELEMENT RuleHead (item+)>
<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item Element CDATA #REQUIRED>
```

Fig. 6 DTD associated with the XML document for physical representation of association rules

and their summarized represen tation. To store rules we use the XML representation proposed in (Baralis et al. 2007), extended in order to store rules according to signature, content and signature&content patterns and to compute the $distinctServices$ measure. The DTD of the proposed representation is reported in Fig. 6. The XML document is composed of a set of rules. Each rule is characterized by the antecedent (`body`) of the rule, its consequent (`head`) and the measures of interest (`support`, `confidence` and `distinctServices`). Both the antecedent and the consequent of rules are represented as sets of items, where each item is a pair `(Element,value)`. Depending on the type of pattern, `Element` can be the name of an input/output parameter, the label `ServiceCategory`, etc. Since each type of pattern represents different characteristics of the analyzed services, we propose to use an XML document for each type.

4 Intensional knowledge querying

W-DREAM allows designers to issue different types of queries and to iteratively refine the search criteria being guided by the summarized representation of available Web services and their past invocations. We define a query q as follows:

$$q = \langle COND_q, t_q \rangle \tag{3}$$

$COND_q$ is a condition used for filtering available services, defined as $item_1 = value_1 \wedge \ldots \wedge item_n = value_n$. Exploiting the intensional representation of Web service descriptions and invocations, the W-DREAM infrastructure proposes to the designer the set of association rules whose antecedent matches the itemsets composing $COND_q$. We denote this set as the *Set of Matching Rules (SMR)*. A mined rule $X \Rightarrow Y \in SMR$ iff

$$\{(item_1, value_1), \ldots, (item_n, value_n)\} \subseteq X \tag{4}$$

The element t_q is the query type, among *'single search'*, *'completion search'* or *'proactive search'*. We will introduce the different query types t_q with the help of a service selection scenario (which is represented through the flowchart shown in Fig. 7).

In the flowchart shown in Fig. 7a, the iterative nature of the selection process is evident. Interactions with the web designer in the flowchart are intended to be performed through the W-DREAM Web interface. The designer firstly formulates a query in the

query formulation area and the system performs the query processing. Given the set of rules SMR returned by the system, the designer browses the list of intensional answers returned by the query and, in case, selects the rule $X \Rightarrow Y \in SMR$ she considers more relevant to guide the selection of the services suitable for her final application. From the transactional representation of services, W-DREAM extracts the descriptions $\mathcal{S}$ of services satisfying R, i.e. the identifiers, category, parameters of services with a transaction containing both X and Y. At this point, the designer might select a service $s \in \mathcal{S}$ (see detailed flowchart portion shown in Fig. 7b). After the first selection, the designer may iteratively repeat the search and selection of a new Web service according to the three different query types.

The *single search* query is equivalent to the first query issued by the designer. Its goal is to search and select a new service independently of previously selected Web services.

In the *proactive search* query, the system iteratively starts an autonomous new search q' of services that can be combined with the selected services, that is, present among their inputs the outputs of already selected services. Let $o_1, \ldots, o_n$ be the type of the output parameters of s. The condition $COND_{q'}$ is formalized as $(inputParameterType = o_1) \wedge \ldots \wedge (inputParameterType = o_n)$ and is used to retrieve intensional descriptions of services with input parameters related to the output of s (i.e., parameters of the same type). In this case, W-DREAM proposes to the designer the set of mined rules matching in the antecedent the itemsets describing $COND_{q'}$. In this case, a mined rule $X \Rightarrow Y \in SMR$ iff $\{(inputParameterType, o_1) \wedge \ldots \wedge (inputParameterType, o_n)\} \cap X \neq \emptyset$. Note that, when the query is generated by the system, the notion of matching is less strict; indeed, we require a non-empty intersection between the query condition and the body of the mined rules.

The *completion search* query is a combination of the two previous types of queries; the designer formulates the $COND_q$ condition and checks one or more services among the already selected ones.

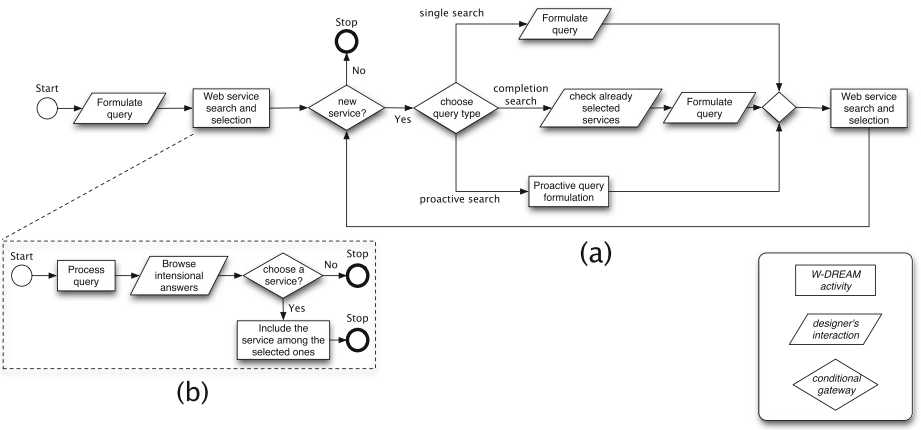


Fig. 7 A Web service selection scenario

Consider our running example and suppose the designer's first query is *"What about services that satisfy ServiceCategory = Hotel Booking?"*. W-DREAM returns the following set SMR of association rules:

	X	⇒	Y	sup	conf
P1.1	(ServiceCategory= "HotelBooking")	⇒	(inputParameterType= "City"), (inputParameterType= "CreditCardType")	5 %	90 %
P1.4	(ServiceCategory= "HotelBooking")	⇒	(outputParameterType= "hotelCategory")	4 %	70 %
P3.1	(ServiceCategory= "HotelBooking"), (City_$IN="Rome")	⇒	(inputParameterType= "CreditCardType")	2 %	100 %
P3.2	(ServiceCategory= "HotelBooking"), (City_$$IN$$ ="Florence")	⇒	(CreditCardType_$$IN$$ = "Visa")	5 %	75 %

If the designer selects as relevant the rule P3.1, becoming aware that all Web services related to hotel booking in Rome require as input parameter the Credit Card type, then WDREAM returns the transactional representation of the set of services satisfying it. Note that the rule selected by the designers contains both information on the service category and parameters, and a value for the input parameter `City`, extracted from the log of past invocations (i.e., according to the signature&content pattern).

Then the designer selects, randomly or on the basis of her experience, a service from the list of services associated with the selected rule, e.g., the following transaction that matches the rule P3.1:

```
{ (serviceID=10),(serviceCategory="HotelBooking"),(operationName="bookHotel"),
  (inputName="City"),(inputParameterType="City"),
  (inputName="CreditCard"),(inputParameterType="CreditCardType"),
  (outputName="HotelName"),(outputParameterType="HotelName"),
  (outputName="Address"),(outputParameterType="Address"),
}
```

At this point , the designer issues another query; e.g. *What about services that satisfy ServiceCategory = On Line Payment?*, to add to the application under development a service for Secure Payment.

Independently from the designer's queries, W-DREAM autonomously applies the new request *What about services that satisfy InputParameterType = HotelName OR inputParameterType=Address?*, to find services that can be combined with the previously selected one. As we noticed above, an OR condition corresponds to pose many queries, one for each operand.

The approach used to formulate the automatic query q' is based on the assumption that the same parameter types are used by all the indexed services, or that a set of matching functions are available in order to identify parameters types representing the same type of information. However, this matching problem is out of the scope of this paper.

5 Architecture

Figure 8 shows the W-DREAM functional architecture. This architecture will be integrated in a design framework for service-oriented applications, where designers are in charge of implementing new applications based on a set of already available Web services. The first step of this framework is based on the explorative search enabled by the W-DREAM architecture. The architecture is composed of three modules: the *W-DREAM-Registry Manager*, the *Intensional Knowledge Manager*, and the *Intensional Query Execution Engine*.

W-DREAM-Registry Manager. This module provides the basic primitives to index Web services coming from different repositories and to store information into the *W-DREAM-Registry* (W-DREAM-R). In particular, W-DREAM-R contains: (i) a reference to the transactional representation of Web service repository S; (ii) optionally, a reference to the transactional representation of invocation logs L; (ii) for each stored Web service, the reference to the WSDL document that describes the Web service interface, the Web service category (according to a given categorization scheme) and a human-readable description of the Web service. Figure 9 reports the XML representation of the W-DREAM-Registry structure.

Intensional Knowledge Manager. This module relies on data mining algorithms to manage a summarized, intensional representation of the W-DREAM-R contents and to store it within the *Intensional Knowledge Repository (IKR)* in an efficient way, in order to support the designer's query execution.

Intensional Query Execution Engine. This module provides the methods to query the IKR and supports the user to perform an exploratory search of web services, as explained in Section 4.

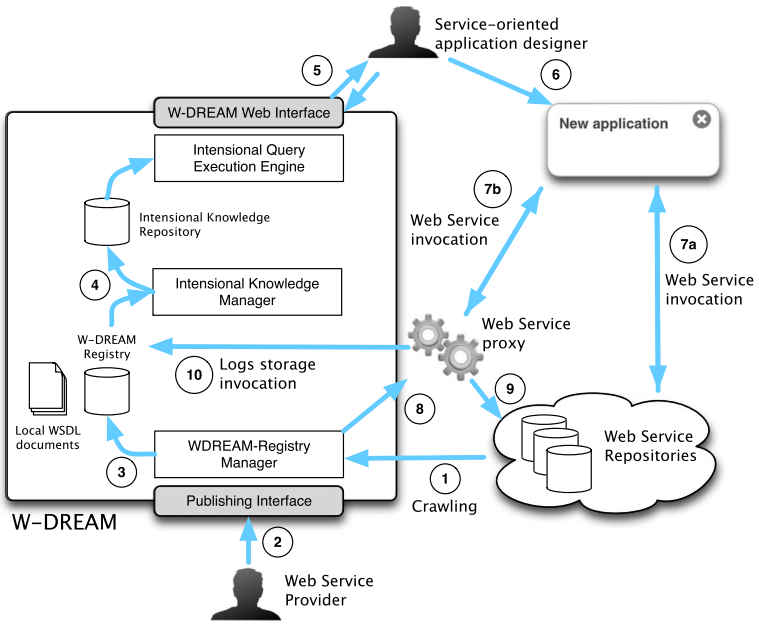


Fig. 8 The W-DREAM functional architecture

```
<W-DREAM_R>
  <transaction_service_dataset ref="localhost/TSD_0XY"/>
  <transaction_logs_dataset ref="localhost/TLD_0XY"/>
  <W-DREAM_Web_services>
    <service name="FindHotel">
      <description>
        ...
      </description>
      <categories>
        <categorizationScheme ref="http://www.unspsc.it/">
          <item>Travel</item>
        </categorizationScheme>
      </categories>
      <interface ref="http://localhost:8080/FindHotel.wsdl"/>
    </service>
    ...
  </W-DREAM_Web_services>
</W-DREAM_R>
```

Fig. 9 W-DREAM-Registry contents

In the following, we provide details about the architecture and interactions between W-DREAM modules. The information about indexed Web services can be collected in two ways. In the first case, the W-DREAM-R Manager is in charge of collecting available Web services by crawling their WSDL documents from the net (step 1). Sources of the crawler are public Web service repositories (e.g., `Seekda`), that are specified within a W-DREAM configuration file. Alternatively, the Web service providers could be directly engaged in the publication of their WSDL documents within the W-DREAM (step 2). The Publishing Interface is quite simple: it enables the provider to select the category and to upload the WSDL documentor to specify the WSDL URI. Within the system, the categorization schemes adopted by the crawled Web service repositories have been pre-loaded (see, for example, Fig. 9). The WSDL documents can be referenced within the W-DREAM Web service specification or locally stored (see, for instance, the `FindHotel` WSDL document on Fig. 9).

After publishing or collecting the WSDL documents, the W-DREAM-Registry Manager is in charge of updating the *transaction service dataset S* (step 3). The Intensional Knowledge Manager works off-line and extracts the intensional answers from the W-DREAM-Registry (step 4). Intensional answers are therefore stored within the Intensional Knowledge Repository. Finally, the Intensional Query Execution Engine is able to serve the queries submitted by the designer (step 5).

Orchestrated invocation of composed Web services from the application will be performed later on based on the WSDL documents (step 7a). However, we recall that our goal consists in allowing designers to easily find Web services of their interest through an explorative search within available repositories and not on composing or invoking them through our system. Hence, the W-DREAM architecture is only in charge of supporting the selection phase, while the other ones will be developed as future work. Figure 8 also shows additional steps when the collection of Web service invocation logs is enabled. These steps are described in the following.

Recording Web service invocation logs As underlined in the previous sections, the exploitation of Web service invocation logs to extract intensional answers may improve the quality of suggestions provided through the summarized representation of Web service repository contents. In fact, it may enable additional kinds of searches that designers can

perform allowing them to focus their attention on a subset of interesting services. Nevertheless, for Web services concerning business activities or subject to privacy issues, the registration of Web service invocation logs should be avoided. Within the W-DREAM architecture, we leave this decision to the Web service providers themselves, who are the owners of service invocation logs: during step 2 (see Fig. 8), the service provider may enable the system to store the invocation logs. Let consider the case in which the provider of a Web service $\mathcal{W}$ enables to store the invocation logs of the service. In this case, a Web service proxy is instantiated by the W-DREAM-Registry Manager (step 8). The proxy is in charge of collecting the SOAP requests (step 7b), forwarding them to the Web service $\mathcal{W}$ (step 9), collecting the $\mathcal{W}$ responses and sending them back to the application. In this way, SOAP messages between the application and the Web service $\mathcal{W}$ are caught by the proxy, that can store invocation logs within the W-DREAM Registry (step 10). To this aim, within the W-DREAM Registry the interface of the Web service $\mathcal{W}$ refers to the proxy, while the WSDL of $\mathcal{W}$ is used to generate the classes of the proxy to interact with the $\mathcal{W}$ implementation (step 9).

6 Experiments

We performed a set of experiments to evaluate: (i) the quality of the extracted patterns and their possible use in a real scenario, (ii) the recall and precision of intensional answers, and (iii) the scalability of the proposed approach.

We used a real Web service repository[4], called OWLS-TC, and a set of synthetically generated datasets. The real public available repository OWLS-TC is composed of 1013 services, associated with their descriptions.

Since the log files are not available for the considered repository, and they are not normally explicitly collected for the other publicly available repositories, the experiments on real data are exclusively focused on signature patterns. The other two types of patterns have been extracted from synthetic datasets, also used to evaluate the scalability of our approach. We implemented a dataset generator that randomly generates a set of services and the relative log files. The generator allows specifying the number of Web services, the maximum number of parameters for each service and the total number of invocations.

All the experiments were performed on a laptop with a 2.2-GHz AMD Turion Dual-Core RM-75 processor and 4.0 Gbytes of main memory, running Kubuntu 10.04.

6.1 Qualitative analysis of the extracted patterns

We extracted signature patterns from the Web service descriptions of the OWLS-TC collection. We performed an initial set of experiments by setting the minimum support threshold to 0.5 %. Table 2 reports some of the most interesting extracted patterns. In the following, we will discuss how they can be used to perform exploratory searches over services and/or give a summary of the main characteristics of classes of services.

Patterns P1-P3 in Table 2 are signature patterns related to books. They highlight which are the common input parameters for the services returning books and can be exploited to refine user's queries. Suppose Alice is in charge of designing an application that allows end

[4] http://projects.semwebcentral.org/projects/owls-tc/

Table 2 A subset of the signature patterns mined from OWLS-TC

PID	Pattern	sup	conf
P1	outputParameterType=Book ⇒ inputParameterType=Author	0.5 %	20.8 %
P2	outputParameterType=Book ⇒ inputParameterType=Title	0.6 %	25.0 %
P3	outputParameterType=Book ⇒ inputParameterType=Publication-Number	0.5 %	20.8 %
P4	ServiceCategory=Communication ⇒ outputParameterType=Film	1.1 %	19.0 %
P5	ServiceCategory=Communication ⇒ outputParameterType=Media	0.6 %	10.3 %
P6	ServiceCategory=Communication ⇒ outputParameterType=Price	0.7 %	12.1 %
P7	ServiceCategory=Communication ⇒ outputParameterType=Quality	3.0 %	51.7 %

users to look for new books to read. Alice knows that, usually, end users do not exactly know which book they are looking for.

Alice queries the W-DREAM by executing a generic query "select services where the output parameter type is *book*" in order to find a service to include in her application. Together with the extensional answer composed of all the services returning a book, she also obtains the SMP composed of patterns P1-P3. Alice can exploit the patterns in SMP to focus her research on a set of services which are more suitable to her application's needs. By exploiting P1-P3, she learns that there are three main types of services returning books, based on the specification of the book author, title and publication number, respectively. Since Alice knows that end users are usually not aware of the exact book they are looking for, the services based on the publication number are not of interest, because end users, probably, do not know that information. End users are probably more interested in services allowing to search for a book by specifying its author or title. Hence, exploiting intensional knowledge, Alice can refine her initial query as "select services where the output parameter type is *book* and the input parameter type is *author* or *title*", obtaining a smaller set of services, more suitable to her needs. Finally, she selects, among the services returned by the refined query, the one to include in her application. She can also decide to select more services, characterized by different parameters, in order to include in her application different book searching forms. Each form, based on a different web service, allows performing a different type of book searching operation.

Patterns can also be useful to summarize the main characteristics of a service category. The set of frequent patterns associated with a specific service category can be used to provide a high view on the available services to designers. Consider the set of patterns P4-P7. They highlight that when the service category is *"Communication"*, then the frequent output types are the following: *Film*, *Media*, *Price* and *Quality*. This summary can be shown to designers and it allows learning which are the types of information (output data) they can obtain by invoking *Communication* services, and hence in which applications they can be used. Similar knowledge has been extracted for all the available service categories.

6.2 Recall and precision analysis

Recall and precision are two commonly used measures for Information Retrieval. Suppose a query engine executes a query q on a dataset D. The set of retrieved data $D_{retrieved}$ (i.e., the set of data returned by the query engine for the given query q) and the set of relevant

data $D_{relevant}$ (i.e., the set of data in D that is relevant with respect to the given query q) are used to define recall and precision.

Recall measures the fraction of relevant data (or services) that is actually returned by the query engine:

$$Recall = \frac{|D_{relevant} \cap D_{retrieved}|}{|D_{relevant}|}$$

Precision measures the fraction of retrieved data (or services) that is relevant:

$$Precision = \frac{|D_{retrieved} \cap D_{relevant}|}{|D_{retrieved}|}$$

Both measures take values in the range [0 %,100 %].

We want to use recall and precision to evaluate the quality of intensional answers. Given a query q, the extensional answer corresponds to the relevant set, while the intensional answer is the retrieved set. However, an intensional answer is a set of patterns (i.e., properties of the data and services of interest) and not data or service descriptions. Hence, a different definition of recall and precision is needed.

Similarly to Mazuran et al. (2012), we define recall and precision, for intensional answer evaluation, as follows.

Recall measures the fraction of relevant data (or services) matched/represented by at least one of the returned patterns:

$$Recall = \frac{|d \in D_{relevant} \text{ matched by at least one pattern } p \in P_{retrieved}|}{|D_{relevant}|}$$

Precision measures the fraction of returned patterns matching/representing at least one relevant data (or service):

$$Precision = \frac{|p \in P_{retrieved} \text{ matching at least one data } d \in D_{relevant}|}{|P_{retrieved}|}$$

where $P_{retrieved}$ is the set of patterns SMP (i.e., the intensional answer) retrieved by querying the frequent pattern set and $D_{relevant}$ the set of data of interest (i.e., the extensional answer).

We performed a set of experiments by considering the four general (high level) queries reported in Table 3. We used a set of queries similar to those we suppose a designer executes when starting to look for Web services tailored to her needs. The experiments were performed on the OWLS-TC, where only signature patterns are available.

For all the queries reported in Table 3 precision is always equal to 100 %, because patterns are extracted from the service dataset, then they are used to answer a query

Table 3 Queries

Query ID	Natural language query	Query as a set of predicates
Q1	Select the services of the travel category	ServiceCategory=travel
Q2	Select the services of the travel category which are characterized by an input parameter of type City	ServiceCategory=travel and inputParameterType=City
Q3	Select the services returning a hotel	outputParameterType=Hotel
Q4	Select the services returning a hotel and allowing to specify a City as input parameter	outputParameterType=Hotel and inputParameterType=City

only if they match the query condition, thus, there are no "false positive" patterns. On the contrary, recall depends on the minimum support threshold enforced during the pattern mining step. Figure 10a, c, e and g report the obtained recall values. Since by lowering the minimum support threshold the number of extracted frequent patterns increases, as expected, also recall increases (a larger set of patterns is potentially returned in response to the user's query). However, one of our goal consists in providing summarized and manageable knowledge to users by means of intensional answers. Hence, we need to pay attention to the cardinality of the intensional answers (i.e., the number of returned patterns). The cardinality of the intensional answer should be lower than the cardinality of the extensional answer (number of retrieved data/services).

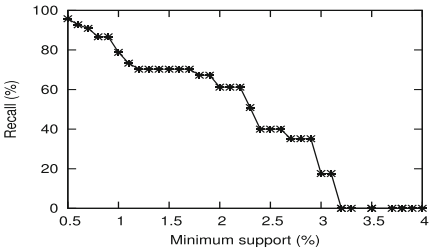
For the four considered queries, the cardinality of the intensional and the extensional answers, when varying the minimum support threshold, are shown in Fig. 10b, d, f and h. The reported charts show that the cardinality of the intensional answer is lower than that of the extensional answer. However, the conciseness of intensional answer is more significant for high support values. The conciseness of intensional answer is indispensable when the extensional answer contains many records. For example, the number of services returned by executing Q1 is 163, while the number of returned patterns ranges from 1 to 41 depending on the enforced threshold (see Fig. 10b). Hence, the intensional answer is more manageable than the extensional one.

6.3 Scalability

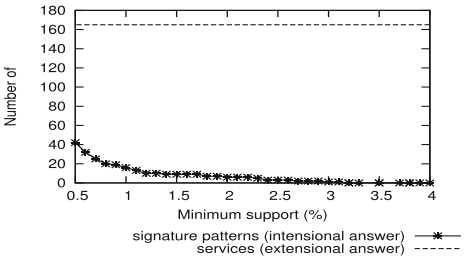
As discussed in the paper, the proposed approach can exploit log files, storing the performed invocations, if available. Since the dimension of the log file can be potentially large for frequently used web services, we decided to analyze the scalability of our approach. In particular, due to the lack of real datasets, we used a set of synthetically generated datasets to evaluate the scalability of our approach with respect to the size of the log file (i.e., number of invocations) and the maximum number of parameters of the Web service operations. From the synthetic datasets, we extracted all the pattern types described in Section 3.

We initially generated a set of datasets by setting the number of services to 1000, the maximum number of parameters per service to 10, and by varying the number of invocations from 100,000 to 1,000,000. We used these datasets to analyze the execution time with respect to the cardinality of the log file (i.e., the number of Web service invocations). We repeated the experiment by setting different minimum support threshold values. Figure 11a reports the obtained results. In particular, in Fig. 11a there is one line for each enforced minimum support threshold value. Since the mining step is based on an association rule mining algorithm, as expected, the execution time increases almost linearly with respect to the cardinality of the log file. The enforced minimum support impacts exclusively on the slope of the curve.

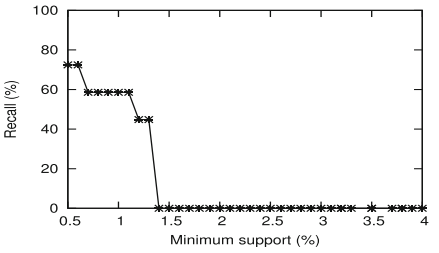
Another dimension of interest is the number of parameters per service. The dataset generator allows setting the maximum number of parameters per service (max_pars). In the generated dataset, each synthetically generated Web service is characterized by a number of parameters in the range [1, max_pars]. We generated a set of datasets by varying the maximum number of parameters from 5 to 15. The cardinality of the generated datasets is set to 1,000,000. The obtained results, reported in Fig. 11b, show that the execution time increases not linearly with respect to the number of parameters. This is related to the number of extracted patterns. In fact, the number of patterns increases not linearly with respect



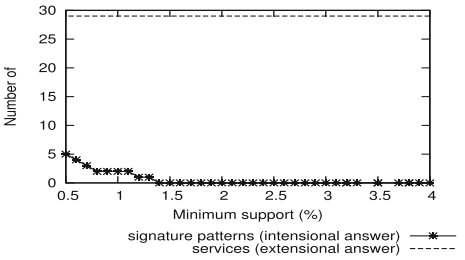
(a) Q1: Recall.



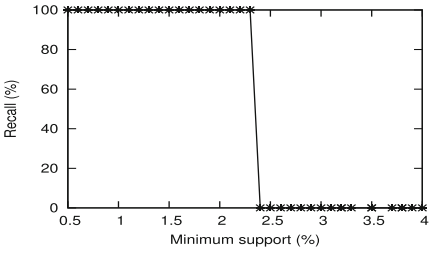
(b) Q1: Cardinality of the result set.



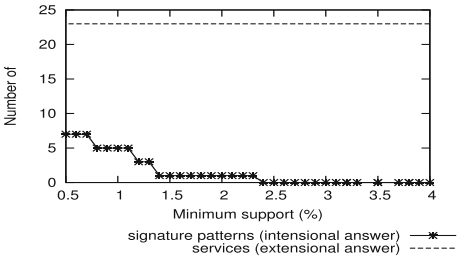
(c) Q2: Recall.



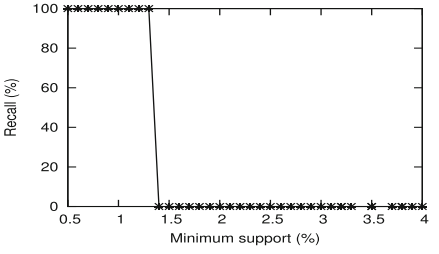
(d) Q2: Cardinality of the result set.



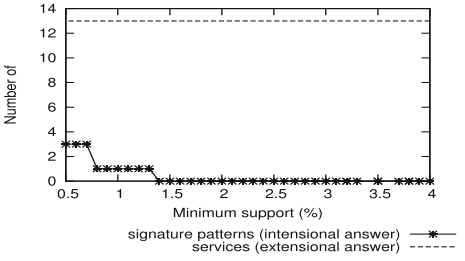
(e) Q3: Recall.



(f) Q3: Cardinality of the result set.



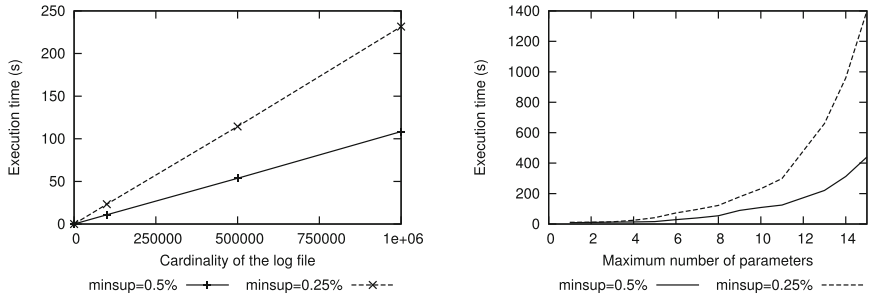
(g) Q4: Recall.



(h) Q4: Cardinality of the result set.

Fig. 10 (Exact matching) Effect of the minimum support threshold on recall and result set cardinality

to the number of parameters. However, the proposed approach scales well also with 15 parameters. Real Web services are usually characterized by few parameters. Hence, we argue that our approach can efficiently mine patterns from real service repositories. For example, for the real Web service repository OWLS-TC, described in the previous section, the average number of parameters per service is 2.7, while the maximum number of



(a) Effect of the cardinality of the log file. Maximum number of parameters = 10.

(b) Effect of the maximum number of parameters. Cardinality of the log file = 1,000,000.

Fig. 11 Execution time of the pattern mining algorithm

parameters is 10. Moreover, only 2 of the 1013 real Web services are characterized by 10 parameters.

7 Related work

Web service discovery and matchmaking Web service discovery and matchmaking systems have been studied to help the designer finding Web services that fit the requirements of the application under development, out of the large amount of available components. Initially web service discovery and search systems were usually based on keyword-based search engines. The proposed approaches analyzed the descriptions of the available Web services, usually in terms of their interfaces, and exploited similarity measures to select the set of services that best match the human proved search-keywords. To overcome the limitations of purely keyword-based matching search engines, semantic Web service discovery techniques have been proposed (e.g., Dong et al. (2004), Chen and Xu (2010), Kiefer and Bernstein (2006), Stern et al. (2007), Plebani and Pernici (2009), Di Sciascio et al. (2007), and Horrocks and Li (2004)). Semantic web service discovery techniques exploit semantically enriched descriptions of Web services to improve precision and recall of traditional systems by means of more precise similarity-based approaches (Dong et al. 2004; Chen and Xu 2010; Kiefer and Bernstein 2006; Stern et al. 2007; Plebani and Pernici 2009) and allow also reasoning on service semantics by means of logical inferencing (Di Sciascio et al. 2007; Horrocks and Li 2004). Differently from the cited works, in our paper we address the Web service search problem from a different prospective. Our approach aims at providing a succinct representation of the available Web services. The extracted (intensional) knowledge can be used to characterize the content of Web service repositories and support an exploratory search approach. This kind of search has been declared as a missing feature in Web service repositories (Sabou and Pan 2005), but not yest properly addressed.

Data mining for Web service discovery The problem of extracting implicit and possibly useful knowledge from data by means of pattern mining algorithms is a well-known problem (Agrawal and Srikant 1994). Algorithms for pattern extraction (e.g., supervised classification (Nayak 2008), clustering (Nayak and Lee 2007)) have been used to mine useful knowledge, not directly deducible by a human being. In Baralis et al. (2007) a lossy

summarization technique for XML data, based on itemsets and association rules, have been proposed. The proposed approach mines the most frequent patterns and uses them for (possibly approximately) answering queries, either when very large XML documents have to be queried (currently unfeasible by means of traditional XQuery engines even when indices are used), or when the actual XML document is not available. The efficiency is achieved by querying instance patterns, which provide a summarized representation of XML documents in the style of materialized views for data warehouses (Harinarayan et al. 1996), rather than traditional indexing techniques. Data mining techniques in the context of Web services have been used in Rouached et al. (2006), where they are applied on execution logs in order to discover and verify Web service properties. Also in Liang et al. (2006) Web service usage patterns are treated through mining techniques at three different levels: user's request, template and instance level. In Nayak and Lee (2007) and Nayak (2008) clustering techniques have been used to group similar Web services in order to improve the service discovery. In these approaches, data mining techniques have not been applied to provide intensional representation of available services for discovery purposes according to an exploratory perspective. To the best of our knowledge, the application of data mining techniques performed within the W-DREAM infrastructure constitutes an innovative perspective, enabling more efficient and effective decision support for the user during the service discovery process.

8 Final remarks

In this paper we proposed W-DREAM, an infrastructure for intensional data and service representation and querying. Within W-DREAM, data mining techniques are applied to infer patterns representing a summarized representation of data and services, that can be directly queried. An experimental evaluation on available and synthetically generated datasets has also been presented, showing the effectiveness and the scalability of the approach. As future work, the adoption of semantic-enriched Web service specifications (e.g., SAWSDL) and the introduction of semantics at the intensional knowledge representation level will be studied to evaluate the positive effects of semantics within the W-DREAM infrastructure. The application of the W-DREAM approach in a real pervasive environment, related to the promotion of artistic and cultural heritage, is currently under development.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large data-bases. In *Proceeding of the 20th International Conference on Very Large Data Bases (VLDB'94)* (pp. 487–499).

Baralis, E., Garza, P., Quintarelli, E., & Tanca, L. (2007). Answering XML queries by means of data summaries. *ACM Transactions on Information Systems*, *25*(3), 1–33.

Chen, L., & Xu, L. (2010). A framework for web service discovery based on ontology similarity. In *SOSE* (p. 197201).

Di Sciascio, E., Di Noia, T., & Donini, F. (2007). Semantic Matchmaking as Non-Monotonic Reasoning: A Description Logic Approach. *Journal of Artificial Intelligence Research*, *29*, 269–307.

Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E., & Zhang, J. (2004). Similarity Search for Web Services. In *Proceeding of the 30th International Conference on Very Large Data Bases (VLDB'04),* (pp. 372-383). Toronto, Canada.

Harinarayan, V., Rajaraman, A., & Ullman, J. (1996). Implementing data cubes efficiently. In *Proceeding of ACM SIGMOD International Conference on Management of Data* (pp. 205–216).

Horrocks, I., & Li, L. (2004). A software framework for matchmaking based on semantic web technology. *Int. Journal of Electronic Commerce (IJEC)*, *8*, 331339.

Hull, R. (2005). Web services composition: A story of models, automata, and logics. In *Proceedings of the IEEE International Conference on Services Computing, pp. xviii–xix. IEEE Computer Society (2005)*. doi:10.1109/SCC.2005.108.

Kiefer, C., & Bernstein, A. (2006). Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. In *Proceeding of 2006 ACM Symposium on Applied Computing* (pp. 1984–1689).

Liang, Q., Chung, J., Miller, S., & Ouyang, Y. (2006). Service Pattern Discovery of Web Service Mining in Web Service Registry-Repository. In *Proceeding of the IEEE International Conference on e-Business Engineering* (pp. 286–293).

Mazuran, M., Quintarelli, E., & Tanca, L. (2012). Data Mining for XML Query-Answering Support. *IEEE Transactions on Knowledge and Data Engineering*, *24*, 1393–1407.

Motro, A. (1994). Intensional answers to database queries. *IEEE Transactions on Knowledge and Data Engineering*, *6*(3), 444–454.

Nayak, R. (2008). Data Mining in Web Service Discovery and Monitoring. *International Journal of Web Services Research*, *5*(1), 63–81.

Nayak, R., & Lee, B. (2007). Web Service Discovery with additional Semantics and Clustering. In *Proceeding of IEEE/WIC/ACM International Conference on Web Intelligence* (pp. 555–558).

Pirotte, A., Roelants, D., & Zimányi, E. (1991). Controlled generation of intensional answers. *IEEE Transactions on Knowledge and Data Engineering*, *3*(2), 221–236.

Plebani, P., & Pernici, B. (2009). URBE: Web service retrieval based on similarity evaluation. *IEEE TKDE*, *21*(11), 1629–1642.

Rao, J., & Su, X. (2005). A Survey of Automated Web Service Composition Methods. In *Proceeding of First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04)* (pp. 43–54).

Rouached, M., Gaaloul, W., van der Aalst, W., Bhiri, S., & Godart, C. (2006). Web Service Mining and Verification of Properties: An Approach Based on Event Calculus. In *OTM Conferences* (pp. 408–425).

Sabou, M., & Pan, J. (2005). Towards Improving Web Service Repositories through Semantic Web Techniques. In *ISWC 2005 Workshop on Semantic Web Enabled Software Engineering (SWESE'05),* (p. 15). Galway, Ireland.

Stern, M., Klein, M., Küster, U., & König-Ries, B. (2007). Diane: An integrated approach to automated service discovery, matchmaking and composition. In *Proceeding of the 16th World Wide Web Conference* (pp. 1033–1042).