

A Framework for Architecture-driven Service Discovery

A. Kozlenkov V. Fasoulas F. Sanchez G. Spanoudakis A. Zisman

Department of Computing, City University, Northampton Square, London EC1V 0HB, UK

Phone: +44 20 7040 8346 Fax: +44 20 70400244

a.kozlenkov@city.ac.uk, {v.fasoulas, sa840, gespan, a.zisman}@soi.city.ac.uk

ABSTRACT

Service discovery has been recognised as an important aspect in the development of service centric systems, i.e. software systems that are constructed based on the composition of web services. In order to develop service centric systems it is necessary to identify web services that can be combined to fulfil the functionality and quality criteria of the system being developed. In this paper we present a framework to support architecture-driven service discovery – that is the discovery of services that can provide the functionalities and satisfy properties and constraints of systems as specified during the design phase of the development lifecycle. Our framework assumes an iterative design process and allows 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 two main components: a UML 2.0 integration module, which derives queries from behavioural and structural UML design models and integrates the results of the queries; and a query execution engine, which performs the queries against service registries. The execution of the query is a two-stage process based on a similarity analysis algorithm.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Technique – *Computer-aided software engineering, modules and interfaces, object-oriented design methods, user interfaces.*

General Terms

Algorithms, Design, Languages.

Keywords

Service centric system, service discovery, profile, messages, queries, operations.

1. INTRODUCTION

The development of service centric systems (SCS), in which software systems are constructed based on the composition of autonomous web services, has been recognised as an important paradigm for software system development. Recently, software systems are being developed, deployed, and consumed in this way, shifting from traditional object-oriented approaches. This new paradigm focuses on the creation, discovery, and composition of autonomous services that can fulfil various functional and quality characteristics of the system.

In the last few years, we have seen the emergence of important

standards and technologies that enable the service centric vision. However, there is still the need to extend current software development practices with new processes, methods, and tools to assist the engineering of complex and dependable SCS.

An important aspect of service centric systems is the ability to support the discovery and composition of services at different stages of the development life cycle of a system. In this paper, we present a framework to support *architecture-driven service discovery* (ASD). By architecture-driven service discovery we mean the identification of services that can provide the functionality and satisfy quality properties and constraints of an SCS as specified by its design models.

Architecture-driven service discovery requires support to address some important challenges, including:

- The extraction of service discovery queries from SCS architecture and design models specifying the functionality and quality properties of such systems;
- The provision of a query language supporting both the expression of arbitrary logical combinations of prioritised functionalities and quality properties criteria for the required services, and *similarity-based queries* of the form "find a service that is similar to service X";
- The efficient matching of service discovery queries against service specifications and return of services that may have varying degrees of match with the queries;
- The assistance to system designers to select services for an SCS in cases where the discovery process identifies more than one candidate services satisfying a query;
- The integration of discovered services into an iterative design process in which SCS architecture and design models may be re-formulated following service discovery.

The above challenges have been identified by industrial partners in the areas of telecommunications, automotive, and software in an integrated European project focusing on service centric system engineering (SeCSE [16]). These challenges constitute the main driver underpinning the framework that we present in this paper.

Our framework adopts an iterative architecture-driven service discovery process in which the discovery activity relies on the ongoing development of the architecture of the SCS and, therefore, the available services identified during this process can be used to amend and reformulate the design models of the system. The reformulation of these models may trigger new service discovery iterations. The result of this process is a complete specification of the SCS architecture models.

The framework assumes the use of UML to specify structural and behavioural design models of the SCS. It includes a *UML 2.0 integration module*, which derives queries from UML design models and integrates back the results of the queries, and a *query*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE'06, May20-28, 2006, Shanghai, China.

Copyright 2006 ACM

execution engine, which performs these 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 variant of a similarity analysis algorithm presented in [15].

The remainder of this paper is structured as follows. In Section 2, we describe the ASD process and framework with its main components, and introduce a scenario that we will use to illustrate our work. In Section 3, we discuss how queries are specified in our approach. In Section 4, we present the query execution engine. In Section 5, we discuss ASD results. In Section 6 we give an overview of implementation aspects. In Section 7 we present some related work. Finally, in Section 8, we summarise our approach and discuss future work.

2. ARCHITECTURE-DRIVEN SERVICE DISCOVERY FRAMEWORK

2.1 ASD Process

Our framework adopts an iterative architecture-driven service discovery process in which service discovery is seen as part of the design of service centric systems. In the ASD process, queries are derived from system design models and identify services that can subsequently be integrated into these models.

Figure 1 shows an overview of the ASD process. The process is iterative and 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). These models are specified and visualised as UML sequence and class diagrams, respectively.

The interactions in SyBM and classes and interfaces in SySM are used to specify queries. The queries 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.

It is also possible for the designers to realise that certain parts of the system cannot be fulfilled by available services. In this case, the designer may alter the design models so as to get the relevant parts of the system functionality from existing legacy systems and components, or by developing new software code. Designers may decide to terminate the ASD process at any moment, or when it is clear that further queries cannot discover services relevant to the design models.

Our approach assumes that SySB and SySM models are expressed in UML (as sequence and class diagrams, respectively). This is because UML is the de-facto standard for designing software systems and can effectively support the design of service-centric

systems as it has been argued in [3][8]. Furthermore, UML has the expressive power to represent the design models that we use, and can provide a basis for specifying ASD queries (see Section 3).

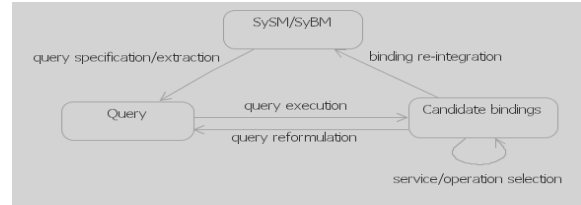


Figure 1: ASD Process Overview

2.2 ASD Framework

In order to support the above process we have developed an ASD framework that is composed of two main components: a *UML 2.0 integration module* and a *query execution engine*.

The *UML 2.0 integration module* is combined with a UML CASE tool (IBM Rational Software Modeler 6.0, in the current implementation) 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* executes the queries by searching for services in different service registries. The search is based on a graph-matching algorithm [15] that computes similarities between queries and service specifications. We assume that service specifications are composed of parts, called *facets*, which provide description of different aspects of services. The facets include information stored in service registries based on standard UDDI and ebXML technologies such as service interface specifications expressed in WSDL[17], behavioural service specifications expressed as BPEL4WS[2] or OMML [4], semantic service specifications expressed in OWL[12], WSMO[19], or WSML[18], and other information types (e.g., cost, quality, textual description) described in XML format.

2.3 ASD Scenario

In order to illustrate our work, in the rest of this paper, we will use an example of the design of a *global positioning service centric system* (GP_SCS) as a subsystem of a car-based Haptical device. The GP_SCS offers its users various functionalities including the identification of routes to get from one place to another, back-on-track re-routing, avoidance of specific areas, display of maps, and location of different points of interest in selected areas.

More specifically, our scenario focuses on the design of an interaction of GP_SCS that can locate the closest point of interest of a certain type (e.g. restaurants, car parks, cinemas) given an address. Based on requirements related to this functionality of the system, assume that a software designer has produced a behavioural model (SyBM) of the above interaction and a structural model (SySM) defining the data types used in the interaction, as shown in Figures 2 and 3, respectively. These models are specified and visualised as UML sequence and class diagrams, respectively. The scenario is taken from an “End-to-end travel” use case in which *Traveller* is the only actor. The sequence diagram in Figure 2 depicts the boundary of the system represented by component Haptical interacting with two broad

global parameters are considered as default values in the query and can be overridden by local 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. These constraints can be applied for `asd_query_package`, `asd_query_message`, and `asd_query_operation` elements. The constraints may be formulated in terms of UML metamodel (e.g., number of parameters in a query operation) or facets metamodel (e.g. textual description, cost).

The constraints include (a) the type of the constraint (hard or soft), (b) the body of the constraint as an OCL [11] expression, and (c) an optional weight of the constraint if the constraint is soft (real value between 0.0 and 1.0). Hard 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 to specify constraints is motivated by the fact that OCL is the standard formal language for specifying constraints for UML models and therefore ASD queries which are based on them.

Following the specification of a query interaction, the tool generates an ASD 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 tool automatically executes the extraction of recursive data structures used in the parameters of the query messages. The resulting query package is represented in XMI 2.0 – the standard XML based format for representing UML 2.0 models.

Example. Consider the design of GP_SCS shown in Figures 2 and 3. Suppose that the designer wants to specify a query (*Location Query 1*) to identify services that can return the location of an address based on its geographical co-ordinates and services that can identify the position of a POI within a certain acceptable distance (messages *FindAddress* and *FindPOI*, respectively, in Figure 2). Suppose that for this query the designer wants to restrict the number of candidate service operations that should be returned for each of the query messages to be 4 and the maximum number of services that should be searched in the registry to be 100.

Figure 5 shows the specification of the query in our framework. Apart from the query name, all the other query parameters are optional. A default value is assumed for a query parameter that is not specified. After specifying the values for the query parameters, the framework generates a query interaction, which is stored in the query package.

As an example of specifying local query parameters, suppose that for message *FindAddress* the designer wants to restrict the number of candidate services returned to be not greater than 10 and the maximum number of services examined to be 100. The designer may also specify a local query constraint (in OCL) on message *FindAddress* such that the textual description of candidate services must contain the term “geocod”.

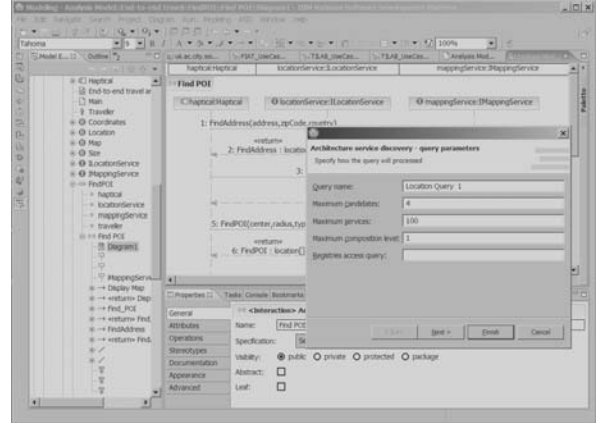


Figure 5: Example of the creation of Location Query 1

4. Query Execution Engine

The ASD query package is submitted to the query execution engine to be processed. The execution of queries is performed in a two-stage process. In the first stage, referred to as *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, referred to as *best operation matching*, the query execution engine searches through the services identified in the filtering phase, to find the operations that have the best match with the soft constraints of the query.

Selection of best operation matching: Detection of the best possible matching between the operations required by an ASD query and the candidate service operations identified in the filtering stage is formulated as an instance of the *assignment problem* as proposed in [15]. More specifically, given the set of operations required by an ASD query Q , $Oper(Q)$ and the set of service operations identified in the filtering stage, $Oper_S(Q)$, an *operation matching graph* is constructed with two disjoint sets of vertices V^Q and V^S defined as

$$V^Q \equiv Oper(Q) \cup DV_k \text{ and } V^S \equiv Oper_S(Q),$$

where DV_k is a set of k special vertices representing dummy operations ($k = |Oper(Q)| - |Oper_S(Q)|$). This formulation assumes, without loss of generality, that $|Oper(Q)| > |Oper_S(Q)|$. If this is not the case, k dummy vertices are added to V^Q where $k = |Oper_S(Q)| - |Oper(Q)|$.

The set of edges of the graph, $E(V^Q, V^S)$, includes all the possible edges between the required operations in V^Q and the retrieved service operations in V^S . These edges are weighted by a measure $D(v_i^Q, v_j^S)$ indicating the overall distance between v_i^Q and v_j^S where $v_i^Q \in V^Q$ and $v_j^S \in V^S$. This measure is computed as the weighted sum of a set of partial distance measures $d_f(v_i^Q, v_j^S)$ quantifying the semantic differences between v_i^Q and v_j^S with respect to each facet f in the descriptions of v_i^Q and v_j^S according to the following distance function (the weights are assumed to be normalized):

$$\begin{aligned} D(F, v_i^Q, v_j^S) &= \sum_{f \in F} w_f d_f(v_i^Q, v_j^S) \text{ if } v_i^Q \in Oper(Q), v_j^S \in Oper_S(Q) \\ D(F, v_i^Q, v_j^S) &= 1 \text{ if } v_i \in DV_k \\ D(F, v_i^Q, v_j^S) &= \infty \text{ if } v_i \text{ should not be mapped onto } v_j^S \end{aligned}$$

where F is the set of facets in the descriptions of operations.

Function D is defined to have a value in the range $[0,1]$ for all the pairs of operations drawn from $Oper(Q)$ and $Oper_S(Q)$. In the case of comparisons between an existing operation and a dummy operation D 's value is defined to be 1. This favours the possibility of mapping an existing operation onto a requested operation rather than leaving without a counterpart. Finally, D is defined to take an infinitum value (∞) in the case of operations which – by virtue of the constraints defined in Q – should not be mapped onto each other. This precludes the matching of such operations when the optimal matching between V^Q and V^S is selected.

Following the computation of the D distances for all the edges of the graph, the matching between the operations in V^Q and V^S is detected in two steps. In the first step, a subset $O(V^Q, V^S)$ of $E(V^Q, V^S)$ that is a total morphism between V^Q and V^S (or onto morphism if $|Oper(Q)| < |Oper_S(Q)|$) and minimises the function $\sum_{v_i, v_j \in O(V^Q, V^S)} D(F, v_i, v_j)$ is selected. $O(V^Q, V^S)$ is selected using standard algorithms for the *assignment problem* [15]. In the second step, $O(V^Q, V^S)$ is restricted to include only the edges whose distance $D(F, v_i, v_j)$ does not exceed a threshold value D_t .

Partial distance functions. For the facets corresponding to soft constraints defined as Boolean tests, the partial distances d_f are defined as 1 if the test returns *false* or the facet is not available for a specific service, and 0, otherwise. The partial distance that is used to compare the facets specifying the signatures of two operations is defined as

$$d_{f=signature}(v^Q, v^S) = w_N * d_L(name(v^Q), name(v^S)) + w_{IN} * d_{PS}(in(v^Q), in(v^S)) + w_{OUT} * d_{PS}(out(v^Q), out(v^S))$$

In this formula, d_L is a linguistic distance built on top of WordNet lexicon [10] and d_{PS} is a function computing the distance between the sets of input or output parameters of two operations. For two sets of parameters $P1$ and $P2$, d_{PS} is computed by finding the best possible morphism pm between the elements of these sets, defined as: $d_{PS}(P1, P2) = \min_{pm} (\sum_{(x,y) \in pm} d_P(x,y))$, where $d_P(x,y)$ is a function that computes the distance between two specific parameters. This distance is computed by finding the best possible matching between the structures of the types of the given operation parameters using a variant of the class distance measures defined in [15].

Our operation matching framework has been designed to support modifications to the set of facets F for service specifications. More specifically, when new facets are added to F , our framework could be extended to support them by incorporating partial distance functions enabling operation comparisons with respect to these facets.

5. ASD Results

The results of an ASD query identified by the query execution engine (i.e. best candidate services with smallest distances) is specified by using the ASD profile. Figure 6 presents the part of the ASD profile for ASD results. The ASD results are represented as a UML package stereotyped as `<<asd_results_package>>`.

The `asd_results_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 a number of UML packages stereotyped as `<<asd_service_package>>`, one for each candidate service identified by the query execution engine. Each `asd_service_package` contains elements representing a concrete

discovered service together with the class diagram of all data types and their relationships used in the XSD schemas reverse engineered from the WSDL specification of this service. The structural model in the `asd_results_package` contains copies of all data types from the `asd_query_package` together with the mapping (stereotyped as `<<asd_mapping_association>>`) to the data types in the service packages. This data mapping is based on the data distances computed for each bound operation in the service against the query message associated to the service.

The operations in an `asd_service_package` may be stereotyped as (i) bound operations `<<asd_bound_operation>>` that denote the service operations with the best match to a query message or the one that the designer selects as the best candidate; (ii) candidate operations `<<asd_candidate_operation>>` that reflect another possible result for the query message, but not necessarily the best match; and (iii) service operations `<<asd_service_operation>>` that are all the remaining operations in the WSDL specification of the service. The above operations are grouped together in a UML component (contained in the `asd_service_package`) stereotyped as either `<<asd_bound_service>>` or `<<asd_candidate_services>>`, depending on the existence of any bound operations.

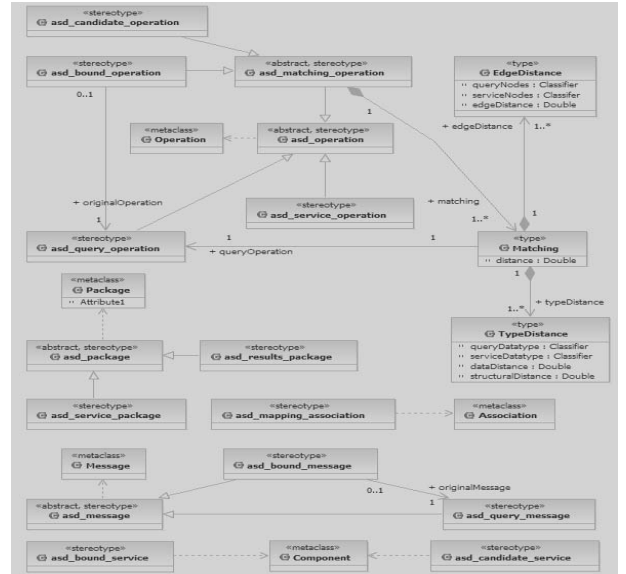


Figure 6: Part of the ASD Profile for ASD Results

The interaction in the `asd_results_package` refines the query interaction by replacing query messages by bound messages (stereotyped as `<<asd_bound_message>>`) corresponding to bound operations. When no operation is found, the query message is not modified.

The framework allows the designer to analyse the results of a query and select candidate operations to become bound operations. After the designer selects a particular service from the returned candidates, the structural model in the `asd_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 designer may use the `asd_results_package` as a basis for a new iteration of the ASD process (see Subsection 2.1).

Example. Consider the execution of *Location Query 1* against a service registry with 45 services. For this query, the query

execution engine selects four distinct services. Tables 1 and 2 show the selected services and their providers with the identified operations and the computed distance between these operations and the respective ASD query messages.

For this example, the best matching for ASD query message *FindAddress* is *findAddresses()* operation from service *AddressManager* (*ArcWeb* provider), and for query message *FindPoi* is *getPoiList()* operation from service *FindNearbyPOIService* (*ViaMichelin* provider). The above operations are stereotyped as <<asd_bound_message>> in the *asd_results_package*.

After analysing and accepting the results for messages *FindAddress* and *FindPOI*, consider the designer wants to create a new query (*Location Query 2*) for operation *Display Map* and start a new iteration of the ASD process.

For the discovery of a service that can return the URL of the map of a given location (*Display Map*), assume the designer wants the provider of this service to be *ViaMichelin* (the designer is now familiar with this provider from the results of the first query). In this case, the designer creates a new query, with a hard OCL constraint to restrict the discovery process to services provided by *ViaMichelin*.

Provider	Service	Operation	Distance
ArcWeb	AddressManager	findAddresses()	0.138
cdyne.com	AddressLookup	AdvancedCheckAddress()	0.145
cdyne.com	AddressLookup	CheckAddress()	0.150
cdyne.com	AddressLookup	CheckAddressW2lines()	0.152

Table 1: Results of query execution for *FindAddress*

Provider	Service	Operation	Distance
ViaMichelin	FindNearbyPOIService	getPoiList()	0.136
ViaMichelin	FindNearbyPOIService	getPoi()	0.140
ViaMichelin	FindNearbyPOIService	getCompactPoiList()	0.140
ArcWeb	PlaceFinderSample	findPlace	0.154

Table 2: Results of query execution for *FindPOI*

For this query, the query execution engine identifies operation *getMapDefinition()* from service *GetBestMapDefinitionService* (*ViaMichelin* provider) as the best match. Figure 7 shows the complete sequence diagram with the bound services and operations for the two queries.

The framework also allows the designer to search for other results by using bound messages from previous iterations of the ASD process, as constraints for a query (context message).

As an example, consider again the query iteration for operation *Display Map*. Suppose that the designer wants to investigate the operations that can be discovered for query message *Display Map*, when this query message is constrained by the bound operation *getPoiList()* identified in *Location Query 1*. In this case, operation *getPoiList()* is stereotyped as context message (<<asd_context_message>>) in the query. For the bound operation *getPoiList()*, the framework has created data mapping associations between the data types used in query message *FindPOI* and those used in operation *getPoiList()*. For this particular case, data type *Coordinates* used in *FindPOI* (see Figure 2) is mapped to data type *GeoCoordinates* in *FindNearbyPOIService* service, and data type *Location* is mapped to data type *Location* in the same service.

Since *Display Map* and *FindPOI* have originally the same parameter *center* of type *Coordinates* (Figure 2), and *Coordinates* is mapped to *GeoCoordinates*, the query will be constrained to

find operations similar to data type *GeoCoordinates*. The results of the query will prefer services from the same provider as that of the context message, since the best matching algorithm considers the minimal distances between operation names, parameters, and (nested) data types.

The query execution engine identifies operation *getMapDefinition()* from service *GetBestMapDefinitionService* (*ViaMichelin* provider) as the best match, which is the same result as found for the previous case (i.e., when the service provider *ViaMichelin* was used as a hard constraint). However, if we consider another query for *Display Map* without any constraints on the provider or context message, the query execution engine returns *getThematicGeographiesForExtent()* from *MapImage* service as the best match for this query.

6. IMPLEMENTATION ASPECTS

The ASD framework has been implemented as an Eclipse plugin for IBM Rational Software Modeler/Architect 6.0 (RSM/RSA). This UML CASE tool has almost complete support for the UML 2.0 notation and XMI 2.0 model serialization, and is based on extensible and open Eclipse 3.0 IDE. The tool offers all the necessary requirements for extensibility and compliance to open standards. Only a small portion of the plugin uses the vendor-specific (IBM) API, but the majority of the UML model manipulation code uses the standard open source UML2 API provided by Eclipse, which greatly enhances the portability of our tool to other UML CASE tools based on Eclipse.

The *ASD profile* is also implemented as an Eclipse plugin, so that it is deployed in RSM and can be applied to UML models via RSM GUIs and programmatically. The specification of the stereotype properties is assisted by the use of Eclipse wizards and property editors. The definitions of OCL constraints are assisted by IntelliSense live syntax checking and help. The query execution engine is currently a part of the plugin, but can be wrapped as a web service, which allows the execution of ASD queries independently of any particular CASE tool.

7. RELATED WORK

Some approaches have been proposed to support architecture-driven service discovery. In [6] 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[12]. Our framework is more general and supports the discovery of services specified in various specification formats (i.e. facets).

Hausmann et al. [5] 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 [7] although details of the matching algorithm are not described.

The METEOR-S [1] system adopts a constraint driven service discovery approach in which the queries are integrated into the composition process of a SCS and are represented as a collection 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.

The approach in [8] 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.

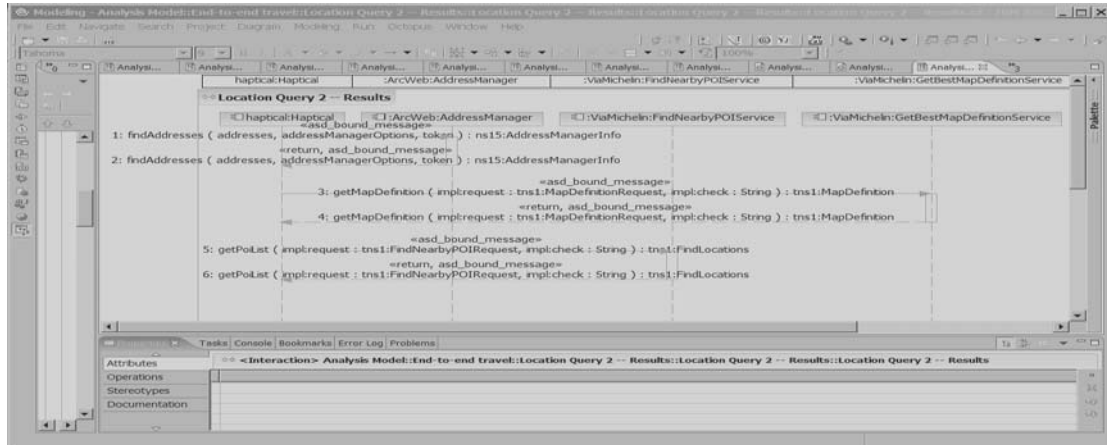


Figure 7: Results of the best matching for the queries

In [4] the authors advocate the use of (abstract) behavioural models of service specifications in order to increase the precision of service discovery process. Similarly, in [14], the authors propose to use service behaviour signatures to improve service discovery. We plan to extend our work to use behavioural specifications, as proposed in [4].

Some specific query languages for web services have been proposed [13][20]. However, none of them can be integrated with UML-based system engineering design process.

Although the above approaches have contributed to the problem of service discovery, an approach that combines service discovery as part of the design process of SCS has not been proposed.

8. CONCLUSIONS

We presented a framework to support architecture-driven service discovery that is integrated with iterative UML-based system engineering design processes. Our framework addresses the five challenges described in the introduction, in particular allowing service discovery to be driven by architecture decisions taken during the development of SCS systems and fulfils the lack of processes and tools to assist the engineering of complex and dependable SCS. Together with industrial partners, we are conducting large-scale experimentation of our framework taking into consideration different types of service specifications ranging from structural, to semantic and behavioural aspects.

9. ACKNOWLEDGMENTS

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).

10. REFERENCES

[1] Aggarwal R., Verma K., Miller J., Milnor W. "Constraint Driven Web Service Composition in METEOR-S", IEEE Int. Conf. on Services Computing, 2004.
 [2] BPEL4WS. "Business Process Execution Language for WS", <http://www.106.ibm.com/developerworks/library/ws-bpel>.
 [3] Gardner T., "UML Modelling of Automated Business Processes with a Mapping to BPEL4WS", 2nd European Workshop on OO and Web Services (ecoop), 2004.

[4] Hall R.J. and Zisman A. "Behavioral Models as Service Descriptions", 2nd Int. Conference on Service Oriented Computing, ICSOC 2004, New York, November 2004.
 [5] Hausmann, J. H., Heckel, R. and Lohmann, M., "Model-based Discovery of Web Services", IEEE International Conference on Web Services (ICWS'04), USA, 2004.
 [6] 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.
 [7] Hoschek W. "The Web Service Discovery Architecture", IEEE/ACM Supercomputing Conf., Baltimore, USA, 2002.
 [8] Klein M. and Bernstein A. "Toward High-Precision Service Retrieval". IEEE Internet Computing, 30-36, January 2004.
 [9] Kotov V. "Towards Service Centric Systems Organisation", <http://www/hpl.hp.com/techreports/2001/HPL2001-54.pdf>.
 [10] Morato J., Marzal M. A., Llorens J., and Moreira J., 2004. "WordNet Application", Proceedings of GWC 2004. The Second Global Wordnet Conf. 2004, Czech Republic.
 [11] OCL. <http://www.omg.org/docs/ptc/03-10-14.pdf>
 [12] OWL-S. <http://www.daml.org/services/owl-s/1.0>, 2003.
 [13] Papazoglou M., Aiello M., Pistore M., Yang J. "XSRL: A Request Language for web services" <http://citeseer.ist.psu.edu/575968.html>
 [14] Shen, Z. and Su, J. "Web Service Discovery Based on Behavior Signature". IEEE International Conference on Services Computing, SCC 2005, USA, July 2005.
 [15] Spanoudakis G, Constantopoulos P., "Elaborating Analogies from Conceptual Models", International Journal of Intelligent Systems, 11(11), pp917-974, 1996.
 [16] SECSE, <http://secse.eng.it/pls/secse/ecolnet.home>.
 [17] WSDL. <http://www.w3.org/TR/wSDL>.
 [18] WSMO. <http://www.wsmo.org/wsmo/wsmo-syntax>
 [19] WSMO. <http://www.w3.org/Submission/2005/SUBM-WSMO-20050603>.
 [20] Yunyao Li Y., Yang H., Jagadish H. "NaLIX: an Interactive Natural Language Interface for Querying XML". SIGMOD 2005, Baltimore, MD, June 2005