

Chapter 4

Unified Discovery and Composition of Heterogeneous Services: The SODIUM Approach

Aphrodite Tsalgatidou, George Athanasopoulos, Michael Pantazoglou,
Arne J. Berre, Cesare Pautasso, Roy Grønmo, Hjørdis Hoff

1 Introduction

Service-Oriented Computing (SOC) is an emerging software engineering trend that promises to reform the way applications are built. Services, the main building blocks in this new engineering trend, provide the means to utilize functionality that is offered by service providers via message exchanges over the Internet. The unique characteristics of a service have been a highly debated research issue (see for example (Kozlenkov et al. 2006), (Czajkowski et al. 2004), or (Vogels 2003, 59)); nonetheless, all researchers agree that a service possesses properties such

as self-description, internet accessibility and message oriented communication. Normally, a service can be described, discovered and invoked using XML-based protocols and standards which lie on top of other proven communication protocols e.g. HTTP.

Web Services (Booth et al. 2004) is the most well known instantiation of the Service-Oriented Computing (SOC) paradigm. Other instantiations include Grid Services (Czajkowski et al. 2004), which emerged from the scientific application domain, or Peer-to-Peer (P2P) Services (Li 2001, 88), which originated from community-oriented systems and applications, e.g. Instant Messaging, File Sharing, etc.

All these services, irrespectively of their type, offer functionality which can be very useful in the development of service-oriented applications. For example, the applications in the Crisis Management domain require functionality which may be provided by services from the Health Care domain or services from the Traffic Management domain. However, such services are usually heterogeneous and are published in heterogeneous registries and networks. Thus, their distinct features, properties, supported protocols and architectural models render them incompatible

(Athanasopoulos, Tsalgatidou and Pantazoglou 2006). Furthermore, it is notable that besides the interoperability issues between different service types, there are also interoperability concerns among services of the same type, for example between P2P Services, as they mostly adhere to proprietary protocols and standards, or between Web Services due to different existing implementations. The latter has given rise to approaches such as the one undertaken by WS-I (Web Service Interoperability Org) which has established the basic interoperability profile (Ballinger et al. 2004) to deal with the discrepancies between Web Services.

The aforementioned service heterogeneity and the lack of interoperability between services of the same type as well as between services of different types constitute a major obstacle towards their widespread utilization in new service-oriented applications of diverse domains. Specifically, the activities of service discovery and service composition, which are two of the most important tasks in service-oriented engineering, become really cumbersome when one has to deal with services adhering to heterogeneous protocols and standards. We believe that this situation can be greatly improved by a unified approach

towards the discovery and composition of heterogeneous types of services; such an approach can alleviate the developer of a service-oriented application from the burden of dealing with the heterogeneity of the current service protocols, standards and architectural models. This is the approach that is followed by the *SODIUM (Service-Oriented Development In a Unified fraMework)* project which in this way supports the exploitation of the useful functionality offered by existing heterogeneous services and thus facilitates the development of service-oriented applications. Specifically, the goal of SODIUM is to create an open and extensible platform comprising appropriate tools and middleware that abstract developers from the underlying characteristics of the addressed service types. At the same time, SODIUM allows each type of service to retain its unique characteristics, without altering or enforcing any restrictions upon the underlying service provision platforms. Hence, developers are able to utilize the distinct traits of each service type. The types of services addressed by the current tools and languages of SODIUM are that of Web Services, P2P Services and Grid Services. Nevertheless, the openness and extensibility of the SODIUM solution

allows for the support of other service types, like UPnP (Newmarch 2005) or sensor services (Gibbons et al. 2003, 22) by the provision of the appropriate extensions and plug-ins.

The contribution of SODIUM is mainly positioned in the Service Integration Layer of the NESSI Framework (NESSI) but it can also be positioned in the Interoperability Layer of this Framework. Specifically, the contribution of SODIUM lies in the bottom and middle layers of the SOC Roadmap proposed in (Papazoglou and Georgakopoulos. 2003, 24), as it provides innovative solutions for the description, discovery and composition of heterogeneous services. It also touches the upper layer of the SOC Roadmap as it provides some support related to the monitoring of compositions of heterogeneous services. Finally, it should be noted that the SODIUM solution for service discovery and composition exploits existing work on semantics and quality-of-service, without however making a specific contribution in these areas.

In the following sections we present the approach employed by the SODIUM project and its outcomes as follows: we by describing a motivating scenario that illustrates the need to integrate heterogeneous

services (section *“Motivating Scenario”*). This scenario comes from the Crisis Management domain and it was implemented by one of the pilot applications developed for the evaluation of the SODIUM platform (Hoff, Hansen and Skogan 2006). We continue by illustrating the existing heterogeneity and discrepancies in the protocols and standards used for service description, discovery, invocation and composition with respect to Web, P2P and Grid Services which hinder the reuse of such services in other service-oriented applications (section *“Heterogeneous Services Computing”*). Then, we exemplify the SODIUM approach and present the main elements of the SODIUM platform (section *“The SODIUM Approach to Service Discovery and Composition”*). Finally we compare our work with other similar approaches (section *“Related Work”*) and present some concluding remarks (section *“Conclusions”*).

2 Motivating Scenario

Our motivating scenario comes from the *Crisis Management* domain, where an important task is to determine how to get to a crisis location and dispatch the appropriate emergency units as fast as possible. For example, in case of an accident with severely injured people, it is critical to reach these persons with the appropriate equipment within minutes. In such cases, if the injury causes lack of oxygen to the brain for 3-5 minutes, brain cells start to die and after approximately 15 minutes the damages are irreversible. Thus, it is vital that properly equipped ambulances and other rescue units are located within a 15-minutes range at all times and places, to increase the possibility of reaching injured people before it is too late.

This requirement is hard to achieve due to the vast set of parameters that need to be taken into account, e.g. accident/injury probability, population density and composition, accessibility, time of day/week/month/year, weather conditions, hospital locations and many others. Furthermore, the integration of information stemming from various systems that calculate all these parameters i.e. weather forecasting systems, traffic management systems, hospital information systems, etc. is a complicated problem. Therefore, the use of a service-oriented approach

in the development of applications satisfying the aforementioned requirements can be beneficial. Some examples of services that provide useful functionality for the implementation of the above scenario are:

- Web services providing weather information such as temperature and precipitation or traffic conditions from roadside speed sensors and video surveillance cameras.
- Grid services providing driving route calculations, weather forecasting information and "response range" calculations based on current positions and conditions.
- P2P services providing information about the locations and status of emergency vehicles and messaging facilities to the emergency vehicles with reposition message commands.

Alas, such existing services and systems, e.g. services and systems from the Health Care Management domain, Weather Forecasting systems, and so on, are highly heterogeneous and thus difficult to be discovered and combined. Therefore, a service-oriented application supporting this scenario needs to be able to integrate heterogeneous services such as the ones mentioned above. Nevertheless, this is not an easy task, as we

mentioned in the introduction, due to the incompatibility of the existing service types. In the following section we illustrate some of the incompatibilities that need to be dealt with. The rest of the paper presents the SODIUM approach which provides a unified solution to the discovery and composition of heterogeneous services (i.e. Web, Grid and P2P services) and which has been used for implementing the motivating scenario described above.

3 Heterogeneous Services Computing

The requirements imposed by real-world applications, such as the ones presented in the previous section, induce the need for discovery and composition of various types of services. However, as we mentioned in the introduction, this is not an easy task due to the heterogeneity that characterizes the various underlying service technologies. In the following paragraphs, we outline the results of a thorough analysis on the technologies of Web, P2P, and Grid services, conducted within the context

of SODIUM, which revealed a number of heterogeneities, and discrepancies spanning across aspects such as service description, discovery, invocation, and composition. For a detailed description of the state-of-the-art analysis, please refer to (Tsalgatidou et al. 2005).

3.1 State-of-the-Art in Service Description

The information conveyed by the description of a service generally falls into one or more of the following categories:

- *Syntactic information* which refers mainly to the structure of the service interface, and the provided operation signatures;
- *Semantic information* which is provided to describe the capability of the service, i.e. its offered functionality;
- *Quality information* which describes the non-functional, qualitative characteristics of the service, such as its reliability, availability, performance, etc.

The *Web Services Description Language (WSDL)* (Christensen et al. 2001) has been established as the *de facto* standard for syntactically describing a service. Still, the peculiarities of the various service types, such as Grid and P2P services, have yielded numeral extensions to the standard. The *Web Service Resource Framework (WSRF)* (Banks 2006) defines a set of extension elements in WSDL in order to describe the properties of the resource being handled by a Grid service. On the other hand, network topology concepts, such as peers or peer groups, need to be described in a P2P service description to allow its invocation. This requirement has yielded extensions, such as the ones described in (Athanasopoulos, Tsalgatidou and Pantazoglou 2006), to the WSDL standard.

Over the last years, a number of diverse protocols were proposed to address the lack of semantic information in WSDL descriptions. The *Web Service Modeling Ontology (WSMO)* (Roman et al. 2005) and *OWL-S* (Martin et al. 2004) frameworks, along with the latest *SAWSDL* (Farrell and Lausen 2007) specification are the most prominent approaches towards this direction. These protocols were further extended to meet the requirements of the Grid, as they were utilized to provide semantic

annotations also to the descriptions of sharing resources (Babik et al. 2006).

Many heterogeneous protocols have also been proposed with respect to the Quality-of-Service (QoS). The *Web Service Level Agreement (WSLA)* (Keller and Ludwig 2003, 57), *WS-QoS* (Tian et al. 2004), and *Web Service Offering Language (WSOL)* (Tosic et al. 2003) are some of the proposed specifications that can be used to describe the QoS of a Web service. As for Grid services, approaches such as the *G-QoSM* (Al-Ali et al. 2002) and the *Globus Architecture for Reservation and Allocation (GARA)* (Foster et al. 1999) provide advanced quality specification and support management of services, resources, and the underlying network infrastructure. Finally, although not directly addressed in terms of language specifications, QoS have been taken into account in the P2P world, and many algorithmic approaches (Sahin et al. 2005) (Vu, Hauswirth and Aberer 2005) have been proposed to optimize the qualitative aspects of P2P networks.

3.2 State-of-the-Art in Service Discovery

Besides the heterogeneity in their descriptions, Web, P2P, and Grid services have employed diverse publication and discovery mechanisms. Registries complying with the *Universal Description, Discovery, and Integration (UDDI)* (Clement et al. 2004) specifications and the *ebXML* (EBXML) standard are commonly used for publishing and discovering Web services. On the other hand, Grid infrastructures, such as *Globus* (Globus) or the latest *gLite* (GLite), have established their own mechanisms for publishing and discovering resources and services within Virtual Organizations. Such mechanisms utilize directories and services which rely on the *Lightweight Directory Access Protocol (LDAP)* (Koutsonikola and Vakali 2004, 66). Completely different discovery approaches are realized by P2P technologies, such as *JXTA* (Li 2001, 88) or *Edutella* (Nejdl et al. 2002), where services and resources are advertised and discovered in a distributed manner, all over the network.

3.3 State-of-the-Art in Service Invocation

Web services are traditionally communicated through the exchange of *SOAP* (Mitra 2003) messages over proven network protocols, such as HTTP, SMTP, etc. The same invocation pattern is also applied to Grid services, with the exception that the service client needs first to acquire the necessary credentials in order to gain access to a Virtual Organization. When it comes to P2P services, their invocation is tightly coupled with the specific P2P technology. Thus, services provided by peers in a *Gnutella* (Gnutella) network are invoked in a different manner than services provided by peers in a JXTA network, and so on. Nevertheless, in most cases, the service client must either join the P2P network as peer, or use an existing *proxy* peer to be able to invoke P2P services.

3.4 State-of-the-Art in Service Composition

The *Business Process Execution Language for Web Services (BPEL4WS)* (Alves et al. 2007) has been established as a standard for composing Web

services into business processes. However, despite its wide adoption, BPEL4WS lacks the flexibility that would allow it to encompass other types of services as well, such as P2P and/or Grid services, also taking into account their specialized characteristics. Research efforts such as the *Kepler* (Altintas et al. 2004) engine have risen to support the execution of Grid service scientific workflows in Grid environments. Other efforts such as the one described in (Gerke, Reichl and Stiller 2005) have been proposed to enable the composition of P2P services in a P2P network. However, to the best of our knowledge, there is no approach, protocol, or standard to enable the composition of Web, P2P, and Grid services.

All the aforementioned protocols and standards suggest a heterogeneous situation that overwhelms developers and hinders the wide utilization of Web, P2P, and Grid services in a single service-oriented application. We believe that this challenge can be effectively addressed by establishing a unified approach in Service-Oriented Computing as regards the various existing and emerging types of services. This is the exact contribution of the SODIUM project, the significant results of which we present in the following sections.

4 The SODIUM Approach to Discovery and Composition of Heterogeneous services

The primary goal of SODIUM is *to facilitate the unified discovery and composition of heterogeneous services* (with focus on Web, P2P, and Grid services), and thus to promote interoperability at the service discovery and composition levels. The SODIUM approach achieves this goal by providing an abstraction layer that hides the technical details of each service type from both developers and end users, without altering or modifying the distinct properties and characteristics of the underlying technologies. The *SODIUM platform* realizes this abstraction layer, and comprises:

- A *Generic Service Model (GeSMO)* that supports the definition of the common as well as distinct characteristics of Web, P2P, and Grid services, thereby providing a solid conceptual basis to the SODIUM approach

- A set of languages, namely:
 - The *Visual Service Composition Language (VSCL)* which supports the visual composition of heterogeneous services
 - The *Unified Service Query Language (USQL)* which supports the unified discovery of heterogeneous services
 - The *Unified Service Composition Language (USCL)* which provides a format for the executable representation of visual compositions made of heterogeneous services
- A set of tools and supporting middleware, namely:
 - The *Composition Visual Editor* which implements the VSCL and supports the visual design of heterogeneous service compositions, as well as their translation to the USCL format
 - The *USQL Service Discovery Engine*, which implements the USQL and supports the unified discovery of different types of services from a wide spectrum of registries, repositories and networks

- o The *USCL Execution Engine*, which supports the execution and monitoring of heterogeneous service compositions that are expressed with the USCL format.

The following figure (Figure 4.1) depicts the overall architecture of the SODIUM platform and also outlines the interactions between the constituent tools.

< Figure 4.1 here >

The SODIUM platform is divided in two main subsets, the “*Composition Suite*” and the “*Runtime Environment*”, which support the design and the run-time phases of the development process, respectively. Nevertheless, as the above figure shows, the constituent tools are loosely coupled and communicate through document exchanges by means of well defined interfaces. Therefore, they can be integrated with other tools in the future so as to create a customized service-oriented development environment.

Along with the SODIUM conceptual model, languages and tools comes a model-driven methodology, which provides a way of composing existing yet heterogeneous services in order to execute a complex task.

In the following sections, we go through presenting and describing the SODIUM results.

4.1 The Generic Service Model

The *Generic Service Model (GeSMO)* provides the conceptual basis upon which the SODIUM languages and tools were developed. The specification of GeSMO was driven by the results of a thorough investigation of the state-of-the-art in Web, P2P, and Grid service technologies, which we outlined in section 3. In general, the model is characterised by its generality, abstraction, simplicity, modularity, expressiveness, and extensibility.

<Figure 4.2 here>

GeSMO has adopted a layered architecture (see Figure 4.2) as follows:

- The *Core Layer* which models all common concepts among the investigated service types, i.e. Web, P2P, and Grid services

- The *Extensions Layer* which sits on top of the Core Layer and caters for the distinct features of each service type, i.e. Web, P2P, and Grid services
- A number of layers, orthogonal to the Core and Extensions layers, which model additional cross-cutting features, such as semantics, quality-of-service, trust & security, and management

Naturally, the fundamental concept of GeSMO is *Service*. With the combination of concepts deriving from the layers described above it is possible to describe a service from multiple points of view. The following figure (Figure 4.3) outlines the different views that have been defined by GeSMO along with their interdependencies:

<Figure 4.3 here>

These service views are briefly described as follows:

- Abstract: This view looks into the service notion from an abstract point of view and tries to identify its relationships with elements of the software engineering field

- Basic: This view pinpoints the minimal set of elements that need to be provided. The elements that are identified within this view may be further analyzed in other sub-views
 - Description: This view focuses on the elements that are related to the description of a service
- Structure: The structure view identifies the structural elements that a service may comprise
- Semantics & QoS: This view identifies the elements of the service model that may have semantic and QoS annotations
- Message Structure: This view provides a look into the structure and the elements of messages that are exchanged among a service and its clients
- Communication: The communication view identifies the elements related to the underlying network communication details i.e. communication protocols that are used, network address, message wire format, etc.

The *Basic*, *Description*, and *Structure* views of GeSMO are shown in the following figures (Figure 4.4, Figure 4.5 & Figure 4.6).

<Figure 4.4 here>

The basic service model depicts a minimal set of concepts and their respective relationships that define the concept of service. This set of constructs according to (Vogels 2003, 59) suffices for the invocation of a service, but it needs to be further extended so as to facilitate the whole set of operations that are supported by the service model, i.e. publication, discovery, invocation, composition, etc.

<Figure 4.5 here>

The service description model provides a detailed specification of the information that a description document may convey. A description document may provide descriptions or links to other documents for the whole or for parts of the information that is depicted in Figure 4.5.

<Figure 4.6 here>

The structure model defines the set of structural elements that a service may be broken down into. For a detailed description of all views defined in our model, please refer to (Tsalgatidou et al. 2005).

GeSMO shares many common concepts with the SeCSE Conceptual Model (Colompbo et al. 2005). However, in GeSMO we primarily focused

on the definition of different points of view with the ultimate goal of describing services independently of their actual technology. In this respect, the conceptual service views provided by GeSMO may be considered as complementary to the SeCSE Conceptual Model.

Within the context of SODIUM, GeSMO served as a multi-purpose tool. More specifically:

- a) It was used as a conceptual basis for the specification of the three SODIUM languages, i.e. the VSCL, USQL, and USCL, as well as for the development of the P2P Service Description Language (PSDL) (Athanasopoulos, Tsalgatidou and Pantazoglou 2006), a WSDL-based format that was utilised in SODIUM for the description and invocation of JXTA P2P services;
- b) It provided a common point of reference that facilitated communication and knowledge exchange among the project stakeholders;
- c) Its abstraction and extensibility drove specific design decisions regarding the SODIUM tools, such as the plug-in based architecture adopted by most of them.

Concluding, we would like to note that, although the specification of GeSMO currently caters for Web, P2P, and Grid services, its extensibility allows for seamlessly accommodating other types of services in the future.

4.2 Visual Service Composition

Visual service composition is supported by SODIUM through the *Visual Service Composition Language (VSCL)* and the *Composition Suite*.

4.2.1 The Visual Service Composition Language (VSCL)

The VSCL language is based on UML (UML 2002), and supports the visual representation of service compositions which leverage Web, Grid and P2P services. Specifically the VSCL leverages UML activity diagram concepts as a basis for the provision of appropriate stereotypes that define all the necessary composition primitives. Nevertheless, VSCL has a conceptual meta-model (see Figure 4.7) that is independent of UML and

other existing graphical modelling languages, but may be realized by a UML profile.

<Figure 4.7 here>

According to the VSCL meta-model, a service *Composition* consists of *Nodes* and *Flows/edges*. The *Nodes* are *TaskNodes*, *ControlNodes*, *ObjectNodes*, *EventNodes* or *TransformationNodes*, whereas the different kinds of *Flow* are used to specify flow of control and data between nodes. Hence, the main concepts of the VSCL language are the *tasks* and the *flow of data and control* between tasks.

A task consists of both an abstract part and a concrete part. It may be coarse grained which means that it can be detailed in a sub composition of tasks. The abstract part is service independent and may be used as a starting point when querying for available and relevant heterogeneous services. The concrete part of the task has information about which service(s) to execute for the specific task. The strength of the task-based approach is that there is one composition graph, and not two – one concrete and one abstract. After services have been selected, the abstract part may still be used to check if new available and better suited services

have emerged. To match task output(s) to the input(s) of the successor task, the possibility for defining transformations has also been included (Grønmo, Jaeger and Hoff 2005). Alternatively, a task may itself be a composition and thus it is decomposed into sub-tasks.

A heterogeneous composition in VSCL may consist of tasks executed by different kinds of services. The types defined in the context of SODIUM are: P2P, Web or Grid services (see Figure 4.8).

<Figure 4.8 here>

As it is shown in the above figure, one or more services may be selected to realize/execute a specific task. When more than one service operation(s) are selected, the developer may state how the execution is to be done. There are three different possibilities. The service operations may be executed in parallel, sequential or in a random order, where the last two are associated with a response time limit. The service operations in the *selectedServiceOperations* list are considered “equal” with respect to the functionality they provide, but other aspects like e.g. QoS characteristics may vary between them.

The detailed specification of the VSCL language has been released as a public deliverable of SODIUM and may be found in (Hoff et al. 2005).

4.2.2 The SODIUM Composition Suite

<Figure 4.9 here>

The SODIUM Composition Suite consists of the following main subcomponents:

- The *Composition Visual Editor* for editing and analyzing service compositions with the use of the VSCL,
- The *VSCL2USCL Translator* that translates a service composition from graphical notation (VSCL) to a lexical XML-based notation (USCL), and
- The *USQL Dialog* for interacting with the USQL Engine in order to discover available Web, P2P, and Grid services.

The *Composition Visual Editor* is the main component of the Composition Suite and has been developed as an Eclipse

(<http://www.eclipse.org>) plug-in (see Figure 4.9). The Composition Visual Editor cooperates with the USQL Engine and the USCL Execution Engine in order to support the discovery of services and the deployment of executable service compositions. Moreover, it supports the:

- Specification of service compositions in multiple levels of abstraction

- Static analysis of service compositions described in VSCL and

Translation of VSCL graphs to USCL documents that can be executed by the Execution Engine.

The Composition Visual Editor supports three approaches for defining heterogeneous service compositions:

- *Top-down approach* for a task-oriented focus where tasks are identified, but no candidate service operations have yet been identified/selected,
- *Bottom-up approach* for a service-oriented focus when service operations to use are pre known, and
- *Dynamic approach* for service operations to be discovered at execution time.

A typical use of the Composition Visual Editor for the specification of service compositions based on the top-down approach is described as follows:

The first step in constructing VSCL graphs is to break down the composition into tasks, which can interoperate in order to finally achieve the overall goal. This initial composition model of tasks is called an *abstract model* since there are no selected concrete services identified yet. The abstract model is used as a basis for searching for appropriate candidate services which can realize each of the abstract tasks. When the appropriate services are discovered and selected, they are associated with the respective tasks and the result is a *concrete model*. Note that, apart from selecting a specific service for the implementation of a task, developers are also allowed to assign USQL queries that will be executed at run-time thus enabling the dynamic binding of services that are discovered, selected and invoked at run-time.

More details regarding the SODIUM Composition Suite are available in (Hoff et al. 2006).

4.3 Unified Service Discovery

Service discovery in SODIUM is supported by the *Unified Service Query Language (USQL)* and its associated engine, called *USQL Engine*, which are briefly described in the following.

4.3.1 The Unified Service Query Language (USQL)

The USQL is an XML-based language providing the necessary structures for the formulation of service-type independent query documents and their responses. A rich, yet extensible set of syntactic, semantic, and QoS search criteria enable service requesters to express their requirements in an accurate and intuitive manner. Moreover, with USQL requesters may express their requirements towards the service, its operations, as well as the messages (i.e. input/output) exchanged by them. Hence, the USQL specification retains its consistency with GeSMO, specifically with respect to the *Structure* view of a service (see Figure 4.6).

Abiding by the principles of GeSMO, the USQL has established a certain level of abstraction so as to support mappings from/to a wide range of service description formats (e.g. WSDL, OWL-S, SAWSDL, etc.) and discovery protocols (e.g. UDDI, ebXML, JXTA, etc.). Thus, it can be used to discover services in a unified manner, regardless of how the latter have been described, or where they have been published. The USQL specification defines two types of documents, namely the *USQLRequest* and *USQLResponse*. The former is used to express service type-independent queries, while the latter is used to convey the results of the query execution.

The following snippet demonstrates an example USQL request document, in accordance to the motivating scenario presented earlier in the chapter:

```

<?xml version="1.0" encoding="UTF-8"?>
<USQL xmlns="urn:usql" version="1.1">
  <USQLRequest id="1167253624765">
    <SearchCriteria>
      <Service>
        <Provider><Value>Locus</Value></Provider>
        <Domain ontologyURI="urn:sodium:ont:crisis">
          <Value>CrisisManagement</Value>
        </Domain>
        <Operation>
          <Semantics ontologyURI="urn:sodium:ont:crisis">
            <Value>GetCallerPosition</Value>
          </Semantics>
          <Input>
            <Part>
              <Semantics ontologyURI="urn:sodium:ont:crisis">
                <Value>PhoneNumber</Value>
              </Semantics>
            </Part>
          </Input>
          <Output>
            <Part>
              <Semantics ontologyURI="urn:sodium:ont:crisis">
                <Value>CallerLocation</Value>
              </Semantics>
            </Part>
          </Output>
        </Operation>
      </Service>
    </SearchCriteria>
  </USQLRequest>
</USQL>

```

The query is intended to search for services which retrieve the location of a caller based on its phone number. A closer look at the requirements included in the query reveals that there is nothing implying or bound to a specific service type; indeed, the USQL request is service type-agnostic.

The requested service belongs to the domain of Crisis Management, as this is expressed by the *Domain* element. The desired functionality as well as the input and output requirements have been captured with the use of the *Semantics* elements specified into the *Operation*, and the requested *Input* and *Output* parts. The concepts used to populate the *Semantics* elements have been taken from a custom domain ontology that was developed for the purposes of SODIUM. However, the USQL is independent of the ontology being used to populate its semantic elements and to this end, any ontology and/or semantic dictionary could be employed. To further constrain the query, and because of existing Service Level Agreements and partnerships, the service requester has also specified the provider of the requested service.

An example USQL response document to the above USQL request is depicted below:

```

<?xml version="1.0" encoding="UTF-8"?>
<USQL xmlns="urn:usql" version="1.1">
  <USQLResponse queryId="1167253624765">
    <Services>
      <Entry rank="1.0">
        <WebService xmlns="urn:ws">
          <Name>CrisisMgmtService</Name>
          <Operation>GetCallerLocation</Operation>
          <Port>GetCallerPositionPort</Port>
          <WSDL>
            http://jemini.di.uoa.gr:8080/sodium/services/CrisisMgmtService.wsdl
          </WSDL>
        </WebService>
      </Entry>
      <Entry rank="0.833">
        <JXTAService xmlns="urn:jxta">
          <Name>PeerPositionService</Name>
          <Operation>GetPeerPosition</Operation>
          <Interface>PeerPositionServiceIF</Interface>
          <PSDL>
            http://www.s3lab.com/services/PeerPositionService.psdl
          </PSDL>
        </JXTAService>
      </Entry>
    </Services>
  </USQLResponse>
</USQL>

```

Apparently, the service discovery process yielded two matches, a standard Web service and a JXTA P2P service, both delivering the desired functionality. The results have been prioritized according to their rank value, which quantifies their degree of match with respect to the search criteria of the USQL request. Note that, although the USQL request was constructed in a service type-independent manner, the information conveyed by each one of the corresponding entries in the USQL response

is strongly associated to the type of the referred services and is adequate to enable their invocation.

4.3.2 The USQL Engine

The *USQL Engine* is a powerful search tool enabling the discovery of heterogeneous services in various types of registries, repositories, and networks. As the name implies, the USQL Engine fully supports the USQL specification and acts as a black box from the user perspective: it accepts USQL request documents as input, and returns corresponding USQL response documents, as output.

The architecture of the USQL Engine is depicted in the following figure:

<Figure 4.10 here>

The USQL Engine is characterized by a high degree of openness and extensibility, which is achieved by using plug-in mechanisms to accommodate the different types of services and registries. Specifically,

the engine was extended in the context of SODIUM and plug-ins were provided to support the discovery of services in UDDI & ebXML registries and JXTA networks. Moreover, appropriate extensions were developed to support the processing of WSDL, SAWSDL, OWL-S, and WS-QoS service descriptions. Let us now briefly describe a typical service discovery process:

Upon receiving a USQL request document, the USQL Engine engages the *USQL Handler* to validate and forwards it to appropriate *Registry Plug-in* components, which are coordinated by the *Registry Selector* and run in parallel. The service description retrieved by the various registries are matched against the search criteria of the USQL request, and the ones that meet them are passed to the USQL Handler. The latter consolidates the results from all Registry Plug-in components and prioritizes them according to their degree of match into a single USQL response document, which is returned back to the service requester.

The USQL Engine provides both a Graphical User Interface (GUI) and a Web service interface. The GUI, namely *USQL Dialog*, has been integrated with the SODIUM Composition Visual Editor and is used by

developers to formulate USQL queries, access the USQL Engine and discover services at design time. At run time, the USCL Execution Engine may invoke the USQL Engine Web service and submit a predefined QoS-enhanced USQL query, in order to select and late-bind a service in a task, from a set of alternative services having the same interface, yet characterized by different quality properties.

The architecture and functionality of the USQL Engine have been described in (Pantazoglou Tsalgatidou and Athanasopoulos 2006, 104). Moreover, the definition of the matchmaking algorithm that has been implemented by the engine can be found in (Pantazoglou Tsalgatidou and Athanasopoulos 2006, 144).

4.4 Execution of Heterogeneous Service Compositions

The execution of compositions consisting of Web, P2P, and Grid services is accomplished in SODIUM by the *Unified Service Composition Language (USCL)* and the related *USCL Execution Engine*.

4.4.1 The Unified Service Composition Language (USCL)

The USCL is an XML-based language intended to be processed by machines, rather than humans. The main feature of the language consists of providing support for composing an open set of services, including Web, Grid and P2P services. The description of the compositions is kept separate from the description of the software components responsible for executing each composition task, in order to enhance the reusability of both. Compositions are modelled as processes, whose structure defines the data and control flow dependencies between the service invocations, as well as the required exception handling behaviour. Components are modelled as services, an abstraction that makes the mechanism used to access the corresponding implementation transparent.

The following diagram (Figure 4.11) depicts the structure of a USCL document:

<Figure 4.11 here>

The root element (*USCL*) of an USCL document can contain a set of *Process*, *Service* and *ServiceType* definitions. In practice, the service definitions and the required service types declarations are usually defined once and included from separate USCL documents. The external operation signature of a *Process* is defined by a set of input and output parameters. Internally, a process contains the list of its component tasks and the data flow (*Parameters* and *Edges*). The *Service* elements store the set of available service types that can be invoked. Similar to *Processes*, also the operation signature of services is composed of a set of input and output parameters. Furthermore, a service can contain multiple *Access Methods* which define different, alternative ways to invoke the functionality provided by the service. Access methods also have input and output parameters, conforming to the template defined in the referenced *ServiceType*. By definition, the input and output parameters of an access method and the ones belonging to the corresponding service type are considered as system parameters (Pautasso, Alonso 2004).

Control flow is specified declaratively as a set of Event-Condition-Action (ECA) rules, which are associated with each service invocation. The actions represent the actual invocation of the service, which is carried out only when the associated event fires and the condition evaluates to true. An event is defined as a predicate over the global state of the workflow (e.g., a rule should fire if a specific service has just successfully completed its invocation; another rule should fire if any out of a set of services has failed). A firing event will trigger the evaluation of the corresponding condition, which is a boolean expression, over the values of data flow parameters. Non-trivial conditions are used to model alternative paths in the execution of the workflow. Moreover, they can also represent loop entry/exit conditions, since the rules associated with the service invocations can fire more than once during the lifetime of the workflow.

The data flow is also modeled declaratively as a graph of edges linking pairs of parameters. This fits quite well with the approach taken by VSCL, where such edges are visualized. Thus it is intended to simplify the mapping between the two languages.

In addition to control and data flow, USCL also features nesting, iteration, recursion, reflection, dynamic binding and several other constructs specifically targeting the composition of heterogeneous kinds of services. The detailed specification of the USCL language is available in (Pautasso, Heinis and Alonso 2005).

4.4.2 The USCL Execution Engine

The *USCL Execution Engine* provides a reliable and scalable platform for executing processes given in the USCL format, which are composed of heterogeneous services (Pautasso, Alonso 2004b). To do so, the architecture of the USCL Execution Engine employs plug-in components which enable it to support an open set of heterogeneous service invocation mechanisms.

The USCL Execution Engine provides a number of Application Programming Interfaces (APIs), which are used for the communication

with the rest of the SODIUM tools, as well as for invocation of Web, P2P, and Grid services.

<Figure 4.12 here>

The functionality provided by the *Deployment API* is used to let the engine know that the processes and services declared in a USCL document are available, thus making the engine forget about the previously deployed items of the same USCL document. In this way, USCL documents can be deployed to the USCL engine so the compositions stored in them can be prepared for execution. By means of this API, the VSCL2USCL Translator component of the SODIUM Composition Suite is able to load USCL compositions to the USCL Execution Engine.

The *Startup API* is mainly used to initiate the execution of a new composition instance. In addition to starting an instance, this API also allows clients to assign values to the input parameters of the newly created instance and to control how the state of the instance is managed by the engine (monitoring & logging). Once a process has been instantiated, it should also be possible to identify it among the other ones

that are concurrently running inside the engine. Thus, after a process instance has been started, it is associated with a unique ID, returned to the client that started its execution. No assumptions should be made about the format of such an ID as this is left unspecified.

The *Monitoring API* allows accessing the execution logs of a given instance of a composition, identified by its ID. These logs provide information about the state of the execution of the composition and can be retrieved at any time during the execution of a composition. In general, the logs contain information about the data of a composition (current values of input/output parameters) as well as meta-data (current execution state of a task, performance profiling information, error messages, and so on). Logs could be presented in a variety of formats (e.g., text, CSV, XML) depending on the intended usage.

The *Invocation Plug-ins API* addresses the requirement of dealing with heterogeneous services, by providing a platform which can be extended to support the invocation of different kinds of services. This amounts to opening up the engine to use different mechanisms for invoking services of different kinds. The same API is also used by the engine for the

execution of USQL queries at runtime, by submitting them to the USQL Engine through its Web service interface.

The USCL Execution Engine supports both synchronous and asynchronous service invocation. For each service invocation to be performed the engine instantiates a new object of the given service invocation plug-in class. Furthermore, the plug-in is executed in a dedicated thread. In this way, the USCL Execution Engine handles the multithreaded issues for the concurrent invocation of multiple services.

The USCL Engine provides for the persistence of the state information of the process instances. The design of this mechanism has been influenced by many requirements, such as performance, reliability, and portability across different data repositories. Access to the state information of the composition instances is provided in terms of a key-value pair which uniquely identifies a certain data (or meta-data) value associated with a process (and task) instance. The state information data model is independent of the physical location of the data, so that it is possible to use caching to exploit locality and – for increased availability – replicate some of the values. Along these lines, in order to provide a level

of scalability, state management can be optimized to keep only a subset of all of the composition instances in memory and, for instance, swap compositions whose execution has been completed to secondary storage, in a so-called process history space. In this way, the USCL Engine gives access to the state of past executions to enable composition profiling and optimization, caching of already computed results as well as lineage tracking analysis.

Finally, in addition to the APIs previously described, the functionality of the USCL Execution Engine is also accessible through a *WSRF Web services API* (Heinis et. al 2005). This feature renders feasible the utilisation of the USCL Execution Engine as a distinct component outside the SODIUM platform.

Thanks to its multi-thread support and efficient resource management, the USCL Execution Engine achieves considerable performance and scalability. A detailed description of these features along with experiment measures may be found in (Pautasso, Heinis and Alonso 2007, 65) and in (Pautasso et al. 2006).

4.5 The SODIUM Service Composition Methodology

The *SODIUM Service Composition Methodology* provides a way of composing existing yet heterogeneous services, in an iterative, incremental four phase evolving manner, as the following figure (Figure 4.13) depicts:

<Figure 4.13 here>

Let us proceed with a description of the four development phases defined by the SODIUM methodology. Along the lines, we also exemplify the use of the various SODIUM tools in order to demonstrate how they should be used according to the principles of the methodology.

4.5.1 Phase 1: Modeling

The first phase of the methodology consists of a number of activities. The first activity is to identify the task to be solved by a new service composition. The coarse-grained task is then detailed into more detailed

tasks. The relationships between tasks are modeled as flow of control and data, and may be characterized as a graph consisting of nodes (tasks) and directed edges (flows). To create complex control flow graphs, parallelism and choices may be introduced. Each of the tasks is given a unique name. In addition, expected input and output parameters may be specified. Wanted service type(s) (Web service, P2P service, and Grid service) may also be specified.

The next two activities may be done in parallel:

- Semantics: By associating the task name and its input/output parameters to a domain, the chance of discovering appropriate service(s) for the task increases. Therefore the user:
 - First identifies available ontologies in order to import a relevant subset of its concept definitions into the Editor.
 - Then uses these concepts to annotate the service category and input/output parameters of each of the tasks.
- QoS: For each of the tasks, the user may specify the required QoS that must be offered by services executing the given task.

The final product of this phase is an abstract composition that serves as input to the next phase of the methodology.

In SODIUM, the first phase of the methodology is supported by the Composition Suite. The following screenshot (Figure 4.14) illustrates an abstract composition that reflects the motivating scenario described in Section 2.

<Figure 4.14 here>

The above figure depicts what happens when an emergency unit receives an emergency telephone call: it has to identify the location of an accident based on a caller's position, to find an ambulance which resides closer to that location and to dispatch it to this location. The outcome of the composition is a map with a route from the ambulance's current position to the accident location and a set of necessary command messages that are transmitted to the ambulance's personnel. Note that, the *Get Caller Info*, *Get Caller Position*, *Get Best Suited Ambulance*, and *Get Map* tasks in the abstract graph of the above figure are composite; thus they are decomposed into sub-tasks, as the little mark at the bottom right edge of each task denotes.

4.5.2 Phase 2: Discovery

The second phase handles discovery of services that can satisfy the requirements of each task. The user may either go back and forth between phases 1 and 2 of the process, or create the whole abstract composition at once and then move on to phase 2 to populate the abstract composition with services, hence transforming it to a concrete composition ready for execution. Since SODIUM aims at supporting dynamic service discovery at runtime, some tasks may be left without an associated service prior to runtime. Instead, these tasks are given a predefined service query, which will be executed at runtime.

Phase 2 is defined as a set of activities performed in sequence. The first activity is to make a semantic description (a query document) by using the query dialog for each of the selected tasks. The query documents are passed directly to the query engine that supports the discovery process based on matchmaking of semantic descriptions. It is assumed that a

service registry is available with the following information provided for each service:

- a service interface description,
- a semantic description that can be used for the matchmaking process, and
- QoS offered that can be exploited for the selection in the next phase.

The final product of phase 2 is a list of candidate services per task.

The service discovery phase of the methodology is handled by the USQL Engine in SODIUM. More specifically, for each task that requires a service, the developer opens the USQL Dialog (see Figure 4.15) from the Composition Visual Editor, creates a USQL query by setting a number of syntactic, semantic, and/or QoS search criteria, and executes the query by submitting it to the USQL Engine. The latter returns a list of matching services, which are appropriately displayed in the USQL Dialog (see Figure 4.15).

<Figure 4.15 here>

4.5.3 Phase 3: Selection

In phase 3, the goal is to narrow down and rank the services based on the QoS requirements. The QoS requirements contain two parts (see (Gronmo and Jaeger. 2005) for further details). The first part contains the absolute QoS constraints that are used to exclude services. The second part contains the optimization criteria that are used to rank the services. Instead of applying a “greedy-based” approach (i.e. ensuring QoS optimization of each task in isolation) for the QoS optimization of a service composition, another one which considers the composition as a whole or in sub-parts is proposed in (Gronmo and Jaeger. 2005). However, the SODIUM platform only supports QoS optimizing of single tasks in isolation.

The QoS-based prioritization and selection will return an assignment of a ranked list of candidate services for each task. Then, the composition

designer chooses one or more concrete services for each task in the abstract composition model. In this way, the developer finalizes the concrete composition model, which is the outcome of phase 3. Often it can be wise to choose more than one service for a task. This is the case if during run-time a service becomes temporarily or permanently unavailable. Then an alternative service performing the same task may compensate the unavailable one.

The outcome of phase 3 is a concrete service composition model with selected services. SODIUM platform supports this phase via the use of the Composition Suite, and the USQL dialog. Having executed a USQL query, the developer needs to go through the list of matching services and manually select the one(s) that seem to be more appropriate for the specific task.

4.5.4 Phase 4: Publishing

In the fourth and final phase, the concrete composition model is used to generate different descriptions about the composed service:

- a) a WSDL document describing the syntactic interface and its technical bindings,
- b) an executable flow document, and
- c) semantic Web service documents (e.g. OWL-S, WSML etc.) and documents describing the offered QoS.

The WSDL file can be automatically generated as shown in (Gronmo et al. 2004, 1), and the executable flow document can be generated as shown in (Kath et al. 2004). The semantic Web service documents can be automatically generated by the transformation tool described in (Grønmo, Jaeger and Hoff 2005). The transformation rules from the concrete composition model to QoS documents have not been specified within the context of SODIUM.

All the generated information (items a-c) is submitted to a Web service registry. Then, third parties can discover and use the composed service. Publishing the new composite web service and registering service descriptions to a web service registry is not supported by the SODIUM

platform, as it was considered outside the scope of the project. Thus within the context of SODIUM platform, only the generation of the executable USCL document is supported, by means of the VSCL2USCL Translator, which compiles and converts VSCL service compositions into USCL processes.

The following figure (Figure 4.16) summarizes the involvement of the SODIUM tools in each phase prescribed by the methodology. In addition to the four phases of the methodology, a final step regarding the execution of the service composition has been included to show the involvement of the SODIUM execution engine.

<Figure 4.16 here>

A detailed view and description of the SODIUM Service Composition Methodology is given in (Gronmo and Hoff 2007). Before concluding this section, we would like to remark that, although the intended users of the SODIUM methodology are primarily the users of the SODIUM platform, the principles of this methodology may be applied to service composition in general, as a way of working.

5 Related Work

SODIUM provides for the development of service-oriented applications by supporting the unified discovery and composition of heterogeneous services. Its contribution lies in the areas of visual service composition (VSCL language and editor), execution of service compositions (USCL language and execution engine), and in service discovery (USQL language and engine). In the following we compare the SODIUM results with existing work in these areas.

The SODIUM composition modeling languages i.e. VSCL and USCL accommodate the visual as well as the textual representation of service compositions. Similar to VSCL, the BPMN (BPMI 2004) notation supports the description of the control and data flow for a business process. BPMN is based on the UML activity diagram and facilitates the visual representation of business processes in a methodology-independent manner. VSCL has been influenced by BPMN and, in addition, it provides constructs which facilitate the description of Web, Grid and P2P services. Moreover, VSCL provides concepts which support the description of the

non-functional aspects of a service such as quality of service properties or associated semantics.

USCL on the other hand is an XML-based service composition language which similar to WS-BPEL (Alves et al. 2007) accommodates the description of the composition control and data flows. In contrast to WS-BPEL, USCL is independent of a specific service technology thus it may support the composition of services irrespectively of their type and underlying service provision platforms. Furthermore, USCL along with the USCL Engine accommodate the description of data transformations in a rich set of transformation techniques which include XSLT (Clark 1999), XQuery (Boag et al. 2007) as well as QVT (QVT-Merge Group 2005) so that the most optimal one in terms of run-time performance and development effort can be applied.

The use of a service-oriented approach to software development introduces the need for service discovery which is only partially addressed by other approaches to service composition development (Altintas et al. 2002). In the areas of Web, Grid, and P2P services, service discovery is performed with the use of custom and incompatible APIs and

discovery mechanisms offered by registries and networks (Clement et al. 2004) (Li 2001, 88). Over the last years, research in the area of service discovery was oriented towards improving and/or extending the existing discovery mechanisms (Paolucci et al. 2002) (Li et al. 2004). Moreover, many approaches were proposed to enhance the overall discovery process and precision (Klein and Bernstein 2004, 30) (Kozlenkov et al. 2006). Still, to the best of our knowledge, the SOC community lacks the means that would enable accessing and querying heterogeneous registries in a unified and standards-based manner. Moreover, exploitation of semantics and QoS within service descriptions proves to be a crucial part of service discovery. USQL and its enacting engine address these issues and constitute a stepping stone to the unification of the various heterogeneous service areas.

6 Conclusions

Service-Oriented Computing (SOC) is an emerging trend that promises to reform current software engineering approaches. Even though SOC aims at facilitating interoperability between the required components for building an application, it still has not come up to this expectation. The proliferation of various service-oriented technologies, e.g. Web, Grid and P2P services, which employ incompatible properties and characteristics, hinders the widespread utilization of those services. With the emerging need to compose such incompatible types of services, the support for their interoperation becomes an issue of high importance.

The SODIUM contribution in service interoperability and integration, presented in this chapter, provides an approach to unified discovery and composition of heterogeneous services. SODIUM provides a Generic Service Model, a set of languages supporting the visual specification of service compositions, the unified querying for services and the execution of heterogeneous service compositions, along with a set of tools that provide for the design, discovery and execution of service compositions. Although the SODIUM platform was originally implemented to support the unified discovery and composition of Web, Grid and P2P services, its

extensibility and modularity features which are exhibited by the whole set of components facilitate the future accommodation of other types of services as well.

Other projects of the Information Society Technologies (IST) Priority of the 6th Framework Programme (FP6) of the European Union which tackle issues similar to the ones tackled by SODIUM are SeCSE (<http://secse.eng.it>), ATHENA (<http://www.athena-ip.org/>), PLASTIC (<http://www-c.inria.fr/plastic/>) and AMIGO (<http://www.hitech-projects.com/euprojects/amigo/>). Specifically, the SeCSE project provides innovative technologies, methods and tools for supporting service-oriented computing, however it does not explicitly tackle heterogeneity in service discovery and composition which is the main focus of SODIUM. The ATHENA project addresses system and application interoperability and focuses mainly on bridging the gap between the business and IT whereas SODIUM concentrates on the interoperability between Web, P2P and Grid services in the development of service-oriented applications. The PLASTIC and AMIGO projects tackle service interoperability in pervasive computing environments by taking advantage of context information to

provide efficient service discovery in multi-network environments (the PLASTIC approach) and by establishing interoperability at a semantic level (the AMIGO approach) while SODIUM addresses service interoperability by offering a unified approach to the discovery and composition of existing heterogeneous services through the SODIUM languages and tools described in the previous sections.

The need to address heterogeneous types of services in a unified manner has been also identified by other organizations such as OMG, which has recently released a request for proposal (UPMS RFP (UPMS 2006)) for the provision of a service meta-model that will leverage the description of services. Two SODIUM results, the VSCL language and the GeSMO model have been submitted to OMG in order to address the needs of the UPMS RFP. Further to addressing the needs of standardization bodies the SODIUM results have been also exploited through other venues. Thus, we would like to note that GeSMO has been used as input by the aforementioned ATHENA project, whilst the USQL and the USQL engine are going to be extended and further utilized by the SeCSE project mentioned above.

It is worth mentioning that the development of the two pilot applications in the Crisis Management and the Health Care domains was greatly enhanced by the SODIUM Platform, despite the prototype phase of its provided tools. Thus, the SODIUM solution has been effectively applied to the construction of systems which integrate functionality from heterogeneous services in these two domains. Furthermore, the independence of the SODIUM platform components facilitated the customization of the application development environment, since developers were given the liberty to tailor the deployment according to their needs.

Concluding, we would like to mention that, all SODIUM tools are open source, provided under the LGPL license (<http://www.gnu.org/licenses/lgpl.html>) with the potential for integration in any commercial environment.

7 Acknowledgements

The work published in this chapter is partly funded by the European Community under the Sixth Framework Programme, contract FP6-04559 SODIUM. The work reflects only the author's views. The Community is not liable for any use that may be made of the information contained therein. Last, the authors would like to thank the rest of the SODIUM project partners, namely ATC S.A. for the project management and Locus and MEDISYSTEM for contributing in the development of the pilots, as well as the rest of the NKUA, SINTEF and ETHZ teams for their invaluable contribution in this work.

8 References

- Al-Ali, R. J., Rana, O. F., Walker, D. W., Jha, S., and Sohail, S. 2002. G-QoS: Grid Service Discovery Using QoS Properties. *Computing and Informatics*, Vol. 21, 363-382, 2002, ISSN 1335-9150.
- Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludscher, B., and Mock, S. 2004. Kepler: An Extensible System for Design and Execution of

Scientific Workflows. 16th Intl. Conference on Scientific and Statistical Database Management, June 2004, Santorini Island, Greece.

Alves, A., Arkin, A., Askary, S., Bloch, B., Curbera, F., Golland, Y., Kartha, N., Liu, C., König, D., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., and Yiu, A. eds. 2006. Web Services Business Process Execution Language Version 2.0. OASIS Standard, April 2007, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html> (accessed December 4, 2007).

Athanasopoulos, G., Tsalgatiidou, A., and Pantazoglou, M. 2006. Unified Description and Discovery of P2P Services. 2006 International Conference on Software and Data Technologies, September 11-14, in Setubal, Portugal.

Babik, M., Gatial, E., Habala, O., Hluchy, L., Laclavik, M., and Maliska, M. 2006. Semantic Grid Services in K-Wf Grid. Second International Conference on Semantics, Knowledge, and Grid, November 1-3, in Guilin, China.

- Ballinger, K., Ehnebuske, D., Ferris, C., Gudgin, M., Nottingham, and M., Yendluri, P., eds. 2004. Basic Profile Version 1.1, Web Service Interoperability Organization. <http://www.wsi.org/Profiles/BasicProfile-1.0-2004-04-16.html>
- Banks, T. 2006. Web services resource framework (WSRF) – Primer v1.2, OASIS, May 2006 <http://docs.oasis-open.org/wsrp/wsrp-primer-1.2-primer-cd-02.pdf>.
- Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., and Siméon, J., eds. 2007. XQuery 1.0: An XML Query Language, W3C, January 2007, <http://www.w3.org/TR/xquery/>.
- Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., eds. 2001. Web Services Architecture, W3C, February 2004, <http://www.w3.org/TR/ws-arch/>.
- Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., eds. 2001. Web Services Description Language (WSDL) 1.1. W3C, March 2001, <http://www.w3.org/TR/wsdl>.

- Clark, J. 1999. XSL Transformation (XSLT) Version 1.0, W3C, November 1999, <http://www.w3.org/TR/xslt>.
- Clement, L., Hately, A., von Riegen, C., and Rogers, T. eds. 2004. UDDI Version 3.0.2., OASIS, October 2004, http://uddi.org/pubs/uddi_v3.htm.
- Colombo, M., Di Nitto, E., Di Penta, M., Distante, D., and Zuccalà, M. 2005. Speaking a common language: a conceptual model for describing service-oriented systems. 3rd International Conference on Service Oriented Computing, (ICSOC 2005), December 17-20, 2005, in Amsterdam, Netherlands.
- Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S., 2004. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution, Version 1.0, The Globus Alliance, http://www.globus.org/wsrif/specs/ogsi_to_wsrif_1.0.pdf.
- EBXML, Electronic Business using Extensible Markup Language, <http://www.ebxml.org> (accessed June 2007).

- Farrell, J., and Lausen, H., eds. 2007. Semantic Annotations for WSDL and XML Schema. W3C, August 2007,
<http://www.w3.org/TR/2007/REC-sawSDL-20070828>.
- Foster, I., Kesselman, C., Lee, C., Lindell, B., Nahrstedt, K., and Roy, A. 1999, A Distributed Resource Management Architecture that Supports Advance, Reservation and Co-Allocation. International Workshop on QoS, June 1-4 1999, in London, England.
- Gerke, J., Reichl P., and Stiller, B. 2005. Strategies for service composition in p2p networks. Second International Conference on E-business and Telecommunication Networks, October 3-7, 2005, in Reading, UK.
- Gibbons, P., Karp, B., Ke, Y., Nath, S., and Seshan, S. 2003. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, Vol. 2 No. 4, October-December 2003: 22-33
- GLite, Lightweight Middleware for Grid Computing,
<http://glite.web.cern.ch/glite/>.
- Globus, The Globus Alliance, <http://www.globus.org/>.

Gnutella, Gnutella Com, <http://www.gnutella.com/>.

Grønmo, R. Skogan, D., Solheim, I., Oldevik, J.. 2004. Model-driven web service development. The International Journal of Web Services Research, Vol.1, No. 4, Oct-Dec 2004: 1-13

Grønmo, R., and Jaeger, M. 2005. Model-Driven Methodology for Building QoS Optimised Web Service Compositions. 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), June 15-17 2005, in Athens, Greece.

Grønmo, R., Jaeger, M., and Hoff, H., 2005, Transformations between UML and OWL-S. In Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA), November 7-10, 2005, in Nuremberg, Germany: 269-283

Grønmo, R., and Hoff, H. 2007. D19: SODIUM Service Composition Methodology. SODIUM, January 2007, <http://www.atc.gr/sodium>.

Heinis T., Pautasso C., Alonso G., Deak O., 2005. [Publishing Persistent Grid Computations as WS Resources](#), In Proc. of the 1st IEEE

International Conference on e-Science and Grid Computing

([e-Science 2005](#)), Melbourne, Australia

Hoff, H., Grønmo, R., Skogan, D., and Strand, A. 2005. D7: Specification of the Visual Service Composition Language (VSCL). SODIUM, June 2005, <http://www.atc.gr/sodium>.

Hoff, H., Skogan, D., Grønmo, R., Limyr, A., and Neple, T. 2006. D9: Detailed Specification of the SODIUM Composition Suite. SODIUM, February 2006, <http://www.atc.gr/sodium>.

Hoff, H., Hansen, T., and Skogan, D., 2006. D5: Specification of requirements for User Applications Part I: Requirements specification for the Locus Pilot. SODIUM, February 2006, <http://www.atc.gr/sodium>

Kath, O., Born, M., Eckert, K. P., Funabashi, M., and Hirai, C. 2004. Towards Executable Models: Transforming EDOC Behaviour Models to CORBA and BPEL. 8th International Enterprise Distributed Object Conference (EDOC 2004). September 20-24, 2004. Monterey, California, USA.

- Keller, A., and Ludwig, H. 2003. The WSLA Framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management, Special Issue on E-Business Management*, Vol. 11 No. 1, March 2003: 57-81.
- Klein, M., and Bernstein, A. 2004. Toward High-Precision Service Retrieval. *IEEE Internet Computing*, Vol. 8 No. 1, Jan-Feb 2004 : 30-36.
- Koutsonikola, V., and Vakali, A. 2004. LDAP: Framework, Practices, and Trends. *IEEE Internet Computing*, Vol. 8 No. 5, Sept-Oct 2004: 66-72.
- Kozlenkov, A., Fasoulas, F., Sanchez, F., Spanoudakis, G., and Zisman, A. 2006. A framework for architecture-driven service discovery. 2006 International Workshop on Service-Oriented Software Engineering, May 27-28, 2006, in Shanghai, China.
- Li, G. 2001, JXTA: a network programming environment. *IEEE Internet Computing*, May-Jun 2001, Vol. 5 No. 3: 88-95.

Li, Y., Zou, F., Wu, Z., and Ma, F. 2004. PWSO: A scalable web service discovery architecture based on peer-to-peer overlay network. 6th Asia-Pacific Web Conference on Advanced Web Technologies and Applications, April 14-17 2004, in Hangzhou, China.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. eds. 2004. OWL-S: Semantic Markup for Web Services. W3C, November 2004, <http://www.w3.org/Submission/OWL-S/>.

Mitra N. eds. 2003. SOAP Version 1.2 Part 0: Primer. W3C, June 2003, <http://www.w3.org/TR/soap12-part0/>.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. 2002. EDUTELLA: a P2P networking infrastructure based on RDF. 11th international Conference on World Wide Web WWW '02, May 07 - 11, 2002. in Honolulu, Hawaii, USA.

NESSI, Networked European Software & Services Initiative, <http://www.nessi-europe.com>.

- Newmarch, J. 2005. UPnP services and Jini clients. 2005 Conference on Information Systems: Next Generations, ICIS 2005, December 11-14 2005, in Las Vegas, Nevada, USA.
- UML, 2002, Unified Modeling Language: Superstructure, version 2.0, OMG, <http://www.omg.org/cgi-bin/doc?formal/05-07-04>.
- BPML, 2004, Business Process Modeling Notation (BPMN) Version 1.0, OMG, May 2004, <http://www.omg.org/docs/bei/05-08-07.pdf>.
- UPMS, 2006, UML Profile and Meta-model for Services (UPMS) Request for Proposal, OMG, <http://www.omg.org/docs/soa/06-09-09.pdf>.
- Pantazoglou, M., Tsalagatidou, A., and Athanasopoulos, G. 2006. Quantified matchmaking of heterogeneous services. In Proceedings of the 7th International Conference on Web Information Systems Engineering (WISE 2006), October 23-26 2006, in Wuhan, China, Springer LNCS 2455, 144-155.
- Pantazoglou, M., Tsalagatidou, A., and Athanasopoulos, G. 2006. Discovering web services and JXTA peer-to-peer services in a unified manner. In Proceedings of the 4th International Conference

on Service Oriented Computing (ICSOC 2006), December 4-7 2006, Chicago, USA, Springer LNCS 4294, 104-115.

Paolucci, M. Kawamura, T., Payne, T., Sycara K., 2002. Importing the Semantic Web in UDDI. 2002 Web Services, E-business and Semantic Web Workshop WES 2002, CAiSE 2002 International Workshop, May 27-28, 2002, in Toronto, Canada.

Papazoglou, M., and Georgakopoulos, D. 2003. Service-oriented computing. Communications of the ACM, Vol. 46 No. 10 October 2003: 24-28.

Pautasso C., Alonso G. 2004. [From Web Service Composition to Megaprogramming](#) In: Proceedings of the 5th VLDB Workshop on Technologies for E-Services (TES-04), Toronto, Canada.

Pautasso C., Alonso G. 2004b. [JOpera: a Toolkit for Efficient Visual Composition of Web Services](#). International Journal of Electronic Commerce (IJEC), 9(2):107-141

- Pautasso, C., Heinis, T., and Alonso, G. 2005. D6: Specification of the Unified Service Composition Language (USCL), SODIUM, June 2005, <http://www.atc.gr/sodium>.
- Pautasso, C., Heinis, T., Alonso, G., Pantazoglou, M., Athanasopoulos, G., Tsalgatidou, A., 2006, D10: Detailed Specification of SODIUM Runtime Environment, SODIUM, <http://www.atc.gr/sodium>
- Pautasso, C., Heinis, T., and Alonso, G. 2007. Autonomic Resource Provisioning for Software Business Processes. *Information and Software Technology*, Vol. 49 No. 1, January 2007: 65-80.
- QVT-Merge Group, 2004. Revised submission for MOF 2.0 Query/Views/Transformations. QVT-Merge Group, <http://qvtp.org/downloads/1.1/qvtpartners1.1.pdf>
- Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, P. 2005. Web Service Modeling Ontology. *Applied Ontology*, Vol. 1 No. 1, 2005: 77-106.
- Sahin, O. D., Gerede, C. E., Agrawal, D., El Abbadi, A., Ibarra, O., and Su, J. 2005. SPiDeR: P2P-Based Web Service Discovery. 3rd

International Conference on Service Oriented Computing (ICSOC 2005), December 13-15 2005, in Amsterdam, Netherlands.

Tian, M., Gramm, A., Ritter, H., Schiller, J. 2004. Efficient selection and monitoring of QoS-aware web services with the WS-QoS framework. 2004 IEEE/WIC/ACM International Conference on Web Intelligence, September 20-24 2004, in Beijing, China.

Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., and Ma, W. 2003. Management Applications of the Web Service Offerings Language (WSOL). 15th Conference On Advanced Information Systems Engineering (CAiSE'03), June 16-20, 2003, in Velden, Austria.

Tsalgaidou, A., Athanasopoulos, G., Pantazoglou, M., Floros, V., Koutrouli, E., and Bouros, P. 2005. D4: Generic Service Model Specification. SODIUM, June 2005, <http://www.atc.gr/sodium>.

Vogels, W. 2003. Web Services Are Not Distributed Objects. IEEE Internet Computing, Vol.7 No. 6, Nov.-Dec. 2003: 59-66.

Vu, L. H., Hauswirth, M., and Aberer, K. 2005. Towards P2P-based Semantic Web Service Discovery with QoS Support. Workshop on

Business Processes and Services (BPS), September 5 2005, in Nancy,
France.