# City Research Online

## City St George's, University of London

**Citation:** Spanoudakis, G. & Zisman, A. (2010). Discovering Services during Service-Based System Design Using UML. IEEE Transactions on Software Engineering, 36(3), pp. 371-389. doi: 10.1109/tse.2009.88

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

**Permanent repository link:** https://openaccess.city.ac.uk/id/eprint/5164/

**Link to published version:** https://doi.org/10.1109/tse.2009.88

**Copyright and Reuse:** Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

# Discovering Services during Service-based System Design using UML

G. Spanoudakis and A. Zisman

**Abstract**—Recently, there has been a proliferation of service-based systems, i.e. software systems that are composed of autonomous services, but can also use software code. In order to support the development of these systems, it is necessary to have new methods, processes, and tools. In this paper we describe a UML-based framework to assist with the development of service-based systems. The framework adopts an iterative process in which software services that can provide functional and non-functional characteristics of a system being developed are discovered, and the identified services are used to re-formulate the design models of the system. The framework uses a query language to represent structural, behavioural, and quality characteristics of services to be identified, and a query processor to match the queries against service registries. The matching process is based on distance measurements between the queries and service specfications. A prototype tool has been implemented. The work has been evaluated in terms of recall, precision, and performance measurements.

**Index Terms**— Design notations and documentation, software process models, search discovery language, service discovery engine

## 1 Introduction

Service-based software system engineering has been recognised as an important paradigm for software system development in which different distributed software services are composed to support the rapid and low-cost development of software systems. Services, in this paradigm, are loosely coupled autonomous software entities that can be deployed remotely across organisational and IT infrastructure boundaries. To enable this paradigm, software services need to be described, discovered, composed, and monitored.

To address this challenge, service integrators, developers, and providers have been collaborating over the last few years developing approaches and tools that can support the emerging paradigm. These approaches and tools include: (a) languages to describe services (WSDL[60], WSCL[59], BPEL4WS[6], OWL-S[37], and WSMO[62]), (b) techniques for service discovery (e.g. semantic matchmaking [2][20][21][25][27][32], behavioural signatures matching [63], and matching of full service behavioural models [19] involving requirements [66], architectural [29], and run-time [52] aspects of service oriented systems); (c) techniques for service composition [3][9][10][45][46], and (d) techniques for service monitoring, validation, verification, and evolution [14][15][19][19]. Despite advances in this area, however, existing techniques still fall short of supporting adequately the development and deployment of complex and dependable service-based systems.

In order to overcome this situation, in this paper we describe a framework that we have constructed to support the development of service-based systems; i.e., software systems that are composed of services but may also use additional software code to provide the required functionality. Our framework is UML-based and assists with the design of structural and behavioural models of service-based systems. The framework adopts an iterative system development process, in which software services that can provide the functionality and quality of service (QoS) properties required by a service-based system being developed are identified, and identified services are used to amend and re-formulate the design models of the system. The reformulated design models are used in other iterations of the development process to trigger the identification of services that can be used in the design models.

The framework makes use of a query language to specify the characteristics of the services to be discovered, and a query processor that can execute the queries against service registries. These characteristics can be related to different aspects of the system to be developed and the services that can be deployed in it, and include structural (aka interface) and behavioural models representing expected functionality from the services, and constraints representing additional structural, behavioural, and quality properties that services should satisfy (e.g., the time and cost of executing certain operations, conditions about the provider of a service). Constraints can be hard or soft. Hard constraints must always be satisfied by services whilst soft constraints may be compromised if a service has a good match with other required characteristics but fails to satisfy them.

Queries are executed in a two-stage process. In the first stage, services that satisfy hard constraints in a query are identified generating a set of candidate services. In the second stage, the candidate services of the first stage are matched with the structural and behavioural models of the query as well as its soft constraints, and those services with the best overall match are returned as the final candidate services. The

• *G. Spanoudakis is with the School of Informatics, City University,Northampton Sqaure, London, EC1V 0HB, UK. E-mail: gespan@soi.city.ac.uk.*
• *A. Zisman is with the School of Informatics, City University,Northampton Sqaure, London, EC1V 0HB, UK.. E-mail: a.zisman@soi.city.ac.uk.*

*Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE

matching of services with a query is based on the computation of distances between service descriptions and queries.

The framework assumes services specified by different perspectives, including *structural* (interface) describing operations of services with their data types using WSDL [60], *behavioural* describing behavioural models of services in BPEL4WS [6], *quality* describing non-functional aspects of services in XML-format, and general information of the services. The identification of services based on distinct aspects provide a more accurate match between queries and services and the consequent discovery of services with the required characteristics, as opposed to techniques that are based only on keywords or interface aspects (e.g., WOOGLE [58] and UDDI [54]), which provide less precise match.

The work presented in this paper has been carried out as part of a large European research programme on service centric systems enginering (SeCSE[48]) and has been based on requirements identified in scenarios of service based systems development within different industrial domains including the telecommunication, automotive, and software industries.

The work presented in this paper extends our previous work described in [29][30][68]. The work in [30], presents initial ideas of the framework and describes the discovery process only in terms of structural matching of queries and service operations. In [68], we describe the results of evaluating the framework for structural matching only in terms of precision measures. The work in [29] describes the discovery process in terms of structural and behavioral matching and evaluates only the precision of the structural matching using a small set of services. In contrast, the main contributions in this paper are: (a) development of constraint language to represent extra conditions when executing service discovery during the design of service-based systems, (b) extension of the discovery process to support structural, behavioral, and quality matching, (c) development of the approach as a web service to allow the framework to be used independently of any CASE tool, (d) thorough evaluation of structural, behavioral, and constraint matching in terms of recall, precision, and performance measures, (e) detail description of the framework, and (f) more complete account of related work.

The remainder of this paper is structured as follows. In Section 2 we present an overview of the framework. In Section 3 we describe the service discovery query used in our work. In Section 4 we present the computation of distances. In Section 5 we evaluate the work in terms of recall/precision and performance. In Section 6, we discuss the main features and limitations of the approach undertaken by the framework. In Section 7, we discuss some related work. Finally, in Section 8 we provide some concluding remarks and outline directions for future work.

## 2 Framework Overview

As described in Section 1, our framework adopts an iterative process. In this process, service discovery is driven by structural and behavioural design models of service-based systems (called SySM and SyBM, respectively). The services identified during this process can be used to reformulate the design models and trigger new service discovery iterations.

The behavioral models used in the discovery process describe interactions between operations of a service-based system that can be provided by web services, legacy systems, or software components, while the structural models specify the types of the parameters of operations in the behavioural models[1].

SySM and SyBM are expressed in UML as class and sequence diagrams, respectively. The use of UML as a basis for our approach is because this language: (a) is the de facto standard for designing software systems and can effectively support the design of service-based systems [11][16][31], and (b) has the expressive power to represent the design models of service-based systems since it can represent modelling of software services, legacy code and software components in a system. Furthermore, UML provides built-in extensibility mechanisms (aka UML profiles) that can be used to define an extension of its meta model for specifying service discovery queries and, thus, enable specification of queries in the same language as the system design models.

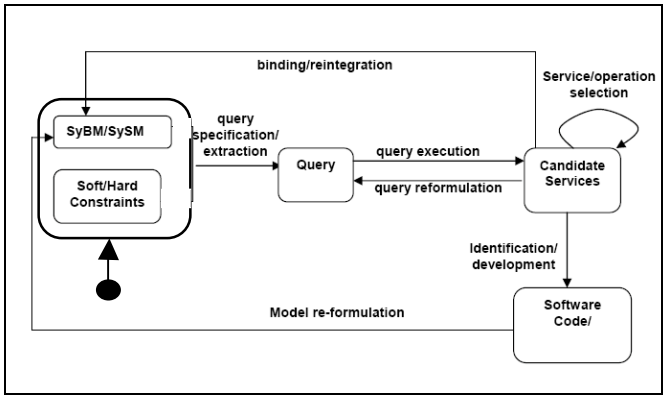


Fig. 1. Overview of framework process

Figure 1 presents an overview of the iterative discovery process of the framework. As shown in the figure, queries are specified in reference to the sequence diagrams in SyBM and the classes and interfaces in SySM, and may include additional constraints about the required services. The candidate services which are identified after the execution of queries can be bound to the SySM and SyBM models by designers. When this happens, SySM and SyBM are reformulated (e.g. by adding message data types and operations of identified services) and their new versions can be used to specify further queries for discovering additional services for other parts of the system. Queries may also be re-formulated and re-executed when the identified services are not adequate. This process can be terminated by the system designer at any time, when all the required services have been discovered, or when it is clear that further queries would not be able to identify services that have a better match with the current design models.

The framework has been implemented as a web service and can be deployed by any client that is able to produce service discovery queries expressed as UML 2.0 models

[1] Examples of such models are given in Section 3.

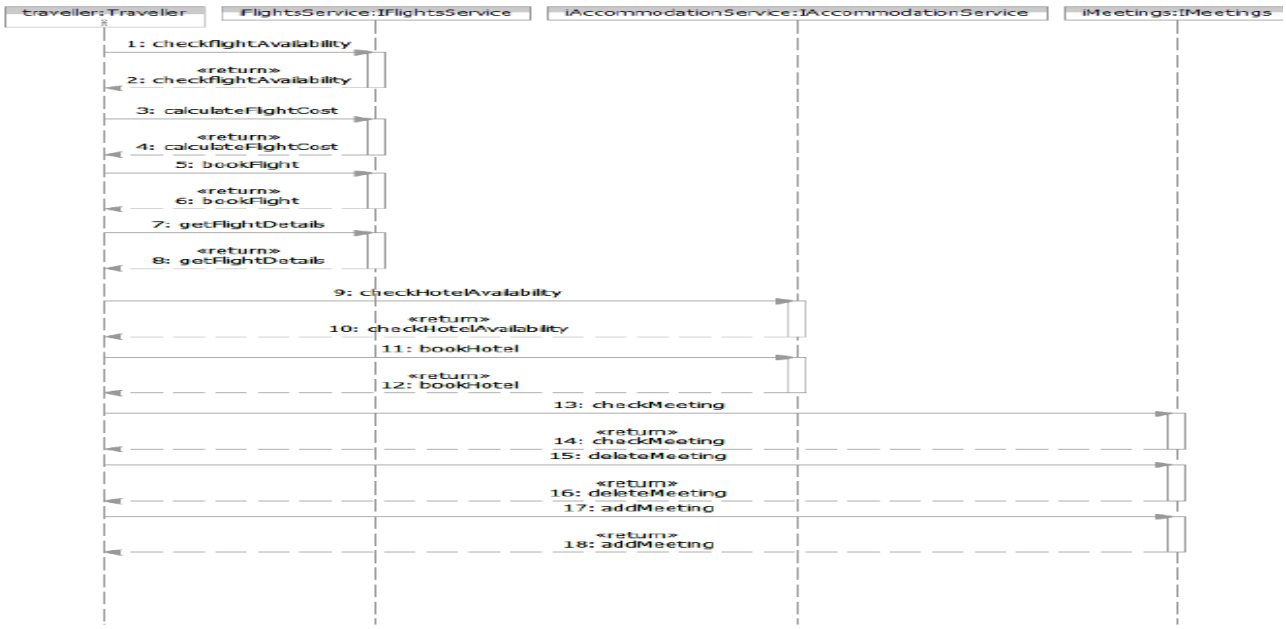


Fig. 2. An interaction of the *ConferenceTravel* system

represented in XMI. Service discovery queries are expressed using appropriate UML sterotypes that we have defined for this purpose. Queries may also include hard and soft constraints, which are expressed in an XML based language that we have developed for this purpose (see Section 3). The deployment of the framework as a web service allows the framework to be used by different types of CASE tools that support UML 2.0 and the representation of such models in XMI.

The execution of a query consists of retrieving different types of service specifications from registries and matching these specifications against the query. The different types of service specifications are called *facets* and include structural, behavioral, and quality specifications. In the current implementation, structural and behavioural descriptions of services are expressed in WSDL[60] and BPEL[6], respectively, and quality descriptions are expressed in XML. The registry used in the current implementation is an eXist database [12]. The framework also provides access to different types of registries through the use of adapters. In this case it is possible to use standard UDDI [54] technologies to store service interface specifications expressed in WSDL together with other service registries to store other types of facets.

## 3 Service Discovery Queries

As discussed earlier, a query may have three different parts, namely (a) structural query model, (b) behavioral query model, and (c) query constraints. The structural and behavioural query models represent functional aspects of the service-based system being developed that need to be fulfilled by the services. The query constraints represent quality aspects (e.g. performance, availability, or cost of service operations) or extra functional aspects (e.g., provider of a service, receiver of a query message) that need to be present in the services. The specification of these parts is discussed in the following.

### 3.1 Structural and Behavioural Query Model

The elements in SySM and SyBM are used to specify queries to indetify services that can be used in the service-based systems. To express a query, system designers must select an interaction from the SyBM model of the system being designed and specify the messages in this interaction that should be realized by service operations that are to be discovered. These messages constitute the so called "query messages" of the query. The specification of an interaction message as a query message is possible by associating the message with the stereotype <<query_message>>. This stereotype is part of a service querying profile that we have defined to enable the specification of queries in UML 2.0 and has been presented in detail in [29].

The service querying profile defines additional stereotypes for different types of UML elements that may exist in a query interaction. These include the stereotypes <<context_message>> and <<bound_message>>. The stereotype <<context_message>> indicates additional structural and behavioural constraints for the query messages. For example, if a context message has a parameter p1 with the same name as a parameter p2 of a query message, then the type of p1 should be taken as the type of p2. The stereotype <<bound_message>> indicates a candidate service operation that is bound to a query message by the designer. All the messages in a query interaction, which are not stereotyped by any of the above stereotypes, are treated as messages irrelevant to the discovery process and, thus, do not restrict the services to be discovered in any way.

As an example of query specification, consider the behavioural and structural design models of a *Conference Travel* support system shown in Figure 2 and Figure 3, respectively. This system allows users to search for and

book flights and hotels as part of trip planning and preparation activities. When a designer wants to find service operations that can provide implementations of the messages *checkHotelAvailability(info:AccommodationInfo, hotel:String):Boolean* and *bookHotel(bookInfo: AccommodationInfo, hotel: String): String* in the sequence diagram of Figure 2, he/she can create a query as a copy of the sequence diagram, and attach the stereotype <<query_message>> to these two messages. The classes representing the data types of the parameters of the two query messages, and all the other classes which are directly or transitively related to them, are automatically identified by the framework and pulled together to formulate the structural model of the query. In the example, these classes are those shown in Figure 3.

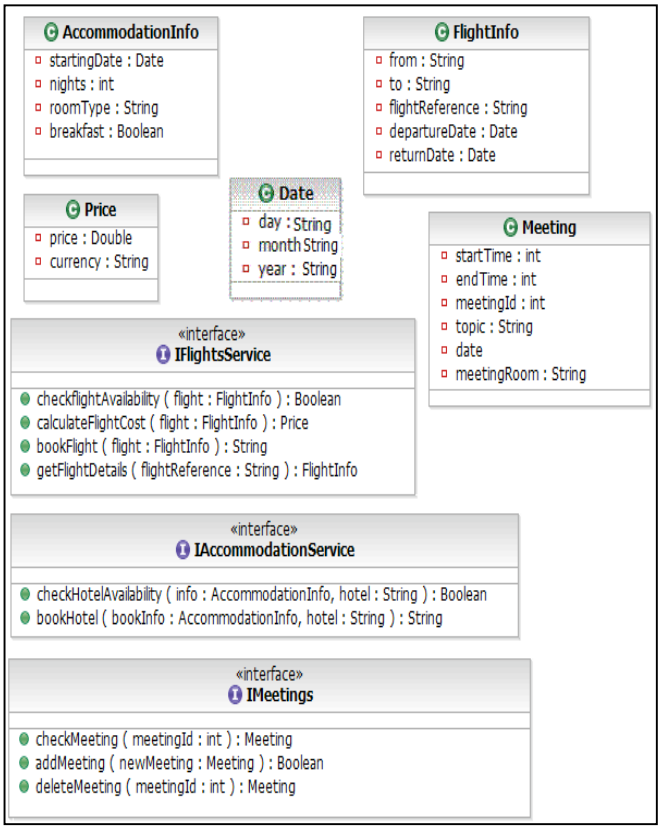


Fig. 3. Structural model of *ConferenceTravel* system

### 3.2 Hard and Soft Contraint Query Language

The specification of hard and soft constraints in queries is based on an XML-based constraint service query language that we have defined as part of the framework and is called *ConstraintSQL*. The specification of the constraints in an XML-based language is motivated by the fact that it is necessary to use XPath [64] expressions to reference elements and attributes in service specifications that are described in XML format.

A soft or hard constraint is defined as a *constraintQuery*. A *constraintQuery* is a logical expression that defines an atomic condition over some element or attribute in a service specification, or a logical combination of atomic conditions that is formed by using logic operators AND and OR. Logical expressions can also be negated.

An atomic condition is defined by a relational expression over the values of two operands (operand1 and operand2). This expression can be specified using the relational operators of the language, namely the operations equalTo, notEqualTo, greaterThan, lessThan, greaterThanEqualTo and lessThanEqualTo. The operands of a relational expression can be of three types, namely *query operands*, *arithmetic expressions* or *constants*.

A query operand identifies an element or attribute in the description of a service (facet) using an XPath [64] expression (see Section 2 for description about facets). An arithmetic expression is used to express a computation over the values of service facet elements and/or attributes and is defined by a sequence of arithmetic operands connected by arithmetic operators (e.g., addition (plus), subtraction (minus), multiplication (multiply), and division (divide)). The operands of an arithmetic expression can be query operands, constants, or functions. A function operand denotes the execution of a complex computation over a series of arguments, which results in a numerical value that can be subsequently used as an operand in the arithmetic expression. A function has a name indicating the function to be executed and a sequence of one or more arguments that might be query operands, constant, or arithmetic expressions, themselves. *ConstraintSQL* offers a set of built-in functions including functions for computing statistics (e.g. mean, standard deviation) and arithmetic functions (e.g. sum, min/max values, power).

```
<?xml version="1.0"?>
<constraintQuery  xmlns="http://tempuri.org/secse/normalQuery"
            name="MaxMeanTimeToComplete"
            weight="0.5" type="soft">
            <logicalExpression> <condition negated="false">
   <lessThanEqualTo>
      <operand1> <queryOperand> <xpathExpression>
        <facet>
            <name>QoS</name> <type>QoS</type> </facet>
            <xpath> //Metrics//Metric[Name =
                "MeanTimeToComplete"]/MinValue </xpath>
            </xpathExpression> </queryOperand> </operand1>
        <operand2> <constant>
           <value>3500</value> <type>STRING</type>
          </constant></operand2>
            </lessThanEqualTo>  </condition> </logicalExpression>
</constraintQuery>
```

Fig. 4. Example of a soft constraint in *CostraintSQL*

A *constraintQuery* has also: (i) a *name* specifying the name of the constraint; (ii) a *type* indicating whether the constraint is hard or soft; and (iii) a *weight* (i.e., a number in the range [0,1]) specifying the significance of the constraint for the service discovery query. The weight in hard constraints is always 1.0, given that a hard constraint needs to be satisfied by all candidate services, while in soft constraints the weight is in the range [0.0, 1.0], since soft constraints are used to rank a service with respect to a query (see Section 4).

Figure 4 presents an example of a soft constraint expressed in *ConstraintSQL*. This constraint specifies that the mean time to execute the operations of a service

should not be more than 3500 milliseconds. As shown in the figure, the constraint is soft and applies to element *//Metrics/Metric[Name = "MeanTimeToComplete"]/MinValue* of facet *QoS* in service descriptions. Furthermore, the weight of the constraint is defined as 0.5.

## 4 Query Execution

Service discovery queries are executed in two phases by a *query processor*. In the first phase, the query processor searches service registries in order to identify services that satisfy the hard constraints of a query. This stage is called *filtering phase* and is based on an exact matching of hard constraints of a query against the service descriptions in the registries. In the second stage, candidate services which have been returned at the filtering stage are matched against structural and behavioral models and soft constraints of a query, and the best candidate services for the query are identified. This stage is called *optimisation phase*

The fit of services with a query is computed during the optimisation stage using three partial distances, namely *signature*, *behavioural*, and *soft constraint* distances. These distances are computed by matching service descriptions with structural model, behavioural model, and soft constraints of a query, respectively. This matching is inexact and even services which do not match exactly with the query may be identified as the best possible candidates.

The structural matching between a query and a service is performed by comparing the signatures of query messages in the structural model against the signatures of the operations of the services. In this case, the WSDL specification of a service and the signatures of the messages in a query are converted into a set of data type graphs (see Subsection 4.1). The matching identifies the mapping and calculates the distances between the elements represented in the graphs.

The behavioural matching between a query and a service is performed by comparing the behavioral specification of the services and the behavioral model of a query. In this case, the behavioural specifications of the service and the behavioural model of the query are converted into state machine models and distances between these state machines are calculated (see Subsection 4.2).

The soft constraint matching between a query and a service is performed by analysing the conditions in the constraint part of a query against service specifications (see Subsection 4.3).

The partial signature, behavioural, and soft constraint distances which are computed between services and a query are aggregated in an overall distance which is then used to select the best service operations for different query messages. The selection of the best service operation for query messages is formulated as an instance of the assignment problem [43], i.e., the problem of 1-1 mapping between query messages and service operations, which minimises a weighted sum of the overall distances between all the mapped service operations and query messages.

There may be some differences in the execution process of a query. These differences are due to the lack of hard, behavioral, and soft constraints in a query, or any combinations of the above constraints. In the case in which there are no hard constraints in a query, the filtering phase is not executed and the partial distances are calculated between all the services in the registries. In the case in which behavioral or soft constraints are not present in a query, the computation of the corresponding partial distance is bypassed and the overall distance is computed by using only the partial distances of the types of constraints specified in a query. Note that structural constraints are always present in a query and, therefore, at least distances based on them are calculated.

In the following, we discuss the computation of signature, behavioural and soft constraints distances and the selection of the best candidate service operations for a query. We also give an example of computing these distances.

### 4.1 Signature Distance

The signature distance between a service operation $S_o$ and a query message $Q_m$ is computed by a function that considers the linguistic distance between the names of the operation and query message, the names of their parameters, and the data types of these parameters as defined below.

**Definition 1:** The signature distance between a service operation $S_o$ and a query message $Q_m$ is computed by function:

$$d_{f\text{-}sig}(Q_m,S_o) = w_N * d_L(name(Q_m), name(S_o)) + w_I * d_{PS}(in(Q_m),in(S_o)) + w_O * d_{PS}(out(Q_m),out(S_o))$$

where, $d_L$ is a linguistic distance function; $d_{PS}$ is a function that computes the distance between input and output parameters of $S_o$ and $Q_m$, and $w_N$, $w_I$, $w_O$ are weights associated with the names, input parameters, and output parameters of the service operation and query message, respectively (with $w_N+w_I+w_O=1$).

The definitions of the $d_L$ and $d_{PS}$ fuctions are given below. Note that $d_{f\text{-}sig} \in [0,1]$ since this function is defined as a linear combination of $d_L$ and $d_{PS}$ which also return values in [0,1] as we discuss below.

**Definition 2:** The linguistic distance between two strings S1 and S2 is computed as:

$$d_L(S1,S2) = |t(S1/S2)| + |t(S2/S1)| + 0.5 * |t(S2) \cap_s t(S1)| \; / \; |t(S2) \cup t(S1)|$$

where,

- t(S1) and t(S2) are sets of tokens in S1 and S2. The tokens in a string S are identified by splitting S into successive substrings starting at the beginning of S, or at a capital letter within S, and ending before the next capital letter,
- t(Si/Sj) is the set of tokens x in t(Si) for which there is no token y in t(Sj) that is a synonym of x (the synonymy of two tokens is determined on the basis of WordNet lexicon [35]),
- $t(S2) \cap_s t(S1)$ is the set of the tokens in S1 and S2 which have synonym or identical tokens in the other set,
- $|\Phi|$ is the cardinality of set $\Phi$.

According to the above definition, the linguistic distance between two strings S1 and S2 is computed by tokenising S1 and S2 into two sets of tokens t(S1) and t(S2) and taking the ratio of the sum of the number of tokens in each of these two sets which have no synonym in the other set and the number of

tokens which have synonyms weighted by 0.5 over the total number of distinct tokens in t(S1) and t(S2) (i.e., the cardinality of the set t(S2) ∪ t(S1)).

Note that $d_L \leq 1$ since $t(S1/S2) \cup t(S1/S2) \cup (t(S2) \cap_s t(S1)) \subseteq t(S2) \cup t(S1)$ and $t(S1/S2) \cup t(S1/S2) \cup (t(S2) \cap_s t(S1))$ and, therefore, $|t(S1/S2)| + |t(S2/S1)| + 0.5*|t(S2) \cap_s t(S1)| \leq |t(S2) \cup t(S1)|$. Also, $0 \leq d_L$ since $t(S1) \neq \emptyset$ and $t(S2) \neq \emptyset$ (every string will have at least one token) and, therefore, $|t(S2) \cup t(S1)| > 0$.

$d_{PS}$ is computed by finding the best possible morphism between the data types of the parameters of a service operation $S_o$ and a query message $Q_m$. To compute this morphism, the query processor first formulates graphs that represent the data types of the input and output parameters of $S_o$ and $Q_m$ and then matches the input graphs and output graphs with each other.

The data type graphs of the input and output parameters of a service operation and a query message are constructed taking into consideration both primitive and non-primitive data types. In the graph of a set P of parameters, a special node representing the root of the graph is created with immediate children nodes, for each parameter pi in set P. The data type associated with parameter pi is added to the graph as a child node of the respective root node (datatype_$p_i$ node). The name of the parameter pi is represented in the graph as the name of the edge between the root node and datatype_$p_i$ node. In the case of a data type that is a non-primitive type, a subgraph for this data type is constructed such that each data type of the attributes in the class representing datatype_$p_i$ is added to the graph as a child of datatype_$p_i$ with the name of the attribute as the name of the respective edge. If the data type of an attribute is also non-primitive the process is repeated for this data type. The process terminates when all the leaf nodes in the graph have only primitive data types.

More specifically, the graph that represents the data types of a set S of parameters $p_1,\ldots,p_m$ with types $T_1, \ldots, T_m$ is a labeled directed graph that includes the following set of edges:

$$Edges(S) = \cup_{i=1,\ldots,m} <p_i, (sn,T_i)> \cup_{i=1,\ldots,m} Edges'(T_i)$$

Edges(S) containts: (i) the edges $<p_i, (sn,T_i)>$ that represent the parameters in S, and (ii) sets of $Edges(T_i)$ which represent the structure of the types $T_i$ of the different parameters $p_i$. In an edge $<p_i, (sn,T_i)>$, $p_i$ is the name of the parameter that labels the edge, *sn* is a special root node that does not represent any specific parameter type and $T_i$ is the type of the relevant parameter. If the type of a parameter is primitive $Edges'(T_i)$ is empty (i.e., $Edges'(T_i)=\emptyset$). Otherwise, for non primitive data types, assuming that $Sfeatures(T_i)$ is the set of structural features of $T_i$, (i.e., the set of attributes and associations of $T_i$) and each feature x in $Sfeatures(T_i)$ relates $T_i$ with another type Type(x), $Edges'(T_i)$ is defined as $Edges'(T_i) = \cup_{x \in Sfeatures(Ti)} \{<name(x), (T_i, Type(x))>\} \cup_{x \in Sfeatures(Ti)} \{Edges'(Type(x))\}$. Thus, the set of nodes which are inteconnected by Edges(S) represent $T_1, \ldots, T_m$ and all the types in their own structures, and are labelled by the names of these types. Also, the edges in Edges(S) represent the structural relations (attributes and associations) between $T_1, \ldots, T_m$ and all the types in their structures, and are labelled by the name of the relevant relation. An example of graphs of input parameter data types is shown in Figure 5 for a query message *checkHotelAvailability(info:AccommodationInfo,hotel:String):Boolean* from the *ConferenceTravel* system described in Section 3, and a service operation *checkRoomAvaliability(room:Room, Starts:Date,Ends:Date):Boolean*, where *Room* is a non-primitive data type with attributes *hotelName:String* and *category:String*.

**Definition 3:** The distance between two sets of parameters P1 and P2 is computed as:

$$d_{PS}(P1,P2) = MIN_{m \in Morphisms(Edges(P1), Edges(P2))} \{(\sum_{(e1,e2) \in m} d_E(e_1,e_2) + abs(|Edges(P2)| - |Edges(P1)|)) / max(|Edges(P2)|, |Edges(P1)|)\}$$

where,

- Morphisms(Edges(P1), Edges(P2)) is the set of all the total morphisms from the edges in Edges(P1) to the edges in Edges(P2)
- abs(exp) is the absolute value of the arithmetic expression,
- $d_E(e_1,e_2)$ is the distance between two edges e1=< e1-name, $(T_1^S, T_1^D)$> and e2=< e2-name, $(T_2^S, T_2^D)$> defined as
$$d_E(e_1,e_2) = w_1.d_L(\text{e1-name, e2-name}) + w_2.d_L(name(T_1^S), name(T_2^S)) + w_3.d_L(name(T_1^D), name(T_2^D)) + w_4.d_{EDGES}(Edges(T_1^D), Edges(T_2^D)))$$
with $T_i^S$ and $T_i^D$ are the source and destination data type nodes respectively; and $w_1$, $w_2$, $w_3$ and $w_4$ weights associated with the linguistic distances of the names of the edges, the names of the data types, and the distances of two edges.
- $d_{EDGES}(Edges(T_1^D), Edges(T_2^D))$ is the distance between the edges of two types, defined as
$$d_{EDGES}(Edges(T_1^D), Edges(T_2^D)) = 1 \quad \text{if } Edges(T_1^D)=\emptyset \text{ or } Edges(T_2^D)=\emptyset$$
$$d_{EDGES}(Edges(T_1^D), Edges(T_2^D)) = MIN_{m \in Morphisms(Edges(T1D), Edges(T2D))} \{(\sum_{(e1,e2) \in m} d_E(e_1,e_2) + abs(|Edges(T_2^D)| - |Edges(T_1^D)|)) / max(|Edges(T_1^D)|, |Edges(T_2^D)|)\} \quad \text{Otherwise}$$

According to Definition 3, the morphism of the structure of the data types of the parameters is determined by finding the matching between the edges of the data type graphs of two parameters that has the least possible sum of edge distances (see formula $(\sum_{(e1,e2) \in m} d_E(e_1,e_2) + |Edges(P2)| - |Edges(P1)|) / |Edges(P2)|$). The distance between two edges is computed by taking into account the linguistic distance between the names of the structural features represented by the edges, the names of the types that have these features ($T_1^S$ and $T_2^S$), and the similarity of the structures of the types that the features point to (see function $d_{EDGES}(Edges(T_1^D), Edges(T_2^D))$). Thus, the computation of $d_E$ analyses recursively the entire structure of the graphs that represent the data types of two parameters.

Note that $0 \leq d_{PS} \leq 1$. This is because $Edges(P1) \neq \emptyset$ and $Edges(P2) \neq \emptyset$ and, thus, $max(|Edges(P2)|, |Edges(P1)|) \geq 0$. Also, for any morphism *m* in $Morphisms(Edges(P1),Edges(P2))$, we have that $|m| = min(|Edges(P2)|, |Edges(P1)|)$. Thus, for all *m* in $Morphisms(Edges(P1),Edges(P2))$ it will be that $\sum_{(e1,e2) \in m} d_E(e_1,e_2)) \leq |m|$ and $\sum_{(e1,e2) \in m} d_E(e_1,e_2) + abs(|Edges(P2)| - |Edges(P1)|)) \leq max(|Edges(P2)|, |Edges(P1)|)$ if $0 \leq d_E(e_1,e_2) \leq 1$. However, $d_E(e_1,e_2)$ is computed recursively as a linear combination of $d_L$, which as discussed earlier takes

values in [0,1], and $d_{EDGES}$. At the end of the recursion, however, $d_{EDGES}$ will be applied on primitive types with no further edges and therefore it will be equal to 1. Hence, in the preceeding computation it will be that $d_E(e_1,e_2) \leq 1$. Similarly it can be shown that for all previous computations of $d_E$ in the recursion it will be $d_E \leq 1$. An example of signature distance is described in Section 4.4.

### 4.2 Behavioural Distance

The behavioural distance between a service operation $S_o$ and a query message $Q_m$ is computed by matching the state machine representing the behaviour expected by interface I that executes $Q_m$ in the query ($SM_Q$) and the state machine representing the behaviour of the service S that provides $S_o$ ($SM_S$) (see Definition 4). The state machines $SM_Q$ and $SM_S$ are generated automatically from the interaction diagram of a query Q and the BPEL specification of a service S in the registry, respectively. The algorithms used to generate the state machines can be found in [29].

**Definition 4:** The behavioural distance between $S_o$ and $Q_m$ is computed as:

$$d_{f\text{-}beh}(S_o, Q_m; k) = d_{beh}(SM_Q, SM_S; k) =$$

$$MIN_{n=0\ldots k}$$

$$\{MIN_{m \in Morphs^{(n)}(SM_Q, SM_S)}$$

$$\{1/(MAX(len(SM_Q),len(SM_S))$$

$$(\sum_{ti \in p \text{ and } m(ti) \neq NULL} d_{f\text{-}sig}(oper(t_i),oper(m(t_i))) +$$

$$(\sum_{ti \in transitions(SM_Q) \text{ and } m(ti) = NULL} 1) +$$

$$(\sum_{tj \in q \text{ and } m^{-1}(tj) = NULL} 1))\}\}\} \quad \text{if } SM_Q.Transitions \neq \emptyset$$

$$\text{and } SM_S.Transitions \neq \emptyset$$

$$d_{f\text{-}Beh}(S_o, Q_m; k) = 1 \quad \text{Otherwise}$$

where,

- $SM_Q$ and $SM_S$ are state machines represented as $SM = <\sigma, O, T, \sigma_I>$ where $\sigma$ is the set of states of SM, O is the set of signatures of the operations provided by SM, T is a set of transitions of SM which are labelled by an operation signature in O, and $\sigma_I$ is the initial state of S ($\sigma_I \in \sigma$),
- $len(SM_Q)$ and $len(SM_S)$ are maximum length of a path in $SM_Q$ and a path in $SM_S$, respectively;
- $Morphs^{(n)}(SM_Q,SM_S)$ is the set of all the possible 1-1 mappings between the transitions of two paths p and q in $SM_Q$ and $SM_S$ that preserve the ordering of the transitions within these paths (i.e. for all transitions $t_i$ and $t_j$ in p such that $t_i \prec_p t_j$ it also holds that $m(t_i) \prec_q m(t_j)$[2]) and leave n transitions in p or q without counterparts (the counterpart of all such transitions will, by convention, be a dummy transition NULL);
- $m^{-1}$ is the inverse mapping of a mapping m from p to q;
- k (flexibility matching) is a parameter defining the maximum number of the transitions of p or q that are allowed not to have a counterpart in the mappings between these paths, $0 \leq k \leq ABS(len(SM_Q) - len(SM_S))$;
- oper(t) is the operation signature that labels a transition t in a state machine;
- $d_{f\text{-}sig}$ is the distance between signatures of two operations.

The algorithm that computes $d_{beh}(SM_Q, SM_S; k)$ is a search algorithm that finds the path q in $SM_S$ which has the best possible match with the single path p of $SM_Q$ , and returns the aggregate distance between these paths as the distance between $SM_Q$ and $SM_S$. In this search, the degree of match between two paths p and q is computed as the sum of the signature distances between the operations which label the mapped transitions of p and q ($d_{f\text{-}sig}(oper(t_i),oper(m(t_i)))$).

The search of a path q in $SM_S$ that has the best possible match with path p in $SM_Q$ is implemented by trying to construct alternative mappings from p onto q ($Morphs^{(n)}(p,q)$) Incrementally. These alternative mappings must preserve the order of the transitions in the two paths (i.e., for all transitions $t_i$ and $t_j$ in p such that $t_i \prec_p t_j$ it should also hold that $m(t_i) \prec_q m(t_j)$). Furthermore, valid mappings are allowed to leave up to k transitions of p and q without a counterpart.

More specifically, the construction of alternative mappings from p onto q is executed by consuming one by one all the transitions $t_p$ in p, comparing these transitions with the transitions $t_q$ in q, preserving the order of $t_p$ and $t_q$ in the paths, and verifying if transition $t_p$ can be (i) accepted, when $t_p$ matches $t_q$; (ii) removed, if $t_p$ does not match $t_q$, but a transition $t_{p+x}$ following $t_p$ in p matches transition $t_q$ (where $1 \leq x \leq k\text{-}L$ and L is the number of transitions that have already been removed or added during the transformation process), or (iii) $t_q$, $t_{q+1}$, $t_{q+x}$ can be added when $t_p$, $t_{q+x}$ and the other transitions in p can be consumed. The mapping that minimizes the distance $d_{f\text{-}beh}(SM_Q, SM_S; k)$ is the one selected. It should be noted that whilst matching the state machine of a query with the state machine of a service any conditions of the later are ignored. This is because it is not possible to establish the equivalence of such conditions without making strong assumptions about naming of internal service variables.

The flexibility of the approach to allow up to k transitions of p to be left without counterparts makes it possible to discover services whose behaviour is similar to the behaviour of the required service, but not identical. It should be noted, however, that while unmapped transitions of p and q are allowed in a mapping m, such transitions contribute a distance of 1 to the aggregate distance of m. Thus, the more the transitions that a mapping leaves without counterparts, the less likely is for m to present the best possible match for p.

Note that, $d_{f\text{-}beh}$ takes always values in range [0,1]. This is a direct implication of its definition when if $SM_Q.Transitions=\emptyset$ or $SM_S.Transitions=\emptyset$. In cases where $SM_Q.Transitions\neq\emptyset$ and $SM_S.Transitions\neq\emptyset$ $d_{f\text{-}beh} \geq 0$ since $MAX(len(SM_Q),len(SM_S) > 1$. We also have that $d_{f\text{-}beh} \leq 1$, since $|Morphs^{(n)}(SM_Q,SM_S)| \leq MAX(len(SM_Q),len(SM_S)$ and $d_{f\text{-}sig}(oper(t_i),oper(m(t_i)) \leq 1$.

An example of the behavioural distance is described in Subsection 4.4.

### 4.3 Soft Constraint Distance

The soft constraint distance between a query message $Q_m$ and a service operation $S_o$ is computed by the function in Definition 5 below:

**Definition 5:** The soft constraint distance between $S_o$ and $Q_m$ is computed as:

$$d_{f_con}(Q_m,S_o) = \sum_{Ci \in Scons(Qm)} w_i d_{con}(C_i,S_o) / \sum w_i \quad \text{if } Scons(Q_m) \neq \emptyset$$

[2] $\prec_p$ is a relation that reflects the linear order of transitions within a path *p*.

$d_{f_con}(Q_m, S_o) = 0$ if $Scons(Q_m) = \emptyset$

where,

- $Scons(Q_m)$ is the set of soft constraints defined for query message $Q_m$ in the query and the global soft constraints of the query which apply to $Q_m$ by default,
- $w_i$ is a weight expressing the significance of the constraint $C_i$ for query message $Q_m$ ($w_i > 0$),
- $d_{con}(C_i,S_o)$ is a distance measure that represents if the constraint $C_i$ is satisfied by service operation $S_o$. This measure is 0 if the constraint $C_i$ is satisfied by $S_o$, and 1 otherwise.

According to this definition, the soft constraint distance between $Q_m$ and $S_o$ is the sum of the weights of the soft constraints that apply to $Q_m$ and are not satisfied by $S_o$ , divided by the sum of the weights of all the soft constraints in the query which apply to $Q_m$. It should also be noted that $d_{f_con}(Q_m,S_o)$ takes always values in the range [0,1] since $d_{con}(C_i,S_o) \in [0,1]$ and, therefore, $\sum_{Ci \in Scons(Qm)} w_i d_{con}(C_i,S_o) \leq \sum w_i$ and $\sum w_i > 0$.

### 4.4 Overall Distance

The overall distance between a query message $Q_m$ and a service operation $S_o$ is computed as the weighted sum of the signature, behavioural, and soft constraint distances between $Q_m$ and $S_o$ as defined below.

**Definition 6:** The overall distance between $Q_m$ and $S_o$ is computed as:

$$D(Q_m, S_o; k) = w_{sig}\, d_{f\text{-}sig}(Q_m, S_o) + w_{beh}\, d_{f\text{-}beh}(Q_m, S_o; k) + w_{con}\, d_{f\text{-}con}(Q_m, S_o)$$

where $w_{sig}$, $w_{beh}$ and $w_{con}$ are weights of signature, behavioural and constraint distances for which $w_{sig} + w_{beh} + w_{con} = 1$.

It should be noted that $D(Q_m, S_o; k)$ returns always a value in the range [0,1] since $d_{f\text{-}sig}$, $d_{f\text{-}beh}$ and $d_{f\text{-}con}$ also return values in the same range, as discussed earlier.

Following the computation of the D distances between all the query messages in a query ($Q_m$) and the set of service operations in registries ($S_o$), the best service operation for each query message is determined by finding the morhism (1-1 mapping) between query messages and service operations which minimises the function

$$MIN_{M \in Morphisms(QM,SO)} \{\sum_{(Qm,So) \in M} D(Q_m, S_o; k)\}$$

for a given value of flexibility matching K (Morphisms($Q_m$,$S_o$) in the above formula denotes the set of all the possible morphisms from $Q_m$ to $S_o$).

More specifically, the computation of the best service operation for each query message is executed by constructing an *operation matching graph* G with (a) two disjoint sets of vertices: one set of vertices representing messages in a query and another set of vertices representing the service operations identified in the filtering stage (or the service operations in the registries when the filtering stage has not been executed); and (b) edges that connect each of the messages in the query with all the operations of the retrieved services, and vice versa. Each edge *e(m,o)* in graph G is weighted by a measure that indicates the overall distance $D(Q_m, S_o; k)$ between a message m in $Q_m$ and an operations o in $S_o$.

Following the computation of the distances between the vertices, the matching between the messages in the query and the operations in the candidate services is detected by selecting a subset E′ of the edges in graph G, such that E′ is a total morphing between the vertices in G, and has the minimal distance values. This subset is selected by applying an instance of the assignment problem algorithm following the approach in [51][3].

### 4.5 Example

As an example of computing the different types of distances between query messages and service operations consider again the query for *ConferenceTravel* system that we introduced in Section 3. This query has two query messages, namely *checkHotelAvailability(info:AccommodationInfo, hotel:String):Boolean* (QM1) and *bookHotel(bookInfo: AccommodationInfo, hotel: String): String* (QM2), and the soft constraint as described in Figure 4. Suppose also that the query is matched with a service, called *HotelService1*, which offers the following 3 operations:

- SO1:HotelService1::*checkRoomAvailability(room:Room,Starts:Date,Ends:Date): Boolean*
- SO2: HotelService1::*reserveRoom(room: Room, From:Date, To:Date): Reservation*
- SO3: HotelService1::*cancelReservation(reservation: Reservation): Boolean*

Assume that *HotelService1* has a QoS facet indicating that the time to execute the operations in the service is 3500 miliseconds (this facet is not shown in the paper due to space limitations).

As an example of the computation of signature distance, consider the signature distance between query message QM1 and the service operation SO1. According to Definition 1, this distance is computed from the linguistic distance $d_L$*(checkHotelAvailability, checkRoomAvailability)* between the names of QM1 and SO1, the distance between the input parameters of QM1 and SO1 $d_{PS}$(in(QM1),in(SO1)), and the distance between the output parameters of QM1 and SO1 $d_{PS}$(out(QM1),out(SO1)). The linguistic distance between QM1 and SO1 is equal to 0.5 as the names of QM1 and SO1 have two identical substrings (i.e., "check" and "availability") and two non identical substrings with no synonyms (i.e., "hotel" and "room"). Hence, $d_L$*(checkHotelAvailability, checkRoomAvailability) = 2/4 = 0.5.*

The computation of $d_{PS}$(in(QM1),in(SO1)) and $d_{PS}$(out(QM1),out(SO1)) is based on the graphs of the data types of QM1 and SO1. Figure 5 shows the graphs of the data types of the input parameters of QM1 and SO1. Based on these graphs, the distance $d_{PS}$(in(QM1),in(SO1)) is 0.415. According to Definition 3, this distance is computed from $d_E$ distances between the pairs of edges that have been mapped by the best possible morphism between the input parameters of QM1 and SO1 which is shown by the dashed lines in Figure 5[4]. The $d_E$ distances between these edges are shown in the grey boxes which appear upon the dashed lines in Figure 5 (each of

[3] When the number of messages in a query is not the same to the number of operations in the candidate services, special vertices are added in the graph representing dummy operations, in order to make the number even.

[4] 0.415=((0.437+0.437+1+0.437+0.437+0+0+0)+1)/9.

these boxes shows the distances ($d_E$(e1, e2), $d_L$(name($T_1^S$),name($T_2^S$)), $d_L$(e1-name,e2-name), $d_L$(name($T_1^D$), name($T_2^D$)), $d_{EDGES}$(Edges($T_1^D$), Edges($T_2^D$)))) for the relevant pair of mapped edges).

As shown in Figure 5, the best morphism between the input parameters of QM1 and SO1 maps the edges day, month and year of data types Date in the two graphs. This is because the linguistic distance between the names of these edges, the names of their starting nodes (Date), the names of their destination nodes (String), and the distance between the edges of their destination nodes are all equal to 0. Note also, that the edge which represents the attribute StartingDate of data type *AccommodationInfo* in QM1 is mapped on the input parameter *Starts* of SO1. These two edges have an overall distance 0.437 which is the minimum possible edge distance excluding the mappings discussed above. For the same reason, the attribute *roomType* of the data type *AccommodationInfo* in QM1 is mapped onto the attribute category of data type *Room* in SO1.

The computation of $d_{PS}$(out(QM1),out(SO1)) is performed in a similar way to the computation of $d_{PS}$(in(QM1),in(SO1)). In the example, $d_{PS}$(out(QM1),out(SO1)) is zero, since both QM1 and SO1 have returned parameters of type *Boolean*.

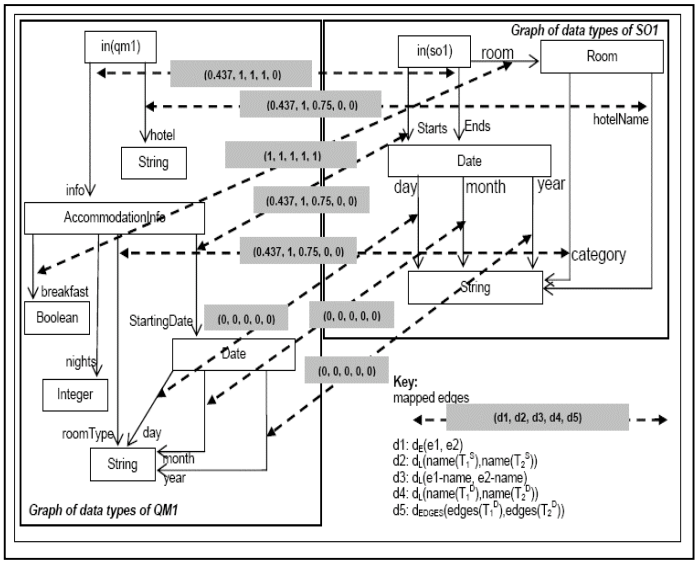


Fig. 5. Graphs for data types of input parameters of QM1 and SO1

Consider $w_N$ = 0.4, $w_I$ = 0.4, and $w_O$= 0.2 the weights associated with the linguistic distance and the input and output parameter distances, respectively. The signature distance for QM1 and SO1 is $d_{f\text{-}sig}$(QM1,SO1) = 0.4*0.5 + 0.4* 0.415 + 0.2 * 0 = 0.366.

The behavioural distance between QM1 and SO1 is 0.209 for K=0. This distance results from the computation of the best possible mapping between the state machine of QM1 and the state machine of SO1. This mapping is shown by the dashed lines in Figure 6 where (i) transition *checkHotelAvailability()* in the state machine of QM1 is mapped onto transition *checkRoomAvailability()* from state S1 to state S1 in the state machine of SO1, and (ii) transition *BookHotel()* in the state machine of QM1 is mapped onto transition *reserveRoom()* from state S1 to state S2 in the state machine of SO1. This particular mapping is selected because (a) it consists of the minimum sum of distances between possible pairs of transitions (the distances between the former and the latter pair of mapped transitions were 0.174 and 0.281, respectively), and (b) the value of the parameter K=0 eliminates other alternative mappings of transitions with the same pairwise distances.

One of these alternatives, for example, is a mapping in which transition *BookHotel()* in the state machine of QM1 is mapped onto transition *reserveRoom()* from state S1 to state S2 in the state machine of SO1 (as in the selected mapping) , but transition *checkHotelAvailability()* in the state machine of QM1 is mapped onto transition *checkRoomAvailability()* from the initial state to state S1 in the state machine of SO1. Although, the pairwise distance between the latter pair of transitions in the alternative mapping is the same as the distance between transition *checkHotelAvailability()* in QM1 and transition *checkRoomAvailability()* from state S1 to state S1 in the selected mapping, the alternative mapping was eliminated since this mapping requires one unmapped transition between two mapped transitions in the state machine of SO1 (i.e., transition *checkRoomAvailability()* from state S1 to state S1) and, as the value of K is zero, no such unmapped transitions are allowed.

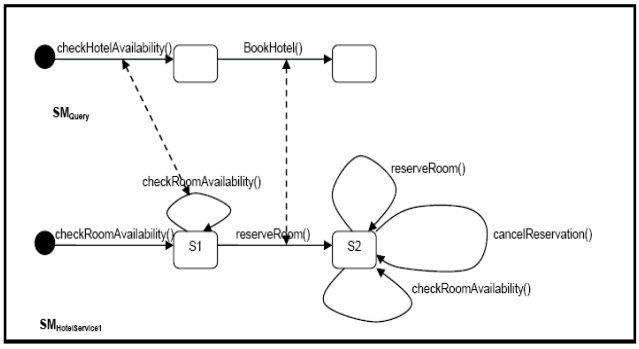


Fig. 6. *ConferenceTravel* and *HotelService1* statemachines

The soft constraint distance between QM1 and SO1 is zero since the constraint specified in Figure 4 matches the performance time describe in the QoS facet of the service.

Suppose in the example that the signature distance has a weight of 0.5, the behavioural distance has a weight of 0.3, and that constraint distance has a weight of 0.2. In this case, the overall distance between QM1 and SO1 is D(QM1,SO1,0) = 0.5*0.366 + 0.3*0.209 + 0.2*0 = 0.2457.

## 5 Evaluation

To evaluate our framework, we have performed a set of experiments, designed to measure and analyse: (a) the recall and precision of the results of service discovery, and (b) the performance of the matching process.

### 5.1 Experimental SetUp

In the experiments we used a registry of services that had been built collectively by the industrial partners of the SeCSE project [48]. The registry included 95 different services offering a total of 316 service operations of different complexities. These services had different service providers and were related to different domains including: (a) online retailing, (b) internet searching, (c) travel planning and booking, and (d)

online banking. All the 95 services had structural specifications expressed in WSDL [60] and QoS specifications, and more than half of them (52) had behavioral specifications expressed in BPEL [6]. The registry had been implemented as an eXist database [12].

In the evaluation, we also used six queries drawn from design models of two service-based systems that had been provided as case studies by the industrial partners of the SeCSE project, namely *ConferenceTravel* (i.e., the system introduced in Section 3), and *PurchaseTransaction* (i.e. a system for purchasing services and goods over the Internet). Three of these queries (Q1, Q2, and Q3) were defined for the *PurchaseTransaction* system and another three were defined for the *Conference Travel system* (queries Q4, Q5, and Q6). The exact form of the queries that we used reflected steps in the design of the relevant systems that had been identified by the industrial partners. The queries had a total of 18 query messages with different complexities, where (i) Q1 and Q5 had two query messages, (ii) Q2 and Q6 had three query messages, and (iii) Q3 and Q4 had four query messages.

The complexity of query messages was determined by the number of edges in the graphs of the data types of the parameters of the message, as in the case of operations (see Subsection 4.1). Based on this measure, a query message was classified as of *low complexity* if it had a data type graph with less or equal to ten edges and of *medium-high* complexity if it had a data type graph with more than ten edges. This boundary value between low and medium-high complexity was determined by a previous analysis of the complexity of service operations in the registry [68]. This previous analysis has demonstrated that 49% of the services in the registry had data type graphs of low complexity and 51% had data type graphs of medium-high complexity. The boundary value represented the median operation complexity found in that study. Furthermore, all the queries used in the experiment included the soft constraint shown in Figure 4, but no hard constraints. Hard constraints were excluded from queries as they could filter out services before the optimisation stage during query execution and, therefore, reduce the query execution time (due to the reduction on the number of services to be matched in the optimization phase) and improve recall and precision. Furthermore, each query was executed twice: once with a flexibility matching level k=0 and once with flexibility matching level k=1.

To evaluate recall and precision we used assessments of the *relevance* of service operations in the registry to the messages in the queries. These assessments of relevance were provided by six different users, who had no involvement in the development of the discovery framework and the specification of the queries and design models used in the experiment, and no knowledge of the algorithms used by the framework. Four of the users had a PhD in Software Engineering or Computer Science and two of them had an MSc in Computer Science and were undertaking a PhD in Software Engineering at the time of the experimentation. All users were familiar with service-oriented system engineering and UML based system design.

The six users who participated in the experiment assessed the relevance of 5688 possible pairs of query messages and service operations (18 query messages × 316 service operations) independently and prior to the execution of queries. Each user was given the queries and access to the descriptions of the services in the registry and asked to assess if the service operation in each of these pairs was relevant to the query message based on two *criteria of relevance*.

The first of these relevance criteria (RC1) was to assess relevance by looking at both the signatures and the behaviour of the query messages and service operations and consider as relevant only operations whose behaviour had an exact fit with what was expected in the query. The second criterion (RC2) was to assess relevance by looking at both the signatures and the behaviour of the query message and the service operations and consider as relevant operations whose behaviour was similar to what was required in the query even if it was not exactly the same.

The provision of generic criteria of relevance to the users was to avoid excessive diversity in their assessments [49]. The selected criteria corresponded to general factors that could be taken into account in the software design process and had a general correspondence to the different types of matching used in the discovery process of the framework. It should be noted, however, that no hints about this correspondence were given to the users.

Based on the assessments of the different users, we derived an aggregate final assessment of the relevance of service operations to query messages with respect to each of the three different relevance criteria. Aggregate relevance assessments were derived based on "voting scheme". More specifically, an operation was deemed relevant to a message with respect to a specific relevance criterion only if least 4 of the 6 users confirmed the relevance of a result (i.e., when at least 65% of the users confirmed the relevance of a result).

### 5.2 Recall and Precision Evaluation Results

The evaluation of recall and precision was used for assessing the ability of the framework to locate as many as possible service operations which are relevant to a specific query (recall), and disregard operations which are irrelevant in the discovery process (precision). In the experiments, recall and precision were measured according to the following formulas:

$$\text{Precision}_c = |SO \cap UO_c| / |SO| \quad (1)$$

$$\text{Recall}_c = |SO \cap UO_c| / |UO_c| \quad (2)$$

In these formulas,

- SO is the set of service operations that were retrieved as possible results for a query Q;
- $UO_c$ is the set of service operations that more than T percent of the users considered to be relevant to query Q according to criterion C; and
- $|X|$ is the cardinality of set X.

Recall and precision measures were computed for individual messages in the different queries using the above formulas. Based on these assessments, after the execution of queries, we measured recall and precision (using the above formulas) for the service operations that were returned for each query message, at 10 successive distance cut-off levels ($d_t$); i.e., for service operations with a distance of up to 0.1, 0.2, … and 1.0 from a query message. The use of different distance cut-off levels spanning the entire range of possible distance values,

enabled the evaluation of recall and precision when considering results at different distance levels. The main findings of the recall and precision evaluation of the framework are discussed below.

**Overall Performance**

Table 1 shows cumulative precision (P) and recall (R) measures taken at successive overall cut-off distance levels ($d_t$) between query messages and service operations (i.e., for operations having up to a $d_t$ distance from the relevant query message). The shown recall and precision measures are averages of recall and precision measures obtained across the individual messages of the different queries for different distance cut-off points using formulas (1) and (2). They are also based on: (i) overall query message to service operation distances computed using $w_s$=0.6, $w_b$=0.35 and $w_{sc}$=0.05 as weights, and (ii) aggregate assessments of relevance that were derived from individual relevance assessments of the six users using the voting scheme discussed in Subsection 5.1. The values of the weights used in the experiments demonstrate the importance of the different factors for the scenarios.

TABLE 1
OVERALL RECALL AND PRECISION FOR ALL QUERIES

| | k=1 | | k=0 | |
|---|---|---|---|---|
| D(≤$d_t$) | R | P | R | P |
| 0.1 | 0.10 | 0.97 | 0.19 | 0.86 |
| 0.2 | 0.54 | 0.74 | 0.67 | 0.42 |
| 0.3 | 0.80 | 0.30 | 1.00 | 0.15 |
| 0.4 | 0.83 | 0.22 | 1.00 | 0.11 |
| 0.5 | 0.95 | 0.07 | 1.00 | 0.03 |
| 0.6 | 1.00 | 0.03 | 1.00 | 0.01 |
| 0.7 | 1.00 | 0.03 | 1.00 | 0.01 |
| 0.8 | 1.00 | 0.03 | 1.00 | 0.01 |
| 0.9 | 1.00 | 0.03 | 1.00 | 0.01 |
| 1 | 1.00 | 0.03 | 1.00 | 0.01 |

The assessments of relevance that were used to evaluate recall and precision in the case of exact matching distances (i.e., when k=0) were different from those used for inexact matching distance (i.e., when k=1). More specifically, for k=0 we used assessments of relevance formulated by users after considering the relevance criterion RC1 (consideration of service and query message operation signatures and behaviour and the existence of an exact behavioural matching). For k=1, we used assessments of relevance formulated by the users after considering the criterion RC2 (consideration of service and query message operation signatures and behaviour and the existence of a non exact but good behavioural matching).

As shown in the table, precision was high for service operations with a distance of up to 0.1 from a query message since 97% and 86% of the retrieved operations in this distance range on average were relevant in the case of inexact and exact matching, respectively. For operations with distances up to 0.2, precision dropped to 74% in the case of inexact matching and 42% in the case of exact matching. Recall reached its maximum value of 1.0 when considering service operations with a distance of up to 0.3 in the case of exact matching and up to 0.6 in the case of inexact matching. These findings indicate that the users of the framework should expect a high accuracy of results when considering service operations whose distance from a query message is up to 0.2 in the case of inexact matching and up to 0.1 in the case of exact matching, but need to consider the relevance of service operations carefully for distances higher than these values depending on the required matching flexibility. Furthermore, to ensure that no relevant results are missed, all service operations with a distance of up to 0.3 or 0.6 need to be considered, when exact and inexact matching are deployed, respectively.

The results shown in Table 1 also indicate that the precision of inexact matching was higher than the precision of exact matching for all distance cut-off levels (see columns *(k=1)-P* and *(k=0)-P* in Table 1). The statistical significance of the difference in precision between inexact and exact matching was tested using the *paired t-test*. The use of this test in checking the statistical significance of comparative evaluations of the precision and recall of IR methods is supported by different studies and evidence that *t-test* produces reliable results even when assumptions about the normality of underpinning data do not hold [23][47].

The *t-test* was applied to pairs of cumulative precision rates that were calculated for each query message of the queries used in the experiments by the two types of matching for each distance cut-off level. The use of the test indicated that the observed differences at all the different distance cut-off levels were statistically significant at $\alpha$=0.05 (the probability *p* yielded by the *t-test* for the different cut-off points ranged from 0.00025 to 0.0425)[5].

The same test was applied to the recall measures of the two types of matching. As shown in Table 1, the recall of exact matching was higher than the recall of inexact matching until the distance cut-off level of 0.5. This difference was also statistical significant at $\alpha$=0.05.

**Effect of Partial Distances on Precision and Recall**

In the evaluation, we also investigated differences in the recall and precision measures produced by different partial distances. Table 2 presents the average recall and precision measures that were obtained at different distance cut-off levels based on the different types of distances computed by the framework, namely the overall (*D*), signature ($d_{f\text{-}sig}$), behavioural ($d_{f\text{-}beh}$), and soft constraint distances ($d_{f\text{-}con}$).

As shown in the table, the precision of results based on overall distances was higher than the precision of results based on signature distances only for all the cut-off distance points up to 0.5 in both inexact and exact matchings. The statistical significance of this difference was examined using the paired *t-test* and found to be statistically significant at $\alpha$=0.05 for all the cut-off distance levels up to 0.5.

The overall distance was also found to generate more precision results than the behavioural distance at the first cut-off point (0.1) for both types of matching (the differences at this level were statistically significant at $\alpha$=0.025 in both cases). However, for higher cut-off points, the picture was mixed. In particular, the differences in the preci-

[5] *p* is the probability of the two samples coming from a population with the same average precision.

sion of the overall and behavioural distance results were not statistically different at the cut-off point 0.2 in the case of exact matching and at the cut-off points 0.2 and 0.3 in the case of inexact matching. Following these two cut-off points, the differences in precision between these two distances became statistically significant again at $\alpha$=0.025 but this time the behavioural distance was found to generate results of higher precision. The reason for this phenomenon might be an increased focus of the users on behavioural aspects when service operations start being dissimilar to query messages.

Finally, the precision of the overall distance clearly outperformed the precision of the constraint distance at all cut-off points and the observed differences were statistically significant at $\alpha$=0.025.

The behavioural distance also produced more precise results than the signature distance at all the distance cut-off points for both types of matching, as shown in Table 2. However, the observed differences in the precision of these two distances were statistically significant at $\alpha$=0.025 for all but the first cut-off point (i.e., 0.1).

TABLE 2
RECALL AND PRECISION OF DIFFERENT DISTANCES

| (k=1) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| d | D | | $d_{f\text{-}sig}$ | | $d_{f\text{-}beh}$ | | $d_{s\text{-}con}$ | |
| $\leq d_t$ | R | P | R | P | R | P | R | P |
| 0.1 | 0.10 | 0.97 | 0.18 | 0.79 | 0.60 | 0.87 | 0.44 | 0.04 |
| 0.2 | 0.54 | 0.74 | 0.62 | 0.50 | 0.74 | 0.72 | 0.44 | 0.04 |
| 0.3 | 0.80 | 0.30 | 1.00 | 0.03 | 0.78 | 0.31 | 0.44 | 0.04 |
| 0.4 | 0.83 | 0.22 | 1.00 | 0.03 | 0.82 | 0.30 | 0.44 | 0.04 |
| 0.5 | 0.95 | 0.07 | 1.00 | 0.03 | 0.82 | 0.23 | 0.44 | 0.04 |
| 0.6 | 1.00 | 0.03 | 1.00 | 0.03 | 0.82 | 0.23 | 0.44 | 0.04 |
| 0.7 | 1.00 | 0.03 | 1.00 | 0.03 | 0.82 | 0.22 | 0.44 | 0.04 |
| 0.8 | 1.00 | 0.03 | 1.00 | 0.03 | 0.82 | 0.18 | 0.44 | 0.04 |
| 0.9 | 1.00 | 0.03 | 1.00 | 0.03 | 0.82 | 0.18 | 0.44 | 0.04 |
| 1 | 1.00 | 0.03 | 1.00 | 0.03 | 1.00 | 0.03 | 1.00 | 0.03 |
| (k=0) | | | | | | | | |
| d | D | | $d_{f\text{-}sig}$ | | $d_{f\text{-}beh}$ | | $d_{s\text{-}con}$ | |
| $\leq d_t$ | R | P | R | P | R | P | R | P |
| 0.1 | 0.19 | 0.86 | 0.21 | 0.52 | 0.87 | 0.64 | 0.51 | 0.01 |
| 0.2 | 0.67 | 0.42 | 0.62 | 0.21 | 0.93 | 0.42 | 0.51 | 0.01 |
| 0.3 | 1.00 | 0.15 | 1.00 | 0.01 | 1.00 | 0.15 | 0.51 | 0.01 |
| 0.4 | 1.00 | 0.11 | 1.00 | 0.01 | 1.00 | 0.15 | 0.51 | 0.01 |
| 0.5 | 1.00 | 0.03 | 1.00 | 0.01 | 1.00 | 0.11 | 0.51 | 0.01 |
| 0.6 | 1.00 | 0.01 | 1.00 | 0.01 | 1.00 | 0.11 | 0.51 | 0.01 |
| 0.7 | 1.00 | 0.01 | 1.00 | 0.01 | 1.00 | 0.10 | 0.51 | 0.01 |
| 0.8 | 1.00 | 0.01 | 1.00 | 0.01 | 1.00 | 0.09 | 0.51 | 0.01 |
| 0.9 | 1.00 | 0.01 | 1.00 | 0.01 | 1.00 | 0.09 | 0.51 | 0.01 |
| 1 | 1.00 | 0.01 | 1.00 | 0.01 | 1.00 | 0.01 | 1.00 | 0.01 |

In the case of recall, the signature distance produced better results than the overall distance for all cut-off points up to 0.5 in the case of inexact matching, and the observed differences were statistically significant at $\alpha$=0.025. In the case of exact matching, however, the differences in the recall of these two distances were neither consistent nor statistically significant.

A mixed picture of recall differences was observed across signature and behavioural distances. More specifically, the behavioural distance generated higher recall for the first two cut-off distance levels (0.1 and 0.2) for both types of matching. The differences for the 0.1 cut-off level were statistically significant at $\alpha$=0.025 for both types matching. At the 0.2 cut-off level, however, the recall difference was statistically significant (at $\alpha$=0.025) only in the case of exact matching. For cut-off levels greater than 0.2, however, the signature distances resulted in statistically better recall measures than behavioural distances (at $\alpha$=0.025) in the case of inexact matching, or equal recall measures in the case of exact matching.

**Effect of Matching Flexibility on Precision and Recall**

The effect of matching flexibility in recall and precision was also analysed in the experiments.

As discussed earlier, inexact matching (k=1) resulted in statistically significant higher average precision than exact matching (k=0) in the case of overall distances across all the distance cut-off points. Also exact matching resulted in statistically significant higher recall than inexact matching for all the distance cut-off levels up to 0.5, as discussed earlier.

The application of the *t-test* on the precision and recall measures associated with the different partial distances across the two types of matching indicated further statistically significant differences. More specifically, the differences in the precision of both the signature distances and the behavioural distances across the two types of matching flexibility were also statistically significant for all the distance cut-off points at $\alpha$=0.025. Also, the recall of behavioural distances in exact matching was higher than the recall of the same distances in inexact matching up to distances of 0.9 and the relevant differences were statistically significant at $\alpha$=0.025. In the case of structural distances, however, no statistically significant differences were observed in the recall measures.

### 5.3 Performance Evaluation Results

The performance evaluation focused on the time to execute queries Q1 to Q6. This time was measured for flexibility matching levels k=0 and k=1 and for service registries with 20, 40, 52, and 95 services, in order to analyze whether increments in these two factors affect the performance of the discovery process. All the queries were executed on an Intel Core 2 Duo machine, with 2.33 GHz and 500 MB Ram.

Table 3 presents the results of the execution times in seconds for Q1 to Q6 and flexibility matching k=0 and k=1. The results shown for each query $Q_i$ represent the average execution time of $Q_i$ that was taken across ten different executions of it for each of the different sizes of the registry used in our experiment. For each query, the table shows the average time taken to: (a) retrieve services from the registry, (b) execute structural matching, (c) execute behavioral matching, (d) execute soft constraint matching, and (e) execute whole query (rows *Total per query*). Table 3 also presents the total average time to execute all queries for each different number of services (column *Avg*) and the number of the query messages in each query (row Qm in the table).

As shown in the table, the time to retrieve services from the

registry grows linearly with the number of services, and it is not affected by the flexibility matching level k and the size of the queries. The high time for retrieving services from the registry, in particular for service registry with 95 services, is due to the fact that the database used to implement the registry is slow. It should be noted, however, that although the registry retrieval time affects the performance of the framework, the implementation of the service registry was not our focus during the development of the discovery framework. Furthermore, the registry retrieval time could be improved by using appropriate indexing schemes or an alternative DBMS for implementing the registry. These indexing schemes could be based on the properties of hard constraints in order to facilitate the filtering phase of the query execution process (see Section 4). However, as mentioned before, we have not used hard constraints in these experiments since we did not want to restrict the number of services to be used in the structural, behavioural, and soft constraint matching process.

The results also show that the time to execute structural matchings for a query grows linearly with the size of the registry. This is indicated in Table 3 by contrasting the average structural matching times of each query for k=0 or k=1 across registries of different sizes. This linearity was confirmed by regression analysis between the execution times recorded for each query and the corresponding size of the service registry. The $r^2$ coefficients that were produced for the different queries by linear regression analysis ranged from 0.82 to 0.91 across the different queries when k=0 and from 0.89 to 0.96 when k=1. Also both the fitted lines and the coefficients of the service size variable (i.e., the coefficient $b$ in the fitted lines $Y = bX+a$ where X is the registry size and Y is the structural matching time) were statistically significant at a=0.025 in all cases. The statistical significance of the fitted line was tested using the *F-test* and the statistical significance of the service size coefficient was tested using the *t-test*.

It should also be noted that in most cases, the structural matching time of queries with fewer query messages was lower than the structural matching time for queries with more query messages. An exception to this occurs in queries Q2 and Q3, as shown in Table 3. In this case, the structural matching time of query Q2 (three query messages) took a bit longer than the structural matching time of query Q3 (four query messages) in some cases. This was again due to the lower complexity of query messages in Q3. Thus, the time to execute structural matching varies not only with the number of query messages, but also with the complexity of these messages.

The results in Table 3 show that the behavioural matching time increased across all queries when a higher flexibility matching factor was used for all the different sizes of the registries. This result was confirmed by the the application of an one-tail *upaired t-test* over the samples of execution times obtained for each query in the ten different trials (ten data points in each set). The test was applied to 18 pairs of sample execution times – six pairs of samples for the three registries of 20, 40 and 52 services (the registry with 95 services was excluded as the additional services in it did not have a behavioural model and, hence, would not affect behavioural matching times). The normality of the underlying data – a condition that is normally required for the application of the t-test – was tested using the Anderson-Darling test [1]. This test confirmed the normality of the two thirds of the used samples. The application of the *t-test* showed that the execution time when k=1 was higher than the time when k=0, and the result was statistically significant at $\alpha$=0.025 in all cases. This result was expected since for higher values of k the number of combinations of paths in the state machines of the query and the service that need to be compared increases. The behavioral matching time increases linearly with the number of services up to 52 services in the registry, but there was no increase for 95 services. This was because in the data set

TABLE 3
PERFORMANCE TIME IN SECONDS FOR EXECUTING QUERIES Q1 TO Q6

| | # Ser | Q1 Qm 2 | Q2 Qm 3 | Q3 Qm 4 | Q4 Qm 4 | Q5 Qm 2 | Q6 Qm 3 | Avg |
|---|---|---|---|---|---|---|---|---|
| **(k=0)** | | | | | | | | |
| **Reg. Retr.** | **20** | 34.9 | 33.7 | 32.6 | 35.6 | 36.9 | 31.5 | **34.2** |
| | **40** | 81.8 | 77.9 | 80.5 | 73.6 | 84.1 | 83.0 | **80.1** |
| | **52** | 101.9 | 104.8 | 104.7 | 98.4 | 107.3 | 103.9 | **103.5** |
| | **95** | 203.0 | 202.4 | 202.7 | 202.6 | 202.5 | 202.2 | **202.6** |
| **Str. Match** | **20** | 5.1 | 7.2 | 7.3 | 8.7 | 5.2 | 7.5 | **6.8** |
| | **40** | 11.1 | 17.8 | 15.3 | 26.0 | 12.8 | 14.7 | **16.3** |
| | **52** | 16.0 | 22.4 | 21.0 | 35.1 | 19.2 | 22.5 | **22.7** |
| | **95** | 63.4 | 79.1 | 102.2 | 139.7 | 67.3 | 101.6 | **92.2** |
| **Beh. Match** | **20** | 17.2 | 16.7 | 16.9 | 14.8 | 15.7 | 17.2 | **16.4** |
| | **40** | 25.1 | 25.4 | 23.8 | 24.8 | 22.7 | 22.8 | **24.1** |
| | **52** | 33.9 | 32.6 | 31.7 | 31.2 | 32.0 | 31.6 | **32.2** |
| | **95** | 29.9 | 29.0 | 28.7 | 33.7 | 30.6 | 28.0 | **30.0** |
| **Soft Cons. Match** | **20** | 0.3 | 0.3 | 0.4 | 0.2 | 0.2 | 0.4 | **0.3** |
| | **40** | 0.4 | 0.6 | 0.6 | 0.6 | 0.2 | 0.3 | **0.5** |
| | **52** | 0.5 | 0.5 | 0.7 | 0.6 | 0.3 | 0.5 | **0.5** |
| | **95** | 1.0 | 1.2 | 3.8 | 2.0 | 1.0 | 0.9 | **1.7** |
| **Total per query** | **20** | 60.3 | 60.9 | 59.7 | 61.9 | 60.2 | 59.2 | **60.4** |
| | **40** | 122.1 | 125.7 | 123.8 | 128.8 | 123.1 | 124.1 | **124.6** |
| | **52** | 156.8 | 164.9 | 162.0 | 169.4 | 162.9 | 162.5 | **163.1** |
| | **95** | 306.3 | 320.2 | 354.1 | 388.9 | 309.8 | 340.3 | **336.6** |
| **(k=1)** | | | | | | | | |
| **Reg. Retr.** | **20** | 37.1 | 25.3 | 20.4 | 21.4 | 28.8 | 28.7 | **27.0** |
| | **40** | 81.8 | 70.6 | 44.5 | 39.5 | 79.4 | 70.3 | **64.4** |
| | **52** | 105.5 | 83.7 | 55.8 | 52.6 | 98.3 | 84.1 | **80.0** |
| | **95** | 201.8 | 205.5 | 201.6 | 201.3 | 201.5 | 201.5 | **202.2** |
| **Str. Match** | **20** | 3.4 | 7.3 | 6.0 | 9.7 | 5.9 | 5.5 | **6.3** |
| | **40** | 9.0 | 14.7 | 14.7 | 22.9 | 12.7 | 14.7 | **14.8** |
| | **52** | 13.4 | 22.3 | 21.0 | 35.0 | 19.1 | 22.4 | **22.2** |
| | **95** | 50.0 | 75.7 | 73.9 | 118.6 | 57.5 | 98.7 | **79.1** |
| **Beh. Match** | **20** | 17.6 | 47.2 | 224.7 | 230.8 | 25.4 | 35.9 | **96.9** |
| | **40** | 30.5 | 47.0 | 317.7 | 322.3 | 30.3 | 47.3 | **132.5** |
| | **52** | 41.0 | 57.9 | 333.9 | 333.3 | 41.1 | 57.2 | **144.1** |
| | **95** | 36.0 | 62.3 | 337.9 | 333.2 | 36.8 | 53.1 | **143.2** |
| **Soft Cons. Match** | **20** | 0.1 | 0.4 | 0.4 | 0.4 | 0.2 | 0.2 | **0.3** |
| | **40** | 0.2 | 0.3 | 0.5 | 0.5 | 0.2 | 0.3 | **0.3** |
| | **52** | 0.3 | 0.5 | 0.6 | 0.6 | 0.3 | 0.5 | **0.5** |
| | **95** | 0.5 | 1.0 | 1.4 | 1.1 | 0.6 | 0.9 | **0.9** |
| **Total per query** | **20** | 61.0 | 82.8 | 254.0 | 264.8 | 63.7 | 72.8 | **133.2** |
| | **40** | 125.3 | 136.0 | 381.0 | 388.6 | 126.6 | 136.0 | **215.6** |
| | **52** | 164.7 | 168.4 | 415.9 | 425.7 | 163.4 | 168.3 | **251.1** |
| | **95** | 295.7 | 352.3 | 623.4 | 661.9 | 303.8 | 361.4 | **433.1** |

used in the experiments there were a total of 52 services with behavioral specifications. The linear increase in the behavioural matching time with respect to the size of the registry was confirmed by regression analysis using the size of the registry as independent variable (X) and the behavioural execution time as the dependent variable (Y). The $r^2$ coefficients that were produced for the different queries by linear regression analysis ranged from 0.87 to 0.94 across the different queries when k=0 and both the fitted lines and the coefficients of the service size variable were statistically significant at a=0.025 in all cases. For k=1, the $r^2$ coefficients ranged from 0.69 to 0.98 in five of the six queries. The fitted lines and the coefficients of the service size variable (*b* coefficients) in these cases were statistically significant at $\alpha$=0.025. In the sixth query (Q2), the $r^2$ coefficient was equal to 0.18 and neither the regression line nor the coefficient of the independent variable was statistically significant (the probability of the *b* coefficient being different than 0 was 0.17 and therefore it was rejected at $\alpha$=0.025).

Table 3 also shows that the behavioural matching time increased with the number of query messages for k=1 (i.e., the times to execute Q3 and Q4 are higher than the times to execute Q2 and Q6, which are higher than the times to execute Q1 and Q5). In the case of k=0, the behavioral matching times recorded were similar for all the different queries given registries of the same size.

The above differences were confirmed by performing *analysis of variance* over three groups of behavioural execution times for different k values and service sizes. The first of these groups (G1) included the execution times in the ten trials of each of Q1 and Q5, the second group (G2) included the behavioural execution times in the ten trials of each of Q2 and Q6 and the third group included the execution times in the ten trials of each of Q3 and Q4. The differences across these three groups were found to be statistical significant for k=1 across each of the service sizes but insignificant when k=0. This was because for k=0 the matching of the paths in the state machines of the queries had to be exact with the paths in the state machines of the services (i.e., no transition in the paths of the state machines of the queries and services could be left without a counterpart). Thus, any path in the state machine of a service that was shorter than the path in the state machine of the query was ignored and, therefore, the number of detailed path comparisons during the behavioural matching decreased.

Although very small, the constraint matching presented some variations with the number of services and number of query messages. This was because the soft constraint had to be evaluated for each single service in a registry and for each of the query messages in a query.

It should be noted that in Table 3, the total time for each combination of query and registry size (rows *Total per query*) is higher than the sum of the registry retrieval, structural matching, behavioral matching, and soft constraint matching times for the given combination. This discrepancy arises because the total time per query includes the time required for additional computations during the execution of a query, namely the time required to (a) create graphs of the data types of the parameters of the query messages, (b) create state machines, and (c) parse constraints.

## 6 Discussion

Overall, the results of the evaluation presented in Section 5 demonstrate the merit of the proposed framework for service discovery and its ability to support this process as part of a service-based system design process that uses UML. This is particularly important if someone considers that the framework relies only on modeling notations and service standards, namely UML, WSDL and BPEL, which are widely used in the software industry and does not require the deployment of further notations whose uptake is limited within the industry (e.g. special purpose semantic service description languages). It should be noted, however, that the framework could be applied to services with different types of behavioural models as long as they can be translated to state machines.

The conducted evaluation has also indicated some basic characteristics of the discovery process that are important for practitioners. More specifically, when deploying the framework, users can expect that in most cases it will be sufficient to consider retrieved service operations having a distance to a query message in the range [0, 0.6] to ensure that no relevant operations are missed (as indicated in Table 1 recall reached its maximum value of 100% for both types of matching within this range). Furthermore, from the service operations which fall in this range only those with a distance of up to 0.1 are highly likely to be accurate (precision ranged from 97% to 86% for distances up to 0.1, depending on the type of maching).

The evaluation also indicated that the incorporation of behaviour and constraint matching into the retrieval process can increase precision for low-to-medium distance results significantly in comparison with a structure only matching process. The evaluation also indicated that the performance of the framework in terms of recall and precision is good even with inexact matching. This makes the use of the framework suitable in stages of software system design where system models are still evolving. Performing service discovery based on exact matching and constraints in such stages would not be appropriate.

Although the computation of behavioural matching for inexact cases (i.e., flexibility matching k>0) is combinatorial, the statemachines representing the queries are normally small.

Clearly, the realization of the discovery process by the presented framework makes assumptions that can limit its applicability in certain circumstances. One limitation is related to the computation of the linguistic distance ($d_L$) which assumes that service operation and query messages signatures are specified using the "Camel" notation where different words within strings are distinguished by starting with a capital letter. Whilst this convention is used widely in practice, our plan is to look at alternative ways of identifying words within strings in future work.

The use of weights in the computation of distances is another assumption that may turn out to be limiting. Weights are used to express the relative importance of

different factors (e.g., service interface, behaviour and QoS constraints) in the discovery process but users may sometimes have difficulty in expressing this importance accurately. The specification of weights has not been the main focus of our work, as our assumption has been that the exact phase in the system design life-cycle when the discovery is performed can indicate the general magnitude of weights.

If the design model that is used for discovery does not include elaborate behavioural models, for example, then the weight used for the behavioural distance could be low. Similarly, when certain parts of a design model are stable and should not be modified, the weights attached to the criteria of the matching process that relate to them should be relatively high. For example, if the data types of the parameters of some query messages should not be changed, the weight attached to the signature distance should be high. Alternatively, designers may decide to specify some hard constraint to ensure that operations won't be matched to the particular query messages unless they have the required parameters.

In general, it should be noted that the framework gives its users the ability to configure its matching process in various ways, including: (i) the assignment of weights to partial distances and constraints, (ii) the definition of constraints of different types (hard vs. soft) and importance, and (iii) the configuration of the flexibility of the behavioural matching process (by setting the value of the parameter k). This ability is one of the key characteristics of our approach and can be used to address different requirements which may arise in the design process.

The relative slow performance found in the evaluation of the approach has been mainly due to the time required to retrieve services from the registries and the matching of complex criteria such as data types of parameters of service and query operations, and behavioural matching. The registry retrieval time can be reduced by using faster DBMS or appropriate indexing schemes. The use of complex querying criteria (e.g. behaviour) is necessary to support service discovery during service-based system design and has been shown to improve the precision of results. Hence, the additional performance cost that they induce is justified. However, if developers prefer to avoid this cost, it is possible to specify queries without behavioural parts.

## 7 Related Work

There have been various strands of research in the literature to support service discovery. In [17], the authors describe some initial work in this area. We present below an account of the various approaches for service discovery.

The structural matching process used in our framework is similar to the work in [67] applied to software libraries. Our work extends this approach by considering matching of behavioral and quality specifications of services.

Approaches based on graph matching have been proposed in [20][26]. The work in [20] uses graph transformation rules for specifying services and service discovery queries. Similarly to our work, these rules represent each service operation by two "source" and "target" object graphs whose nodes and edges correspond to data entities and relationships between them, respectively. Our matching criteria are more flexible as they are based on distance measures which quantify similarities between the graphs.

Work on similarity analysis based on WordNet have been proposed in [28][55][63]. The approach in [63] uses four similarity assessment methods to support service matching, namely lexical, attribute, interface, and quality-of-service (QoS) similarity. In our approach, the distance of the parameters is computed by finding the best possible morphism between the data types of the operation parameters. Moreover, the quality matching in our framework is not restricted to specific types of quality aspects.

The work in [55] combines WordNet-based techniques and structure matching for service discovery. This approach identifies similarities between WSDL [60] specifications by comparing the structures and identifiers of the operations, messages, and data types in WSDL descriptions. Details of how the degree of similarity between data types is calculated are not described in the paper. The structural matching used in our work also considers the names of the operations, parameters, and data types, as well as the structure of primitive and complex data types in service specifications and structural query models. In our work, the similarity of data type structures is computed by considering the morphism of the graphs representing the data types. Unlike the work in [55], our approach does not compare WSDL specifications only, but it compares UML design models with WSDL specifications. In addition, our work differs from the technique in [55], since it uses behavioral and other types of constraint matchings.

The WSDL-M2 approach [28] uses lexical matching to calculate linguistic similarities between concepts, structural matching to evaluate the overall similarity between composite concepts, and combines vector-space model techniques with synonyms and semantic relations based on WordNet. The structural matching is based on maximum weight bipartite problem in which weights in the edges are denoted by lexical similarities of the two elements associated with the edge. Our work differs from WSDL-M2 since, in addition to lexical similarity of concepts and parameters, it considers the structure of the data types of the parameters, as well as the behavioral and quality aspects of the system being developed and services.

The approach in [22] uses service descriptions based on operation signatures that can be queried through XQuery. This approach is primarily focused on interface queries where operation signatures are matched using string matching. This form of matching is very limited, as it cannot account for small variations in operation signature specifications such as the use of different parameter names or orderings of parameters.

The use of behavioral matching for service discovery have been advocated in [18][19][34][50]. In [19], the approach uses (abstract) behavioural models of service specifications to increase the precision in service discovery. This approach locates services that satisfy task requirement properties expressed formally in temporal logic, by using a lightweight automated reasoning tool. Our approach can support the use of behavioral service specifications as those proposed in [19]. The approach in [50] proposes a behavioral model for services which associates messages exchanged between services with activi-

ties performed within services. A query language based on first-order logic that focuses on properties of behavior signatures is used to support the discovery process. The work in [34] advocates the use of behavioral specifications represented as BPEL for service discovery for resolving ambiguities between requests and services and use a tree-alignment algorithm to identify matching between request and services.

The work in [18] proposes an approach for service discovery based on the use of behavioural models for services represented as WSCL [59] conversation protocols. The behavioural matching in this work is based on graph matching representing user's requirements and service specifications in WSCL. More specifically, the approach transforms graphs by the use of editing operations and computes the distances between the graphs. These distances take into account the cost to insert and suppress edges and vertices in the graph, the cost to edit a vertice in the graph, and the linguistic differences of WSCL interactions (e.g., identifiers, types, and documents). Our behavioural matching is similar to the approach in [18]. However, in our work behavioural matching is based on the comparison of state machines representing UML sequence diagrams (queries) and service specifications in BPEL4WS. Moreover, in our approach the mappings between the state machines preserve the order of the transitions and allow for a pre-defined number of transitions not to be matched (flexibility matching).

Semantic web matchmaking approaches have been proposed to support service discovery based on logic reasoning of terminological concept relations represented on ontologies [1][2][5][21][25][27][28][32][41][55][56]. The METEOR-S [2] system adopts a constraint driven discovery approach in which queries are integrated into the composition process of a service-based system. In [1], semantic, temporal, and security constraints are considered during service discovery. In our framework, extra constraints concerned with structural, behavioral, and quality aspects of the system are considered.

In [21] the discovery of services is addressed as a problem of matching queries specified as a variant of Description Logic (DL) with service profiles specified in OWL-S [37]. The matching process is based on the computation of subsumption relations between service profiles and supports different types of matching. Our view is that our framework is more flexible as it can support the discovery of services specified in various specification formats (facets).

The work in [27] extends existing approaches by supporting explicit and implicit semantic by using logic based, approximate matching, and IR techniques. The work in [56] proposes QoS-based selection of services. In [25], the authors present a goal-based model for service discovery that considers re-use of pre-defined goals, discovery of relevant abstract services described in terms of capabilities, and contracting of concrete services to fulfill requesting goals. Our work differs from the above approaches since it supports the discovery of services not only based on the linguistic distances of the query and service operations and their input and output parameters, but also on the structure of the data type graphs of these parameters, the behavior of the system and services, quality aspects of the system and services, and extra conditions that cannot be specified by the models of the system. Moreover, our approach is not restrictive to return exact matches, but instead it returns a set of best matches for a request. These best matches provide the designer an opportunity to choose the most adequate service and to become more familiar with the available services.

Approaches for service discovery based on service capabilities have been proposed in [5][41]. The work in [41] uses DAML-S to describe service capabilities while in [5] services are described in OWL. In [41] service requests are matched against service advertisement by comparing outputs (inputs) of the request with outputs (inputs) of the advertisement. The approach considers four degrees of matching namely exact, plugin, subsumes, and fails. The work in [5] reduces these four degrees of matching to three degrees namely exact, inclusive, and weak. The approach in [5] also considers discovery of pervasive services based on context and QoS characteristics. As in our approach, this approach uses weights to denote the importance and preference of non-functional properties.

Other approaches have been proposed to support quality-of-services aware composition and service level agreements [7][36][40]. Although existing approaches have contributed to assist service composition an approach that uses these compositions as part of the development of service-based systems has not been proposed.

There have been proposals for specific query languages to support web services discovery [4][38][39][44][65]. In [4] the authors propose BP-QL a visual query language for business processes expressed in BPEL. The behavioral part of the query language used in our framework also supports querying BPEL specifications. However, our language is based on UML interaction diagrams, which is widely used to describe behavioral aspects of software systems. The query language proposed in [44] is used to support composition of services based on user's goals. NaLIX [65], which is a language that was developed to allow querying XML databases based on natural language, has also been adapted to cater for service discovery.

In [38], the authors propose USQL (Unified Service Query language), an XML-based language to represent syntactic, semantic, and quality of service search criteria. The query language used in our framework is more complete, since it accounts for the representation of behavioral aspects of the system. Moreover, in our query language, structural and behavioral criteria are represented by complete UML class and interaction models including the specification of complex types. Our constraint query language allows for the specification of not only quality aspects of the system, but also extra conditions concerned with structural and behavioral criteria. An extension of USQL that incorporates the behavioral part of our query language has been proposed in [39].

Although the above approaches have contributed to the problem of service discovery, none of them supports service discovery as part of the design process of service-based systems. Also, to the best of our knowledge, there are no other approaches that focus on service discovery based on structural, behavioral, and quality descriptions of services at the same time, as well as approaches that support service requests based on structural, behavioral, quality, and extra constraints of the system being developed. In our view, in many settings, service discovery needs to be integrated with main stream software development processes as the UML-based design

process that we assume in this paper and, in this respect, the approach that we have presented in this paper has clear elements of novelty.

## 8 Conclusion and Future Work

In this paper we have presented a framework to support the development of service-based systems by discovering services that can fit in the design of such systems. Our framework adopts an iterative process in which structural and behavioural design models of service-based systems, together with extra quality and non-quality constraints, are used to identify services that can fulfill functional and non-functional characteristics of the systems. The identified services are used to reformulate the design models, and trigger new service discovery iterations. The approach is UML-based as structural and behavioural design models are represented as UML class and sequence diagrams, respectively.

A query language enabling the specification of the characteristics of the services to be discovered has been developed. A query in this language contains: (a) a structural model, (b) a behavioural model, and (c) a constraint specification. The structural and behavioural models in a query are derived from UML design models of the system being developed. The constraint language allows for the representation of additional structural, behavioural, and quality properties that services should satisfy.

Queries are executed in a two-stage process. The first stage is a filtering phase, in which services that satify the hard constraints in a query are identified. The second stage is an optimization phase in which the services returned by the first stage which have the best match with the structural, behavioural, and soft constraints in a query are selected. This part of the process is flexible enough to allow the selection of services which might not have behavioural and/or other types of descriptions (excluding WSDL descriptions, which are always required). The matching is based on the computation of partial distances between service descriptions and queries.

A prototype tool has been implemented and used in an evaluation of the framework in terms of recall, precision, and performance. The results of this evaluation were positive indicating: (a) high precision (86-97%) of results (services) with low distance to queries (<0.1); (b) high recall of services for distances of up to 0.5 (95%-100%); and (c) statistically significant increase in precision when flexible behavioural service-query matching was included in queries (for low service-query distances of up to 0.2 the increase was from 18% to 32%). The results also confirmed that the time for all different types of matching and the overall query execution time grow only linearly with the number of services in registries and, therefore, the framework can scale to registries of large sizes.

We are currently extending the framework to support creation and negotiation of service level agreements during the developmnt of service-based systems, service discovery based on behavioural composition, and verification of design models.

## Acknowledgment

The work reported in this paper has been funded by the European Commission under the Information Society Technologies Programme as part of the project SeCSE (contract IST-511680).

## References

[1] Anderson, T. W.; Darling, D. A. "Asymptotic theory of certain "goodness-of-fit" criteria based on stochastic processes". Annals of Mathematical Statistics 23:193–212, 1952
[2] Aggarwal R., Verma K., Miller J., Milnor W. "Constraint Driven Web Service Composition in METEOR-S", IEEE Int. Conf. on Services Computing, 2004.
[3] Albert P., Henocque L., and Kleiner M. Configuration-Based Workflow Composition. Int. Conf. on Web Services (ICWS 2005), USA, July 2005.
[4] Beeri C., Eyal A., Kamenkovich S., and Milo T. "Querying Business Processes, 32nd Int. Conf. on Very Large Data Bases, 2006.
[5] Ben Mokhtar S., Preuveneers D., Georgantas N., Issarny V., and Berbers Y. "EASY: Efficient semantic Service discovery in pervasive computing environments with QoS and context support". Journal of Systems and Software 81: 785-808, 2008.
[6] BPEL4WS. www-106.ibm.com/developerworks / webservices/ library/ws-bpel.
[7] Canfora G., Di Penta M., Esposito R., Perfetto F., and Villani M.L. "Service Composition (re)Binding Driven by Application-Specific QoS". 4th Int. Conf. on Service Oriented Computing (ICSOC), December, 2006
[8] Cardoso J. and Sheth A. "Semantic e-Workflow Composition", Journal of Intelligent Information Systems, 21(3):191-225.
[9] Chafle G., Chandra S., Mann V., and Nanda M.G. "Orchestrating Composite Web Services Under Data Flow Constraints". Int. Conf. on Web Services, 2005.
[10] Courbis C. and Finkelstein A. "Weaving Aspects into Web Service Orchestration". Int. Conf. on Web Services, USA, 2005.
[11] Deubler M., Meisinger M., and Kruger I. "Modelling Crosscutting Services with UML Sequence Diagrams", ACM/IEEE 8th Int. Conf. on Model Driven Engineering Languages and Systems, MoDELS 2005, 2005.
[12] eXist. http://exist.sourceforge.net
[13] Faloutsos C. and Oard D. "A Survey of Information Retrieval and Filtering Methods", Tech. Report CS-TR3514, Dept. of Computer Science, Univ. of Maryland, 1995.
[14] Foster H., Uchitel S., Magee J., and Kramer J. "Compatibility Verification for WS Choreography", Int. Conf. on Web Services, 2004
[15] Fu X., Bultan T., and Su J. "Conversation Protocols: A Formalism for Specification and Verification of Reactive Services". Theoretical Computer Science, 328(1-2):19-37, November 2004.
[16] Gardner T., "UML Modelling of Automated Business Processes with a Mapping to BPEL4WS", 2nd European Workshop on OO and Web Services, 2004.
[17] Garofalakis J., Panagis Y., Sakkopoulos E., and Tsakalidis "A. Web Service Discovery Mechanisms: Looking for a Needle in a Haystack". Int. Workshop on Web Engineering, Hypermedia Development and Web Engineering principles and Techniques: Put them in Use, (ACM Hypertext 2004), 2004.
[18] Grirori D., Corrales J.C., and Bouzeghoub M. "Behavioral Matching for Service Retrieval", Int. Conf. on Web Services, ICWS 2006, 2006.
[19] Hall R.J. and Zisman A. "Behavioral Models as Service Descriptions", 2nd Int. Conference on Service Oriented Computing, ICSOC 2004, 2004.
[20] Hausmann, J. H., Heckel, R. and Lohmann, M., "Model-based Discovery of Web Services", IEEE Int. Conf. on Web Services (ICWS'04), USA, 2004.
[21] Horrocks, I., Patel-Schneider, P.F. and van Harmelen, F. "From SHIQ and RDF to OWL: The making of a Web ontology language", J. of Web Semantics, 1(1), 7-26, 2003.
[22] Hoschek W. "The Web Service Discovery Architecture", IEEE/ACM Supercomputing Conf., Baltimore, USA, 2002
[23] Hull, D. 1993. "Using statistical testing in the evaluation of retrieval experiments", 16th Annual ACM SIGIR Conference on Research and Development in information Retrieval, 1993

[24] Jones S., Kozlenkov A., Mahbub K., Maiden N., Spanoudakis G., Zachos K., Zhu X., Zisman A.: Service Discovery for Service Centric Systems, eChallenges 2005, Slovenia, October 2005.
[25] Keller U., Lara R., Lausen H., Polleres A., and Fensel D. "Automatic Location of Services", Proc. of 2nd European Semantic Web Conference (ESWC), Greece, 2005.
[26] Klein M. and Bernstein A. "Toward High-Precision Service Retrieval". IEEE Internet Computing, 30-36, January 2004.
[27] Klusch M., Fries B, and Sycara K. "Automated Semantic Web Service Discovery with OWLS-MX", 5th Int. Conf. on Autonomous Agents and Multiagent Systems, 2006
[28] Kokash N., van den Heuvel W.J., D'Andrea V. "Leveraging Web Services Discovery with Customizable Hybrid Matching", Int. Conf. on Web Services, ICWS 2006, 2006.
[29] Kozlenkov A., Spanoudakis G., Zisman A., Fasoulas F., Sanchez F. "Architecture-driven Service Discovery for Service Centric Systems" International Journal of Web Services Research, special issue on Service Engineering,, 4(2):81-112, 2007
[30] Kozlenkov A., Spanoudakis G., Zisman A., Fasoulas V., and Sanchez F. "A Framework for Architecture Driven Service Discovery". Int. Workshop on Service Oriented Software Engineering – IW-SOSE'06, 2006.
[31] Kramler G., Kapsammer E., Kappel G., and Retschitzegger W. "Towards Using UML 2 for Modelling Web Service Collaboration Protocols", 1st Conf. on Interoperability of Enterprise Software and Applications (INTEROP-ESA), 2005.
[32] Li L. and Horrock I. "A Software Framework for Matchmaking based on Semantic Web Technology", Int. WWW Conference Workshop on E-Services and the Semantic Web, 2003.
[33] Mahbub K. and Spanoudakis G. Run-time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience, Int. Conf. on Web Services, 2005.
[34] Mikhaiel R. and Stroulia E. "Interface- and Usage-aware Service Discovery", 4th Int. Conf. on Service Oriented Computing, 2006.
[35] Morato J., Marzal M. A., Llorens J., and Moreiro J. "WordNet Application", 2nd Global Wordnet Conference, 2004.
[36] Nguyen X.T., Kowalczyk R., and Han J. "Using Dynamic Asynchronous Agregate Search for Quality Guarantees of Multiple Web Services Compositions", 4th Int. Conf. on Service Oriented Computing, 2006.
[37] OWL-S. http://www.daml.org/services/owl-s/1.0, 2003.
[38] Pantazoglou M., Tsalgatidou A., and Athanasopoulos G.. "Discovering Web Services in JXTA Peer-to-Peer Services in a Unified Manner", 4th Int. Conf. on Service Oriented Computing, 2006.
[39] Pantazoglou M., Tsalgatidou A., and Spanoudakis G. "Behavior-aware, Unified Service Discovery", Service-Oriented Computing: a look at the inside Workshop, SOC@Inside'07, co-located with ICSOC, 2007.
[40] De Paoli F., Lulli G., and Maurino A. "Design of Quality-Based Composite Web Services", 4th Int. Conf. on Service Oriented Computing, 2006.
[41] Paolucci M., Kawamura T., Payne T.R., and Sycara K. "Semantic Matching of Web Services Capabilities". Int. Semantic Web Conference, Italy, 2002.
[42] Papazoglou M.P., Traverso P., Dustdar S., Leyman F., and Kramer B. "Service-Oriented Computing Research Roadmap". ftp://ftp.cordis.lu/pub/ist/docs/directorate_d/st-ds/services-research-roadmap_en.pdf.
[43] Papadimitriou C. and Steiglitz K. "Combinatorial Optimisation: Algorithms and Complexity", Prentice-Hall Inc.
[44] Papazoglou M., Aiello M., Pistore M., Yang J. "XSRL: A Request Language for web services" http://citeseer.ist.psu.edu/575968.html
[45] Di Penta M., Esposito R., Villani M.L., Codato R., Colombo M., and Di Nitto E.. WS Binder: a Framework to enable Dynamic Binding of Composite Web Services. Int.l Workshop of Service Oriented Software Engineering, Shanghai, May 2006.
[46] Pistore M., Traverso P., Bertoli P., and Marconi. "A. Automated Synthesis of Composite BPEL4WS Web Services". Int. Conf. on Web Services (ICWS2005), USA, July 2005.
[47] Sanderson M. and Zobel J. "Information Retrieval System Evaluation: Effort, Sensitivity and Reliability", 28th Annual ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 162-169, August 2005
[48] SeCSE, http://secse.eng.it/pls/secse/ecolnet.home.
[49] Seracevic T. "Relevance: A Review of the Literature and a Framework for Thinking on the Notion in Information Science – Part III: Behaviour and Effects of Relevance", Journal of the American Society for Information Science and Technology, 58(13): 2126-2144, 2007
[50] Shen, Z. and Su, J. "Web Service Discovery Based on Behavior Signature". IEEE Int. Conf. on Services Computing, 2005.
[51] Spanoudakis G, Constantopoulos P., "Elaborating Analogies from Conceptual Models", Int. Journal of Intelligent Systems, 11(11): 917-974, 1996.
[52] Spanoudakis G., Zisman A. and Kozlenkov A. "A Service Discovery Framework for Service Centric Systems", Int. Conf. on Services Computing (SCC 2005), USA, July 2005.
[53] Swinscow T.D.V., "Statistics at Square One", BMJ Publishing Group 1997, http://bmj.bmjjournals.com/collections/statsbk/index.shtml
[54] UDDI. http://www.uddi.org.
[55] Wang Y. and Stroulia E. "Semantic Structure Matching for assessing Web-Service Similarity", 1st Int. Conf. on Service Oriented Compusting, 2003.
[56] Wang X., Vitvar T., Kerrigan T., and Toma I. "A QoS-Aware Selection Model for Semantic Web Services", 4th Int. Conf. on Service Oriented Computing, 2006
[57] WebServiceX, http://www.webservicex.net/WS/default.aspx
[58] Woogle, http://haydn.cs.washington.edu:8080/won/wonServlet
[59] WSCL. Web Services conversation language. http://www.w3.org/TR/wscl10.
[60] WSDL. http://www.w3.org/TR/wsdl.
[61] WSML. http://www.wsmo.org/wsml/wsml-syntax
[62] WSMO.http://www.w3.org/Submission/2005/SUBM-WSMO-20050603.
[63] Wu J. and Wu Z. "Similarity-based Web Service Matchmaking". IEEE Int. Conf. on Services Computing, SCC 2005, July 2005.
[64] XPath. http://www.w3.org/TR/xpath.
[65] Yunyao L.Y., Yanh H., and Jagadish H. "NaLIX: an Interactive Natural Language Interface for Querying XML", SIGMOD 2005, Baltimore, June 2005.
[66] Zachos K., Zhu X., Maiden N., and Jones S. "Seamlessly Integrating Service Discovery into UML Requirements Processes". Int. Workshop of Service Oriented Software Engineering (IW-SOSE 2006), May 2006.
[67] Zaremski A.M. and Wing J.M. "Signature Matching: A Tool for Using Software Libraries", ACM Transactions on Software Engineering and Methodology, 4(2):146-170, 1995.
[68] Zisman A. and Spanoudakis G. UML-based Service Discovery Framework, 4th Int. Conf. on Service Oriented Computing, 2006.

**George Spanoudakis** holds BSc, MSc and PhD degrees in Computer Science. He is a Professor at City University in London and has more than 14 years of research experience and has been the principal investigator of several national, European and industry funded research projects. His principal research interests are in the area of software engineering with a recent focus on the development of techniques for runtime software systems verification and methods and techniques for engineering service-centric systems. George has published extensively in the area of software engineering and served on the program and organizing committees of more than 50 Int. Conf.s and workshops in this area. He is also an associate editor of the International Journal of Software Engineering and Knowledge Engineering. For more information see: http://www.soi.city.ac.uk/~gespan.

**Andrea Zisman** holds PhD, MSc, and BSc degrees in Computer Science. She is a Reader in the Deaprtment of Computing, City University London. Andrea has been research active in the areas of software and service engineering where she has published extensively. Her research interests are in service discovery, validation of service-based systems, and consistency management and traceability of software artefacts. Andrea has given tutorials in many Int. Conf.s and has served in the organising and program committees of various Int. Conf.s and workshops, has acted as a reviewer for many international journals, and has co-edited special issues of journals. Andrea has been principal and co-investigator in several European, EPSRC, and industry funded research projects. For more information see: http://www.soi.city.ac.uk/~zisman