A QUERY LANGUAGE FOR SERVICE DISCOVERY

Andrea Zisman, George Spanoudakis
Department of Computing, City University London, Northampton Square, London EC1V 0HB, U.K.

James Dooley
School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, U.K.

Keywords: Service Discovery, Structural, Behavioural, Constraints.

Abstract: To support the discovery of services during development and execution time of service-based systems, it is necessary to have ways of expressing the characteristics of the services to be discovered and the applications that will use them. In this paper, we present SerDiQueL, an XML-based query language that allows for the description of service discovery queries expressing structural, behavioural, quality, and contextual characteristics of services to be discovered. The language supports the identification of services during both development and execution of service-based systems and is supported by prototyped query processors performing similarity analysis between services and queries.

1 INTRODUCTION

Service discovery is an important activity for service oriented computing. Over the last few years various approaches (Hausmann et al., 2004) (Li and Horrock, 2003) (Shen and Su, 2005) have been proposed to support identification of services that can fulfill the functionality and quality aspects of service-based systems during the design, deployment, and use of these systems. Some of these approaches support the identification of services during the development of service-based systems (viz. static service discovery), while other approaches allow for the identification of services during the execution time of these systems (viz. dynamic service discovery).

The need for static service discovery arises when it is necessary to develop systems that are constructed as composition of services available in different service registries. The need for dynamic service discovery arises when it is necessary to identify services during execution time of these systems due to unavailability or malfunctioning of services participating in the systems; changes in the context, structure, behaviour, and quality of these services; changes in the context of the system; or availability of new services.

In any of the above situations, it is necessary to have ways of expressing the characteristics of the services to be discovered. More specifically, for static service discovery, it is necessary to have a query language that allows the representation of structural, behavioural, and quality aspects of a service-based system being developed in order to identify services that match these aspects and use these services in the design of the system. For dynamic service discovery, it is necessary to have a query language that can express not only structural, behavioural, and quality aspects, but also contextual aspects of deployed services that need to be replaced in service-based systems and contextual aspects of the environment where a system is executed.

For an example of static service discovery, consider the development of a service-based system that assists users to plan a trip. In this case, it may be necessary to identify services that allow to check for availability of flights, calculate the cost of a flight, get the details of a flight, and book a flight, in this exact order, with some specific parameters and a restriction that the time to execute those actions by the service should not be more than two seconds. For an example of dynamic service discovery, consider a service-based system that schedules journalists' daily tasks. Suppose this application has a service that identifies the whereabouts of

Zisman A., Spanoudakis G. and Dooley J. (2009).
A QUERY LANGUAGE FOR SERVICE DISCOVERY.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 55-65
DOI: 10.5220/0002260400550065
Copyright © SciTePress

journalists. Assume that this service fails. In this case, it may be necessary to discover services that can identify journalists at a certain location.

In this paper we present SerDiQueL, an XML-based service discovery query language allowing the description of service requests expressing structural, behavioural, quality, and contextual characteristics of the systems and services to be identified during both static and dynamic service discovery. SerDiQueL supports queries that can be performed in both push and pull modes during the execution of service-based systems.

SerDiQueL is composed of three sub-queries. The first sub-query describes structural characteristics of the services to be discovered that comply with the application being developed, or services already being used by a system. The second sub-query expresses behavioural characteristics of services and systems representing the existence of a certain functionality or sequence of functionalities, the sequence and order in which certain functionalities should be executed, pre-conditions, and dependencies between functionalities. The third sub-query represents extra constraints. These extra constraints may involve (i) quality characteristics, (ii) contextual characteristics, or (iii) extra structural and behavioural conditions that cannot be represented in the structural and behavioural sub-queries of the language. Examples of these extra constraints are the time or cost to execute a certain operation in a service, the specific receiver of a message, the provider of a service, or the number of parameters in a service operation.

SerDiQueL has been developed as part of a large programme of research to support service discovery in static (Kozlenkov et al., 2007) and dynamic pull/push modes (Zisman et al., 2008). The language supports the representation of service queries that are executed by a service discovery framework. In this paper, we discuss the language and give a brief overview of the framework. In our work, we assume services specified from different perspectives such as interface (WSDL (WSDL)), behavioural (BPEL4WS (BPEL4WS)), quality, context, and textual descriptions in XML format. The similarities between service queries and service specifications are computed by using distance functions.

The remainder of this paper is structured as follows. Section 2 presents an overview of our framework to support static and dynamic service discovery. Section 3 describes SerDiQueL and gives example queries to illustrate it. Section 4 presents related work. Finally, Section 5, provides an overall discussion, concluding remarks, and future work.

2 OVERVIEW OF SERVICE DISCOVERY FRAMEWORK

Our service discovery framework has been developed to support both static (Kozlenkov et al., 2007) and dynamic identification of services (Zisman et al., 2008). In the case of static service discovery, our framework advocates an iterative process in which service-based systems are developed based on available services. More specifically, the framework supports the identification of services that provide functional and non-functional characteristics of service-based systems during the design phase of these systems. The identified services are used to formulate and amend the design models of these systems. The reformulations of the design models may trigger new service discovery iterations. In the case of dynamic service discovery, our framework advocates identification of services in both reactive and proactive ways, based on subscribed service requests, to replace services in a service-based system during execution time.

In both cases, the framework assumes services described from different perspectives by a set of XML-based facets. These facets include (i) textual facets describing general information of the services in an XML format, (ii) structural facets describing operations of services with their data types using WSDL (WSDL), (iii) behavioural facets describing behavioural models of services in BPEL4WS (BPEL4WS), (iv) quality of service facets describing non-functional aspects of services, and (v) context facets describing quality aspects of a service that change dynamically.

Figure 1 shows the overall architecture of our service discovery framework. As shown in the figure, the main components of the framework are: (a) service requestor, (b) query processor, and (c) service registry intermediary. The framework also uses external service registries and, is invoked by an external client application, and uses special servers and listeners to support notification of changes in services and application environment.

The external client application supports the creation of service requests to be executed both statically and dynamically. These service requests may contain structural, behavioural, quality, and contextual characteristics. The service requestor receives a service request from the client application, as well as context information about the services and application environment in the case of dynamic service discovery. It prepares service queries to be evaluated, organises the results of a query, and returns these results to the client application. In addition, it manages push query execution mode subscriptions, receives information from listeners about services that become available or about changes to existing services, in the case of dynamic service discovery. The query processor is responsible to parse the different parts of a query and evaluate these parts against service specifications in the various service registries. As shown in the figure, the query processor is formed by three sub-components, namely (i) structural, (ii) behavioural, and (iii) constraint matchmakers. Each of these sub-components is responsible to evaluate a different part of a query (see Section 3).

Our framework assumes that constraints in a query can be contextual or non-contextual. A contextual constraint is concerned with information that changes dynamically during the operation of the service-based system and/or the services that the system deploys, while a non-contextual constraint is concerned with static information related to structural, behavioural, and quality aspects of the services and systems. The non-contextual constraints can be hard or soft. A hard constraint must be satisfied by all discovered services for a query and are used to filter services that do not comply with them. A soft constraint does not need to be satisfied by all discovered services, but are used to rank candidate services for a query.

The evaluation of a query is executed in a two-stage process. In the first stage, the constraint matchmaker is invoked in order to evaluate hard constraints in the query. This is a *filtering* stage, which selects candidate services from the registries that match exactly the hard constraints of a query. The second stage is an *optimization* stage in which the candidate services returned from the first stage are matched against structural, behavioural, and soft constraints in a query. When a query does not have hard constraints, the structural, behavioural, and soft constraints parts of a query are matched against all the services in the registries. The above matching is executed by the structural, behavioural, and constraint matchmakers, respectively and is based on the computation of structural, behavioural, and constraint distances.

In the case of static service discovery, the matching in the second stage of the process returns *n*-best services for a query *(n* can be specified in a query or can be equal to ten by default). The identified best services are used by the designer of a service-based system under development to select the services to be used by the system. Details of the static discovery process and distance measures are beyond the scope of this paper, but can be found in (Kozlenkov et al., 2007) (Zisman and Spanoudakis, 2006).

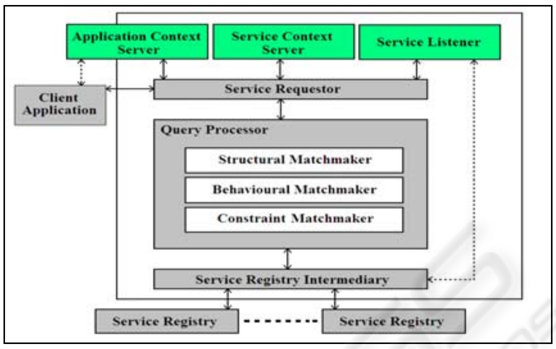

Figure 1: Architecture overview of framework.

In the case of dynamic service discovery, the matching in the second stage of the process returns the best service for a query also based on the computation of structural, behavioural, and constraint distances as described in (Zisman et al., 2008). The dynamic service discovery supports both pull and push modes of query execution. For the push mode of query execution, the framework assumes a proactive approach in which services are identified in parallel to the execution of a service-based system based on subscriptions of application environment, services, and queries associated with these services, so that replacement services can be identified, when notification of changes in services and application environments are pushed to listeners (Zisman et al., 2008). These notifications of changes are supported by the use of service context server, application context server, and service listeners.

The *service registry intermediary* supports the use of different service registries and the discovery of services stored in different types of registries. It provides an interface to access services from various registries. The framework allows accessing services from registries organized as faceted structure, as proposed in the SeCSE project (SECSE). More specifically, in the registries, a service is specified by a set of XML-based facets, namely (i) textual facets, (ii) structural, (iii) behavioural, (iv) quality of service, and (v) context facets.

3 SerDiQueL

SerDiQueL is an XML-based language that allows for the specification of structural, behavioural, quality, and contextual characteristics of services to be discovered or service-based systems being developed. The language has been developed based on requirements for supporting both static and dynamic service discovery identified by industrial partners in the areas of telecommunications, automotive, software, media, and banking in the European research projects (GREDIA) (SECSE), and the framework described in Section 2. These requirements point out the need for:

- Generating service discovery queries from design models of service-based systems specifying functional and non-functional properties of such systems;
- Generating service discovery queries from characteristics of services that have already been deployed in systems, but may need to be replaced;
- Providing service discovery queries that express arbitrary logical combinations of prioritised functional, non-functional, and contextual properties of required services, and similarity-based queries of the form "find a service that is similar to service X";
- Efficient matching of service discovery queries against different types of service specifications and return of services with varying degrees of match with the queries;
- Supporting service discovery in both pull and push execution mode (based on subscriptions);
- Proactive dynamic service discovery to increase efficiency especially at runtime;
- Integrating discovered services into an iterative design process in which service-based systems design models may be re-formulated after the discovery of services.

Figure 2 presents the overall XML schema of SerDiQueL. As shown in the figure, a query specified in the language (ServiceQuery) has three elements representing structural, behavioural, and constraint sub-queries. The division of a query into these three sub-queries is to (i) allow the representation of these three types of information and (ii) support the representation of queries with arbitrary combinations of these types of information. A ServiceQuery also has a unique identifier, a name, and one or more elements describing different parameters for a query. A parameter element is defined by a name and a value. Examples of parameters that can be used in a query are: (a) name of the query, (b) type of the query (e.g., static or dynamic), (c) mode of execution (push or pull), (d) author of the query, and (e) number of n-best services to be returned by a query.

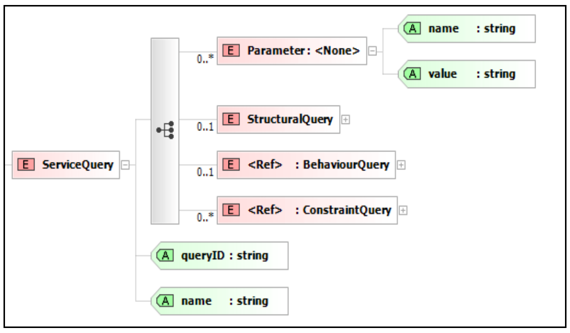

Figure 2: Overview schema of SerDiQueL.

3.1 Structural Sub-query

The structural sub-query describes structural aspects of (i) a service-based system being developed (in the case of static service discovery) or (ii) a service that needs to be replaced and is being used by a service-based system (in the case of dynamic service discovery).

The description of structural aspects for case (i) is based on design models of this system. Our service discovery framework assumes design models expressed in UML class and sequence diagrams represented as XMI documents, due to the popularity of using UML for designing software systems in general, and service-based system in particular (Deuble et al., 2005) (Gardner, 2004). However, the structural sub-query could be based on other types of design models representing the functionality of a system.

In order to support the definition of structural aspects of a system under development based on UML models, we have developed a UML 2.0 profile (Kozlenkov et al., 2007). The profile defines a set of stereotypes for different types of UML elements such as messages in sequence diagrams, or operations and classes defining the types of arguments in the messages in the class diagrams.

For example, messages in a sequence diagram may be stereotyped as: (i) *query messages* representing service operations needed in identified services; (ii) *context messages* representing 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); (iii) *bound messages* representing concrete service operations that have been discovered in previous query executions.

In our framework, structural sub-queries for a service-based system being developed are automatic generated from the class and sequence diagrams of a

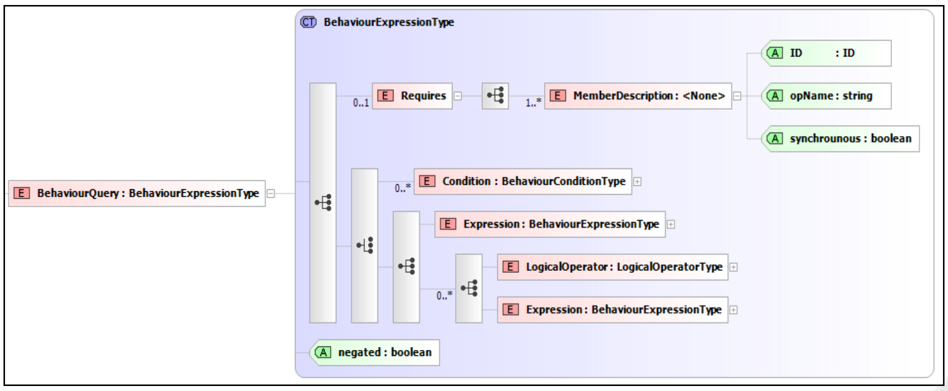
BehaviourExpressionType
ID : ID
opName : string
synchrounous : boolean
0..1
Requires
1..*
MemberDescription : <None>
BehaviourQuery : BehaviourExpressionType
0..*
Condition : BehaviourConditionType
Expression : BehaviourExpressionType
LogicalOperator : LogicalOperatorType
0..*
Expression : BehaviourExpressionType
negated : boolean

Figure 3: XML Schema for behavioural sub-query.

system based on the selection of messages from the designer of the system. The description of structural aspects of a service-based system based on design models of these systems supports the representation of operations being searched in different services together with the representation of the input and output parameters of these operations and their respective data types. This is important to assist with the matching of structural aspects of the systems with structural aspects (interface descriptions) of available services. Moreover, it supports the development of a service-based system based on the characteristics of available services instead of on requirements that may never be able to be fulfilled by existing services.

The description of structural aspects for case (ii) is represented as the WSDL (WSDL) specification of the service to be replaced. In this case, SerDiQueL supports a complete representation of the structural aspects of a service to be identified as interface descriptions. In the framework, structural sub-queries for a service that needs to be replaced during execution time are automatic generated based on the notification that a service became malfunctioning, unavailable, or there have been changes in the characteristics of the service or in the context of the application environment. In both static and dynamic service discovery, structural sub-queries are matched against interface descriptions of services specified as WSDL considering the names of the operations and the data types of the parameters of the operations. Details of the matching process is found in (Zisman et al., 2008).

3.2 Behavioural Sub-query

The behavioural sub-query is matched against behavioural service specifications. In our framework, the matching of the behavioural sub-queries is executed based on the comparisons of paths representing the query and service specification. More specifically, the behaviour matching is executed by (i) transforming behavioural service specifications into state machines, (ii) extracting all the possible paths from a service statemachine, (iii) transforming the behavioural sub-query into paths, and (iv) verifying if the paths representing the query can be matched against a path of the statemachine of a service. Given its popularity, we assume behavioural service specifications represented in BPEL4WS (BPEL4WS). However, the behaviour sub-query can be matched against other types of behavioural service specifications that can be represented as or transformed into statemachines.

The behavioural sub-query is based on temporal logic supporting the representation of behavioural aspects of required services. In particular, it supports the description of queries that verify (a) the existence of a certain functionality, or a sequence of functionalities, in a service specification; (b) the order in which certain functionalities should be executed by a service; (c) dependencies between functionalities; (d) pre-conditions; and (e) loops. Figure 3 shows a graphical representation of the SerDiQueL's XML schema for behavioural sub-queries. As shown in the figure, a behavioural sub-query is defined as (a) a single condition, a negated condition, or a conjunction of conditions, or (b) a sequence of expressions separated by logical operators. A behavioural sub-query also allows for the specification of requires elements.

Requires elements define one or more service operations that need to exist in service specifications, represented as members (element MemberDescription). These member elements are used in various conditions and expressions of a query. The existence of requires elements in service specifications are verified as an initial step during

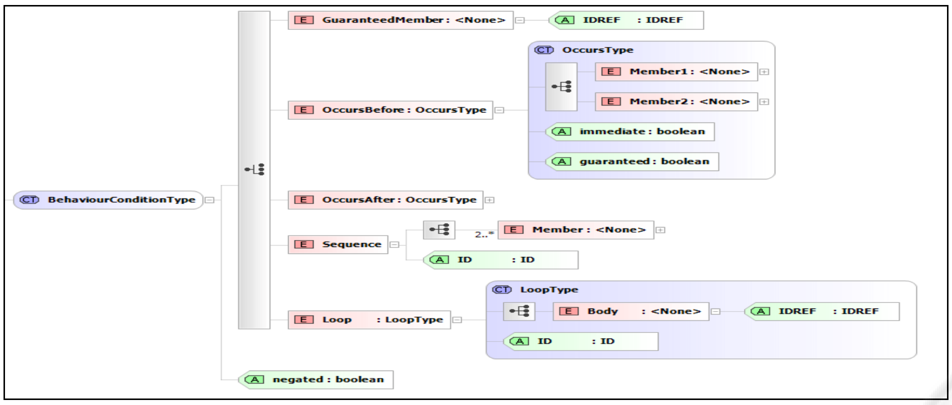

Figure 4: XML Schema for behavioral Sub-query.

the execution of a sub-query, in a fail-fast processing mode, instead of being verified during the analysis of the conditions and expressions that use these elements, providing optimization for the sub-query execution process. A member element has three attributes, namely (a) ID, indicating a unique identifier for the member within a query; (b) opName, specifying the name of an operation described in the structural sub-query, (for the case of dynamic service discovery, this attribute may also contain the port type for this operation for the WSDL description in the structural sub-query); and (c) synchronous, a boolean attribute indicating if the service operation needs to be executed in a synchronous or asynchronous mode in the service.

A condition can be negated and is defined as *GuaranteedMember*, *OccursBefore*, *OccursAfter*, *Sequence*, or *Loop* elements, as shown in Figure 4. A *GuaranteedMember* represents a member element (i.e., service operation) that needs to occur in all possible traces of the execution in a service. This element is defined by attribute IDREF that references requires, sequence, or loop elements. The *OccursBefore* and *OccursAfter* elements represent the order of occurrence of two member elements (*Member1* and *Member2*). They have two boolean attributes, namely (a) *immediate*, specifying if the two members occur in sequence or if there can be other member elements in between them, and (b) *guaranteed*, specifying if the two members need to occur in all possible traces of execution in a service. A *Sequence* element defines two or more members which must occur in a service in the order represented in the sequence. It has an identifier attribute that can be used by the guaranteed member element. A *Loop* element specifies a sequence of members that are executed several times. It has a unique identifier and is defined as a statement that references other elements.

Expressions are defined as sequences of requires elements, conjunctions of conditions, or other nested expressions connected by logical operators AND and OR. The definition of requires elements within an expression (E1) supports the cases where the non-existence of requires elements in a service does not invalidate the selection of this service, if other expressions in the sub-query that are disjointed with the expression are matched against the service.

```
<tnsb:BehaviourQuery>
<tnsb:Requires>
<tnsb:MemberDescription ID="login"
    opName="locS.login" synchrounous="true" />
<tnsb:MemberDescription ID="getLocation"
    opName="locS.retrieveJournalistLocation"
    synchrounous="true"/>
<tnsb:MemberDescription ID="getAllJournalists"
      opName="locS.getJournalistList"
      synchrounous="true" />
<tnsb:MemberDescription ID="select"
     opName="locS.selectJournalist"
      synchrounous="true" /> </tnsb:Requires>
<tnsb:Expression>
 <tnsb:Condition> <tnsb:GuaranteedMember
IDREF="login"/>
 </tnsb:Condition></tnsb:Expression>
<tnsb:LogicalOperator
operator="AND"/><tnsb:Expression> <tnsb:Condition>
 <tnsb:Sequence ID="getJournalistLocation">
    <tnsb:Member IDREF="getAllJournalists" />
    <tnsb:Member IDREF="select" />
     <tnsb:MemberIDREF="getLocation"/>
 </tnsb:Sequence></tnsb:Condition>
<tnsb:Condition>
 <tnsb:OccursBefore immediate'"false"
                    guaranteed="false">
    <tnsb:Member1 IDREF="login" />
    <tnsb:Member2 IDREF="getAllJournalists" />
 </tnsb:OccursBefore>
</tnsb:Condition> </tnsb:Expression>
</tnsb:BehaviourQuery>
```

Figure 5: Example of behavioural sub-query.

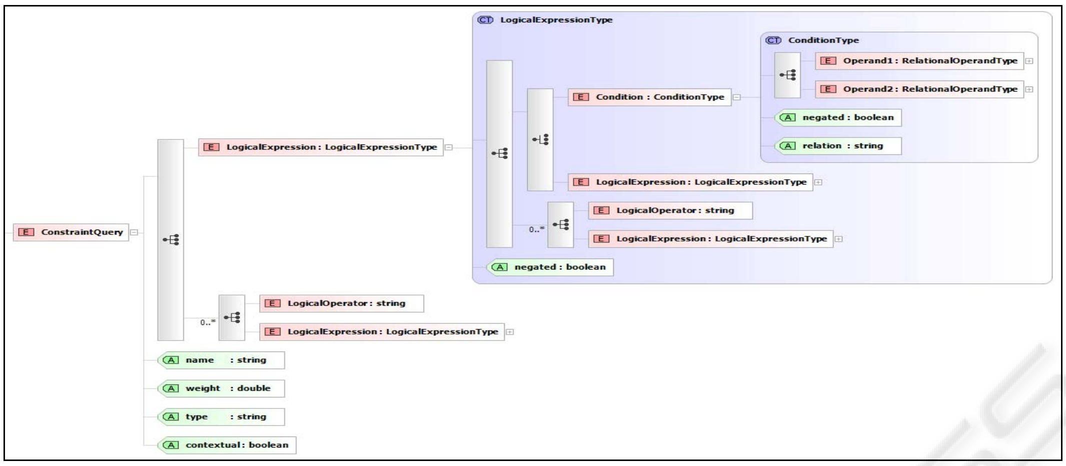
LogicalExpressionType
ConditionType
Operand1 : RelationalOperandType
Operand2 : RelationalOperandType
Condition : ConditionType
negated : boolean
relation : string
LogicalExpression : LogicalExpressionType
LogicalExpression : LogicalExpressionType
LogicalOperator : string
0..*
LogicalExpression : LogicalExpressionType
negated : boolean
ConstraintQuery
LogicalOperator : string
0..*
LogicalExpression : LogicalExpressionType
name : string
weight : double
type : string
contextual : boolean

Figure 6: XML schema for constraint sub-query.

As an illustration consider the example described in Section 1 for the service-based system that schedules journalists' tasks and assume that service S1 which identifies the location of journalists fails. In this case, consider a query for identifying a service to replace S1 with the following behavioural conditions: (i) the service needs to have operations similar to operations *login*, *getJournalistList*, *selectJournalist*, and *retrieveJournalistLocation; (ii)* a user of the service always need to be authenticated; (iii) operations *getJournalistList*, *selectJournalist*, *and retrieveJournalistLocation* need to be executed in this order; and (iv) operation *login* should be executed before operation *getJournalistList*. Figure 5 shows the description of the behavioural sub-query in SerDiQueL. As shown in the figure, the above conditions are described by the use of the elements *Requires* (case i), *GuaranteedMember* (case ii), *Sequence* (case iii), and *OccursBefore* (case iv).

3.3 Constraints Sub-query

The constraint sub-query described different types of extra conditions that need to be fulfilled by a service.

These extra conditions may include (a) quality aspects, (b) contextual aspects, or even (c) extra structural and behavioral aspects of a service that cannot be represented by the structural and behavioural sub-queries.

As described in Section 2, a constraint can be classified as contextual or non-contextual. The non-contextual constraints in a sub-query can be evaluated against any type of service specification (facet) in the service registries. The contextual constraints are evaluated against *context facets.* These context facets are associated with services and describe context information of the operations in these services. Context information is specified as context operations that are executed at run-time. The framework assumes the existence of context services that provide context information. Details of the context constraint matching are described in (Spanoudakis et al., 2007).

Figure 6 shows a graphical representation of SerDiQueL's XML schema for specifying constraints. As shown in the figure, a constraint sub-query is defined as a single logical expression, a negated logical expression, or a conjunction or disjunction of two or more logical expressions, combined by logical operators.

A constraint sub-query has four attributes, namely (a) *name*, specifying a description of the constraint; (b) *type*, indicating whether the constraint is hard or soft; (c) *weight*, specifying a weight in the range of [0.0, 1.0]; and (d) *contextual*, a boolean attribute indicating whether the constraint is contextual or non-contextual. The weight is used to represent prioritisations of the parameters in a query for soft constraints. When the value of the contextual attribute is *true*, the query may contain *ContextOperand* elements. If the value is *false*, the query may contain *NonContextOperand*.

A logical expression is defined as a condition, or logical combination of conditions, over elements or attributes of service specifications (for non-contextual constraints) or over context aspects of service operations (for contextual constraints). A condition can be negated and is defined as a relational operation (*equalTo, notEqualTo, lessThan, greaterThan, lessThanEqualTo, greaterThanEqualTo, notEqualTo*) between two operands, which can be non-contextual, contextual, constants, or arithmetic expressions.

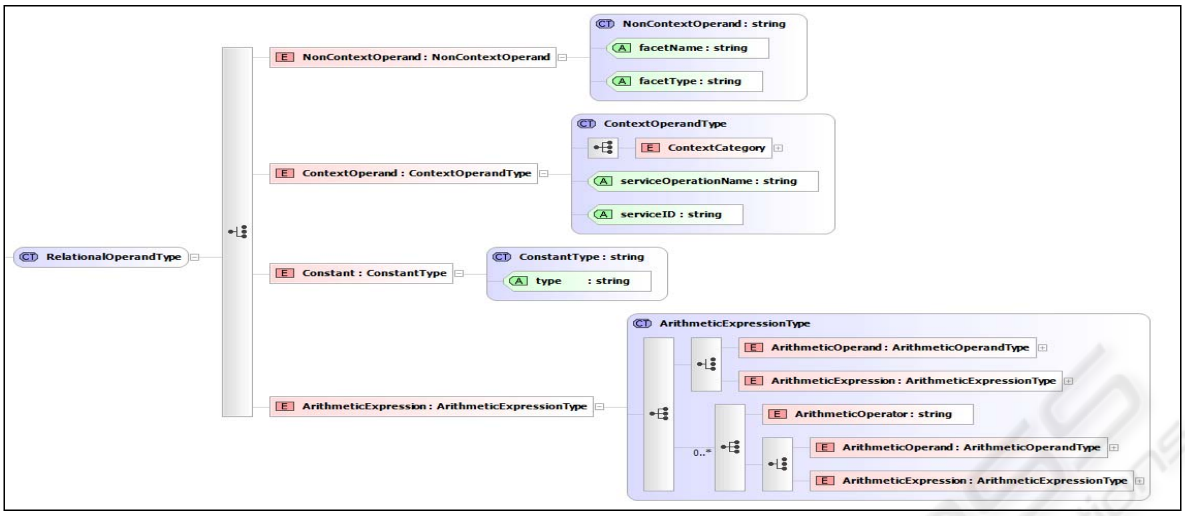
NonContextOperand : string
facetName : string
facetType : string
NonContextOperand : NonContextOperand
ContextOperandType
ContextCategory
serviceOperationName : string
serviceID : string
ContextOperand : ContextOperandType
RelationalOperandType
ConstantType : string
type : string
Constant : ConstantType
ArithmeticExpressionType
ArithmeticOperand : ArithmeticOperandType
ArithmeticExpression : ArithmeticExpressionType
ArithmeticOperator : string
0..*
ArithmeticOperand : ArithmeticOperandType
ArithmeticExpression : ArithmeticExpressionType
ArithmeticExpression : ArithmeticExpressionType

Figure 7: XML schema for constraint sub-query.

As shown in Figure 7, a non-contextal operand (element *NonContextOperand*) has two attributes, namely (a) *facetName*, specifying the name of the service specification and (b) *facetType*, specifying the type of the service specifications to which the constraint will be evaluated. The operand contains an XPath expression indicating elements and attributes in the service specification referenced in *facetName* attribute. Therefore, the constraints can be specified against any element or attribute of any facet in the registries.

A contextual operand (element *ContextOperand*) specifies operations that will provide context information at runtime. More specifically, a contextual operand describes the *semantic category* of context operations instead of the signature of the operation represented by sub-element *ContextCategory*. This is due to the fact that context operations may have different signatures across different services. A contextual operand is defined by (a) attribute *serviceOperationName*, specifying the name of the service operation associated with the contextual operand, and (b) attribute *serviceID*, specifying the identifier of a service that provides the operation. The value of attribute *serviceID* is specified when the context operand provides the specification of a context operation of a known service. This is normally the case when the context operation is associated with a service-based system for which the value of a context aspect of the system needs to be dynamically identified during the evaluation of a query (e.g., location of a mobile device application). In this case, attribute *serviceID* referes to the service-based system itself. Otherwise, the value of *serviceID* is specified as "any" (see Figure 7).

A *ContextCategory* element represents the semantic category of an operation, instead of its actual signature. It is defined as a relation between two categories (*Category1* and *Category2*). These categories can be either a reference to a document or a constant. A document category (element *Document*) has an attribute type indicating if the document is an ontology or a context facet, and contains an XPath expression referencing elements in the document. In the case of an ontology document, an attribute with the URL indicating the location of the ontology that describes the context operation is used. The language can support different ontologies for describing context operation categories since it does not make any assumption of the structure and meaning of the ontologies used, apart from the fact that the ontologies need to be described in XML. A context category in a query is evaluated against context facets of candidate services. This evaluation verifies if a candidate service has a context operation with semantic category that satisfies the categories specified in a query.

Arithmetic expressions define computations over the values of elements or attributes in service specification or context information. They are defined as a sequence of arithmetic operands or other nested arithmetic expressions connected by arithmetic operators. The arithmetic operators are: *addition*, *subtraction*, *multiplication*, and *division* operators. The operands can be contextual, non-contextual, constants, or functions.

```
<tnsa:ConstraintQuery name="C1" type="SOFT"
   contextual="false" weight="0.5">
 <tnsa:LogicalExpression>
  <tnsa:Condition relation="EQUAL-TO">
   <tnsa:Operand1>
    <tnsa:NonContextOperand facetName="OSem"
     facetType="Beh">
          //BPEL/process/sequence/*[1]/@operation
    </tnsa:NonContextOperand> </tnsa:Operand1>
   <tnsa:Operand2>
     <tnsa:Constant
type="String">login</tnsa:Constant>
     </tnsa:Operand2></tnsa:Condition>
    </tnsa:LogicalExpression>
</tnsa:LogicalExpression>
</tnsa:ConstraintQuery>
<tnsa:ConstraintQuery name="C2" contextual="true"
                       type="SOFT" weight="0.5">
  <tnsa:LogicalExpression>
     <tnsa:Condition relation="LESS-THAN">
     <tnsa:Operand1>
      <tnsa:ContextOperand
    serviceOperationName="retrieveJournalistLocatio
n"
    serviceID="any">
     <tnsa:ContextCategory relation="EQUAL-TO">
      <tnsa:Category1>
       <tnsa:Document

location="http://eg.org/CoDAMoS_Extended.xml"
        type="ONTOLOGY">string(/owl:Class/@rdf:ID)
       </tnsa:Document></tnsa:Category1>
     <tnsa:Category2> <tnsa:Constant type="STRING">
          GREDIA_RELATIVE_TIME</tnsa:Constant>
     </tnsa:Category2></tnsa:ContextCategory>
    </tnsa:ContextOperand></tnsa:Operand1>
  <tnsa:Operand2>
   <tnsa:Constant type="STRING">SECONDS-5
 </tnsa:Constant></tnsa:Operand2>
</tnsa:Condition></tnsa:LogicalExpression>
</tnsa:ConstraintQuery>
```

Figure 8: Example of SerDiQueL query.

A function supports the execution of a complex computation over a series of arguments. The results of these computations are numerical values that can be used as an operand in an arithmetic expression. A function has a name and a sequence of one or more arguments. Each of these arguments may be also a contextual or non-contextual operand, constant, or arithmetic expression.

In order to illustrate, consider service S1 that identifies the location of journalists in the example in Section 1. Assume two constraints for the service to be identified to replace S1, namely: (a) contextual constraint specifying that the time to retrieve the location of a journalist should not be more than 5 seconds, and (b) non-contextual constraint concerned with the fact that a user is authenticated before accessing other functionalities of the service.

Figure 8 shows these two constraints in SerDiQueL. Constraint C1 verifies if the first operation in the service's behavioural specification is the operation login (case (a)). Constraint C2 specifies that any candidate service that can identify the location of journalists (i.e., services that match operation retrieveJournalistLocation) needs to have a context operation classified in the category GREDIA_RELATIVE_TIME in ontology http://eg.org/CoDAMos_Extended.xml, and the result of executing this operation has to less than SECONDS-5 for this service to be accepted.

4 RELATED WORK

Several query languages have been proposed to support web services discovery (Beeri et al., 2006) (Pantazoglou et al., 2006) (Pantazoglou et al., 2007) (Papazoglou et al., 2002) (Yunyao et al., 2005). In (Beeri et al., 2006), the authors propose BP-QL a visual query language for business processes expressed in BPEL. SeDiQueL also supports querying BPEL specifications. However, it differs from BP-QL since it supports the specification of structural, quality, and contextual conditions in the query, and the behavioural conditions can be matched against other types of behavioural service specifications. The query language proposed in (Papazoglou et al., 2002) is used to support composition of services based on user's goals. NaLIX (Yunyao et al., 2005), a language developed to allow querying XML databases based on natural language, has also been adapted to cater for service discovery. In (Pantazoglou et al., 2006), the authors propose USQL (Unified Service Query language), an XML-based language to represent syntactic, semantic, and quality of service search criteria. SeDiQueL is more complete, since it accounts for the representation of behavioral aspects of the application being developed and services to be discovered, as well as context characteristics of services and application environments. An extension of USQL that incorporates behavioral based on UML sequence diagrams has been proposed in (Pantazoglou et al., 2007). The behavioural sub-query of SerDiQueL is not restricted only to the representation of sequence of operations.

Semantic matchmaking approaches have been proposed to support service discovery based on logic reasoning of terminological concept relations represented on ontologies (Hausmann et al., 2004) (Klein and Bernstein, 2004) (Klusch et al., 2006) (Li and Horrock, 2003). In (Hausmann et al., 2004), the discovery of services is addressed as a problem of matching requests specified as a variant of Description Logic (DL). The work in (Klein and Bernstein, 2004) extends existing approaches by supporting explicit and implicit semantics using logic based, approximate matching, and IR techniques. Our work differs from the above approaches as it supports explicit and separate representation of service requests based on various types of characteristics for both static and dynamic discovery.

In (Hall and Zisman, 2004) the authors advocate the use of behavioural models of services to increase the precision of the discovery process. Similarly, in (Shen and Su, 2005) the authors use service behaviour signatures to improve service discovery. These approaches do not support service discovery based on quality and contextual characteristics of the services.

Context awareness in service discovery has been proposed in (Bormann et al., 2005) (Choonhwa and Helal, 2003) (Doulkeridis et al., 2006). In (Doulkeridis et al., 2006), context information is represented by key-value pairs attached to the edges of a graph representing service classifications. This approach does not integrate context information with behavioural and quality matching and, since context information is stored explicitly in the service repository, this repository must be updated following context changes. The approach in (Bormann et al., 2005) uses ontologies to express service queries, service descriptions, and context information. Context information in this approach can also be used as an implicit input to a service.

Overall, most of the proposed approaches support service discovery for certain criteria whilst SerDiQueL allows the specification of comprehensive queries structural, functional, quality, and contextual characteristics of services and service-based applications at the same time.

5 CONCLUSIONS, DISCUSSION AND FUTURE WORK

In this paper we have described SerDiQueL, a query language for specifying service discovery queries that can be executed during the development and execution of service-based systems. SerDiQueL allows the representation of structural, behavioural, quality, and contextual characteristics of the services to be discovered and the service-based systems needing them.

A parser for SerDiQueL has been implemented in Java. We have used SerDiQueL in a service discovery framework that we have developed to support static and dynamic service discovery. The use of SerDiQueL in this framework has shown that the language can support the description of comprehensive queries expressing different characteristics of the services to be discovered.

In particular, the representation of the structural criteria in the query is flexible and can accommodate both the description of design models of the systems being developed and the services to be replaced in them. The behavioural sub-query supports the specification of different behavioural criteria of services and service-based systems and is not constrained by the use of proprietary languages (e.g., BPEL4WS). The language also supports behavioural sub-queries to be evaluated in a *fail-fast mode*, optimizing the matching of these sub-queries with service specifications. The behavioural sub-query does not support control structures and conditional loops, since these types of operators require knowledge of the state values of the services during query evaluation, which in general are not available. The constraint sub-query can specify a wide spectrum of complex conditions ranging from quality and contextual criteria, to extra structural and behavioural aspects that cannot be specified in the structural and behavioural sub-queries. The language supports the use of different ontologies for describing context operation categories and different types of context operations for which values can be extracted from a context server. In addition, the constraint sub-query allows for the description of any type of quality aspects that are described by service specifications.

We have used SerDiQueL to describe service discovery queries in both design and execution time of service-based systems in different applications in the telecommunication, automotive, software, media, and banking domains with positive results. More specifically, the use of SerDiQueL to support the development of service-based systems has shown an average precision of 91% for services with very low distance to queries (less than 0.1 in a scale between [0.0 and 1.0]), and a recall of 94% for services with distance of up to 0.5 with queries. Initial evaluation of the performance of matching queries specified in SerDiQueL with structural, behavioural, quality, and contextual constraints against service specifications of the types described in this paper, has shown average times of (i) 8,522 msec when matching 20 services; (ii) 18,153 msec for 40 services, and (iii) 28,754 msec for 60 services.

We are currently investigating the use of SerDiQueL in other service discovery frameworks and developing an editor to support the specification of SerDiQueL queries.

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 and GREDIA (contract FP6-34363).

REFERENCES

Bormann, F., et al., 2005. Towards Context-Aware Service Discovery: A Case Study for a new Advice of Charge Service. *In 14th IST Mob. & Wireless Comm.* Summit.

Beeri C., Eyal A., Kamenkovich S., Milo T., 2006. Querying Business Processes. *In 32nd International Conference on Very Large Databases, Korea.*

BPEL4WS. http://www128.ibm.com/developerworks/library/specification/ws-bpel/

Choonhwa L., Helal, S., 2003. Context Attributes: An Approach to Enable Context-awareness for Service Discovery. *In Symposium on Applications & the Internet.*

Deubler M., Meisinger M., Kruger I., 2005. Modelling Crosscutting Services with UML Sequence Diagrams. *In ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems.*

Doulkeridis, C., Loutas, N., Vazirgiannis, M., 2006. A System Architecture for Context-Aware Service Discovery. *In Electr. Notes Theor. Comp. Sci. 146(1): 101-116.*

Gardner T., 2004. UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. *In 2nd European Workshop on OO and Web Services (ecoop).*

GREDIA Project. http://www.gredai.eu

Hall, R.J., Zisman, A., 2004. Behavioral Models as Service Descriptions. *In International Conference on Service Oriented Computing, New York.*

Hausmann, J.R., Heckel, R., Lohman, M., 2004. Model-based Discovery of Web Services. *In Int. Conf. on Web Services.*

Klein, M., Bernstein, A., 2004. Toward High-Precision Service Retrieval. *In Internet Computing, 30-36.*

Klusch, M., Fries, B., Sycara, K., 2006. Automated Semantic Web Service Discovery with OWLS-MX. *In International Conference on Autonomous Agents and Multiagent Systems.*

Kozlenkov, A., Spanoudakis, G., Zisman, A., Fasoulas, V., Sanchez, F., 2007. Architecture-driven Service Discovery for Service Centric Systems. *In International Journal of Web Services Research, special issue on Service Engineering, 4(2), April-June.*

Li, L., Horrock, I., 2003. A Software Framework for Matchmaking based on Semantic Web Technology. *In Workshop on E-Services & the Semantic Web.*

Pantazoglou, M., Tsalgatidou, A., Athanasopoulos G., 2006. Discovering Web Services in JXTA Peer-to-Peer Services in a Unified Manner. *In 4th International Conference on Service Oriented Computing.*

Pantazoglou, M., Tsalgatidou, A., Spanoudakis, G., 2007. Behavior-aware, Unified Service Discovery. *In Proceedings of the Service-Oriented Computing: a look at the inside Workshop, SOC@Inside.*

Papazoglou, M., Aiello, M., Pistore, M., Yang, J., 2002. XSRL: A Request Language for web services, http://citeseer.ist.psu.edu/575968.html

SECSE Project. http://secse.eng.it

Shen, Z., Su, J., 2005. Web Service Discovery based on Behavior Signatures. *In Int. Conf. on Service Computing.*

Spanoudakis, G., Mahbub, K., Zisman, A., 2007. A Platform for Context-Aware Run-time Service Discovery. *In IEEE Int. Conf. on Web Services, USA.*

WSDL. http://www.w3.org/TR/wsdl

Yunyao, L.Y., Yanh, H., Jagadish, H., 2005. NaLIX: an Interactive Natural Language Interface for Querying XML. *In SIGMOD, Baltimore*.

Zisman, A., Spanoudakis, G., 2006.UML-based Service Discovery Framework. *In 4th International Conference on Service Oriented Computing, ICSOC, USA.*

Zisman, A., Spanoudakis, Dooley, J., 2008. A Framework for Dynamic Service Discovery. *In IEEE International Conference on Automated Software Engineering, ASE, Italy, September.*