ISSN 2047-3338

Discovering Services during Design Phase Using UML

Arram Sriram[1] and Shaik Shah Nawaz[2]

[1,2]Department of CSE, Aurora Engineering College, Bhongir, Nalgonda, A.P, India

arram.sriram@gmail.com, shahshaik2006@gmail.com

***Abstract*– Service discovery has been recognized as an important aspect in the development of service centric systems, i.e., software systems which deploy web services. To develop such systems, it is necessary to identify services that can be combined in order to fulfill the functionality and achieve quality criteria of the system being developed. In this paper, we present a framework supporting design impelled and ranked services detection (DIRSD) - that is the exploit of services in QOS bead ranking order, which can provide functionalities and satisfy properties and constraints of systems as specified during the software design phase. Detailed system design is a compulsion and an assumption about iterative design process that allows the formulation of design models of service-centric systems based on the exploited services. The framework is composed of a query extractor, which derives queries from behavioral and structural UML design models of service centric systems, and a query execution engine that executes these queries against service registries. The paper describes a prototype tool that we have developed to demonstrate and evaluate our framework and the results of a set of preliminary experiments that we have conducted to evaluate it.**

***Index Terms*– Design, UML, Services and Framework**

I. INTRODUCTION

THE development of service centric systems (SCS) - that is the construction of software systems that deploy autonomous web services that can fulfill various functional and quality characteristics - is increasingly recognized as an important paradigm of software system development [1] [2] [3].

This paradigm requires the extension of current software development practices with new processes, methods, and tools to support the effective discovery and composition of web services into an SCS which, in addition to such services, may also use legacy code or software components. Depending on the stage that it occurs within the development life-cycle of an SCS, web service discovery (or simply "service discovery" for the purpose of this paper) can be distinguished into [4]: early service discovery (ESD) - this is service discovery that occurs in the requirements analysis phase in the development life-cycle of an SCS and is driven by its requirements specification, Design-Impelled and ranked service Detection (DIRSD) - this is service discovery that occurs during the Structural and behavioral model design of the software, and ranked service discovery (RSD) - this is service discovery that occurs during the deployment of an SCS and is concerned with the replacement of existing services of an SCS that ranked low based on QOS with a service that ranked high in QOS during the execution of the system.

The work presented in this paper focuses on a framework to support design-impelled and ranked service detection. This form of service discovery requires the development of capabilities to address some important challenges including:

(i) The extraction of service discovery queries from SCS architecture and design models specifying the functionality and quality properties of such systems.

(ii) The provision of a query language supporting both the expression of arbitrary logical combinations of prioritized 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".

(iii) The efficient matching of service discovery queries against service specifications and return of services that may have varying degrees of match with the queries.

(iv) 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 or services that do not satisfy a query entirely.

(v) The integration of the discovered services into an iterative design process in which SCS architecture and design models may be re-formulated following the discovery of services.

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) ("Electronic source," n.d.). These challenges constitute the main driver underpinning the DIRDS framework that we present in this paper.

Our framework adopts an iterative design-impelled service detection and ranked process, and assumes the use of any modeling language to specify structural and behavioral design models of SCS. The framework includes a query extractor, which derives queries from UML design models, and a query 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 which satisfy certain functional and quality criteria are located and maintained for

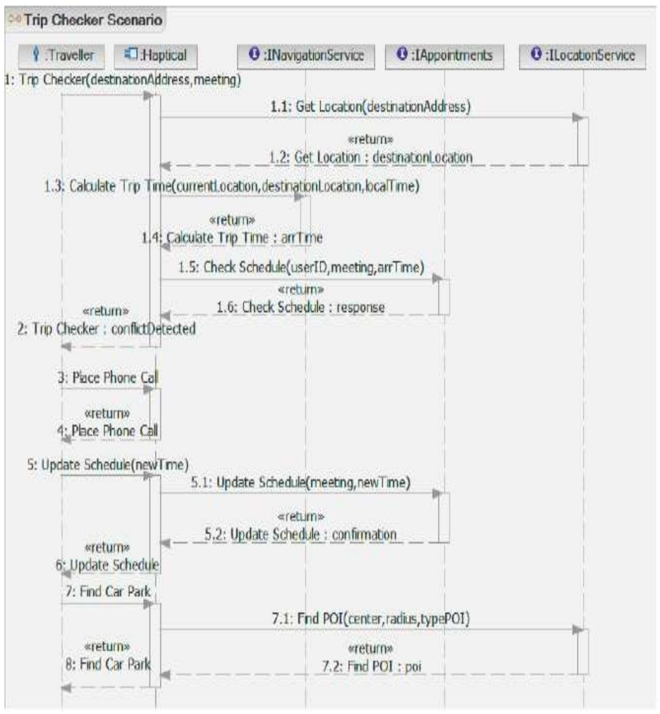

Trip Checker Scenario
:Traveller
:Haptical
:INavigationService
:IAppointments
:ILocationService
1: Trip Checker(destinationAddress,meeting)
1.1: Get Location(destinationAddress)
«return»
1.2: Get Location : destinationLocation
1.3: Calculate Trip Time(currentLocation,destinationLocation,localTime)
«return»
1.4: Calculate Trip Time : arrTime
1.5: Check Schedule(userID,meeting,arrTime)
«return»
1.6: Check Schedule : response
«return»
2: Trip Checker : conflictDetected
3: Place Phone Call
«return»
4: Place Phone Call
5: Update Schedule(newTime)
5.1: Update Schedule(meeting,newTime)
«return»
5.2: Update Schedule : confirmation
«return»
6: Update Schedule
7: Find Car Park
7.1: Find POI(center,radius,typePOI)
«return»
8: Find Car Park
«return»
7.2: Find POI : poi

Fig. 1. A Sample Behavioral Model of a Service Centric System

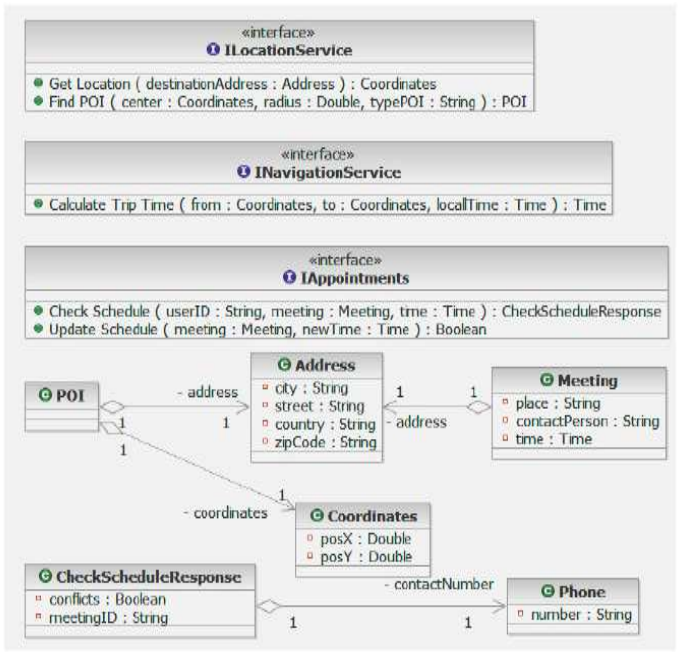


Fig. 2. *Structural Model*: Counter Part of the Behavioral Model in Fig. 1

further processing. In the second stage, the fit of the services located in the first stage with the services required by the query is assessed by (a) computing distances between the descriptions of the former and the latter services, and (b) selecting the set of the former services that has the minimum aggregate distance to the services required by the query.

The remainder of this paper is structured as follows. In Section 2, we introduce a scenario for DIRSD that we use subsequently to demonstrate our approach. In Section 3, we describe our DIRSD process and framework with its main components. In Section 4, we describe the query language used in our framework. In Section 5, we discuss the mechanisms for the query execution and distance functions underpinning it. In Section 6, we give an overview of the implementation of our framework. In Section 7, we present the results of an initial set of experiments that we have conducted to evaluate our framework. In Section 8, we give an account of related work. Finally, in Section 9, we summarize our approach and outline directions for future work.

II. APPROACH OF DESIGN IMPELLED AND RANKED SERVICE DETECTION

The sequence diagram in Fig. 1 specifies five operations, these operations are: Get Location, Calculate Trip Time, Check Schedule, Update Schedule, and Find POI.

The exact signatures and the types of the parameters of the operations in the diagram of Figure 1 are specified in the class diagram of Fig. 2.

A. Query Specification

An DIRSD query is specified by the system designer who selects an interaction I from SyBM, creates a copy of I called query interaction (I'), selects the messages in I' that should be realized by operations of the services to be discovered, and specifies various constraints on these operations (e.g. restrictions on the number of parameters) or on the interaction as a whole (e.g. service provider).

DIRSD query and its results are specified by using a UML 2.0 profile that we have developed (viz. DIRSD profile). 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 (e.g. cost, textual description). Figure 4 presents the part of the DIRSD profile used for specifying an DIRSD query (see Section 5 for DIRSD results). In this case, a UML package is stereotyped as an <<query_package>>.

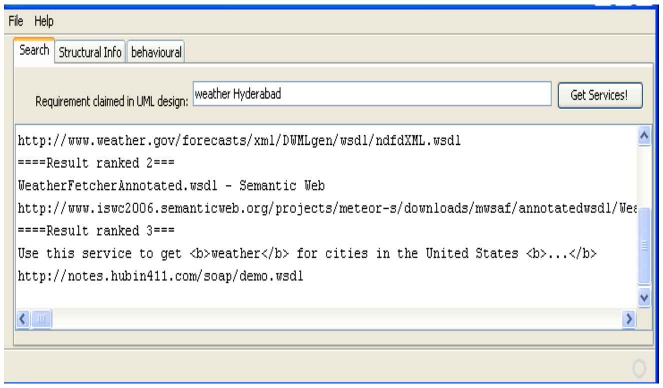
The messages of the interaction may be stereotyped as: (i) query messages <<query_message>> that indicate the service operations that should be discovered; (ii) context messages <<context_message>> that imply additional constraints for the query messages (e.g. if a context message has a parameter p1 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 <<bound_message>> that are bound to concrete operations that have been discovered by executing DIRSD 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 <<query_operation>>.

The DIRSD 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 <<query_package>> and <<query_message>> stereotypes can specify query parameters. Some of these parameters are inherited from the abstract stereotype <<query_element>>. The query parameters specified for query_package are global (i.e., applied to the whole query). The query parameters specified for query_message are local (i.e., applied to specific messages of the query interaction). The 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 <<constraint>> provide specific selection criteria for choosing services based on their various characteristics. These constraints can be applied for query_package, query_message, and 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 [6] 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 DIRSD queries which are based on them.

Following the specification of a query interaction, the tool generates a DIRSD 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.



Requirement claimed in UML design: weather Hyderabad

```
http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl
====Result ranked 2===
WeatherFetcherAnnotated.wsdl - Semantic Web
http://www.iswc2006.semanticweb.org/projects/meteor-s/downloads/mwsaf/annotatedwsdl/Wea
====Result ranked 3===
Use this service to get <b>weather</b> for cities in the United States <b>...</b>
http://notes.hubin411.com/soap/demo.wsdl
```

Fig. 3. Example of the Query Execution

B. Query Execution Engine

The DIRSD 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 a DIRSD query and the candidate service operations identified in the filtering stage is formulated as an instance of the assignment problem as proposed in [7]. More specifically, given the set of operations required by an DIRSD query Q, Oper(Q) and the set of service operations identified in the filtering stage, OperS(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 DVk is a set of k special vertices representing dummy operations (k = |Oper (Q)| − |OperS(Q)|). This formulation assumes, without loss of generality, that |Oper (Q)| > |OperS(Q)|. If this is not the case, k dummy vertices are added to V^Q where k = |OperS(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 $D(v^Qi,v^Sj)$ indicating the overall distance between v^Qi and v^Sj where $v^Qi \in V^Q$ and $v^Sj \in V^S$. This measure is computed as the weighted sum of a set of partial distance measures $df(v^Q i,v^Sj)$ quantifying the semantic differences between v^Qi and v^Sj with respect to each facet f in the descriptions of v^Qi and v^Sj according to the following distance function (the weights are assumed to be normalized):

$$D(F,v^Q_i,v^S_j) = \Sigma_{f\in F}\, w_f d_f(v^Q_i,v^S_j) \text{ if } v^Q_i \in Oper(Q),\ v^S_j \in Oper_S(Q)$$

$$D(F,v^Q_i,v^S_j) = 1 \qquad \text{if } v_i \in DV_k$$

$$D(F,v^Q_i,v^S_j) = \infty$$

if v_i should not be mapped onto v^Sj 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 OperS(Q). In the case of comparisons between an existing operation and a dummy operation D's value is defined to be 1. This favors 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 morphs between V^Q and V^S (or onto morphs if $|Oper(Q)| < |OperS(Q)|$)) and minimizes the function $\Sigma(v^Qi,v^Sj) \in O(V^Q ,V^S)$ $D(F,v^Qi, v^Sj)$ is selected. $O(V^Q,V^S)$ is selected using standard algorithms for the assignment problem [7]. In the second step, $O(V^Q,V^S)$ is restricted to include only the edges whose distance $D(F,v^Qi, v^Sj)$ does not exceed a threshold value Dt. Partial distance functions. For the facets corresponding to soft constraints defined as Boolean tests, the partial distances df 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, dL is a linguistic distance built on top of WordNet lexicon [5] and dPS 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, dPS is computed by finding the best possible morphism pm between the elements of these sets, defined as:

$$d_{PS}(P\hat{1},P\hat{2}) = \hat{m}in_{pm} \ (\Sigma_{(x,y) \ \in \ pm} \quad d_P(x,y)),$$

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 [7].

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.

- *Ranking:*

However, service consumers are interested in not only the functionalities of web services, but also their quality of service (QoS) which are non-functional attributes (e.g. response time, availability etc.) that may have impact on the quality of service provided by Web services. If there are multiple web services providing the same functionality in detected results of the proposed framework the QoS ranking specified can be used to refine the search? If the QoS claims made by service providers are trustworthy, the service selection is simple, either the service with lowest response time and highest availability is selected. But the problem is that the services provider may publish inaccurate QoS information to attract more customers, or the published QoS information may be out of date. To resolve this problem, it should be allowed to rate the QoS of the web service selected by the consumers and the aggregation of these service ratings over a specific period of time should be taken into consideration in ranking process of web services so that the probability of finding the best service for a customer can be increased.

- *Approach:*

Web service provider provides the QoS initially at the time of service registration. Service consumers rate the services after its usage. The web-service ranking will be the aggregate of initial QoS information provided by web service provider stored in XML format and the feedback ratings by the service consumers. The initial QoS information and feedback ratings can be averaged to derive the ranking for the web service to be published. Using this QoS ranking selection of appropriate web service in business-to-business interactions can greatly benefit.

- *QoS Information*

Quality of Service, or QoS, is “a combination of several qualities or properties of a service”. It is a set of nonfunctional attributes that may influence the quality of the service provided by a Web service. Some QoS parameters are given below: Availability is the probability that system is up and can respond to consumer requests. Reliability is the ability of a service to perform its required functions under stated condition for a specific period of time. Performance is the measure of the speed to complete a service request. It is measured by latency, throughput and response time. Cost is the measure of the cost of requesting a service.

The process of detecting web services based on the framework discussed in related work will be initiated, if services are found to match both the functional and QoS requirements and ratings requirements have also been specified, then the Web Service Broker ranks the services based on consumer’s QoS and ratings requirements. Service consumer then selects the web service with the highest rank.

The ranking of the web service will be done based on QOS information available, usage frequency and communal relation.

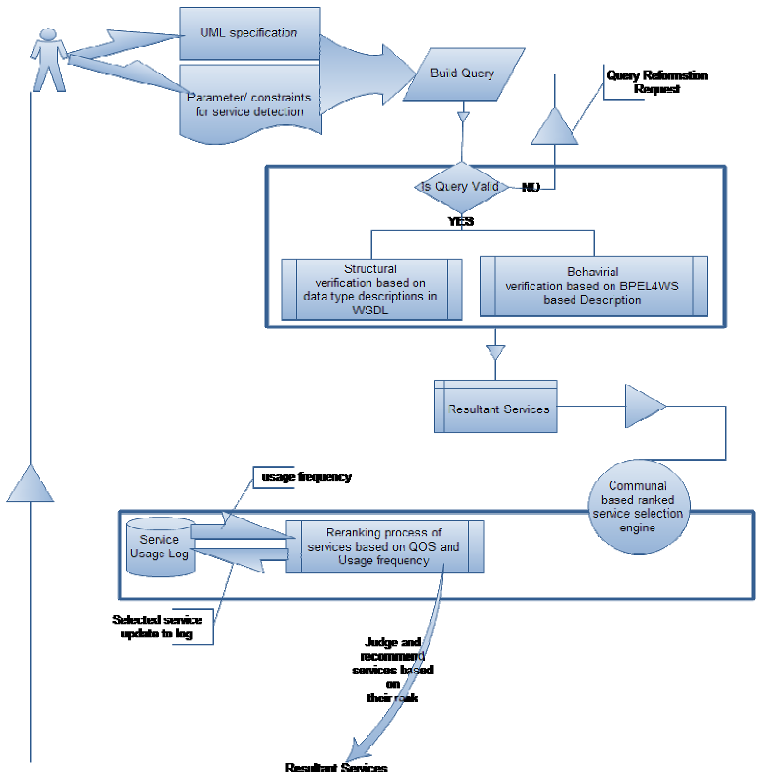

UML specification
Parameter/ constraints for service detection
Build Query
Query Reformstion Request
Is Query Valid
NO
YES
Structural verification based on data type descriptions in WSDL
Behavirial verification based on BPEL4WS based Description
Resultant Services
usage frequency
Communal based ranked service selection engine
Service Usage Log
Reranking process of services based on QOS and Usage frequency
Selected service update to log
Judge and recommend services based on their rank
Resultant Services

Fig. 4. DIRSD-Block Diagram

III. PROPOSED ALGORITHM FOR WEB SERVICE MATCHING, RANKING AND SELECTION

Block diagram (Fig. 4) shows the high level algorithm for service matching, ranking and selection:

- funMatch returns a set of services meeting the functional requirements.
- qosMatch returns the services that meet the QoS requirements.
- ratingMatch returns the set services whose ratings are equal and above than rating requirements, from those returned by method qosMatch and sort them in descending order.
- qosRank computes the QoS ranks for those services returned by method qosMatch and sort them in descending order of QoS rank.
- selectService returns a set of services depending upon the maximum number of services to be returned in response to the discovery request.

- *DIRSD Results:*

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

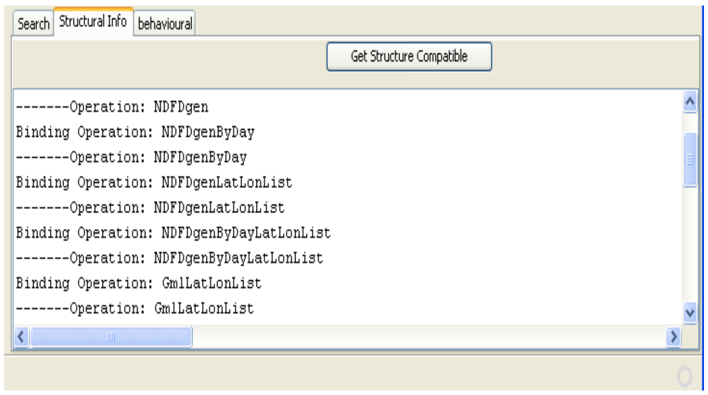
The results_package contains a refinement of the query interaction used by the designer to create the query together with the structural model for the elements in the interaction, and a number of UML packages stereotyped as <<service_package>>, one for each candidate service identified by the query execution engine. Each 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 results_package contains copies of all data types from the query_package together with the mapping (stereotyped as <<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 service_package may be stereotyped as (i) bound operations <<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 <<candidate_operation>> that reflect another possible result for the query message, but not necessarily the best match; and (iii) service operations; <<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 service_package) stereotyped as either <<bound_service>> or <<candidate_services>>, depending on the existence of any bound operations.

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

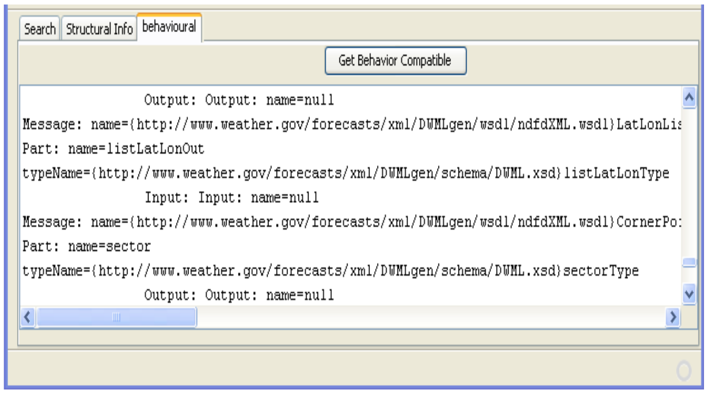
```
/* Algoritm for web service matching, ranking and selection */
discoverServices(funReq, qosReq, ratingReq, maxServices)
{
//discover services meeting the functional requirements
funMatches = funMatch(funReq);
if QoS requirements specified
qosMatches = qosMatch(funMatches, qosReq);
else
return selectService(funMatches, maxServices, "byRandom");
if Rating requirements specified
Matches = ratingMatch(qosMatches, qosReq, ratingReq);
return selectService(Matches, maxServices, "byQoS&Rating");
else
Matches = qosRank(qosMatches, qosReq);
return selectService(qosMatches, maxServices, "byQoS");
}
```

Fig.5. High Level Algorithm for Service Matching, Ranking and Selection



```
-------Operation: NDFDgen
Binding Operation: NDFDgenByDay
-------Operation: NDFDgenByDay
Binding Operation: NDFDgenLatLonList
-------Operation: NDFDgenLatLonList
Binding Operation: NDFDgenByDayLatLonList
-------Operation: NDFDgenByDayLatLonList
Binding Operation: GmlLatLonList
-------Operation: GmlLatLonList
```

Fig. 6. *DIRSD Results:* Structural Information



```
                Output: Output: name=null
Message: name={http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl}LatLonLis
Part: name=listLatLonOut
typeName={http://www.weather.gov/forecasts/xml/DWMLgen/schema/DWML.xsd}listLatLonType
                Input: Input: name=null
Message: name={http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl}CornerPo:
Part: name=sector
typeName={http://www.weather.gov/forecasts/xml/DWMLgen/schema/DWML.xsd}sectorType
                Output: Output: name=null
```

Fig. 7. *DIRSD Results:* behavioral Information

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 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 results_package as a basis for a new iteration of the DIRSD process.

Table 1: Results of query execution for message GetLocation in TripChecker Query

Provider	Service	Operation	Distance	Rank
ViaMichelin	GeocodingService	getLocationsList()	0.11374	3
FIAT	WNavigation	getPosition()	0.14051	2
FIAT	YNavigation	getPosition()	0.15241	1
ViaMichelin	ReverseGeocodingService	getLocationsList()	0.15941	4

Table 2: Results of query execution for message weather

www.weather.gov	ndfdXMLPort	NDFDgenLatLonList	0.138	1
www.iswc2006.semanticweb.org	WeatherPort	LatLonListCityNames	0.152	2
notes.hubin411.com	GetWeather	getWeather	0.145	3

IV. CONCLUSION

We presented a framework to support design impelled and ranked services detection that is integrated with iterative UML-based system engineering design processes. Our framework addresses the varous challenges that common vulanrable in service detection, in particular allowing service detection to be impelled by design 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 behavioral aspects that ranked based on QOS.

REFERENCES

[1] Chammabasavaiah K., Holley K., and Tuggle E.M. (2003), Migrating to a Service-Oriented Architecture. Retrieved May 22, 2006, from http://www28.ibm.com/developerworks/webservices/library/ws migratesoa/.

[2] Kramler G., Kapsammer E., Kappel G., and Retschitzegger W. (2005). Towards Using UML 2 for Modelling Web Service Collaboration Protocols, 1st Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA '05), Switzerland, February 21-25.

[3] Papazoglou, M. (2003). Service-Oriented Computing: Concepts, Characteristics and Directions, Keynote for 4th International Conference on Web Information Systems Engineering, December 10-12.

[4] Jones S., Kozlenkov A., Mahbub K., Maiden N., Spanoudakis G., Zachos K., Zhu X., and Zisman A. (2005). Service Discovery for Service Centric Systems, eChallenges 2005, Slovenia, October 19-21.

[5] M. Deubler, M. Meisinger, and I. Kruger, “Modelling Crosscutting Services with UML Sequence Diagrams,” Proc. ACM/IEEE Eighth Int’l Conf. Model Driven Eng. Languages and Systems, 2005.

[6] eXist, exist.sourceforge.net, 2009.

[7] J. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, “Web Service Discovery Mechanisms: Looking for a Needle in a Haystack,” Proc. Int’l Workshop Web Eng., Hypermedia Development and Web Eng. Principles and Techniques: Put Them in Use, 2004.

Shaik Shah Nawaz has received his M.Tech degree in Computer Science. Currently, he is working as Senior Associate Professor in the Department of Computer Science & Engineering, Aurora Engineering College, Bhongir, Nalgonda, A.P, India. He has 18-years working Experience in Software Industry. His areas of interests are Software Engineering, Testing, Network Security, Computer Networks, Wireless Communications, Data Mining and Data Warehousing.

Arram Sriram is pursuing his M.Tech in Software Engineering (Dept. of CSE) in Aurora Engineering College, Bhongir, Nalgonda, A.P, India. His areas of interests are data mining and knowledge Discovery, Software Engineering, software project management, Testing, Network Security, unified modeling language, and Mobile computing.