4 Unified Discovery and Composition of Heterogeneous Services: The SODIUM Approach

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

4.1 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; 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, such as HTTP.

Web Services (Booth et al. 2004) is the best-known instantiation of the SOC paradigm. Other instantiations include Grid Services (Czajkowski et al. 2004), which emerged from the scientific application domain, and Peer-to-Peer (P2P) Services (Li 2001, 88), which originated from community-oriented systems and applications, such as instant messaging and file sharing.

All these services, regardless 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 divers 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, owing 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. 2006) to deal with the discrepancies between Web Services.

The 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 to 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 toward the discovery and composition of heterogeneous types of services; such an approach can relieve 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 Web Services, P2P Services, and Grid Services. Nevertheless, the openness and extensibility of the SODIUM solution allow for the support of other service types, such as 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. Specifically, SODIUM lies in the bottom and middle layers of the SOC road map proposed in Papazoglou and Georgakopoulos (2003, 24), since it provides innovative solutions for the description, discovery, and composition of heterogeneous services. It also touches the upper layer of the SOC road map, 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 describe a motivating scenario that illustrates the need to integrate heterogeneous services (section 4.2). 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 4.3). Then, we exemplify the SODIUM approach and present the main elements of the SODIUM platform (section 4.4). Finally, we compare our work with similar approaches (section 4.5) and present some concluding remarks (section 4.6).

4.2 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 three to five minutes, brain cells start to die, and after approximately 15 minutes the damage is irreversible. Thus, it is vital that properly equipped ambulances and other rescue units be located within a 15-minute 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 owing 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 requirements mentioned above can be very beneficial. Some examples of services that provide useful functionality for the implementation of the above scenario are the following:

1. Web Services providing weather information, such as temperature and precipitation, or traffic conditions from roadside speed sensors and video surveillance cameras
2. Grid Services providing driving route calculations, weather forecasting information, and ''response range'' calculations based on current positions and conditions
3. 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 was mentioned in the introduction, due to the incompatibility of the existing service types; some of the incompatibilities that need to be dealt with are presented in the following section.

The rest of the chapter presents the SODIUM approach, which provides a unified solution to the discovery and composition of heterogeneous services (i.e., Web, Grid, and P2P services) that has been used for implementing the motivating scenario described above.

4.3 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. 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. A detailed description of the state-of-the-art analysis is available in Tsalgatidou et al. (2005).

4.3.1 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:

1. Syntactic information, which refers mainly to the structure of the service interface, and the provided operation signatures
2. Semantic information, which is provided to describe the capability of the service (i.e., its offered functionality)
3. Quality information, which describes the nonfunctional, qualitative characteristics of the service, such as its reliability, availability and performance.

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, 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 specification (Farrell and Lausen 2007), are the most prominent approaches in this direction. These protocols were further extended to meet the requirements of the Grid, as they were also utilized to provide semantic annotations 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), the WS-QoS (Tian et al. 2004), and the 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) cater for 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 has 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.

4.3.2 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 specifications (UDDI; Clement et al. 2004) and the ebXML standard (EBXML 2002) are commonly used for publishing and discovering Web Services. On the other hand, Grid infrastructures, such as Globus (http://www.globus.org) or the latest gLite (http://glite.web.cern.ch/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.

4.3.3 State of the Art in Service Invocation

Web Services are traditionally communicated through the exchange of SOAP messages (Mitra 2003) over proven network protocols, such as HTTP and SMTP. 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 network (http://www.gnutella.com) 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 a peer, or use an existing proxy peer to be able to invoke P2P services.

4.3.4 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 engine (Altintas et al. 2004) 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.

4.4 The SODIUM Approach to the 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. Specifically, the SODIUM solution comprises the following:

1. 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
2. A set of languages:

- 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

3. A set of tools and supporting middleware:
• 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
• The USCL execution engine, which supports the execution and monitoring of heterogeneous service compositions that are expressed with the USCL format.

Figure 4.1 depicts the overall architecture of the SODIUM solution and also outlines the interactions between the constituent tools.

The SODIUM solution is divided in two main subsets, the composition suite and the runtime environment, which support the design and the runtime phases of the development process, respectively. Nevertheless, as figure 4.1 shows, the constituent tools are loosely coupled and communicate through document exchanges by means of well-defined

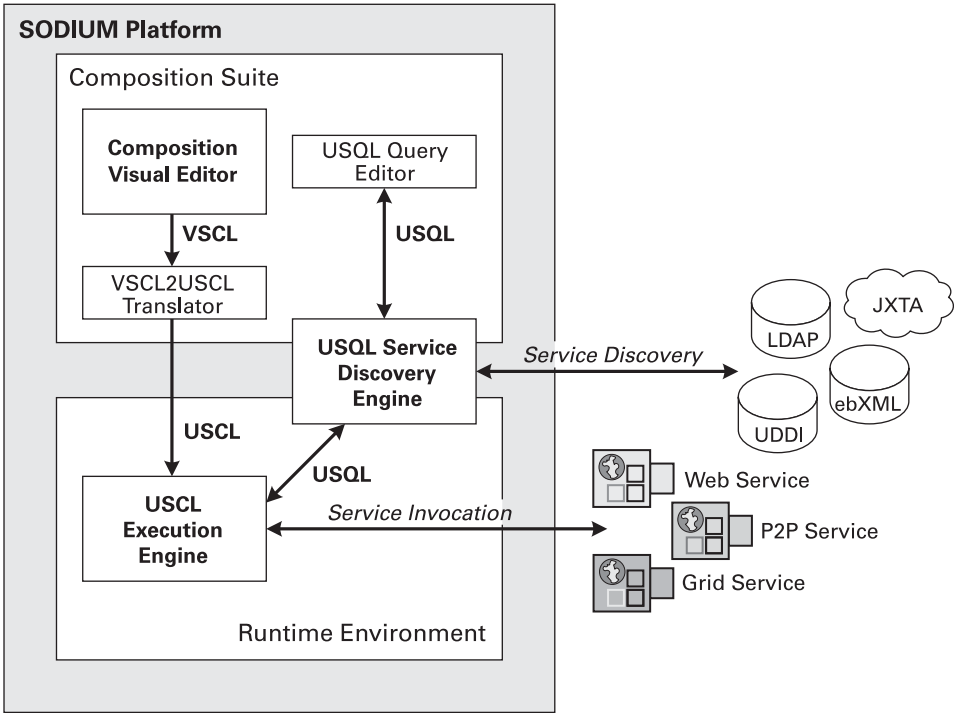

Figure 4.1
The SODIUM Platform

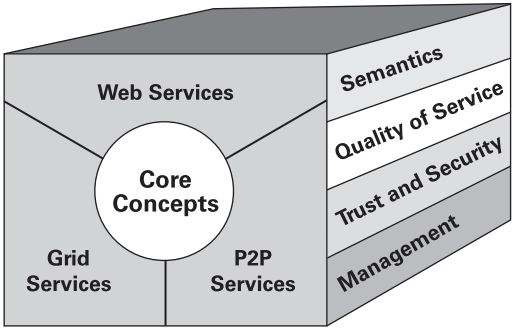

Figure 4.2
Architecture of the SODIUM Generic Service Model

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 present and describe the SODIUM results.

4.4.1 The Generic Service Model

The generic service model (GeSMO) provides the conceptual basis upon which the SODIUM languages and tools were developed. Its specification was driven by the results of a thorough investigation of the state of the art in Web, P2P, and Grid service technologies outlined above. In general, the model is characterized by its generality, abstraction, simplicity, modularity, expressiveness, and extensibility.

GeSMO has adopted a layered architecture (see figure 4.2).

1. The *core layer*, which models all common concepts among the investigated service types (i.e., Web, P2P, and Grid services)
2. 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)
3. A number of layers, orthogonal to the core and extensions layers, which model additional cross-cutting features, such as semantics, quality of service, trust and 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. Figure 4.3 outlines the different views that have been defined by GeSMO along with their interdependencies.

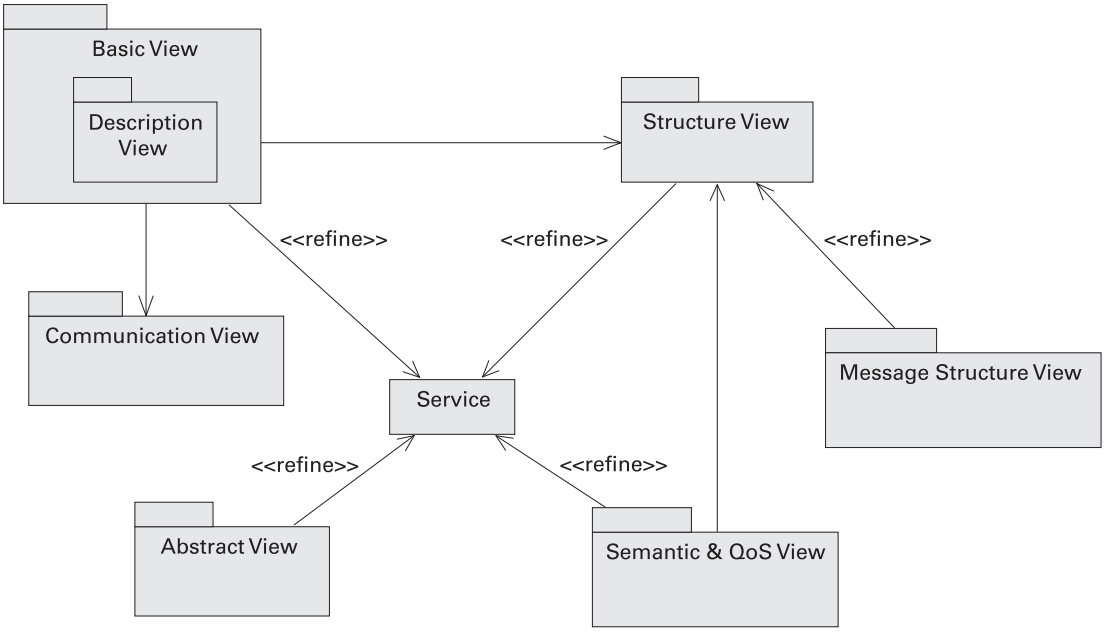

Figure 4.3
Views defined by the SODIUM Generic Service Model (GeSMO)

These service views are briefly described as follows.

1. Abstract view: It looks into the service notion from an abstract point of view and tries to identify its relationships with elements of the software engineering field.
2. Basic view: It 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 subviews.
3. Description view: This view focuses on the elements that are related to the description of a service.
4. Structure view: It identifies the structural elements that a service may comprise.
5. Semantics & QoS view: This view identifies the elements of the service model that may have semantic and QoS annotations.
6. Message Structure view: It provides a look into the structure and the elements of messages that are exchanged among a service and its clients.
7. Communication view: It 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 figures 4.4, 4.5, and 4.6.

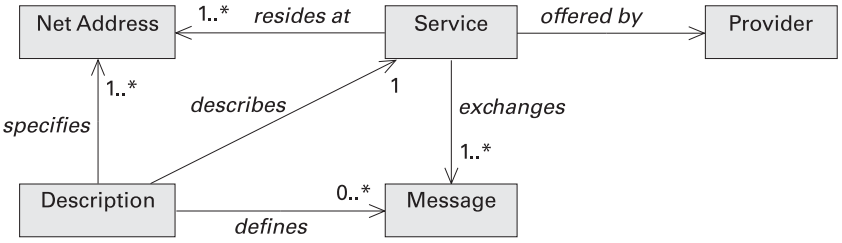

Figure 4.4
The Basic service view defined by GeSMO

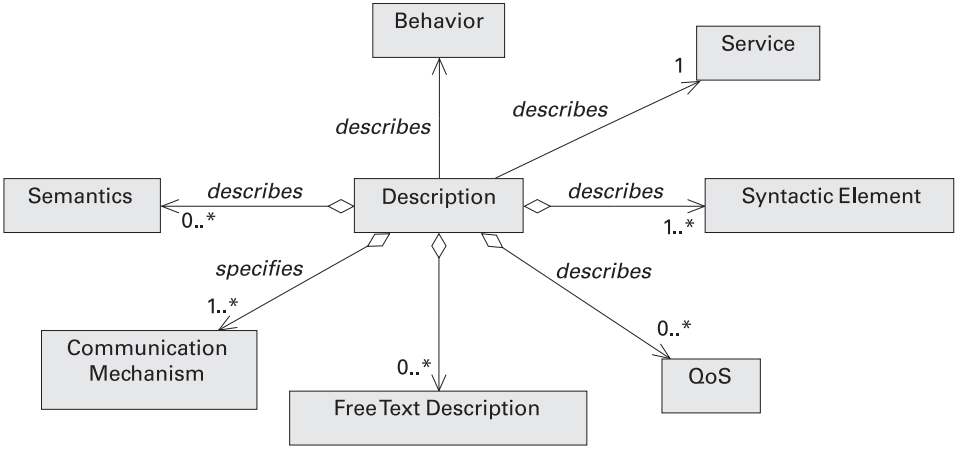

Figure 4.5
The Description service view defined by GeSMO

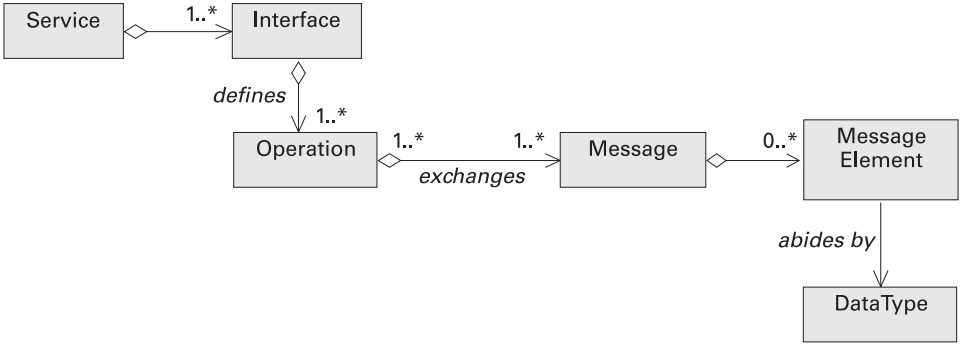

Figure 4.6
Structure view of a service as defined by GeSMO

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.).

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.

The structure model defines the set of structural elements that a service may be broken down into. A detailed description of all views defined in GeSMO is available in Tsalgatidou et al. (2005).

GeSMO shares many common concepts with the SeCSE Conceptual Model (Colombo et al. 2005). However, in GeSMO we focused primarily 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 multipurpose tool. More specifically:

1. It was used as a conceptual basis for the specification of the three SODIUM languages (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 utilized in SODIUM for the description and invocation of JXTA P2P services.
2. It provided a common point of reference that facilitated communication and knowledge exchange among the project stakeholders.
3. Its abstraction and extensibility drove specific design decisions regarding the SODIUM tools, such as the plug-in–based architecture adopted by most of them.

In conclusion, the GeSMO specification currently caters for Web, P2P, and Grid services, nevertheless its extensibility allows for seamlessly accommodating other types of services.

4.4.2 Visual Service Composition

SODIUM supports the visual service composition through the Visual Service Composition Language (VSCL) and the Composition Suite.

The Visual Service Composition Language (VSCL) The VSCL is based on UML v2.0 (UML 2002), and supports the visual representation of service compositions which leverage Web, Grid, and P2P services. Specifically, the VSCL uses the concepts of the UML activity

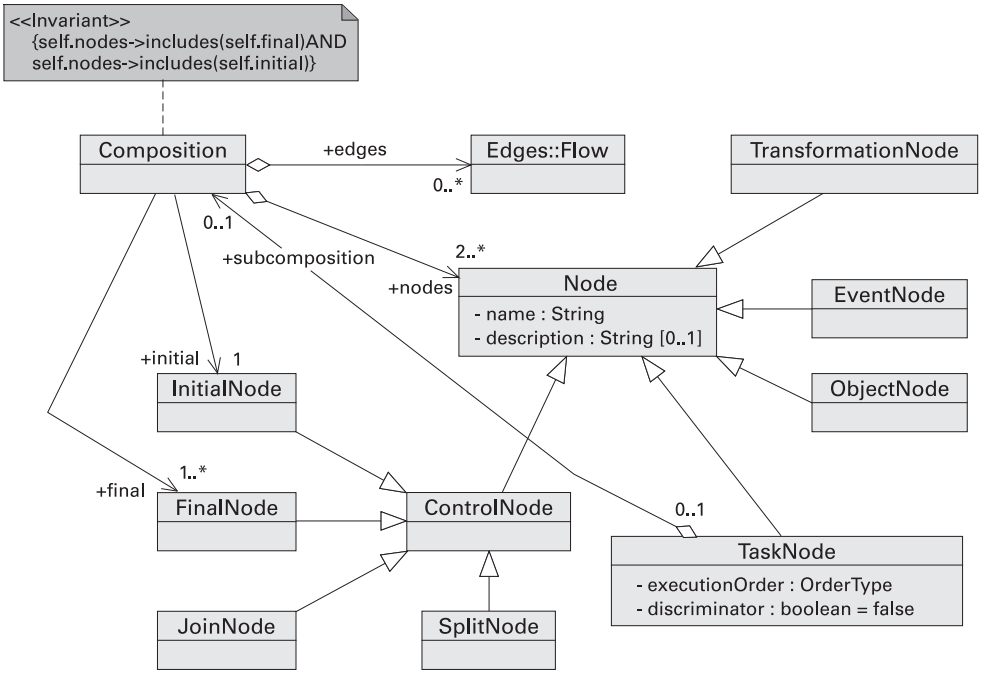

Figure 4.7
The VSCL conceptual meta-model

diagram as a basis for the provision of appropriate stereotypes that define all the necessary composition primitives. Nevertheless, the VSCL has a conceptual meta-model (see figure 4.7) that is independent of UML and other existing graphical modeling languages, however, it can be realized by a UML profile.

According to the VSCL meta-model, a service composition consists of nodes and flows/edges. The nodes are task nodes, control nodes, object nodes, event nodes, and transformation nodes; the different kinds of flow are used to specify flow of control and data between nodes. Hence, the main concepts of the VSCL 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 a task has information about which service(s) to execute for this specific task. The strength of the task-based approach is that there is one composition graph, rather than two, incorporating both concrete and abstract parts. After services have been selected, the abstract part may still be used in order to check if new available and better-suited services have emerged. The possibility for defining

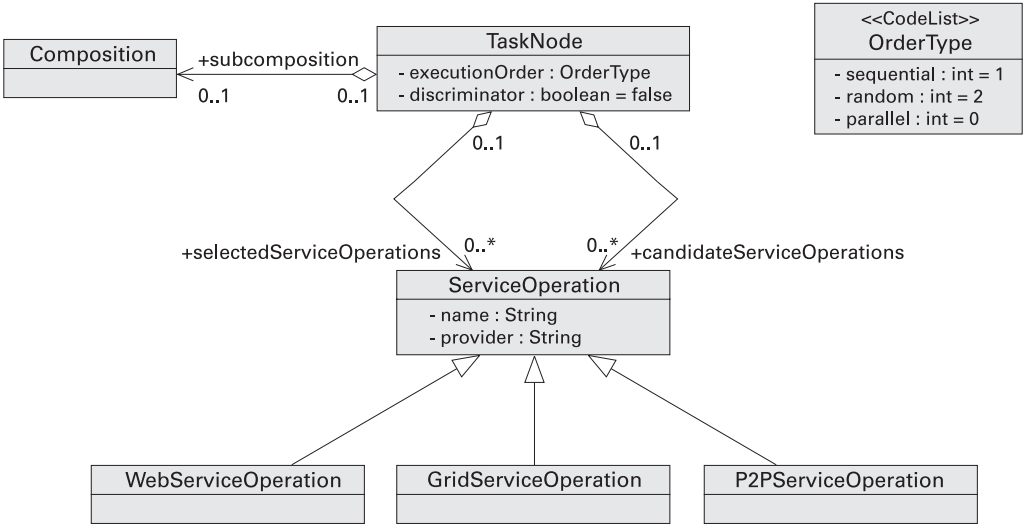

Figure 4.8
Relationship between tasks and services in the VSCL meta-model

transformations between the output(s) of a task and the input(s) of its successor task has also been included (Grønmo, Jaeger, and Hoff 2005). A task may itself be a composition, and thus it is decomposed into subtasks.

A heterogeneous composition in VSCL can consist of tasks executed by different kinds of services. The types defined in the context of SODIUM are P2P Services, Web Services, and Grid Services (see figure 4.8).

As shown in figure 4.8, one or more services may be selected to realize/execute a specific task. When more than one service operation is 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 random order, the last two being 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, such as QoS characteristics, may vary between them.

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

The SODIUM Composition Suite The SODIUM Composition Suite consists of the following main subcomponents:

1. The composition visual editor, for editing and analyzing service compositions with the use of the VSCL

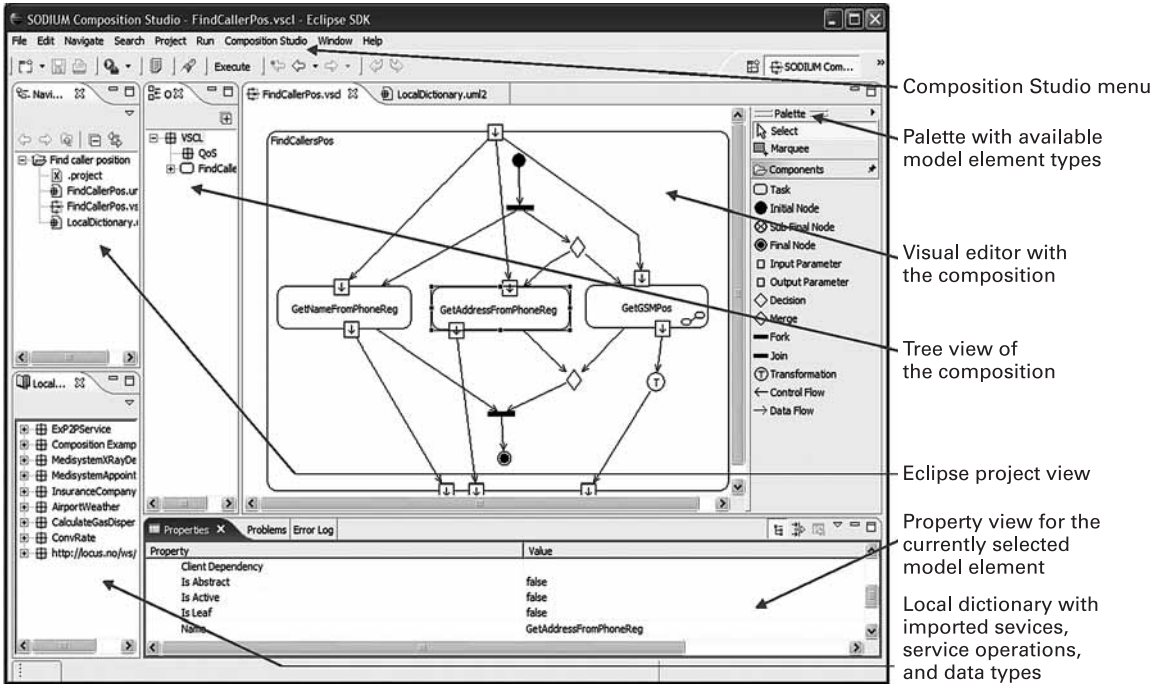

Figure 4.9
The SODIUM Composition Suite

2. The VSCL2USCL translator, which translates a service composition from a graphical notation (VSCL) to a lexical XML-based notation (USCL)
3. The USQL dialogue, 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). It 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 following:

1. Specification of service compositions in multiple levels of abstraction
2. Static analysis of service compositions described in VSCL
3. 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:

1. Top-down approach for a task-oriented focus where tasks are identified, but no candidate service operations have yet been identified/selected
2. Bottom-up approach for a service-oriented focus when service operations to use are preknown
3. 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:

1. The first step in constructing VSCL graphs is to break down the composition into tasks, which interoperate in order to achieve the overall goal. This initial composition model of tasks is called an abstract model because no selected concrete services have yet been identified.
2. 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 runtime, thus enabling the dynamic binding of services that are discovered, selected, and invoked at runtime.

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

4.4.3 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 below.

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 enables service requesters to express their requirements in an accurate and intuitive manner. Moreover, with USQL, requesters can express their requirements toward a service and/or 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, the USQLrequest and USQLresponse. The former

```
<?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>
```

Figure 4.10
Example of a USQL request document

is used to express service type-independent queries, whereas the latter is used to convey the results of the query execution.

The snippet in figure 4.10 is an example USQL request document, in accordance with the motivating scenario presented earlier in the chapter.

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 popu-

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL xmlns="urn:usql" version="1.1">
 <USQLResponsequeryId="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>
```

Figure 4.11
Example of a USQL response document

late 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 in figure 4.11.

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 of the corresponding entries in the USQL response is strongly associated with the type of the referred services and is adequate to enable their invocation.

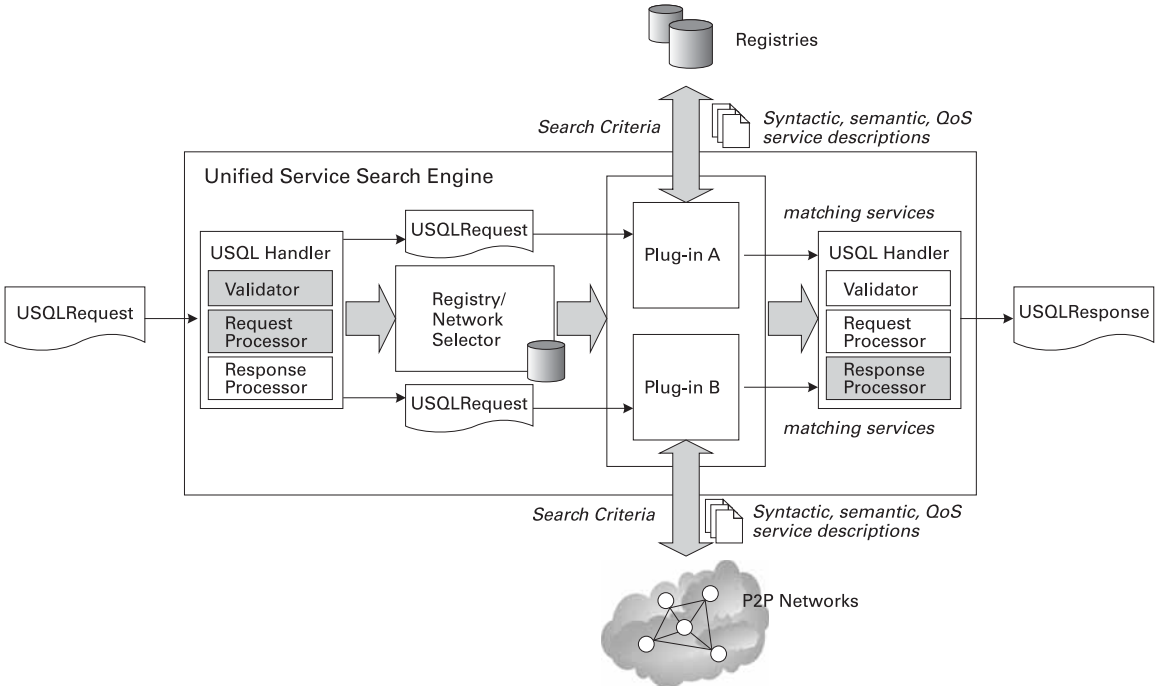

Figure 4.12
Architecture of the USQL Engine

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 figure 4.12.

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 and 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 forward it to appropriate registry plug-in components, which are coordinated by the registry selector and run in parallel. The service descriptions 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 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 suite and is used by developers to formulate USQL queries, access the USQL Engine, and discover services at design time. At runtime, 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 (2006b, 104). The definition of the matchmaking algorithm that has been implemented by the engine can be found in Pantazoglou, Tsalgatidou, and Athanasopoulos (2006a, 144).

4.4.4 Execution of Heterogeneous Service Compositions

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

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, that are responsible for executing each composition task, in order to enhance the reusability of both. Compositions are modeled as processes whose structure defines the data and control flow dependencies between the service invocations, as well as the required exception handling behavior. Components are modeled as services, an abstraction that makes the mechanism used to access the corresponding implementation transparent.

Figure 4.13 depicts the structure of a USCL document.

The root element (USCL) of a USCL document can contain a set of process, service, and service type definitions. In practice, the service definitions and the required service type declarations are 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, 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 alternative ways to invoke the functionality provided by the service. Access

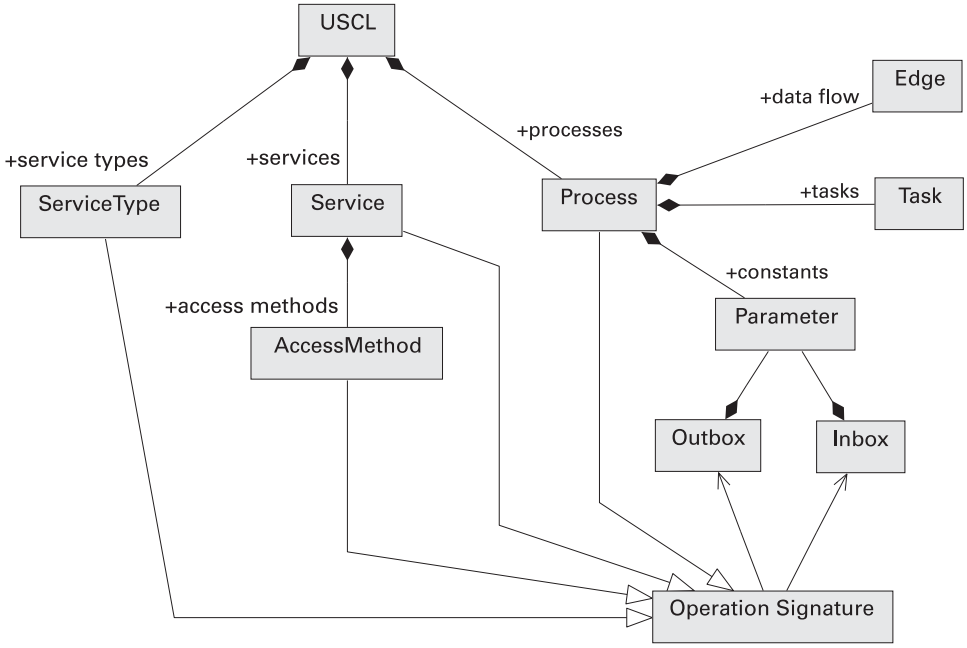

Figure 4.13
Overview of the structure of a USCL document

methods also have input and output parameters conforming to the template defined in the service type. By definition, the input and output parameters of an access method, and the ones belonging to the corresponding service type, are considered system parameters (Pautasso and Alonso 2004a).

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 one 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. Nontrivial 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.

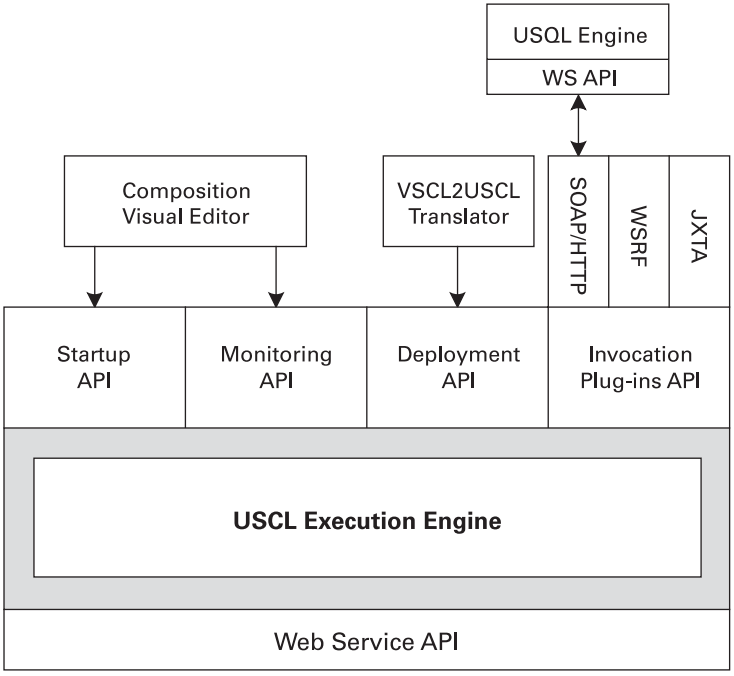

Figure 4.14
APIs provided by the USCL Execution Engine

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).

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 and 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) (see figure 4.14), which are used for communication with the rest of the SODIUM tools, as well as for the invocation of Web, P2P, and Grid services.

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, the 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 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 and logging). Once a process has been instantiated, it should be possible to identify it among others concurrently running inside the engine. Thus, after a process instance has been started, it is associated with a unique ID and then it is 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 a 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 metadata (current execution state of a task, performance profiling information, error messages, and so on). Logs can 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 metadata) 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, and 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 makes feasible the utilization of the USCL Execution Engine as a distinct component outside the SODIUM platform.

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

4.4.5 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. (See figure 4.15.)

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

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 refined 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. Expected service type(s) (Web Service, P2P Service, and Grid Service) can also be specified.

The next two activities may be done in parallel:

1. *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 (1) identifies available ontologies in order to import a relevant subset of its concept definitions into the editor and then (2) uses these concepts to annotate the service category and input/output parameters of each of the tasks.
2. *QoS* For each of the tasks, the user may specify the 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.

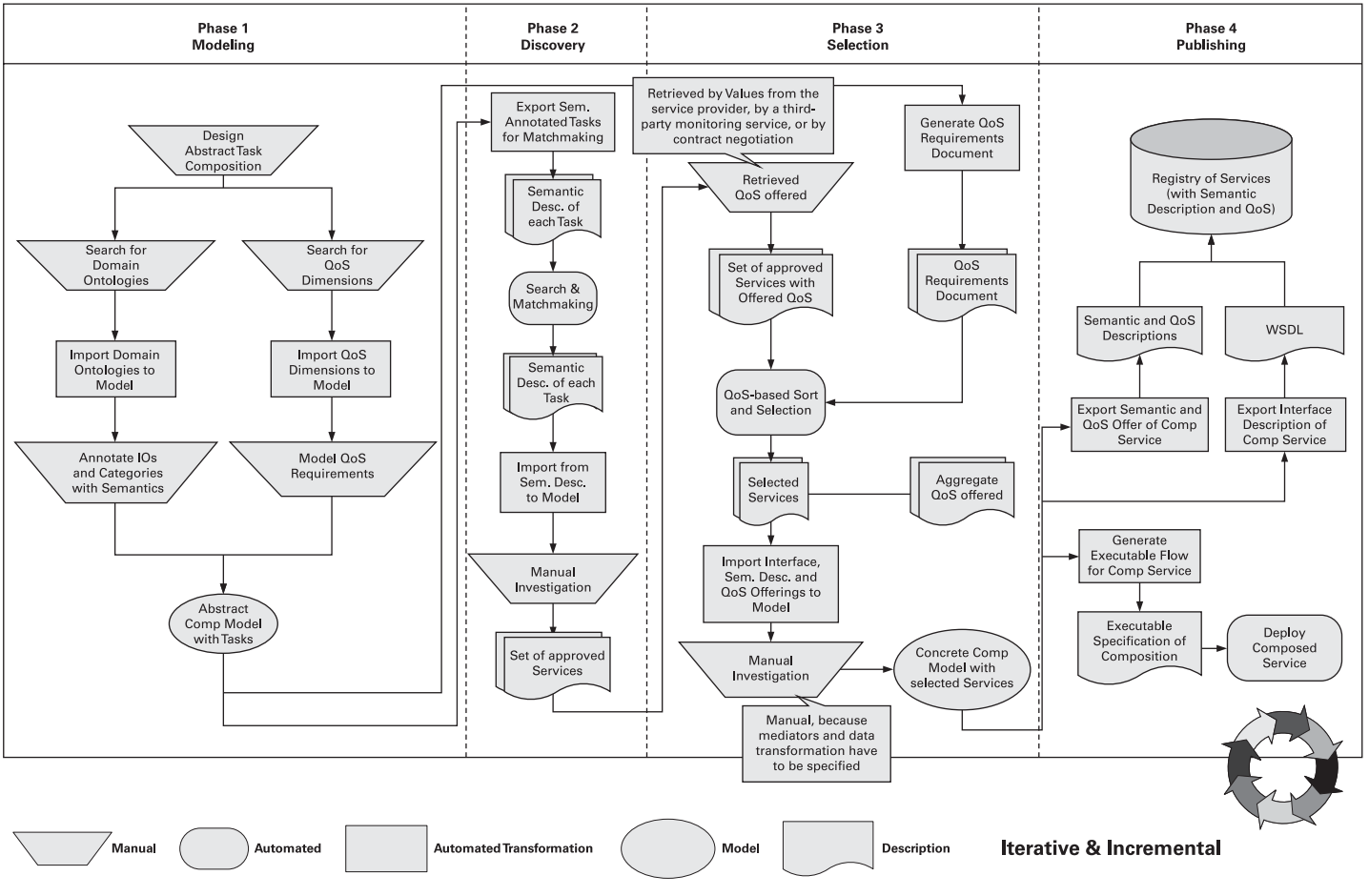

Figure 4.15
The SODIUM Heterogeneous Service Composition methodology

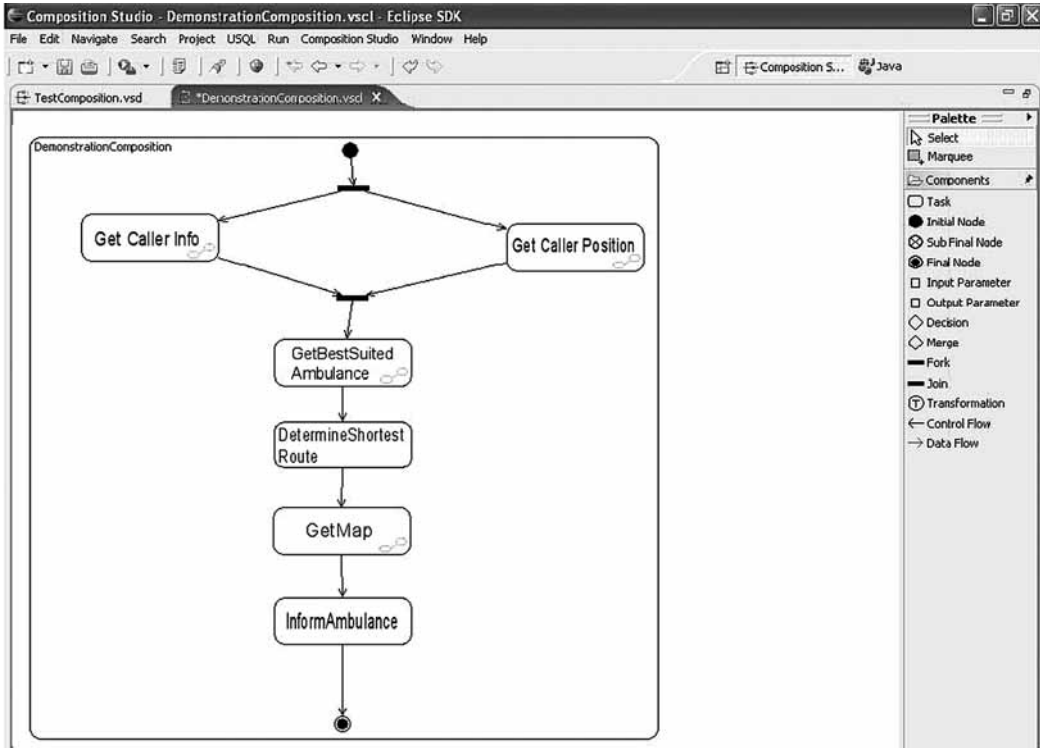

Figure 4.16
Abstract service composition created with the SODIUM Composition Suite

In SODIUM, the first phase of the methodology is supported by the Composition Suite. The screenshot in figure 4.16 illustrates an abstract composition that reflects the motivating scenario described in section 4.2.

Actually, the figure 4.16 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 is closest to that location, and to dispatch it to that 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 are composite; thus they are decomposed into subtasks, as the little mark at the bottom right edge of each task denotes.

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, thereby transforming it into 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 dialogue 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:

1. A service interface description
2. A semantic description that can be used for the matchmaking process
3. 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. More specifically, for each task that requires a service, the developer opens the USQL Dialogue (see figure 4.17) 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 Dialogue (see figure 4.17).

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 Grønmo 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), another approach could be used, as the one proposed in Grønmo and Jaeger (2005), which considers the composition as a whole or in subparts. However, the SODIUM platform supports only QoS optimization 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 runtime a service becomes temporarily or permanently unavailable. Then, an alternative service, performing the same task, may compensate for the unavailable one.

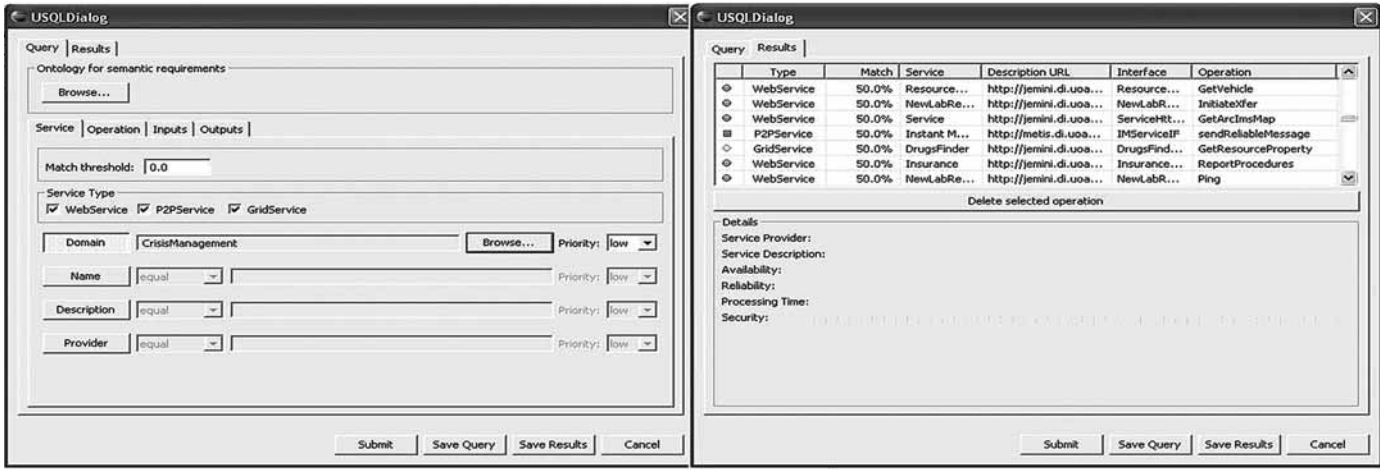

Figure 4.17
The USQL Dialog is used both for editing USQL queries and displaying search results

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

Phase 4: Publishing In the fourth and final phase, the concrete composition model is used to generate different descriptions about the composed service:

1. A WSDL document describing the syntactic interface and its technical bindings
2. An executable flow document
3. 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 Grønmo 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). All the generated information may be submitted to a Web Service registry where, third parties can discover and use the composed service. The transformation rules from the concrete composition model to QoS documents have not been specified within the context of SODIUM.

Publishing 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 the SODIUM platform, only the generation of the executable USCL documents is supported, by means of the VSCL2USCL translator, which compiles and converts VSCL service compositions into USCL processes.

Figure 4.18 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.

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

4.5 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 com-

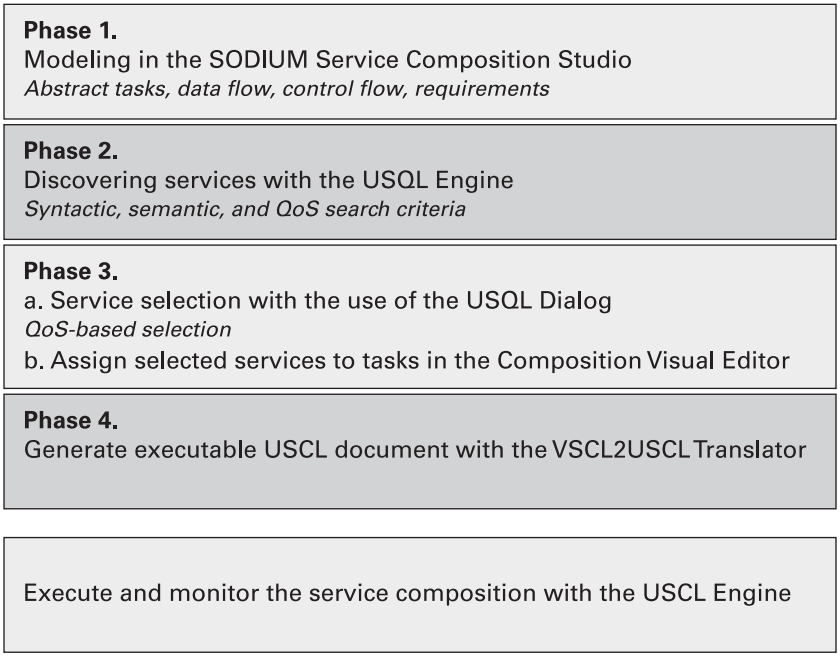

Figure 4.18
SODIUM methodology and SODIUM tools

positions (USCL language and execution engine), and service discovery (USQL language and engine). In the following we compare the SODIUM results with existing work in these areas.

The two SODIUM composition modeling languages (i.e., the VSCL and the 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. The 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.

The USCL, on the other hand, is an XML-based service composition language which, like 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, and thus it may support the composition of services regardless of their type and underlying service provision platforms. Furthermore, the USCL, along with the

USCL Engine, accommodates the description of data transformations in a rich set of transformation techniques which include XSLT (Clark 1999), XQuery (Boag et al. 2007), and QVT (QVT-Merge Group 2004), so that the optimal one in terms of runtime 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. 2004). 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 toward 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.

4.6 Conclusions

Service-oriented computing is an emerging trend that promises to reform current software engineering approaches. Even though SOC aims at facilitating interoperability between the components required 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 that are exhibited by the whole set of its components facilitate the accommodation of other types of services as well.

Other projects of the Information Society Technologies (IST) Priority of the 6th Framework Program (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/). The SeCSE project provides innovative technologies, methods, and tools for supporting service-oriented computing, but 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 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), whereas 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 also been identified by other organizations such as OMG, which 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 also been exploited through other venues. Thus, we would like to note that GeSMO has been used as input in the ATHENA project, whilst the USQL and the USQL engine have been extended and further utilized in the SeCSE project.

The SODIUM solution has been effectively applied to the construction of two pilot applications which integrate functionality from heterogeneous services in the crisis management and the health care domains. It is worth mentioning that the development of these two applications was greatly facilitated and improved by the SODIUM platform, despite the prototype phase of the provided tools. 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.

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.

Acknowledgments

The work published in this chapter was partly funded by the European Community under the Sixth Framework Program, contract FP6–04559 SODIUM. The work reflects only the authors' views. The Community is not liable for any use that may be made of the information contained therein. The authors would like to thank the rest of the SODIUM project

partners, namely: ATC, S.A., for the project management, Locus and MEDISYSTEM for contributing to the development of the pilots and the rest members of the NKUA, SINTEF, and ETHZ teams for their invaluable contribution to this work.

References

Al-Ali, R. J., Rana, O. F., Walker, D. W., Jha, S., and Sohail, S. 2002. G-QoSM: Grid Service discovery using QoS properties. *Computing and Informatics* 21: 363–382.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., and Mock, S. 2004. Kepler: An extensible system for design and execution of scientific workflows. In *16th International Conference on Scientific and Statistical Database Management*, pp. 423–424. Santorini, Greece.

Alves, A., Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C., König, D., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., and Yiu, A. eds. 2007. 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.

Athanasopoulos, G., Tsalgatidou, A., and Pantazoglou, M. 2006. Unified description and discovery of P2P services. 2006. In *First International Conference on Software and Data Technologies.* Setubal, Portugal. INSTICC Press.

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

Ballinger, K., Ehnebuske, D., Ferris, C., Gudgin, M., Nottingham, M., and Yendluri, P., eds. 2006. Basic Profile Version 1.1, Web Services Interoperability Organization. http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html.

Banks, T. 2006. Web Services Resource Framework (WSRF)—Primer v1.2. OASIS, May. http://docs.oasis-open.org/wsrf/wsrf-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. http://www.w3.org/TR/xquery/.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., eds. 2004. Web Services Architecture. W3C Working Group Note, February. http://www.w3.org/TR/ws-arch/.

BPMI. 2004. Business Process Modeling Notation (BPMN) Version 1.0. OMG, May. http://www.omg.org/docs/bei/05-08-07.pdf.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., eds. 2001. Web Services Description Language (WSDL) 1.1. W3C note, March. http://www.w3.org/TR/wsdl.

Clark, J. 1999. XSL Transformation (XSLT) Version 1.0. W3C Recommendation, November. 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. 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. In *Service Oriented Computing, Computing ICSOC 2005: 3rd International Conference.* 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/wsrf/specs/ogsi_to_wsrf_1.0.pdf.

EBXML 2002, OASIS/ebXML Registry Services Specification v2.0. http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf.

Farrell, J., and Lausen, H., eds. 2007. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, August. 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 reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service.* London, England.

Gerke, J., Reichl, P., and Stiller, B. 2005. Strategies for service composition in P2P networks. In *Proceedings of the Second International Conference on E-Business and Telecommunication Networks.* Reading, U.K. INSTICC Press.

Gibbons, P., Karp, B., Ke, Y., Nath, S., and Seshan, S. 2003. IrisNet: An architecture for a world-wide sensor web. *IEEE Pervasive Computing* 2, no. 4: 22–33.

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

Grønmo, R., and Jaeger, M. 2005. Model-driven methodology for building QoS optimised Web Service compositions. In *5th IFIP International Conference on Distributed Applications and Interoperable Systems* (DAIS 2005). Athens, Greece. LNCS 3543. Springer.

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

Grønmo, R., Skogan, D., Solheim, I., and Oldevik, J. 2004. Model-driven Web Service development. *International Journal of Web Services Research* 1, no. 4 (October–December): 1–13.

Heinis, T., Pautasso, C., Alonso, G., and Deak, O. 2005. Publishing persistent Grid computations as WS resources. In *Proceedings of the 1st IEEE International Conference on E-Science and Grid Computing* (e-Science 2005), pp. 328–335. Melbourne, Australia.

Hoff, H., Grønmo, R., Skogan, D., and Strand, A. 2005. D7: Specification of the Visual Service Composition Language (VSCL). SODIUM, June. 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.

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

Kath, O., Blazarenas, A., Born, M., Eckert, K.-P., Funabashi, M., and Hirai, C. 2004. Towards executable models: Transforming EDOC behaviour models to CORBA and BPEL. In *8th International Enterprise Distributed Object Computing Conference* (EDOC 2004). Monterey, Calif.

Keller, A., and Ludwig, H. 2003. The WSLA framework: Specifying and monitoring service level agreements for Web Services. *Journal of Network and Systems Management* 11, no. 1 (March): 57–81.

Klein, M., and Bernstein, A. 2004. Toward high-precision service retrieval. *IEEE Internet Computing* 8, no. 1: (January–February): 30–36.

Koutsonikola, V., and Vakali, A. 2004. LDAP: Framework, practices, and trends. *IEEE Internet Computing* 8, no. 5 (September–October): 66–72.

Kozlenkov, A., Fasoulas, F., Sanchez, F., Spanoudakis, G., and Zisman, A. 2006. A framework for architecture-driven service discovery. In *2006 International Workshop on Service-Oriented Software Engineering.* Shanghai, China.

Li, G. 2001. JXTA: A network programming environment. *IEEE Internet Computing* 5, no. 3 (May–June): 88–95.

Li, Y., Zou, F., Wu, Z., and Ma, F. 2004. PWSD: A scalable Web Service discovery architecture based on peer-to-peer overlay network. In *6th Asia–Pacific Web Conference on Advanced Web Technologies and Applications.* Hangzhou, China. LNCS 3007.

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

Mitra, N., ed. 2003. SOAP Version 1.2 Part 0: Primer. W3C, June. 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. In *11th international Conference on World Wide Web* (WWW '02). Honolulu.

NESSI, Networked European Software & Services Initiative. http://www.nessi.com/Nessi/.

Newmarch, J. 2005. UPnP services and Jini clients. In *Proceedings of the 2005 Conference on Information Systems: Next Generations* (ICIS 2005). Las Vegas, Nev.

Pantazoglou, M., Tsalagatidou, A., and Athanasopoulos, G. 2006a. Quantified matchmaking of heterogeneous services. In *Proceedings of the 7th International Conference on Web Information Systems Engineering* (WISE 2006). Wuhan, China. LNCS 2455, pp. 144–155.

Pantazoglou, M., Tsalgatidou, A., and Athanasopoulos, G. 2006b. 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). Chicago. LNCS 4294, pp. 104–115.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K., 2002. Importing the Semantic Web in UDDI. In *Proceedings of E-Services and the Semantic Web Workshop (WES 2002), CAiSE 2002 International Workshop.* Toronto.

Papazoglou, M., and Georgakopoulos, D. 2003. Service-oriented computing. *Communications of the ACM* 46, no. 10 (October): 24–28.

Pautasso, C., and Alonso, G. 2004a. From Web Service composition to megaprogramming. In *Proceedings of the 5th VLDB Workshop on Technologies for E-Services* (TES-04). Toronto.

Pautasso, C., and Alonso, G. 2004b. JOpera: A toolkit for efficient visual composition of Web Services. *International Journal of Electronic Commerce* 9, no. 2: 107–141.

Pautasso, C., Heinis, T., and Alonso, G. 2005. D6: Specification of the Unified Service Composition Language (USCL). SODIUM, June. http://www.atc.gr/sodium.

Pautasso, C., Heinis, T., and Alonso, G. 2007. Autonomic resource provisioning for software business processes. *Information and Software Technology* 49, no. 1: 65–80.

Pautasso, C., Heinis, T., Alonso, G., Pantazoglou, M., Athanasopoulos, G., and Tsalgatidou, A. 2006. D10: Detailed Specification of SODIUM Runtime Environment. SODIUM. http://www.atc.gr/sodium.

QVT-Merge Group. 2004. Revised Submission for MOF 2.0 Query/Views/Transformations RFP. QVT-Merge Group. http://www.omg.org/cgi-bin/apps/doc?ad/04-04-01.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* 1, no. 1: 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. In *3rd International Conference on Service-Oriented Computing* (ICSOC 2005). Amsterdam.

Tian, M., Gramm, A., Ritter, H., and Schiller, J. 2004. Efficient selection and monitoring of QoS-aware Web Services with the WS-QoS framework. In *2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 152–158. Beijing.

Tosic, D. V., Pagurek, B., Patel, K., Esfandiari, B., and Ma, W. 2003. Management Applications of the Web Service Offerings Language (WSOL). In *15th Conference on Advanced Information Systems Engineering* (CAiSE'03). Velden, Austria. Published in Advanced Information Systems Engineering LNCS 2681 (2008), pp. 1029–1052.

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

UML. 2002. Unified Modeling Language: Superstructure, Version 2.0. OMG. http://www.omg.org/cgi-bin/doc?formal/05–07–04.

UPMS. 2006. UML Profile and Meta-model for Services (UPMS) Request for Proposal. OMG. http://www.omg.org/docs/soa/06–09–09.pdf.

Vogels, W. 2003. Web Services are not distributed objects. *IEEE Internet Computing* 7, no. 6 (November–December): 59–66.

Vu, L.-H., Hauswirth, M., and Aberer, K. 2005. Towards P2P-based semantic Web Service discovery with QoS support. Presented at Workshop on Business Processes and Services (BPS). Nancy, France. Published in *BPM Workshops*, LNCS 3812 (2006), pp. 18–31.