Developing Scientific Workflows from Heterogeneous Services

A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou
Dept. of Informatics & Telecom.
University of Athens (NKUA)
Athens 15784, Greece
{atsalga, gathanas, michaelp} @di.uoa.gr

C. Pautasso, T. Heinis
Department of Computer Science
Swiss Federal Institute of Technology
ETH Zentrum, 8092 Zurich, Switzerland
{pautasso, heinist} @inf.ethz.ch

R. Grønmo, Hjørdis Hoff, Arne-Jørgen Berre
SINTEF Information and Communication Technology
P.O.Box 124 Blindern, N-0314 Oslo, Norway
{roy.gronmo, hjordis.hoff, arne.j.berre}@sintef.no

M. Glittum
Locus S.A. (LOCUS)
Leif Weldingsvei 6-8, Sandefjord, Norway
mg@locus.no

S. Topouzidou
Athens Technology Center (ATC)
Rizariou 10, Halandri, Athens, Greece
s.topouzidou@atc.gr

ABSTRACT

Scientific WorkFlows (SWFs) need to utilize components and applications in order to satisfy the requirements of specific workflow tasks. Technology trends in software development signify a move from component-based to service-oriented approach, therefore SWF will inevitably need appropriate tools to discover and integrate heterogeneous services. In this paper we present the SODIUM platform consisting of a set of languages and tools as well as related middleware, for the development and execution of scientific workflows composed of heterogeneous services.

1. INTRODUCTION

Scientific WorkFlows (SWFs) have emerged as a response to problems encompassing scientific problem solving techniques and workflow features [4]. The main characteristic of SWFs is the need for integration of heterogeneous systems and components that provide specific functionalities or data. This integration is not an easy task and it is one of the main research issues in this area.

Current trends in software engineering signify a move from component-based to service-oriented development, therefore, we believe that the development of SWFs will be benefited by a service-oriented approach.

There are many types of services, including Web, Peer-to-Peer (P2P) and Grid services, which employ different/incompatible architectural models, protocols, and standards for service description, discovery and composition. However, to our best of knowledge, there is no infrastructure or tools available for facilitating the integration and interoperability of such services.

In this paper we present a platform called SODIUM (Service Oriented Development In a Unified fraMework) which provides a set of languages, tools and corresponding middleware, for modeling and executing SWFs composed of heterogeneous services (Web, P2P and Grid services).

In a service-oriented approach, the various workflow tasks can be executed by various types of services (rather than being programmed from scratch). Initially, these services may not be known. Therefore, following a top-down approach for developing a SWF, we need to model requirements for services which will satisfy specific workflow tasks. SODIUM provides a Visual Composition Language (VSCL) and an associated VSCL Editor which support this modeling task. The following step is to search for appropriate services which can satisfy the requirements of each task. There exist a large number of heterogeneous services with incompatible protocols and standards which makes their discovery a cumbersome task. SODIUM provides a Unified Service Query Language (USQL) and an associated query engine that support the discovery of heterogeneous services in a unified way. Both semantic and Quality-of-Service (QoS) information are utilized to improve the discovery. However, the USQL language and its associated engine are not responsible for maintaining such information; rather, they rely on existing service descriptions (e.g. OWL-S, WSDL-S, WS-QoS, etc.), which are maintained by service providers and are published in the various registries or networks. Selected services substitute the requirements in each workflow task resulting in a concrete SWF model. Next, VSCL graphs are mapped to USCL (Unified Service Composition Language) descriptions which are executed by the SODIUM workflow execution engine. The main purpose of the latter is to

provide an efficient, reliable and scalable platform for executing SWFs composed of heterogeneous services.

In the next section, we present a motivating scenario that illustrates the need for the integration of heterogeneous services in SWFs. Section 3 depicts the overall architecture of the SODIUM platform and the constituent components. Section 4 compares our work with existing related approaches and finally Section 5 gives concluding remarks.

2. MOTIVATING SCENARIO

Our motivating scenario is from the Crisis Management area, where an important task is to determine how to get to a crisis location 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 approx. 15 minutes there are permanent damages. Thus, it is vital that properly equipped ambulances and other rescue units are within a 15-minutes range at all times and places.

This requirement is very difficult to be satisfied due to the vast set of parameters such as accident/injury probability, population density and composition, accessibility, time of day/week/month/year, weather conditions, hospital locations and many others, which need to be considered.

A SWF providing for this scenario could take advantage of information and resources offered by existing or emerging services. Such services may be:

- 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, historical incident 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.

It is therefore imperative that a SWF supporting this scenario is able to integrate heterogeneous services such as the ones mentioned above.

3. SODIUM Architecture

Figure 1 provides an overview representation of the SODIUM platform. As we can see, SODIUM introduces a let of languages, tools and associated middleware.

The languages introduced by the SODIUM platform are:

- A *Visual Service Composition Language (VSCL)* for designing workflow graphs at multiple levels of details.
- A *Unified Service Composition Language (USCL)* to facilitate the construction of executable workflows composing of and invoking various types of services.
- A *Unified Service Query Language (USQL)* to cater for the open and unified discovery of heterogeneous services enabling the preservation of the autonomy of service registries.

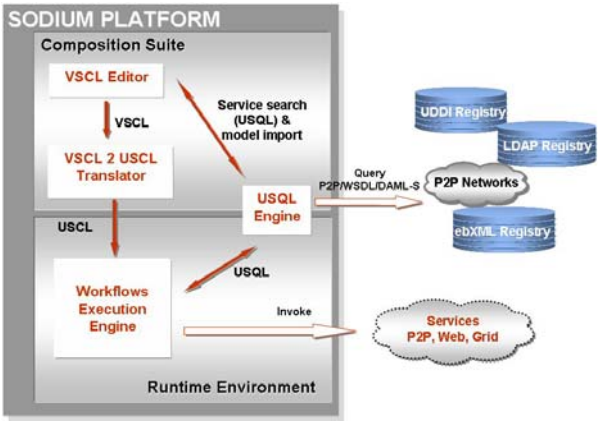

Figure 1: SODIUM Platform overview architecture

With respect to tools and middleware, the SODIUM platform provides the following:

- A Visual Service Composition Suite comprising:
  - A *Visual Editor* enabling the construction and analysis of VSCL Graphs.
  - A Translation mechanism enabling the transformation of the VSCL graphs into USCL.
- A Run Time Environment comprising components necessary for the execution of the composite services:
  - A search engine, namely *USQL Engine*, which submits queries to heterogeneous service registries, utilizing USQL.
  - A *Workflow Execution Engine* which executes workflows written in USCL. The workflow engine invokes the different types of services and/or submits USQL queries to the USQL Engine and subsequently invokes the returned services.

In the following, we describe the three main components of the SODIUM platform, i.e. the Visual Editor, the USQL Engine and the Workflow Execution Engine.

3.1 Visual Editor

The Visual Editor is the SODIUM tool for creating SWF models as VSCL graphs. The first step in constructing VSCL graphs is to break down the workflow into tasks, which can interoperate in order to finally achieve the overall goal. This workflow model of tasks is called an *abstract model* since there are no selected concrete services identified in this phase. This abstract model is used to search for appropriate candidate services to realize each of the abstract tasks. When chosen services are selected for each abstract task, the result is a *concrete model*. VSCL is based on the Unified Modeling Language (UML) [22].

There are extensions to handle Web services, P2P services, Grid services, semantics, QoS, and the relationship between the abstract and the concrete model.

Figure 2 shows an abstract workflow model based on the crisis management scenario presented in section 2. It is abstract since we have only identified the tasks without any concrete service implementations. Each task is defined with enough semantic and QoS information so as to enable a proper search for candidate services. Each task is represented as a UML activity with a stereotype indicating the type of service we are looking for. The goal is to create a service-oriented workflow that monitors the position of ambulance vehicles and sends messages to the ambulances so that they can reposition themselves to achieve a better coverage in a given area. The simplified scenario proposes an ideal ambulance coverage to mean at least one ambulance with a maximum 15-minutes response range for any given position within the area. More realistic scenarios would take into account issues like population density and accident frequency that require better ambulance coverage in certain areas.

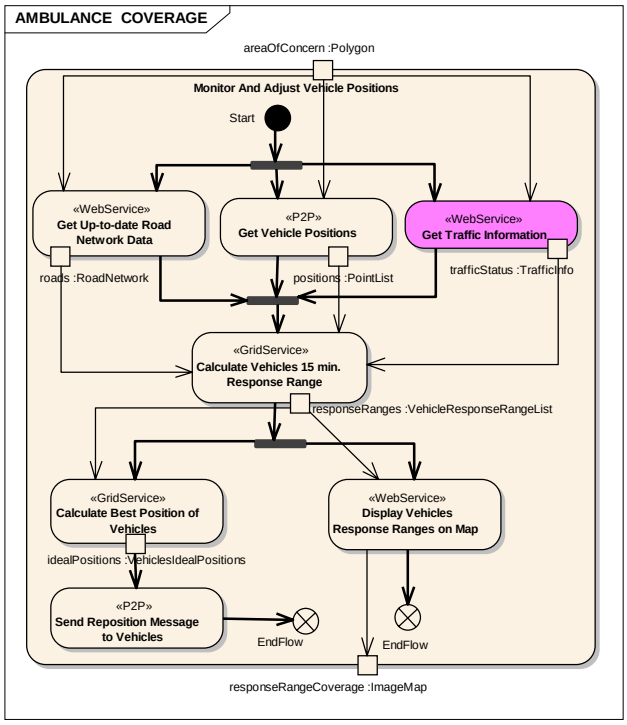

Figure 2: Workflow with heterogeneous services

The workflow takes as input an *areaOfConcern* (i.e. geographical region of concern) which is dispatched at the same time to three different services running in parallel: (1) a Web service returning up-to-date road network data, (2) a P2P service returning the current position of all ambulance vehicles and (3) a Web service returning current traffic information. Results of the three services are forwarded to a Grid service, which calculates the 15-minutes response range for each vehicle. This information is sent to a Web service used to display it on a map covering the original *areaOfConcern* and to a Grid service to calculate the ideal repositioning of the vehicles. These positions are sent to each vehicle by invoking a P2P service. The map produced by the Web service is the output data object returned by the workflow.

All tasks in the abstract workflow model are associated with Quality-of-Service (QoS) requirements and semantically grounded definitions. This improves the precision of the query for concrete services since inappropriate services are omitted from the search results.

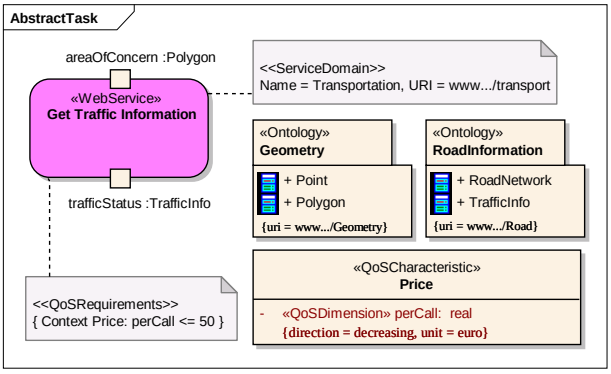

Figure 3: Detailed task with QoS requirement and semantics

Figure 3 illustrates how we can model QoS and semantics for the *Get Traffic Information* task of Figure 2. A note is attached to the task with a pre-defined stereotype *QoSRequirement*. The QoS requirement specifies that the 'price per call' of the service must be less-than-or-equal-to 50 euro. The 'price per call' is a QoS concept that needs to be defined within a UML class with the stereotype *QoSCharacteristic*. The *Get Traffic Information* task is also semantically annotated by defining a service domain and by defining the input and output data objects as semantic types. The semantic types of the data objects are placed in a UML package stereotyped as *Ontology*. The ontology package corresponds to an existing domain ontology identified by the UML tagged value *uri*. The *areaOfConcern* input data object is assigned the *Polygon* type which is a semantic type defined within the *Geometry* ontology, represented by a UML package. It should be noted, that any domain ontology could be practically used as the source for semantically annotating tasks. VSCL, along with USQL, are flexible enough so as to accommodate different ontologies. Nevertheless, an upper domain ontology has been established and is maintained by the USQL Engine, within the scope of SODIUM. More details regarding the upper ontology are provided in the next section (3.2), as well as in [23].

The semantic modeling of services is an extension to UML provided by the SODIUM project. For the QoS notation we

recommend the use of the OMG's UML profile for modeling QoS and Fault Tolerance [14].

A concrete workflow model extends the abstract workflow model with chosen services (that have been discovered via the USQL Engine) for each of the abstract tasks. The concrete workflow model can then be automatically translated into the lexical USCL language which can be executed by the workflow execution engine [24]. The next section explains how the USQL engine is used to discover appropriate services to register in the concrete workflow model.

3.2 USQL Engine

The *USQL Engine* is used for discovering services by searching heterogeneous service registries. The engine implements *USQL (Unified Service Query Language)*, an XML-based query language providing all necessary structures and elements to cater for the unified and standards-based service requests over heterogeneous registries and/or networks. USQL allows requestors to formulate expressive, semantically enhanced queries, which reflect their actual needs and requirements. Queries may also be enriched with the specification of QoS criteria, so as to bring the resulting services as close as possible to the demands of real-world scientific applications.

An example USQL request reflecting the service requirements of Figure 3 is depicted in Figure 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="1.0">
  <find_servicesRequest>
    <Where>
      <Service serviceType="WebService">
        <ServiceDomain ontologyURI="...">Transportation</ServiceDomain>
        <ServiceQoS>
          <ServicePrice currency="EUR" context="perCall" valueIs="equalOrLess">
            50
          </ServicePrice>
        </ServiceQoS>
        <Operation>
          <OperationName>GetTrafficInformation</OperationName>
          <Input>
            <name>areaOfConcern</name>
            <semantics ontologyURI="...">Polygon</semantics>
          </Input>
          <Output>
            <name>trafficStatus</name>
            <semantics ontologyURI="...">TrafficInfo</semantics>
          </Output>
        </Operation>
      </Service>
    </Where>
  </find_servicesRequest>
</USQL>
```

Figure 4: A sample USQL request

The main concept underlying the USQL Engine framework is the abstraction regarding registry details, from the requestor's perspective. This is achieved with the adoption of a domain-centric categorization of the various supported registries, depending on the service advertisements they host. Domain information provided by the requestor is exploited by the engine so as to identify, access and query the appropriate registries in a transparent manner.

The USQL Engine accomplishes this type of domain-driven registry categorization with the introduction of an *Upper Ontology*, which provides a set of classes along with their properties and relations, thus allowing for the application of reasoning within the course of service discovery. The upper ontology associates registries with domains, and concepts with domains. Concepts may be either operations or data that are relative to a specific domain, and are used for semantically annotating USQL requests, as well as in the matchmaking process. Moreover, an ontology mapping mechanism is used to ensure interoperability, as far as support for service descriptions abiding by different ontology frameworks and/or vocabularies is concerned. Maintenance of the Upper Ontology is part of the USQL Engine configuration process, so that service requestors can focus on the thorough and consistent expression of their queries.

The USQL engine architecture is distinguished by its high degree of openness and extensibility, which is achieved by using plug-in mechanisms in order to accommodate virtually any type of service, registry, as well as their governing protocols and standards. The plug-ins used for this purpose can be integrated in a flexible manner, so as to enable different configurations and to broaden the range of supported registries.

The USQL Engine is a crucial component within the context of a scientific workflow engine facilitating the discovery of heterogeneous services that are used for solving a scientific problem. The service discovery results are used to transform abstract workflow graphs conveying orchestrated tasks and their respective requirements into concrete service workflows that are then executed by the SODIUM workflow engine. Alternatively, the USQL Engine may be used at run-time for the discovery of appropriate services to fulfill specific tasks within the workflow.

3.3 Workflow Execution Engine

The engine receives USCL documents containing the definitions of the workflows and exposes an API for initiating, monitoring, and managing their execution. In addition, workflows can be exposed as Web services [10]; hence, it is possible to access them from client applications through standard interfaces.

The core infrastructure used to run USCL workflows is described as follows. The execution of a workflow begins with a request sent through the corresponding API of the engine. The engine API queues the request and handles it by creating and enacting a new workflow instance. The current state of the execution of a process is used to determine which tasks should be invoked based on the control and data flow dependencies that are triggered by the completion of the previous tasks.

Invocation of the tasks equals to invocation of their respective services and is achieved with the appropriate plug-ins. These provide a concrete implementation of the service invocation mechanisms and protocols. After the

execution of the task has been finished, the state of the corresponding process is updated and the execution of the workflow continues. Execution of the workflow is separated from the execution of its tasks since these operations have a different level of granularity and quite often the execution of a task may last significantly longer than the time taken for scheduling it. Thus, the engine supports parallel invocation of multiple tasks belonging to the same process. Furthermore, a slow task does not affect the execution of other processes running concurrently because these two operations are handled asynchronously by different threads.

The SODIUM workflow engine provides for the persistence of the state information about the process instances following a design that has been influenced by many requirements, such as performance, reliability, and portability across different data repositories. Access to the state information of the workflow instances is provided in terms of a key-value data model 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 which complies with what is required in a scientific workflow setting, state management can be optimized to keep only a subset of all of the workflow instances in memory and, for instance, swap workflows whose execution has been completed to secondary storage, in a so-called process history space. In this way, the SODIUM engine gives access to the state of past executions to enable workflow profiling and optimization, caching of already computed results as well as lineage tracking analysis. Additionally, this functionality can be implemented with several different storage technologies along the full spectrum of the persistence versus performance overhead trade off. In the context of the crisis management example, this functionality helps to optimize the performance of the workflow execution as follows. The expensive re-computation of the 15-minutes response range can be avoided if neither the vehicle positions, nor the traffic conditions have changed with respect to a previous execution. Similarly, it is possible to avoid the potentially large download of the same road network data for repeated executions of the workflow with the same input *areaOfConcern*.

The architecture of the SODIUM workflow execution engine employs plug-ins to support an open set of heterogeneous service invocation mechanisms.

3.4 Web Service Plug-in

The first of such plug-ins is responsible for the invocation of standards-compliant Web services described by WSDL [2]. These services are remotely accessible through the SOAP protocol [1]. By utilizing the information returned by a USQL response[1] which provides the URI of a WSDL document describing the selected service, the port, service and operation name as well as the arguments for the operation, the Web service plug-in invokes the Web service by dynamically assembling a SOAP message and sending it to the service provider. Upon receiving a response, the results are extracted and passed back to the workflow engine. For example, in order to invoke the "Get Traffic Information" Web service of Figure 3, the plug-in uses the URI to the WSDL returned by the USQL response to the USQL query of Figure 4, selects the port bound to the SOAP protocol and sends a request message to the *GetTrafficInformation* operation. This message contains the XML serialization of the input parameter *areaOfConcern*.

3.5 Grid Service Plug-in

The Grid service plug-in has been developed in accordance to the Web Services Resource Framework (WSRF) [3] specifications, which have been defined as to shift from the rather stateless paradigm of Web services to the stateful model of Grid services. To do so, WSRF loosely couples a Web service with a stateful resource and provides well-defined methods to access and manage its state.

Thus, in contrast to using only the service URI as is the case for Web services, Grid services also require a resource instance identifier. Therefore, in addition to the arguments passed to the Web service plug-in, the Grid service plug-in also requires a resource instance identifier.

3.6 P2P Service Plug-in

As far as P2P services are concerned, JXTA [19] is one of the most well known platforms for the development of P2P service-oriented applications, therefore JXTA services have been appointed as a supported type of P2P services, with respect to the SODIUM platform. Nevertheless, the mechanisms and facilities provided by the JXTA platform for the description and invocation of JXTA P2P services are rather limited or vague. To address this, we follow an approach based on the use of enhanced description documents and a library of java classes facilitating the binding and invocation of JXTA P2P services.

The pursued approach does not modify the infrastructure or protocols that are used by the JXTA services; rather, it enhances them so as more advanced discovery and binding mechanisms can be used. Furthermore, the P2P service model is not infringed, since the workflow engine actually

[1] A USQL query is submitted to the USQL engine either at design time (in which case service-related information returned by the USQL response is integrated in the VSCL graph, then mapped to a USCL document and subsequently utilized by the execution engine for invoking the service) or at run time, thus supporting dynamic discovery of services.

becomes a node in the respective P2P network, through the plug-in.

4. RELATED WORK

SODIUM provides for the service-oriented development of SWFs. Its contribution lies in the areas of SWF modeling (VSCL with editor), execution (USCL with execution engine), and in service discovery (USQL with query engine). In the following we compare the SODIUM results with existing work in each of these areas.

Similar to KEPLER [5][12] SODIUM's workflow modeling languages (VSCL and USCL) are primarily based on data flow constructs, as this is the most common representation used to model scientific computations. While KEPLER is confined to Web services, SODIUM allows for the discovery and composition of Grid, as well as P2P services, providing support for semantics and QoS metadata which can be used for optimized service selection [9]. Triana [21] is a framework for composing scientific applications. Unlike USCL, the supported workflow language does not provide explicit support for control constructs, while its visual representation is not standard-based with respect to VSCL. For more information on other research projects (e.g., Askalon, Unicore, Karajan) related to scientific workflow management, we refer the reader to [13] and [25].

Considering the scale of the data and the computational resources required by scientific applications, scientific workflow systems must take into account resource management and scheduling features typical of high performance computing environment and tools [7], [11]. In this regard, the Pegasus workflow mapping system [8] shares with SODIUM the idea of mapping a workflow between different levels of abstraction, where an abstract workflow is dynamically bound with run-time information describing the Grid resources that are used to execute its activities. A different kind of mapping is related to providing data transformation capabilities so that mismatching scientific data sources and incompatible tools can be integrated e.g., [6]. Thanks to its extensibility, SODIUM features a rich set of data transformation techniques such as XSLT [16], XQuery [17] as well as QVT [15] 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 SWF development introduces the need for service discovery which is only partially addressed by other approaches to SWF development [20]. 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 [18] [19]. Nevertheless, service-oriented development lacks a query language 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 engine address these issues and constitute a stepping stone to the unification of the various heterogeneous service areas.

5. DISCUSSION AND CONCLUSIONS

Service Oriented Computing (SOC) is a new trend in software engineering. SOC is already affecting the development of business oriented systems and we believe that it will inevitably affect the development of SWFs turning them into service compositions. However, the heterogeneity in protocols and standards of existing service types is a major obstacle for the discovery of services and their integration in scientific workflows.

In this paper we briefly described a platform called SODIUM which provides tools, languages and related middleware for supporting the whole lifecycle of SWFs (i.e. from requirements modeling to their execution) composed of heterogeneous services. Specifically, SODIUM supports abstract as well as concrete modeling of workflows (by providing the VSCL language and editor), uniform discovery of constituent heterogeneous services (through USQL and the query engine) and execution of service workflows (through the USCL Engine).

The open and extendable architecture of SODIUM doesn't alter the underlying protocols and infrastructure used by the various services, but rather hides the specific details from the workflow developers. Furthermore, besides the service types currently supported, i.e. Web, Grid and P2P, SODIUM provides for the easy integration of any other service type.

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

6. REFERENCES

[1] W3C, 2001, *SOAP 1.1*, http://www.w3.org/TR/SOAP

[2] W3C, 2001, *WSDL 1.1*, http://www.w3.org/TR/wsdl

[3] OASIS, 2004, *WSRF*, ***http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf***

[4] Singh, M. P., and Vouk, M. A. *"Scientific Workflows: Scientific Workflow Meets Transactional Workflow."* NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions, Athens, GA, USA, 1996

[5] I. Altintas et al, ***Kepler: An Extensible System for Design and Execution of Scientific Workflows***, SSDBM'04, 21-23 June 2004, Santorini Island, Greece.

[6] Shawn Bowers and Bertram Ludäscher ***An Ontology-Driven Framework for Data Transformation in***

Scientific Workflows In Procs of DILS'04, Leipzig, Germany, Springer, LNCS, volume 2994.

[7] J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd, *GridFlow: Workflow Management for Grid Computing*, Proc. of the 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan, May 2003.

[8] E. Deelman et al, ***Pegasus : Mapping Scientific Workflows onto the Grid***, *Across Grids Conference 2004*, Nicosia, Cyprus, 2004.

[9] R. Grønmo, M. C. Jaeger *Model-Driven Methodology for Building QoS-Optimised Web Service Compositions*. In Procs of DAIS '05 Athens, Greece.

[10] T. Heinis, C. Pautasso, G. Alonso, O. Deak, ***Publishing Persistent Grid Computations as WS Resources***, In: Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science 2005), Melbourne, Australia, December 2005.

[11] S. Krishnan, P. Wagstrom, and G. von Laszewski. ***GSFL: A Workflow Framework for Grid Services***. *Technical Report, Argonne National Laboratory*, Preprint ANL/MCS-P980-0802, August 2002.

[12] B. Ludäscher et al, ***Scientific Workflow Management and the Kepler System***, *Concurrency and Computation: Practice & Experience*, Special Issue on Scientific Workflows, to appear, 2005.

[13] Bertram Ludäscher and Carole Goble, ***Special Section on Scientific Workflows***, SIGMOD Record, 34(3), September 2005.

[14] OMG, *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*, OMG Final Adopted Specification, ptc/04-09-01

[15] QVT-Merge Group. *Revised submission for MOF 2.0 Query/Views/Transformations* OMG document: ad/2004-10-04 version 1

[16] XSLT, ***http://www.w3.org/TR/xslt***

[17] XQuery, ***http://www.w3.org/TR/xquery/***

[18] UDDI, ***http://www.uddi.org***

[19] JXTA Technology, ***http://www.jxta.org***

[20] I. Altintas, E. Jaeger, K. Lin, B. Ludaescher, A. Memon, *"A Web Service Composition and Deployment Framework for Scientific Workflows"*, ICWS 2004

[21] Matthew Shields, Ian Taylor, *"Programming Scientific and Distributed Workflow with Triana Services"*, In *Proceedings of Workflow in Grid Systems Workshop in GGF10*, at Berlin, Germany, March 2004

[22] OMG, "*Unified Modeling Language: Superstructure, version 2.0*", OMG Final Adopted Specification, ptc/04-10-02

[23] Aphrodite Tsalgatidou, George Athanasopoulos, Michael Pantazoglou, *"Semantically Enhanced Discovery of Heterogeneous Services"*, 1st International IFIP/WG12.5 Working Conference on Industrial Applications of Semantic Web (IASW2005), 25-27 August 2005, Jyväskylä, Finland

[24] David Skogan, Hjørdis Hoff, Roy Grønmo, Tor Neple, *"D9-Detailed Specification of the SODIUM Service Composition Suite"*, SODIUM (IST-FP6-004559) project's deliverable, June 2005

[25] Jia Yu and Rajkumar Buyya, "*A Taxonomy of Workflow Management Systems for Grid Computing*", Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005