UML-Based Service Discovery Framework

Andrea Zisman and George Spanoudakis

Department of Computing
City University
Northampton Square, London EC1V 0HB, UK
{a.zisman, gespan}@soi.city.ac.uk

Abstract. The development of service centric systems, i.e software systems constructed as compositions of autonomous services, has been recognised as an important approach for software system development. Recently, there has been a proliferation of systems which are developed, deployed, and consumed in this way. An important aspect of service centric systems is the identification of web services that can be combined to fulfill the functionality and quality criteria of the system being developed. In this paper we present the results of the evaluation of a UML-based framework for service discovery. This framework supports the identification of services that can provide the functionality and satisfy properties and constraints of service centric systems as specified during their design. Our approach adopts an iterative design process allowing for the (re-) formulation of the design models of service centric systems based on the discovered services. A prototype tool has been developed and includes (a) a UML integration module, which derives queries from behavioural and structural UML design models and integrates the results of the queries; and (b) a query execution engine, which performs queries against service registries based on similarity analysis.

1 Introduction

Service centric systems (SCS) has been recognised as an important paradigm for software system development in which service integrators, developers, and providers need to create methods, tools, and techniques to support cost-effective development and use of dependable services and service oriented applications. In the SCS paradigm software systems are constructed based on the composition of autonomous web services. Moreover, this paradigm centres on the creation, discovery, and composition of autonomous services that can fulfil various functional and quality requirements. Recently, software systems are being developed, deployed, and consumed in this way. The emergence of important standards in the last years has enabled the SCS vision. However, new processes, methods and tools are necessary to support the engineering of complex and dependable SCS.

Our interest relies in the engineering of hybrid service centric systems, i.e. software systems that are composed of services, but may also use legacy code or software components when no services can be found to fulfil the requirements and functionalities of the system. To assist the engineering of hybrid SCS, we developed a

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 402 – 414, 2006.

UML-based framework supporting service discovery. This framework allows the identification of services that can provide the functionality and satisfy properties and constraints of SCS specified during the design phase of the development life-cycle. It also supports the design of SCS by allowing for the (re-)formulation and amendment of the design models based on the services that have been discovered. Our framework uses UML to specify structural and behavioural design models of an SCS being developed and includes two main components: (a) a *UML 2.0 integration module*, which derives queries from UML design models and integrates back the results of the queries, and (b) a *query execution engine*, which performs the queries against service registries. The execution of queries is based on a two-stage approach. In the first stage, services that satisfy certain functional and quality criteria are located. In the second stage, the similarity of these services against additional functional and quality discovery criteria is assessed based on a similarity analysis algorithm [23].

The use of UML as a basis for our approach is due to several reasons: (a) UML is the de facto standard for designing software systems and can effectively support the design of SCS as it has been argued in [5][7][16]; (b) the use of services as well as legacy code and software components in the system; and (c) the expressive power of UML to represent the design models necessary in our approach and to specify queries to identify the services.

Our framework addresses important challenges and requirements that have been identified by industrial partners in the areas of telecommunications, automotive, and software in an integrated European project on service centric systems engineering (SeCSE [21]).

This paper focuses on the evaluation of our UML-based framework in terms of its precision. This evaluation has been conducted by three users with substantial knowledge in the areas of service centric engineering and object oriented modelling. In the evaluation, a total of 48 query iterations with various level of complexity have been performed in two different scenarios. These queries have been executed against a service registry with 97 real services and a total of 1028 operations with different numbers of parameters, data types associated with the parameters, and complexity. The results of the evaluation are encouraging and well accepted by our industrial partners, as described in the paper.

The remainder of this paper is structured as follows. In Section 2, we present an overview of the UML-based framework. In Section 3, we describe the evaluation of the framework and present the results of the experiments. In Section 4 we present some related work. Finally, in Section 5, we conclude and discuss future work.

2 Overview of UML-Based Service Discovery Framework

The UML-based service discovery framework adopts an iterative process in which the service discovery activity relies on the ongoing design of SCS and the available services identified during this process can be used to amend and reformulate the design models of the system. The reformulation of the design models may trigger new service discovery iterations. The result of this iterative process is a complete specification of the SCS structural and behavioural design models. In the framework, queries

are derived from system design models and support the identification of services that can subsequently be integrated into these models. The framework uses structural (SySM) and behavioural (SyBM) models of SCS expressed in UML as sequence and class diagrams, respectively.

Figure 1 shows an overview of the iterative process. The process starts from the construction of initial system structural (SySM) and behavioural (SyBM) models by the system designers. The SyBM model describes interactions between operations of an SCS that can be provided by web services, legacy systems or software components. The SySM model specifies the types of the parameters of the operations in SyBM, and constraints for these operations and their parameters (e.g., variants, pre- and post-conditions). When the result of the discovery process is not adequate, designers may decide to reformulate their queries and run the process again. It is also possible, that during the process the designers realise that parts of the system cannot be fulfilled by available services. In this case, designers may alter the design of the system to reflect the fact that the relevant part will be realised by existing legacy code and components, or by the development of new software code. Designers may terminate the process at any time or when further queries cannot discover services that match the existing design models.

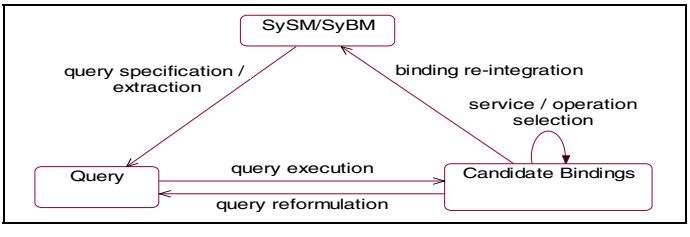

Fig. 1. Process overview

The interactions in SyBM and classes and interfaces in SySM are used to specify queries which are used to identify candidate services and operations that can fulfil parts of (or all) the functionality of the system. Designers may select some of the discovered services and operations and bind them to the design models. This binding results in a reformulation of both the SyBM and the SySM models. The new versions of these models may be used to specify further queries to discover other services that can satisfy more elaborated functionality, properties, and constraints of the system. When the results of the queries are not adequate, designers may reformulate their queries and execute them again.

The UML-based framework is composed of two main components. The *UML 2.0 integration module* component is combined with a UML CASE tool and is responsible for (a) extracting queries specifying the service functionality, properties, and constraints from the design models, based on the designer's selections, and (b) integrating the discovered candidate services back into the design models. The *query engine* component executes the queries by searching for services in different service registries. The search is based on a graph-matching algorithm [23] that computes similarities between queries and service specifications. We assume that service specifications

are composed of parts, called *facets*, which describe different aspects of services. Facets include information stored in service registries based on standard UDDI and ebXML technologies such as service interface specifications expressed in WSDL [29], behavioural service specifications expressed as BPEL4WS [4] or OMML [8], semantic service specifications expressed in OWL [19], WSMO [31], or WSML [30], quality of service information, and other information types (e.g., textual description) described in XML format.

Query Specification and Result. Queries are specified by system designers who select an interaction I from SyBM, create a copy of I called *query interaction* (*I'*), select the messages in *I'* that should be realised by operations of services to be discovered, and specify various constraints on these operations or the interaction as a whole.

A query and its results are specified by using a UML 2.0 profile that we have developed. The profile defines a set of stereotypes for different types of UML elements that may be found in (a) query interaction (e.g., messages), (b) results of query execution (e.g. messages, services), or (c) SySM model of a system that are referenced by elements of the query interaction (e.g., operations, classes that define the types of the arguments of interaction messages) or result parameters. The profile also contains metamodels of the facets that may be used for specifying services. A query is represented as a UML package stereotyped as <<asd_query_package>>[1]. A detailed description of the profile can be found in [15].

The messages of the interaction may be stereotyped as: (i) query messages <<asd_query_message>> that indicate the service operations that should be discovered; (ii) context messages <<asd_context_message>> that imply additional constraints for the query messages (e.g. 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); and (iii) bound messages <<asd_bound_message>> that are bound to concrete operations that have been discovered by executing the queries in previous iterations. All the messages in a query interaction, which are not stereotyped by any of the above stereotypes, are treated as unrelated messages in I'. These messages should not restrict the services to be discovered in any way and do not play any role in the query execution apart from being copied back to the results of a query execution. The operations corresponding to the query messages are stereotyped as <<asd_query_operation>>. The Profile also defines stereotype properties, which are used to specify parameters and constraints for the elements to which the stereotypes containing these properties are applied. Both <<asd_query_package>> and <<asd_query_message>> stereotypes can specify query parameters.

Query parameters are used to limit the search space and the amount of information returned by the query execution engine (e.g., the number of services to be returned), and are specified as scalar values. Query constraints stereotyped as <<asd_constraint>> provide specific selection criteria for choosing services based on their various characteristics. The constraints may be formulated in terms of UML metamodel or facets metamodel.

A constraint includes (a) a type (hard or soft), (b) an OCL [18] expression, and (c) an optional weight if the constraint is soft (real value between 0.0 and 1.0). Hard

[1] The stereotype names used in the profile have prefix "asd" indicating the name of the UML-based framework in the SeCSE project (Architecture-driven Service Discovery-ASD).

constraints must be satisfied by all the discovered services and operations. Soft constraints influence the identification of the best services/operations but may not be satisfied by all the services/operations that are discovered. The use of OCL is motivated by the fact that OCL is the standard formal language for specifying constraints for UML models and, therefore, queries which are based on these models. Apart from functional constraints, OCL expressions can be used to describe quality of service constraints in our framework.

Following the specification of a query interaction, the framework generates a query package that contains the context and query messages of the query, the classes that define the types of the parameters of these messages, as well as other classes that may be directly or indirectly referenced by these classes.

The result of a query identified by the query execution engine (see below) is also specified by using the profile and is represented as a UML package stereotyped as <<asd_results_package>>. The result package contains a refinement of the query interaction used by the designer to create the query together with the structural model for the elements in the interaction, and various UML service packages, for each candidate service identified by the query execution engine.

The service packages contain elements representing concrete discovered services together with all data types used in the XSD schemas reversed engineered from the WSDL specification of the services. The attributes and relationships of these data types are represented as a class diagram. The operations in the service packages can be *bound*, *candidate*, or *uncharacterised*. A bound operation signifies the service operation with the best match to a query message. A candidate operation reflects another possible result for the query message, but not necessarily the best match. The uncharacterised operations are other operations in the services WSDL specifications.

The framework allows the designer to analyse the results of a query and select candidate operations to become bound operations. After a particular service from the returned candidates is selected, the structural model in the results package is automatically updated with concrete data of the chosen service, and the interaction is modified to reflect the binding of the services and operations. The result package can be used as a basis for a new iteration.

Query Execution Engine. The query package is submitted to the query execution engine to be processed. The execution of queries is a two-stage process. In the first stage (*filtering*), the query execution engine searches service registries in order to identify services with operations that satisfy the *hard* constraints of a query and retrieves the specifications of such services. In the second stage (*best operation matching*), the query execution engine searches through the services identified in the filtering phase, to find operations that best match the soft constraints of the query.

Detection of the *best possible matching* between the operations required by a query and the candidate service operations identified in the filtering stage is formulated as an instance of the *assignment problem* following the approach proposed in [23]. More specifically, an *operation matching graph* G is constructed with (a) two disjoint sets of vertices: one set of vertices represent operations required by a query and another set of vertices represent the service operations identified in the filtering stage; and (b) edges that connect each of the operations in the query with all the operations of the retrieved services, and vice versa. Each edge *e(vi,vj)* in graph G is weighted by a measure that indicates the overall distance between vertices vi and vj. This measure

has a value between [0.0, 1.0] and is computed as the weighted sum of a set of *partial distances* quantifying the semantic differences between vi and vj, with respect to each facet F in the description of vi and vj.

Following the computation of the distances between the vertices, the matching between the operations in the query and the operations in the candidate services is detected in two steps. In the first step, a subset S of the edges in graph G is selected, such that S is a total morphing between the vertices in G and has the minimal distance values (this subset is selected by applying an assignment problem algorithm [23])[1]. In the second step, the subset S is restricted to include edges with distances that do not exceed a certain threshold value.

The partial distances are computed based on functions that take into consideration the distance of the signature of two operations. These functions account for the linguistic distance of the names of the operations and distance between the set of input and output parameters. The distance between the set of parameters is computed by finding the best matching between the structures of the data types of these parameters.

The best matching is identified by comparing edges in the graphs representing the structure of the data types of the input and output parameters. The graph of the input and output parameters of an operation is constructed taking into consideration both primitive data types and non-primitive data types. In the graph, the operation name is represented as the root of the graph with immediate $input\_{pi}$ and $output\_{po}$ children nodes, for each input and output parameter in the operation, respectively. The data type associated with an input parameter or output parameter is added to the graph as a child node of the respective $input\_{Pi}$ node or $output\_{po}$ node ($datatype\_{pi}$ and $datatype\_{po}$ nodes). The name of the input and output parameters are represented in the graph as the name of the edges between $input\_{pi}$ and $datatype\_{pi}$, and $ouput\_{po}$ and $datatype\_{po}$. In the case of a data type $datatype\_{i}$ 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 $datatype\_{i}$ is added to the graph as a child of $datatype\_{i}$ 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 node edges of the graph has only primitive data types.

An example of graphs of parameter data types is shown in Figure 2 for a query operation *getBusinessInfo(business:Business):string* (represented in white) and service operation *getStockDailyValueByValueXML(getstockdailyvalue:GetStockDailyValue): string* (represented in grey). In the figure, the dashed lines represent the matching of edges of the input and output parameters in both operation graphs based on the similarity of the names of the edges and their respective data types. For instance, edge exchange of complex type *Stock* is matched to edge *strStockExchange* in complex type *GetStockDailyValue*.

The matching process can support modifications to the set of facets F for service specifications. When new facets are added, the matching process can be extended by incorporating partial distance functions for enabling operation comparisons with respect to the new facets. A detailed description of the process with the distance functions is presented in [15].

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

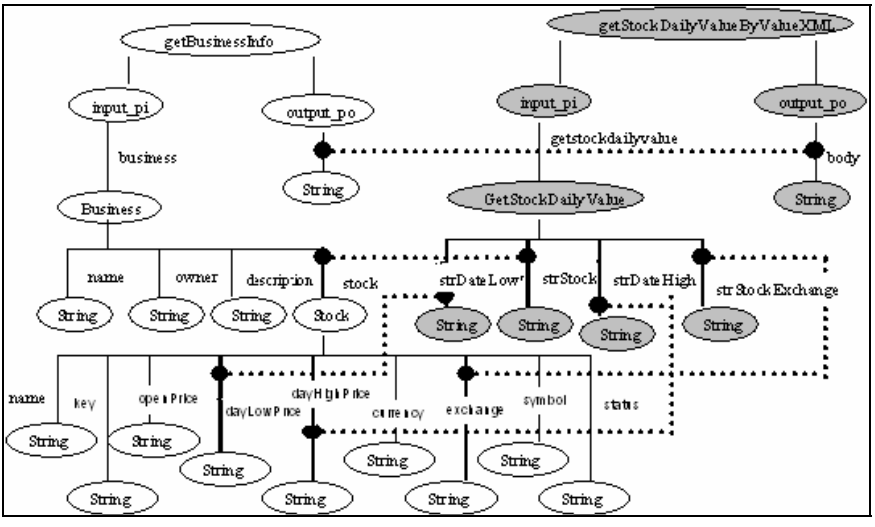

Fig. 2. Examples of data type graphs

3 Evaluation of UML-Based Service Discovery Framework

To evaluate our UML-based service discovery framework, we conducted a set of experiments using descriptions of real services that were identified in the Internet and queries that were constructed as part of two different design scenarios. The objectives of these experiments were to: (i) evaluate the service discovery precision that can be achieved using our framework, and (ii) investigate the effect of different factors on this precision including the complexity of queries, the number of query messages, the use of OCL constraints in them, the use of iterative queries incorporating context messages.

Precision was measured according to its standard definition in the information retrieval literature [6], i.e: $Precision_i = | SO \cap UO_i | / |SO|$, where SO is the set of service operations returned for a query Q; UO_i is the set of the retrieved operations for query Q (SO) that a user i considered to be adequate candidate answers for the query (i.e., relevant operations); and $|X|$ is the cardinality of set X.

3.1 Experimental Set Up

Service Registry. In our experiments, we used a service registry containing descriptions of 97 real services that were taken from various service providers including Across Communications [1], Arc Web [3], ViaMichelin [26], WebServiceX [27], Woogle [28], and Xignite [33]. Eighty two of these services were: (a) communication services (i.e., services that perform communication activities such as sending a fax, making a phone call, sending a text message); (b) location services (e.g., services identifying points of interest, verifying postal addresses, identifying best routes between locations); and (c) business services (e.g.. services providing stock information and market news). The remaining 15 services in the registry were not related to any of the above categories. The services in the registry had a total of 1028 operations of different complexity (see Section 3.2 below).

Queries. The queries used in our experiments were specified in reference to two SCS design scenarios. The first scenario (*AirportTrip)* was concerned with the design of a global positioning SCS offering its users various functionalities including: (i) identification of certain locations and airports in different cities, (ii) identification of best route to airports, (iii) checks for traffic problems in certain routes, (iv) displaying of maps, news reports and weather forecasts, and (v) translation of news reports between different languages. The second scenario (*BrokerInfo*) was concerned with the design of SCS stock purchasing system allowing users to: (a) purchase shares in different stock exchanges, (b) get stock market news, (c) get information about different companies, (d) find present and historical information about stocks, (e) get equity option information, (f) get information about exchange rates, and (g) send transaction documents by fax.

For each of the above scenarios we created a behavioural model (SyBM) of the intended system interactions and a structural model (SySM) defining the data types used in the behavioural model. These models can be found in [25]. Based on these models we specified 24 different service discovery queries for each of the scenarios. These queries were constructed in a way that ensured their variability with respect to four different characteristics that were introduces to investigate whether different types of queries may affect the precision of the results obtained for them and are described below.

(a) *Query complexity*: We used queries of *low* and *medium-high* complexity. The complexity of a query was measured by the number of edges in the graph of the data types of the parameters of the query messages in the query (see Section 2). Based on this measure, low and medium-high complexity queries were defined as queries that had query messages whose data type graphs had up to 10 edges and more than 10 edges, respectively. The threshold distinguishing between the *low* and *medium-high* complexity queries was identified by an analysis of the complexity of the service operations in the registry. This analysis shown that 49% of the service operations had data type graphs with less than 10 edges, 39% of the service operations had data type graphs with 10 to 19 edges, and 12% of the service operations had data type graphs with 20 or more edges. To have query complexity categories representative of the complexity of the operations in the registry, we set the threshold complexity value to 10 representing the median complexity of the operations in the registry.

(b) *Number of query messages*: We used queries with one, two and three query messages.

(c) *Existence or absence of context messages*: We used queries with no context messages and queries with one context message.

(d) *Existence or absence of OCL expressions*: We used queries with no OCL constraints and queries with one OCL constraint. The same type of OCL constraint was used for all the queries of the latter type. This constraint was global and restricted the services that should be returned by a query to be offered by a particular service provider (the form of the constraint was *self.description.Provider.contains('nameOfProvider')*).

Table 2 shows a summary of the different types of queries used in the experiment. The numbers (1) to (24) in the cells of the table represent a different query type. A specific query was created for each query type in each of the two scenarios. Furthermore, we set the number of candidate service operations that should be returned for each query message in a query to be three. This number was fixed to guarantee that precision would be measured in a consistent way across all queries. The value three was selected as it

Table 2. Different types of queries used in the experiment

		No Context Message						With Context Message					
#Query Msg		1		2		3		1		2		3	
OCL		N	Y	N	Y	N	Y	N	Y	N	Y	N	Y
Complexity	Low	1	2	5	6	9	10	13	14	17	18	21	22
	Med/High	3	4	7	8	11	12	15	16	19	20	23	24

was the average number of returned relevant service operations in a sample of queries that we executed prior to the main experiment and, therefore, it would allow for the retrieval of all relevant operations in a query. The complete list of the queries used in the experiment is presented in [25].

Users. In the experiments three different users indicate whether the operations discovered by each query were relevant to the corresponding query message. These users had substantial knowledge in the areas of service centric engineering and object oriented modelling. Each user provided relevance assessments for all 48 queries executed across the scenarios.

3.2 Experimental Results

Tables 3 and 4 show an overview of the precision that we observed in our experiments. More specifically, Table 3 presents the average precision that was recorded for all the queries executed for each of the scenarios and users, and the general precision across all scenarios, queries and users. These results show that on average the precision of the service discovery results for all users in all scenarios is 67%. Also the average measures recorded for the different scenarios were not significantly different: 68% for *AirportTrip* scenario, 67% for *BrokerInfo* scenario. Although some differences were observed for the different users, there was no specific trend for the different scenarios.

Table 4 shows the precision for the different types of queries averaged across the different scenarios and users. As shown in the table (*#Query Mes / Total AVG* row), queries with a larger number of query messages had slightly higher precision than queries with fewer messages (precision ranged from 60% in queries with 1 message to 65.7% in queries with 2 messages and 68.8% in queries with 3 messages). Also, we observed that queries of low complexity had a lower precision than queries of medium/high complexity (60.7% and 70%, respectively, as shown in row *Complexity / Total AVG*). Both these observations confirmed the expectation that as the specification of a query becomes more elaborated (more query messages, messages with more complex data types), precision improves as the models provide a basis for computing more fine-grain distance measures. This trend, however, was not confirmed in the case of OCL constraints where queries with no OCL constraints had higher precision than queries with OCL constraints (69.2% and 60%, respectively as shown in row *OCL / Total AVG*). This was a consequence of the form of the constraint used, which restricted results to services provided by a specific provider and in some queries the required provider did not offer any service with operations relevant to the queries.

The detailed results of our experiments for each scenario and each user with the distance measures of the operations can be found in tables presented in [25].

Our experiments have also demonstrated that the average distance of a discovered service operation that is considered to be relevant to a query message is less than the

Table 3. Summary of precision results

Scenario	User	Average per User and Scenario	
AirportTrip	**U1**	0.71	**AVG:** 0.68
	U2	0.68	
	U3	0.64	
BrokerInfo	**U1**	0.65	**AVG:** 0.67
	U2	0.65	
	U3	0.70	
		Total Average	0.67

Table 4. Summary of precision results by query characteristics

	No Context Message							
	No OCL				**With OCL**			
	1	**2**	**3**	**AVG**	**1**	**2**	**3**	**AVG**
Low Complexity	0.34	0.84	0.68	0.61	0.34	0.69	0.78	0.60
Med/High Complexity	0.84	0.72	0.81	0.79	0.84	0.47	0.57	0.62
Average	0.59	0.78	0.74	0.7	0.59	0.58	0.675	0.61
	With Context Message							
	No OCL				**With OCL**			
	1	**2**	**3**	**AVG**	**1**	**2**	**3**	**AVG**
Low Complexity	0.44	0.86	0.76	0.68	0.44	0.69	0.48	0.54
Med/ High Complexity	0.78	0.5	0.74	0.67	0.78	0.67	0.70	0.72
Average	0.61	0.59	0.75	0.67	0.61	0.68	0.59	0.63
#Query Mes / Total AVG	**1**		0.600	**2**		0.657	**3**	0.688
Complexity / Total AVG	**Low**		0.607	**Med/High**		0.700		
OCL / Total AVG	**No OCL**		0.692	**With OCL**		0.620		
Context /Total AVG	**No Context**		0.660	**With Context**		0.653		

average distance of a discovered service operation that is considered not to be relevant to a query message. Furthermore, the difference between these two average distances is statistically significant. This is evident from Table 5 which shows the average distances between operations which were considered to be relevant and not relevant to a query message. The average distances shown in this table have been calculated across the different query scenarios for the individual users who participated in the experiments. The table also shows the standard deviation of the distances of relevant and not relevant operations to query messages and the number of observed cases.

The statistical significance of the difference between the average distance of relevant service operations and query messages and the average distance between not relevant service operations and query messages was checked using the *t-test* assuming samples with non equal variances [24]. The values of the t-statistic that were calculated for the three different users are also shown in Table 5 (row *t-value*) and demonstrate that the probability of the difference in the average distances be incidental was almost 0. Thus, the differences in the averages can be considered as statistically significant (at α=*0.01*). It should also be noted that the average distance of relevant operations to query messages was less than the average distance of non-relevant operations to query messages. These two observations demonstrate that the distance functions which underpin the querying process implemented by the framework produce distance measures which can differentiate between relevant and not relevant operations in a way that is compliant with assessments provided by designers.

Table 5. Average distances of relevant and irrelevant operations

	U1		U2		U3	
	Relevant Ops	**Not relevant Ops**	**Relevant Ops**	**Not relevant Ops**	**Relevant Ops**	**Not relevant Ops**
AVG distance	0.1326	0.1660	0.1342	0.1655	0.1320	0.1687
Standard deviation	0.0233	0.3567	0.0250	0.0383	0.0220	0.0402
# observations	276	155	274	159	257	174
t-value	10.46		-9.225		-10.986	
d-f	229		237		243	

Overall, the average precision measured in our experiments (i.e., 67%) is an encouraging result. Our results are comparable to the results achieved in [32]. An evaluation of the approach in [32] has shown precision measures between 42% and 62% for similarity analysis of names and types of parameters of service operations (*interface similarities*). Furthermore, the UML-based framework is supposed to be used in an interactive process in which the discovery activity relies on the ongoing development of the design of an SCS and the available services identified during the process can be used to amend and reformulate the design models of the system by the SCS engineer. Thus, in our view a precision of 67% provides a good basis for obtaining more precise results in subsequent discovery queries defined using amended and more elaborated design models by SCS designers.

4 Related Work

Semantic matchmaking approaches have been proposed to support service discovery based on logic reasoning of terminological concept relations represented on ontologies [2][10][12][14][17]. The METEOR-S [2] system adopts a constraint driven service discovery approach in which queries are integrated into the composition process of a SCS and represented as collections of tuples of *features*, *weight*, and *constraints*. In our approach, the queries contain information about features, weights, constraints, and parts of the design models of the SCS being developed. In [10] the discovery of services is addressed as a problem of matching queries specified as a variant of Description Logic (DL). The work in [14] extends existing approaches by supporting explicit and implicit semantic by using logic based, approximate matching, and IR techniques. 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. 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 give the designer the opportunity to choose the most adequate service and become more familiar with the available services and, therefore, design the system based on this availability. Matching based on the structure of data types is important during the design phase of (hybrid) SCS since they specify the functionality and constraints of the system being constructed during design phase.

Hausmann et al. [9] propose the use of graph transformation rules for specifying both queries and services. The matching criteria in our work are more flexible and are

based on distance measures quantifying similarities between the graphs. Another approach that uses graph-matching is [11] although details of the matching algorithm are not described. The approach in [13] focuses on interface queries where operation signature checking is based on string matching and cannot account for changes in the order or names of the parameters. In [8] the authors advocate the use of (abstract) behavioural models of service specifications in order to increase the precision of service discovery process. Similarly, in [22], the authors propose to use service behaviour signatures to improve service discovery. We plan to conduct new evaluations using behavioural specifications, as proposed in [8].

Some specific query languages for web services have been proposed [20][34], although they cannot be integrated with UML-based system engineering design process. The use of UML to support SCS has been advocated in [5][7][16]. However, none of these approaches combines service discovery as part of the UML-based design process of SCS. When comparing to the existing discovery approaches, our UML-based service discovery framework has demonstrated that UML can be used to support design of SCS and service discovery.

5 Conclusions

In this paper we have presented the results of the evaluation of a UML-based framework to support service discovery in terms of its precision. Our UML-based framework adopts an iterative design process for service centric systems (SCS) and allows the (re-)formulation and amendment of design models of SCS based on discovered services. The framework identifies services based on queries derived from UML behavioural and structural models of SCS. The results of our experiments have shown that on average the precision of the UML-based framework is around 67%. The experiments have also demonstrated that the average distance between relevant discovered service operations and query messages was less than the average distance between not relevant discovered service operations and query messages, and that the difference between these average distances was statically significant. We are conducting new evaluations of our framework that take into consideration behavioural service models and quality constraints.

Acknowledgements. 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] Across Communications, http://ws.acrosscommunications.com/
[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] Arc Web, http://www.esri.com/software/arcwebservices/
[4] BPEL4WS. “Business Process Execution Language for WS”, http://www.106.ibm.com/developerworks/library/ws-bpel.
[5] Deubler M., Meisinger M., and Kruger I. "Modelling Crosscutting Services with UML Sequence Diagrams", ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2005, Jamaica, October 2005.

[6] 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.
[7] Gardner T., “UML Modelling of Automated Business Processes with a Mapping to BPEL4WS”, 2nd European Workshop on OO and Web Services (ecoop), 2004.
[8] Hall R.J. and Zisman A. “Behavioral Models as Service Descriptions”, 2nd Int. Conference on Service Oriented Computing, ICSOC 2004, New York, November 2004.
[9] Hausmann, J. H., Heckel, R. and Lohmann, M., “Model-based Discovery of Web Services”, IEEE International Conference on Web Services (ICWS’04), USA, 2004.
[10] 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.
[11] Hoschek W. “The Web Service Discovery Architecture”, IEEE/ACM Supercomputing Conf., Baltimore, USA, 2002.
[12] 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.
[13] Klein M. and Bernstein A. “Toward High-Precision Service Retrieval”. IEEE Internet Computing, 30-36, January 2004.
[14] Klusch M., Fries B, and Sycara K. “Automated Semantic Web Service Discovery with OWLS-MX”, 5th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), Japan, 2006
[15] Kozlenkov A., Spanoudakis G., Zisman A., Fasoulas V., and Sanchez F. “A Framework for Architecture Driven Service Discovery”. International Workshop on Service Oriented Software Engineering – IW-SOSE’06, in conjunction with ICSE’06, Shanghai, May 2006.
[16] Kramler G., Kapsammer E., Kappel G., and Retschitzegger W. "Towards Using UML 2 for Modelling Web Service Collaboration Protocols", Proc. Of the 1st Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA '05), 2005.
[17] Li L. and Horrock I. “A Software Framework for Matchmaking based on Semantic Web Technology”, 12th Int. WWW Conference Workshop on E-Services and the Semantic Web, 2003
[18] OCL. http://www.omg.org/docs/ptc/03-10-14.pdf
[19] OWL-S. http://www.daml.org/services/owl-s/1.0, 2003.
[20] Papazoglou M., Aiello M., Pistore M., Yang J. “XSRL: A Request Language for web services” http://citeseer.ist.psu.edu/575968.html
[21] SeCSE, http://secse.eng.it/pls/secse/ecolnet.home.
[22] Shen, Z. and Su, J. “Web Service Discovery Based on Behavior Signature”. IEEE International Conference on Services Computing, SCC 2005, USA, July 2005.
[23] Spanoudakis G, Constantopoulos P., “Elaborating Analogies from Conceptual Models”, International Journal of Intelligent Systems, 11(11), pp917-974, 1996.
[24] Swinscow T.D.V., "Statistics at Square One", BMJ Publishing Group 1997,: http://bmj.bmjjournals.com/collections/statsbk/index.shtml
[25] UML-based Framework.. http://www.soi.city.ac.uk/~zisman/ASD_Evaluation
[26] ViaMichelin, http://ws.viamichelin.com/wswebsite/gbr/jsp/prs/MaKeyFeatures.jsp
[27] WebServiceX, http://www.webservicex.net/WS/default.aspx
[28] Woogle, http://haydn.cs.washington.edu:8080/won/wonServlet
[29] WSDL. http://www.w3.org/TR/wsdl.
[30] WSML. http://www.wsmo.org/wsml/wsml-syntax
[31] WSMO.http://www.w3.org/Submission/2005/SUBM-WSMO-20050603.
[32] Wu J. and Wu Z. "Similarity-based Web Service Matchmaking". IEEE International Conference on Services Computing, SCC 2005, USA, July 2005.
[33] Xignite, http://www.xignite.com/
[34] Yunyao L.Y., Yanh H., and Jagadish H. “NaLIX: an Interactive Natural Language Interface for Querying XML”, SIGMOD 2005, Baltimore, June 2005.