Unified Discovery and Composition of Heterogeneous Services

A. Tsalgatidou [1], G. Athanasopoulos[1], M. Pantazoglou[1], H. Hoff[2], D. Skogan[2], A.-J. Berre[2]

[1] National & Kapodistrian University of Athens (NKUA),
Department of Informatics & Telecommunications, Athens 15784, Greece
{atsalga, gathanas, michaelp}@di.uoa.gr
[2] SINTEF Information and Communication Technology
P.O.Box 124 Blindern, N-0314 Oslo, Norway
{hjordis.hoff, david.skogan, arne.j.berre}@sintef.no

Abstract. Service-Oriented Development (SOD) is currently gaining momentum and has been marked as the future trend in software development. The building blocks of a service-oriented system are services which may be instantiated by web, grid and p2p services. For SOD to prevail there is a need to support the discovery and composition of services. However, this is not an easy task, due to the heterogeneity and incompatibility between the architectural models, protocols, and standards employed by web, grid and p2p services for their description, discovery and composition. In this paper, we present a unified approach for discovering heterogeneous services and composing them in a visual manner. The approach comprises a set of complying languages and their enacting tools.

1 Introduction

Current trends in software engineering signify a shift from component-based to service-oriented development (SOD). SOD envisages the rapid and cost-effective development of interoperable, loosely-coupled, distributed applications over-the-internet. Services are the building blocks of such applications and are mainly instantiated by web, grid and p2p services. Service-oriented applications -rather than being built from scratch- are being developed as service compositions by exploiting the ever growing number of available services. However, the discovery and composition of appropriate services becomes a tedious task, due to the heterogeneity and incompatibility among the architectural models, protocols, and standards employed by web, grid and p2p services for their description, discovery and composition. To our best of knowledge, there is no infrastructure or tools available for facilitating the integration and interoperability of such services. This paper presents an approach which attempts to bridge this gap by offering appropriate languages and corresponding tools to support the discovery and composition of heterogeneous (web, p2p, grid) services, in an open and unified manner. This approach leverages the development process by alleviating the developer from the specific details of each type of service. The provided languages and tools may be easily extended so as to accommodate additional protocols and standards, or even other types of services that were not originally addressed.

The merits of the proposed approach may be exploited when following either a top-down or a bottom-up development approach. For example, in a top-down service composition approach a developer needs to specify the comprising tasks as well as the control and data flow between them. These tasks can be executed by various types of services (rather than being programmed from scratch) which may not be known initially. Hence, apart from the composition flow a developer needs to model the requirements for appropriate services, which can satisfy specific composition tasks. To support this modeling task, we provide a Visual Composition Language (VSCL) and its associated tool (described in section 2). The next step in top-down service composition is the search of appropriate services, which can satisfy the requirements of each task in the service composition. The ever growing number of web, p2p and grid services, which abide by incompatible protocols and standards, renders discovery a daunting task. To address this, we provide a Unified Service Query Language (USQL) and its associated query engine (described in section 3), that support the discovery of heterogeneous services in a unified way. Both semantic and Quality-of-Service (QoS) information are utilized to improve the discovery process and results. Selected services substitute the requirements in each composition task resulting in a concrete service composition model.

In the next sections (2 and 3), we present our proposed approach, describing the languages and tools that enable the composition and discovery of heterogeneous services, respectively. Section 4 compares our work with existing related approaches and finally section 5 illustrates our concluding remarks.

2 Visual Service Composition

There exist some graphical tools for composing homogeneous value-added services, but none for composing new services from a number of heterogeneous services. To bridge this gap, we have developed a graphical language called VSCL (Visual Service Composition Language) and an associated software tool which are described next.

2.1 The Visual Service Composition Language

The Visual Service Composition Language (VSCL) was developed in order to support the unified, visual composition of heterogeneous services (web, p2p and grid services). The language provides various elements which are used for specifying:

- composition *tasks*,
- *service operation(s)* which can satisfy composition tasks,
- *queries* (expressed in the USQL language which is described in section 3) associated with composition tasks which can be used for service discovery at execution time,
- the order of execution of the various tasks (*control flow, decision/merge, fork/join*),

- data and data flows between the different tasks (*data flow*),
- how output data from preceding task(s) are transformed into expected input data for the succeeding task (*transformation nodes*),
- semantic and QoS annotations on tasks/services and their input and output parameters, and finally
- the interface of the new composite service (*composition*).

From the user perspective, the different service types are handled in the same manner through a common service operation abstraction. The actual service type is denoted as metadata on each service operation associated with a given task.

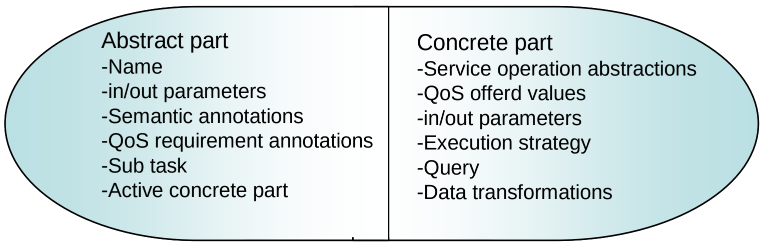


Fig. 1. The task concept

The core concept in the composition model is the Task. Each task has an abstract part and a concrete part (see Fig. 1). The abstract part defines the task to be performed in an implementation independent manner, whereas the concrete part specifies service operations that can realize the task or a query to find existing service operations at execution time. A service operation contains enough information to invoke the particular operation, but hides the complexity of the underlying heterogeneous service for the user. It is a great advantage to have both the abstract and the concrete aspects in the same model. The abstract part is used when searching for appropriate candidate services to realize each of the abstract tasks. The concrete part is being specified after having discovered and selected the appropriate service operations (or after having decided to assign a query at a task) to be executed at run time. Having the abstract and concrete aspects in the same model is also beneficial with respect to composition maintenance.

We have chosen to base the VSCL on the activity diagram part of OMG's UML 2.0 specification [19]. The extensions to UML are mainly introduced to handle:

- the service heterogeneity,
- the task concept with its abstract and concrete parts,
- the semantic and QoS annotations
- the data transformations of in/out parameters between tasks and between a task and its associated service operations.

UML was selected instead of BPMN [21] due to its extendibility, XMI-format and because it is a standard for software development.

2.2 The Visual Service Composition Suite

To support the language we have built a modelling tool called the Visual Service Composition Suite. This is a plug-in to the Eclipse development platform, which is a comprehensive, powerful and extensible development environment. We have utilized a number of the available Eclipse plug-ins in our development, e.g. UML2[18] and EMF[16] for representation and code generation and GEF[17] for building the graphical editor. Important design criteria have been taken into account in order to produce a flexible and extensible tool that may be integrated with other tools (e.g. query, execution and repository tools).

More specifically, the Visual Composition Suite consists of the following subcomponents:

- a visual editor for editing compositions,
- a language translator for translating the graphical notation to a lexical XML-based notation, and
- an analyzer for validating the composition against a set of rules (see Fig. 2).

The Visual Composition Suite supports the following three approaches that may be applied when defining heterogeneous service compositions:

- *top-down approach* when the developer identifies abstract sub-tasks[1] without knowing which service operations can satisfy them,
- *bottom-up approach* when the developer imports descriptions of pre-known services and uses service-operations to specify concrete tasks[2] in the compositions
- *dynamic approach* when queries are associated with the task(s) and are used for service discovery at execution time.

The compositions can be annotated with QoS and semantic information which serves as input for service discovery and service selection/matchmaking. Transformation of messages between the different tasks, and between each task and its selected service operations, can be defined when the message format does not match. The language translator takes a graphical composition as input and generates a lexical representation that serves as input to a composition execution engine. In a pure web service context this execution language could probably have been BPEL, although this has not yet been investigated. The other language translator that has been investigated is a translator that takes the abstract part of the tasks as input and generates service queries based on that. These queries may either be passed to the Query engine (see next section on USQL Engine) or be included in the lexical language. More details about the Visual Composition Suite and its integration with other tools may be found at [28].

[1] An abstract task contains only a requirements specification with parameters and semantic and/or QoS annotations, without having any specific service operations associated to it.

[2] A concrete task is a task with associated service operations

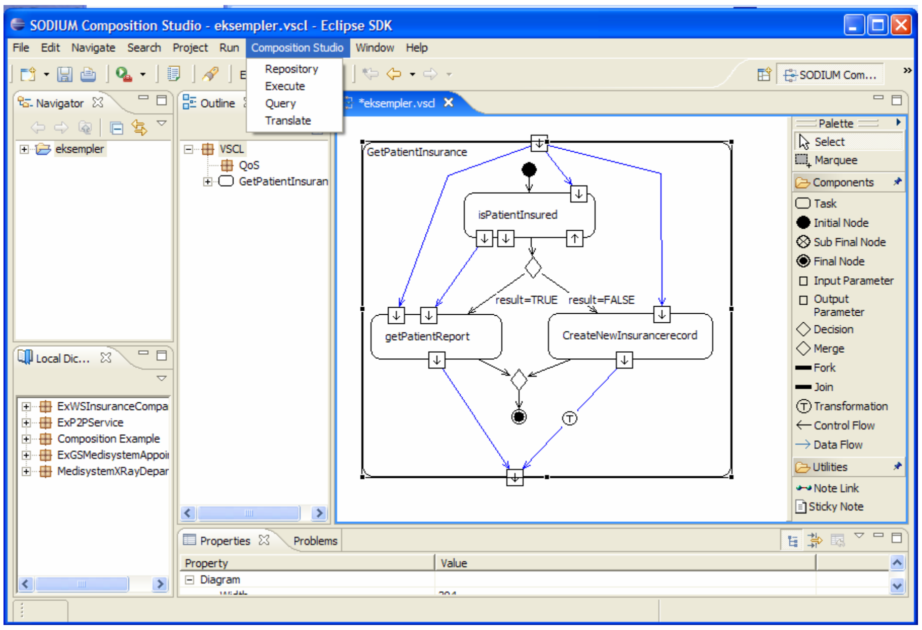


Fig. 2. Example screenshot of the Visual Composition Suite

3 Unified Service Discovery

Service discovery takes a significant role in SOD; it enables developers to find and re-use existing services that meet their application requirements, and thus greatly leverages the overall development process. In our approach, the discovery of services is conducted through the use of the Unified Service Query Language (USQL) and its enacting query engine, namely the USQL Engine. Both the language and the engine are described in the next paragraphs.

3.1 The Unified Service Query Language (USQL)

The *Unified Service Query Language (USQL)* is an XML-based language that facilitates service requesters in expressing service requirements in a consistent and unified manner. The language is distinguished by the following qualities:

- **Simplicity:** USQL remains simple enough to be understandable and usable by a wide range of users, from novice ones to IT professionals. Although described by an XML grammar, the USQL documents are comprehensible by humans.
- **Expressiveness:** The language enables its users to express their service requirements in a consistent and rich way, by employing the necessary elements for representing syntactic, semantic, as well as Quality-of-Service (QoS) search criteria.

- **Abstractiveness:** The USQL specification has taken into consideration and exploited the various existing registry and service description standards and protocols, without requiring from its users to have knowledge of their underlying structure. Thus, users of the USQL are able to construct their query documents without having to worry about technical aspects and details of the candidate services.
- **Extensibility:** The language is rendered open and extensible, so as to allow the definition of extra features deriving from other kinds of services that need to be supported, without affecting the already defined elements and supported types of services.

USQL defines two types of messages, namely the request and response. USQL requests can be formulated in a unified way, regardless of the actual type of the candidate services. Abiding by the Generic Service Model (GeSMO), which is briefly described in [1], the language allows the assessment of requirements at the service level, the operation level, and the message (i.e. input/output) level. Moreover, a well defined set of operators can be used to bring the various syntactic, semantic, and QoS criteria as close as possible to the actual real-world requirements.

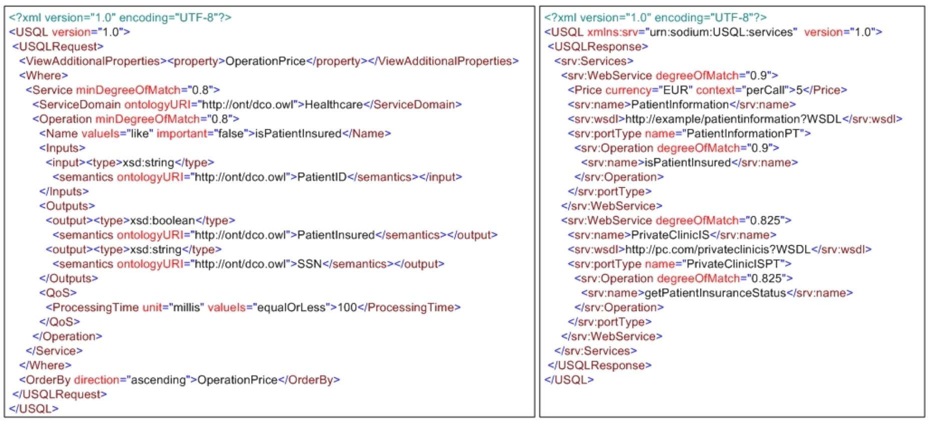
A USQL request is structured in a way that resembles traditional data query languages (e.g. SQL). More specifically, it comprises the following parts:

- **ViewAdditionalProperties** that allow requesters to explicitly specify information to be contained in the response message for each returned matching service and/or operation, apart from some basic properties that are always returned to requester.
- **From** that enables the explicit identification of the registries and networks that will be queried. Each registry or network is identified by a unique URI, and the ontology where it comes from.
- **Where** which is the most important part of the request, as it contains the actual service requirements; syntactic, semantic, as well as QoS search criteria can be expressed in a rich manner.
- **OrderBy** which is used to sort the matching services in the response, according to specific properties (e.g. price, degree of match, etc.).

From the above parts, only the *Where* is required, whilst the others are optional and can be omitted from a USQL request. As it is explained in paragraph 3.2, our infrastructure is capable of implicitly identifying the appropriate registries and networks for querying, based on the service requirements.

Although USQL requests are independent of the type of services they target for, the corresponding USQL responses must be concrete and must convey adequate information that will enable the immediate invocation of the matching services. Therefore, each service entity in a response is strongly bound to the type of service it conveys and provides specific information for this type of service. Additionally, it may convey syntactic, semantic, and QoS information that was originally asked for in the respective USQL request. This additional information is usually human-intended and plays a key role in service selection.

In order to clarify the above features of USQL, the following figure illustrates a USQL request along with its resulting response.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="1.0">
 <USQLRequest>
  <ViewAdditionalProperties><property>OperationPrice</property></ViewAdditionalProperties>
  <Where>
   <Service minDegreeOfMatch="0.8">
    <ServiceDomain ontologyURI="http://ont/dco.owl">Healthcare</ServiceDomain>
    <Operation minDegreeOfMatch="0.8">
     <Name valueIs="like" important="false">isPatientInsured</Name>
     <Inputs>
      <input><type>xsd:string</type>
       <semantics ontologyURI="http://ont/dco.owl">PatientID</semantics></input>
     </Inputs>
     <Outputs>
      <output><type>xsd:boolean</type>
       <semantics ontologyURI="http://ont/dco.owl">PatientInsured</semantics></output>
      <output><type>xsd:string</type>
       <semantics ontologyURI="http://ont/dco.owl">SSN</semantics></output>
     </Outputs>
     <QoS>
      <ProcessingTime unit="millis" valueIs="equalOrLess">100</ProcessingTime>
     </QoS>
    </Operation>
   </Service>
  </Where>
  <OrderBy direction="ascending">OperationPrice</OrderBy>
 </USQLRequest>
</USQL>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<USQL xmlns:srv="urn:sodium:USQL:services"  version="1.0">
 <USQLResponse>
  <srv:Services>
   <srv:WebService degreeOfMatch="0.9">
    <Price currency="EUR" context="perCall">5</Price>
    <srv:name>PatientInformation</srv:name>
    <srv:wsdl>http://example/patientinformation?WSDL</srv:wsdl>
    <srv:portType name="PatientInformationPT">
     <srv:Operation degreeOfMatch="0.9">
      <srv:name>isPatientInsured</srv:name>
     </srv:Operation>
    </srv:portType>
   </srv:WebService>
   <srv:WebService degreeOfMatch="0.825">
    <srv:name>PrivateClinicIS</srv:name>
    <srv:wsdl>http://pc.com/privateclinicis?WSDL</srv:wsdl>
    <srv:portType name="PrivateClinicISPT">
     <srv:Operation degreeOfMatch="0.825">
      <srv:name>getPatientInsuranceStatus</srv:name>
     </srv:Operation>
    </srv:portType>
   </srv:WebService>
  </srv:Services>
 </USQLResponse>
</USQL>
```

Fig.3 A USQL request and its respective USQL response

The depicted USQL request expresses the requirements for the first task displayed in the Visual Editor screenshot (Fig. 2). A closer look at the request reveals that it is service type-agnostic: indeed, there is no indication regarding the type of the candidate service. The task requires an operation that takes as input the id of the patient and returns his/her insurance status. Besides that, the patient's Social Security Number (SSN) is required, in the case where he/she is insured. The requester specifies that this operation should not exceed 100 milliseconds of processing time and thus includes this QoS requirement in the USQL request[3]. Moreover, he/she needs to see the price for each matching operation, so as to make a final decision upon which to use in the service composition. This need is captured with the use of the *OperationPrice* property specified in the *ViewAdditionalProperties* element. As it can be seen in the USQL request, the matching entities will be sorted in ascending order, according to their price. Finally, the requester has decided to relax the minimum degree of match requirement both for the service and the requested operation, while he/she also stated that the operation name syntactic requirement is not important. These relaxations will be taken into consideration by the matchmaking algorithm, which determines the degree of match for each service entity and its operations in the USQL response.

Upon the successful execution of the USQL request, a USQL response is returned, as it is shown in Fig. 3. The response contains two web services, ordered according to their price in ascending order. Each service entity conveys adequate information that can be used for its immediate invocation.

3.2 The USQL Engine

The USQL Engine supports heterogeneous service discovery over heterogeneous registries and networks. It is responsible for processing and forwarding a USQL

[3] Although our approach enables the expression of QoS requirements in service discovery requests, neither the USQL language nor the USQL Engine are responsible for maintaining the QoS properties of services, or for confirming their reliability.

request to the appropriate service registries and networks, applying advanced matchmaking activities and, finally, formulating the corresponding USQL response. The key feature of the engine that makes it a departure from other existing service query engines, is its ability to accommodate virtually any type of registry and network, and to process any type of service description, thus remaining independent of the various service description protocols and standards. This flexibility is achieved by its open architecture, which is primarily based on the use of plug-ins.

Thus, the USQL Engine functionality and the range of registries it explores is augmented with the use of appropriate registry plug-ins which enable the access and querying of registries of specific types, such as UDDI, ebXML, etc [6]. In turn, each registry plug-in can be configured so as to process specific service description protocols, such as WSDL [6], OWL-S [2], WSML [12], WS-QoS [3], etc.

As it can be seen in the figure below (Fig. 4), both the USQL Engine and its registry plug-ins are flexible and re-configurable; the appropriate registry plug-in is plugged to the engine whenever a query to a specific type of registry is required. Each plug-in implements a registry-specific handler in order to encapsulate the registry-specific discovery API, while it uses appropriate syntactic, semantic, and QoS handler components, in order to “understand” and process the advertisements hosted by its affiliated registry. In addition, a form of parallelism is achieved in the execution of queries through the use of a multi-thread mechanism. This mechanism simultaneously activates a set of registry plug-ins, which are responsible for dispatching a USQL request to a set of diverse registries and networks and processing the returned results. For instance, with the use of the USQL Engine, it becomes feasible to forward a USQL request to a UDDI registry which contains WSDL, WSML, and WS-QoS service descriptions and, at the same time, to a JXTA [4] network where JXTA service advertisements have been augmented with the use of OWL-S [5].

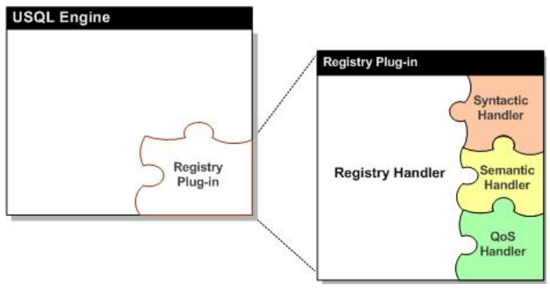
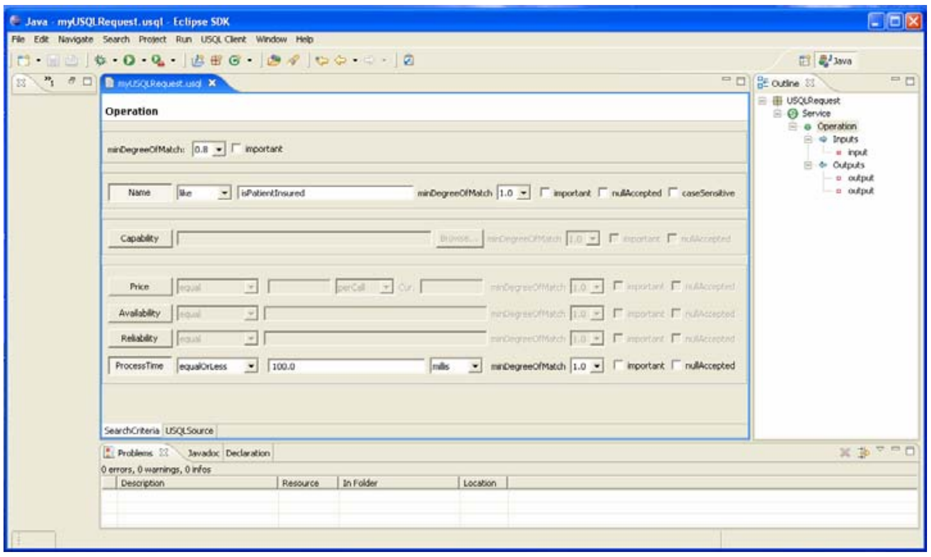


Fig. 4. High-level structure of a USQL Engine registry plug-in

We would like to note that, the USQL Engine is capable of implicitly identifying the target registries and networks, in order to access and query against them. The USQL Engine attains this substantial feature by maintaining and using a domain-specific upper ontology. Briefly described, the ontology enables the categorization of registries and networks, according to the application domain of the services which they advertise. Registry entries are manually inserted and maintained in the ontology. The ontology also associates domains with domain-specific concepts which are used for populating USQL semantic requirements as well as during the application of semantic matchmaking[4]. More details regarding the structure and usage of the ontology are provided in [1]. Hence, the selection of the appropriate registries for a query is performed implicitly, according to the service domain that has been explicitly specified in the USQL request. Alternatively, the requester may specify the registries he/she wants to query against, by making use of the *From* element in the USQL request, as explained in 3.1. Either way, service requesters are alleviated from the burden of acquiring knowledge related to the various registries and their underlying technologies and can focus their efforts in describing their requirements accurately using the USQL language.

The USQL Engine exposes its functionality through a web service interface, and thus can be seamlessly integrated in a service-oriented development environment. In addition, a USQL editor is currently being developed as a plug-in to Eclipse [12], in order to ease the task of constructing and submitting USQL requests to the engine. The editor provides a user-friendly form-based interface for the creation of USQL request documents. A screenshot of the editor interface is depicted in the following figure (Fig. 5), which displays the form used for describing the operation-level search criteria of the first task that is depicted in the Visual Editor (Fig. 2). On the right hand of the screenshot, a tree view is used to handle the USQL request structure; requested services comprise operations, which in turn may contain a set of inputs and outputs.


Java - myUSQLRequest.usql - Eclipse SDK
File Edit Navigate Search Project Run USQL Client Window Help
myUSQLRequest.usql
Outline
Java
Browse...
SearchCriteria USQLSource
Problems Javadoc Declaration

Operation

minDegreeOfMatch: 0.8 ☐ important

Name | like | isPatientInsured | minDegreeOfMatch 1.0 | ☐ important ☐ nullAccepted ☐ caseSensitive

Capability | | minDegreeOfMatch 1.0 | ☐ important ☐ nullAccepted

Price | equal | | perCall | Cur. | | minDegreeOfMatch 1.0 | ☐ important ☐ nullAccepted

Availability | equal | | minDegreeOfMatch 1.0 | ☐ important ☐ nullAccepted

Reliability | equal | | minDegreeOfMatch 1.0 | ☐ important ☐ nullAccepted

ProcessTime | equalOrLess | 100.0 | millis | minDegreeOfMatch 1.0 | ☐ important ☐ nullAccepted

USQLRequest
- Service
  - Operation
    - Inputs
      - input
    - Outputs
      - output
      - output

0 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location

Fig. 5. Screenshot of the USQL Editor

In this paragraph, we focused on how the USQL Engine manages to access and query against various heterogeneous registries and networks in a unified manner, and

[4] It should be noted that the USQL Engine provides minimal ontological support. Ontology mapping is not currently supported, although a semi-automated approach for achieving this is under investigation.

we briefly described the supported plug-in mechanism. For more details on the architecture and internal structure of the USQL Engine, we refer to [20].

4 Related Work

The proposed languages and corresponding tools cater for the development of applications following a service-oriented development approach. Specifically, their contribution lies in the areas of service composition modeling (VSCL with associated tool) and in service discovery (USQL with query engine). In the following we compare these two results with existing work in each of these areas.

Similar to BPMN [21] the VSCL language provides graphical notations for modeling service compositions. However, BPMN's current version lacks in precise modeling of data objects and data flows. Furthermore, BPMN currently may be mapped to compositions of web services only [25], whereas VSCL can be used for the composition of Web, Grid, as well as P2P services, providing support for semantics and QoS metadata which can be used for optimized service selection [15]. WS-BPEL [26] on the other hand, is the prevailing, XML based language used for the specification of executable web service compositions. However, WS-BPEL has received criticism for its lack in supporting semantics [27]. Triana [22] is a framework for composing service-oriented applications. Unlike VSCL, the supported workflow language does not provide explicit support for control constructs, while its visual representation is not standard-based with respect to VSCL. For more information on other research projects (e.g., Askalon, Unicore, Karajan) related to workflow specification and management, we refer the reader to [23, 24].

As far as service discovery is concerned one of the most prevailing issues seems to be the incorporation of semantics in service descriptions as well as their exploitation during the discovery process. Woogle [10] has developed a set of algorithms that enable searching for web service operations that are similar to a given one. The underlying idea of the search engine is the grouping of inputs and outputs into semantically meaningful concepts. Thus, syntactic information in service advertisements attains semantics and can be exploited in a more fruitful manner. The USQL with its associated engine enable the search of similar operations through the establishment of detailed USQL requests with rich and strict criteria at the operation level. As opposed to Woogle, our approach exploits the emerging semantic standards for service description, which are expected to greatly enhance and automate the course of service discovery. Various search engines are currently available on the Internet which group services according to their category [9], [11]. However, they are confined to web services only, while the USQL Engine allows for the categorization of not only web services but also of p2p, and grid services, with the use of the domain-specific ontologies, Other approaches [12] have also established a methodology for effectively discovering services. However, the overall process consists of a set of activities (depending in when does the service discovery actually apply), which render it rather complicated; on the contrary, the combination of the USQL language, the USQL Editor and the USQL Engine provides with a uniform, user-friendly, high-level and straightforward way for discovering heterogeneous services, regardless of the time the discovery process takes place.

5 Discussions and Conclusions

Service oriented development (SOD) is a new trend in software engineering. SOD is already affecting the development of business oriented systems turning them into service compositions. However, the heterogeneity in protocols and standards of existing service types is a major obstacle for the discovery of services and their integration in service compositions.

Within this paper we have presented a set of languages and their corresponding tools which facilitate the discovery and composition of heterogeneous services in an open and unified manner. Specifically VSCL along with the Visual Composition Suite enable the visual composition of heterogeneous services, whilst USQL along with the USQL Engine facilitate the discovery of heterogeneous services over heterogeneous registries and networks.

The provided languages and tools enable the unified discovery and composition of web, grid and p2p services. Nevertheless, both the languages and the corresponding tools are open and extendable so as to accommodate additional protocols and standards as well as other types of services.

Furthermore, the ensued approach for composing heterogeneous services doesn’t inflict any modifications on the underlying middleware and protocols that are used by the various services, but rather hides the specific details from the composition developers. Thus, it abstracts developers from the specific details of each type of service and boosts the development process.

Concluding we should note that the aforementioned languages and tools are still under development, thus our future plans include their finalization and testing. Specifically all languages and tools will be evaluated against two pilot case scenarios; one from the healthcare management and the other from the crisis management application domains. Our testing results will be used for measuring the effects of our approach in the development of applications composed of heterogeneous services. Last but not least, we plan to further extend our work by addressing other types of services apart from the web, grid and p2p services that are originally addressed.

Acknowledgement. This work has been partially supported by the European Commission under the contract IST-FP6-004559 (project SODIUM: Service Oriented Development in a Unified fraMework).

6 References

1. A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, “Semantically Enhanced Discovery of Heterogeneous Services”, 1st International IFIP/WG12.5 Working Conference on Industrial Applications of Semantic Web (IASW2005), 25-27 Aug. 2005, Jyväskylä, Finland

2. OWL-S, http://www.w3.org/Submission/OWL-S/
3. WS-QoS, http://www.wsqos.net/
4. JXTA, http://www.jxta.org/
5. D. Elenius, M. Ingmarsson, "Ontology-based Service Discovery in P2P Networks", International Workshop on Peer-to-Peer Knowledge Management (P2PKM), 22 Aug. 2004, Boston, Massachusetts, USA
6. A. Tsalgatidou, T. Pilioura, "An Overview of Standards and Related Technology in Web Services", International Journal of Distributed and Parallel Databases, Special Issue on E-Services, 12(2): 135-162, Sep. 2002
7. WSML, http://www.wsmo.org/wsml/
8. SeCSE, http://secse.eng.it/pls/secse/ecolnet.home
9. Binding point, http://www.bindingpoint.com/
10. X. Dong, A. Halevy, J. Madhavan, E. Nemes, J. Zhang, "Similarity Search for Web Services", Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004
11. Salcentral, http://www.salcentral.com
12. Eclipse, http://www.eclipse.org
13. R. Grønmo, M. C. Jaeger, "Model-Driven Semantic Web Service Composition", Asia-pacific Software Engineering Conference (APSEC 2005), Taipei, Taiwan, December 2005
14. R. Grønmo, M. C. Jaeger, H. Hoff, "Transformations between UML and OWL-S". In European Conference on Model Driven Architecture – Foundations and Applications (ECMDA'05), Nurnberg, Germany, Nov. 2005.
15. R. Grønmo, M. C. Jaeger, "Model-Driven Methodology for Building QoS-Optimised Web Service Compositions". In the 5th IFIP Intl Conference on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, Jun. 2005
16. EMF project, "Eclipse Modeling Framework", v.2.2.0, Dec. 2005, http://www.eclipse.org/uml2/
17. GEF project, "Graphical Editing Framework", Oct. 2005, http://www.eclipse.org/gef/
18. UML2 project, "EMF-based UML 2.0 Metamodel Implementation", Dec. 2005, v. 2.0.0, http://www.eclipse.org/uml2/
19. OMG, "Unified Modelling Language (UML), Superstructure", ver. 2.0, Jul. 2005.
20. A. Tsalgatidou, M. Pantazoglou, G. Athanasopoulos, "Semantically Enhanced Unified Service Discovery", W3C Workshop on Frameworks for Semantics in Web Services, Digital Enterprise Research Institute (DERI), Innsbruck, Austria, 9-10 Jun. 2005
21. BPMN, Business Process Modeling Notation (BPMN). 2003, Business Process Management Initiative (BPMI),http://www.bpmi.org/
22. M. Shields, I. Taylor, "Programming Scientific and Distributed Workflow with Triana Services", In Proceedings of Workflow in Grid Systems Workshop in GGF10, at Berlin, Germany, Mar. 2004
23. B. Ludäscher and Carole Goble, Special Section on Scientific Workflows, SIGMOD Record, 34(3), Sep. 2005
24. J. Yu, R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing", Technical Report, GRIDS-TR-2005-1, Grid Computing and

Distributed Systems Laboratory, University of Melbourne, Australia, Mar. 10, 2005
25. S. White, “Using BPMN to Model a BPEL Process”, white paper, http://www.bpmn.org/Documents/Mapping BPMN to BPEL Example.pdf
26. OASIS , Web Service Business Process Execution Language (WS-BPEL), 2004, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
27. Woodman, S.J., et al. “Notations for the Specification and Verification of Composite Web Services”, In The 8th International IEEE Enterprise Distributed Object Computing, Conference. 2004. Monterey, California, USA
28. D. Skogan, H. Hoff, R. Grønmo, T. Neple, “D9-Detailed Specification of the SODIUM Service Composition Suite”, SODIUM (IST-FP6-004559) project’s deliverable, Jun. 2005