SEMANTIC TYPE CHECKING IN SCIENTIFIC WORKFLOWS

By

Kheiredine Derouiche

A thesis submitted for MPhil

School of Electronics and Computer Science,

University of Southampton,

United Kingdom.

December, 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Semantic type checking in scientific workflows

by Kheiredine Derouiche

Scientists are increasingly utilizing Grids to manage large data sets and execute scientific experiments on distributed resources [1]. Scientific workflows are used as means for modelling and enacting scientific experiments [2]. Windows Workflow Foundation (WF) is a major component of Microsoft's .NET technology which offers lightweight support for long-running workflows. It provides a comfortable graphical and programmatic environment for the development of extended BPEL-style workflows but offers little support for ensuring that the resulting workflows are complete, robust and meaningful in the user's scientific domain.

Workflow building tools rely on the developer's understanding of multiple services and the data required to execute them. Syntactic type definitions of these data are not meaningful enough to ensure type safety, which are only discovered during execution. We aim to enrich type definitions with semantics in order to guide developers to resolve type mismatch issues at design time.

The approach we have taken is to develop SAWDL-compliant annotations for workflow and use them with a semantic reasoned to guarantee semantic type correctness in scientific workflows.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the Algerian Ministry of Higher Education and Scientific Research for funding my research.

I would also like to thank my supervisor Denis Nicole for his supervision, and continuous support.

I would, finally, like to thank my parents for providing me with their support during my studies, and their continuous encouragement in good and bad times.

Chapter 1 Introduction

Scientists now routinely utilize computational tools and information repositories to conduct their experiments. Such resources are made available with programmatic access as Web Services. This e-Science approach enables scientists and researchers to collaborate. Grid computing builds infrastructures for e-Science to support global distributed collaboration [3]. Research and development efforts within the Grid community have produced protocols, services, and tools that address the challenges of the field. The Globus Toolkit and UNICORE are two popular Grid systems that have provided a rich set of services for different scientific domains. Scientists want tools that allow them to bring together the power of various computational and data resources by developing and executing their own *scientific workflows*. Resources are supplied by third parties and the operations provided are often incompatible with each other. Resolving resource mismatches requires the designer's intervention, which can be difficult and time-consuming task for scientists. Another major problem is the unreliable handling of failed workflows. Such complexities should be hidden from the user by the workflow system.

Web Services provide a basis for distributed, service-oriented systems. Web Service standards such as WSDL provide syntactic descriptions of Web Service functionalities using XML Schemas to describe composite data types and method interfaces. These standards fail to capture the domain semantics of scientific data. Web Services also fail to provide reliability during execution because they lack the isolation property. Current failure handling mechanisms fail to mitigate the effect of failure on the overall execution of the workflow. Semantic ontologies provide an approach to define hierarchies of failure at different levels. Thus, allowing us to define more intelligent handling mechanisms. In this thesis, we present an implementation that successfully integrates semantics into a standard industrial business workflow management system, thus allowing the automatic detection and resolution of service mismatches in workflows at design time. In addition, we demonstrate how compensations can be used semantically to resolve workflow failure issues.

This thesis is structure as the following: In Chapter 2, we provide a general overview of the Semantic Web and current standards. In Chapter 3, we discuss Web Services standards and the emerging efforts in the Semantic Web Services field. In Chapter 4, we discuss the issue of Web Service composition and the related problems in current standards. In Chapter 5, we present our solution and the implementation of the framework. In Chapter 6, we conclude by reemphasising our achievements and the contribution of our work and identify the scope of our future research.

Chapter 2 The Semantic Web

2.1 Introduction

The World Wide Web consists of documents in various formats, including text, images, audio, and videos. These documents are usually unorganised and can only be consumed by humans. Such data can be difficult to manage and understand by software agents and unreliable. The Semantic Web is a vision in which the existing Web will include an unambiguous notion of meaning in data and services. This will make the knowledge organized and machine understandable [4].

The Semantic Web is slowly being commercialized and deployed by companies and communities. Companies such as Celcorp, Ontoprise, and Unicorn already offer semantic integration solutions [5].

The Semantic Web benefits from an open source development community. Applications of ontologies such as Friend of a Friend (FOAF) and Dublin Core are already widespread within the Web community. The academic community has developed applications that demonstrate the potential power of the Semantic Web [6-8].

2.2 Enabling Technologies

2.2.1 RDF/RDFS

The Semantic Web consists of a set of core enabling technologies usually illustrated through a layered diagram referred to as “Semantic Web Layer Cake” [9], show in Figure 1.

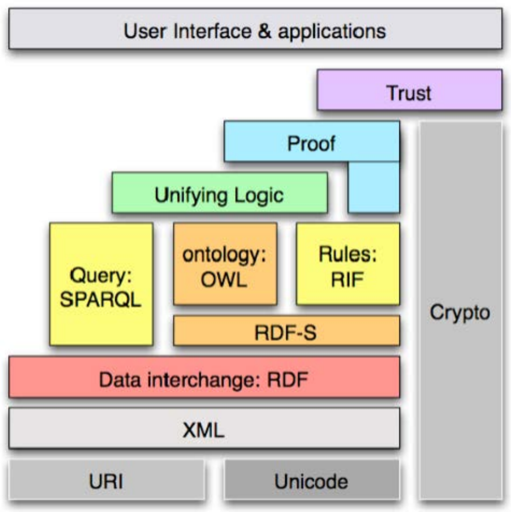

Figure 1 - The Semantic Web Layer Cake

The Resource Description Framework (RDF) [10] is a W3C specification. RDF can be used to associate information with any resource with an URI. These statements are made in the form of subject-predicate-object expressions, called triples. The subject is a resource, always identified by a Uniform Resource Identifier (URI). The predicate is a resource representing a relationship. The object is a resource or a Unicode string literal. RDF Schema is an extensible knowledge representation language intended to structure RDF resources.

2.2.2 OWL

The Web Ontology Language (OWL) [11] is a family of language specifications for defining and instantiating ontologies. The specifications define a type system along with additional constraints. OWL is based on the earlier languages OIL and DAML+OIL. OWL allows information to be processed by applications using reasoning techniques. OWL currently has three increasingly expressive languages OWL Lite, OWL DL, and OWL Full.

OWL Full contains all OWL language constructs and provides free, unconstrained use of RDF constructs. OWL Full allows classes to be treated as entities, whereas in OWL DL and OWL Lite only instances of a class are individual entities. OWL DL is a sublanguage of OWL which places a number of constraints on the use of the OWL language constraints. For example properties inverse of, symmetric, and transitive can never be specified for datatype properties. OWL Lite abides by all the restrictions OWL DL puts on the use of the OWL language constructs. In addition, it forbids the use of some constructs such as `owl:oneOf`, `owl:unionOf`, and `owl:disjointWith`. In practice, working with OWL Full is generally too complex for a logical reasoner to use for logical deduction, but OWL DL is both complete and decidable and hence easier to reason over and use.

Chapter 3 Semantic Web Services

3.1 Service Orientation and Web Services

Interoperability problems are important in Business-to-Business (B2B) electronic commerce. To solve such problems, much effort was invested in the Enterprise Application Integration (EAI) and B2B Integration field. This has led to the development of various solutions, most of which use adapters to connect legacy systems. The Common Object Request Broker Architecture (CORBA) [12] from the Object Management Group (OMG) and the Distributed Common Object Model (DCOM) [13] from Microsoft were two major efforts that attempted to achieve interoperability in distributed systems. CORBA's inherit complexity and DCOM's dependence on Windows led to their failure to achieve universal uptake. The protocols failed properly to interoperate due to some differences in their implementation such as the format for payloads and message representation of communication endpoints. These protocols did not provide a proper type system, and applications were reduced to extracting semantics via HTML parsing. An important factor in CORBA's and DCOM's decline was XML. Microsoft gave up on DCOM and developed SOAP [14] with its partners. This protocol used XML as the on-the-wire encoding for remote procedure calls. With the success of SOAP as a market strategy, numerous vendors moved their efforts toward the Web Services market. This boosted the rapid the advance that Web Services enjoys now.

Web Services play a major role in Service-Oriented-Architecture (SOA) [15] as they fulfil many of its requirements of platform-independence and interoperability. Web Services are built on a set of open core standards defined by standards bodies such as the World Wide Web Consortium (W3C) and the Organization of Structured Information Standards (OASIS). Simple Object Access Protocol (SOAP), a W3C recommendation, is an XML-based RPC protocol. SOAP encoded messages can be delivered using transport protocols such as HTTP and SMTP. Web Services Description Language (WSDL) [16], currently in version 2.0, is another W3C recommendation. WSDL is an XML-based language that provides a model for describing Web Services. WSDL defines services as a set of endpoints operating on messages. Universal Description Discovery and Integration (UDDI) [17] is an open initiative sponsored by OASIS. It is an XML-based registry enabling businesses to publish service listings and discover each other. Although UDDI Business Registries (UBR) were discontinued early 2006, the standard is still supported in several vendors' products and services. While SOAP and WSDL have been widely adopted by software vendors, UDDI has not gained wider adoption in industry despite the fact that enterprises are increasingly deploying Web Services. UDDI's complexity and lack of functionality are among the features that discouraged its uptake by software developers. DISCO was Microsoft's version of UDDI. DISCO documents were published to clients through a Web Server, and they provided links to resources describing the Web Service.

In addition to these core standards, additional specifications have been developed or are being developed to extend Web Services capabilities. These specifications are generally referred to as WS-*. Commercial and industrial interest in SOA and Web Services contributed greatly to their adoption by vendors and to the fast evolution of several standards. Figure 2 illustrates the Web Services building blocks for developing distributed applications.

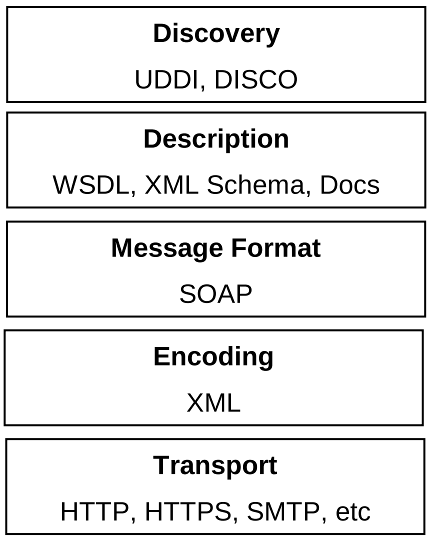

Figure 2 - Web Services building blocks

Figure 3 illustrates a common usage scenario for Web Services that can be defined by three phases: *Publish, Find,* and *Bind*; and three entities: the service requester, which invokes the services; the service provider, which responds to requests; and the registry where services can be published and advertised. Service descriptions are published by service providers to a service registry. These services are discovered by querying or browsing the registry.

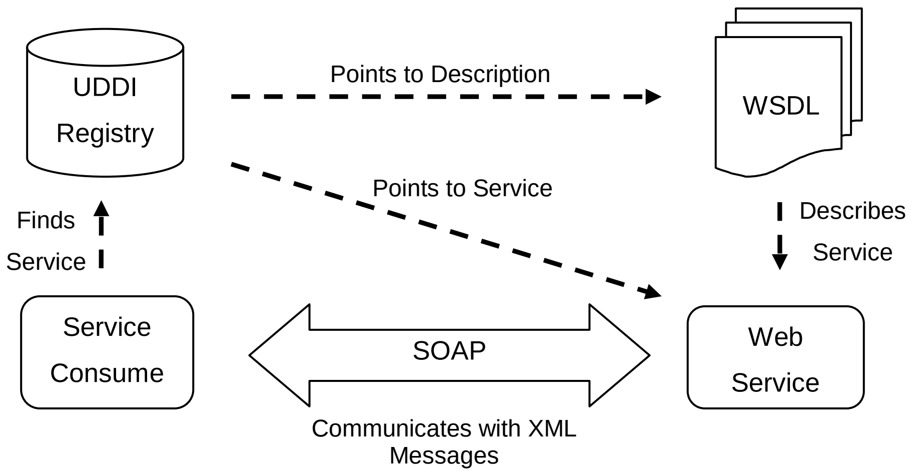

Figure 3 - Web Service usage scenario

3.2 The Semantic Approach to Web Services

The semantic approach to Web Services aims to enable the automatic discovery, composition and execution of Web Services. Traditional technologies for Web Services (WSDL) only provide descriptions at syntactic level, making it difficult to interpret the domain meaning of inputs and outputs flowing between requesters and providers. In the same way that Semantic Web technologies allow semantic markup of data on the Web to make it machine understandable, Web Service can now be augmented with semantic annotations to make them discoverable by software agents, as well as composable and executable.

In the next section, we review three research efforts addressing the Semantic Web Services issues.

3.2.1 The Web Ontology Language for Services: OWL-S

OWL-S [18] (formerly known as DAML-S) is an OWL ontology that includes three primary subontologies: the service profile, process model, and grounding. The service profile is used to describe the capabilities of the service. The process model describes how the service is performed. The grounding specifies how the service is actually invoked. The service profile and process model provide characterizations of a service, and the grounding provides details related to message format, transport protocol. Figure 4 shows the top level ontology classes and the relationships between them. For example, the *presents* property represents a relationship between a *Service* and a *Profile*.

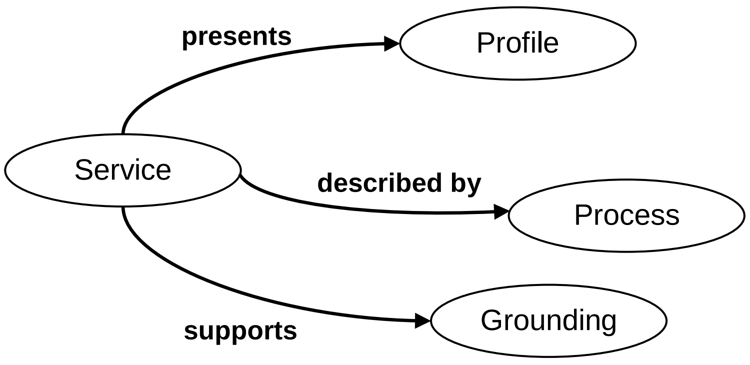

Figure 4 - Top level of Service Ontology

Each service described using OWL-S is represented by an instance of the OWL class *Service*, which has properties that associate it with a process model, one or more groundings, and optionally one or more profiles. A process model provides the complete description of how to interact with the service at an abstract level, and the grounding supplies the details of how to embody those interactions in real messages to and from the service. Each service profile can be thought of as a summary of the process model aspects plus additional advertising information. Several types of grounding exist for OWL-S; the default one employs WSDL.

The Service Profile

The OWL-S profile specifies the capabilities of services. Discovering services that satisfy a request is accomplished by exploiting the OWL-S profile structure and the references to OWL concepts. The principal elements in a profile include the *inputs*, *outputs*, *preconditions* and *effects* (IOPEs) associated with the service — it is required to list all IOPEs. The IOPEs describe the functional aspect of the service, i.e. the service expects data as input and returns data as output. IOPEs specify the preconditions that need to be satisfied and the effects during the execution. Services are usually stateless i.e. they do not change the state of information, preconditions and effects in this case are not necessary. Figure 5 shows a partial example of a profile for a service, expressed in OWL. The profile describes the input the service takes. The other parameters are not described because they are unimportant.

```
<BLASTProfile rdf:ID="WUBLAST">
  <serviceName>BLAST Service</serviceName>
  <hasInput rdf:resource="&blast_process;#DNASequence_In"/>
  ......
</BLASTProfile>
```

Figure 5 - A partial OWL-S profile

The Process Model

The process model specifies the possible patterns of interaction with a Web Service. There are two types of processes that can be invoked: atomic and composite. Atomic processes are single black-box processes. Composite processes can consist of atomic and composite processes linked using control flow flow constructs such as sequences, conditional branches and loops. A third type, the simple process, is a non-invocable and abstracted view of atomic and composite processes. A process in OWL-S has a set of associated features (IOPEs) linked by properties such as hasInput, has Output, etc.

```
<AtomicProcess rdf:ID="blastp">
    <hasInput rdf:resource="#blastp_In">
</AtomicProcess>

<Input rdf:ID="blastp_In">
    <paramterType
        rdf:resource="http://www.mygrid.org.uk/ontology#DNA_sequence"/>
</Input>
```

Figure 6 - An OWL-S Process

Figure 6 shows the atomic process corresponding to the profile in Figure 5. The atomic process specifies that it has an input DNA sequence using the property hasInput and points to its semantic type using the parameterType property. The ontology used is the *myGrid* domain Bioinformatics ontology.

The Grounding

The grounding ontology of OWL-S is used to specify how abstract information detailed by atomic processes is realized by concrete information in deployed Web Services. Grounding maps each atomic process to a WSDL operation, and relates each OWL-S process input and output to elements of the XML serialization of operation input and output messages. Mappings enable the translation of semantic inputs to the appropriate WSDL messages for the service execution, and translate back output messages to semantic descriptions.

Figure 7 shows the corresponding grounding of the blast operation. The wsdlOperation property of WsdlAtomicProcessGrounding specifies the portType/operation pair from WSDL. The wsdlInputMessage property is mapped to the request message in WSDL. The wsdlInput property specifies mappings between OWL-S parameters and WSDL message parts.

```
<WsdlGrounding rdf:ID=”Grounding_BLAST”>
   <hasAtomicProcessGrounding
      rdf:resource=”#WsdlGrounding_blastp”/>
</WsdlGrounding>

<WsdlAtomicProcessGrounding rdf:ID=”WsdlGrounding_blastp”>
 <owlsProcess rdf:resource=”&blast_process;blastp”/>
 <wsdlOperation rdf:resource=”#blastp”/>

 <wsdlInputMessage>
       <xsd:anyURI rdf:value=”&BLASTGroundingWSDL;#blastp_Input”>
 </wsdlInputMessage>

 <wsdlInputs rdf:parseType=”Collection”>
  <WsdlInputMessageMap>
   <owlsParameter rdf:resource=”&blast_process;#DNASequence_In”/>
    <wsdlMessagePart>
             <xsd:anyURI rdf:value=”BLASTGroundingWSDL;#sequence”/>
    </wsdlMessagePart>
  </WsdlInputMessageMap>
 </wsdlInputs>
</WsdkAtomicProcessGrounding>

<WsdlOperationRef rdf:ID=”blastp_operation”>
 <portType>
  <xsd:anyURI rdf:value=”&BLASTGroundingWSDL;#blastp_PortType”/>
 </portType>
 <operation>
  <xsd:anyURI rdf:value=”&BLASTGroundingWSDL;#blastp_op”/>
 </operation>
</WsdlOperationRef>
```

Figure 7 - An OWL-S Grounding

Implementations

Task Computing is a project that has been developed at the Fujistu Laboratories in America [19]. The framework provides an interface for a collection of services and devices such as agendas, display devices, and email clients. An execution environment was developed to consume the provided semantic descriptions. Users are guided to compose simple workflows that accomplish tasks such as locating an address from a contact card and printing the directions there. The environment relies on semantic reasoning to aid select compatible services. Workflows are limited to sequences, where services are connected via their inputs and outputs.

The authors argue that WSDL definitions provide functional descriptions of services, thus requiring programmers to understand the semantics of these services. Hence, they introduce Semantic Service Descriptions (SSDs), service layer semantic descriptions that can be applied to different components of a service, for example inputs, outputs, and class entities.

The authors propose using OWL-S as one possible implementation of a semantic description language. They further explain that service composition can rely on input and output semantic compatibility or entities hierarchical relationships.

3.2.2 The Web Service Modelling Ontology: WSMO

WSMO [20] is an ontology for the description of Web Service. The definition of WSMO hinges on the following four concepts: *Web Services, Goals, Ontologies* and *Mediators*. The following list provides an explanation of the meaning of the four concepts.

Web Services expose the interface of businesses on the Internet. They describe the capabilities of the Web Service, and how these capabilities are fulfilled.

Goals represent the objectives that a client seeks to fulfil. These objectives are characterized by post conditions that describe the information state the client desires, and effects that describe the state of the world that the client desires to achieve.

Ontologies provide a formal specification of the domain. Ontologies provide formal semantic to exchanged information by facilitating interoperation, and specify the precise terminology accepted by Web Services facilitating the definition of semantic descriptions.

Mediators provide a general mechanism to overcome interoperability issues between Web Services. They provide a mapping between different ontologies concerned with related domains.

Goal, WebService and Ontology components are linked by four types of mediators as follows:

- OO mediators link ontologies to ontologies,
- WW mediators link web services to web services,
- WG mediators link web services to goals, and finally,
- GG mediators link goals to goals.

A few tools and APIs are available for WSMO. WSMO Studio is a WSMO compliant editor available as an Eclipse plugin. WSML Rule Reasoner is a reasoner implementation for Web Services Modelling Language (WSML). WSMO4J is a Java API for building WSMO based applications. The Web Services Execution Environment (WSMX) is the execution environment for Semantic Web Services based on WSMO. Due to the lack of technical documentation and working scenarios, WSMO is not being adopted by academic and industrial researchers. Efforts on WSMO focus on producing a conceptually complete and sound framework for describing Web Services rather than a lightweight working solution. The WSMO project uses the WSML as an ontology language rather than OWL, which is a W3C recommendation. WSMX is limited to WSML, which provides syntax and semantics for WSMO. This limits the usability of WSMO since most ontologies are defined in OWL.

3.2.3 Semantic Annotations for Web Service Description Language

Introduction

Current Web Services technologies are built around SOAP and WSDL. These technologies provide a solid foundation for resolving integration problems but do not scale well when it comes to search and mediation. Automation in Web Services requires more than XML descriptions of the data structure and syntax. This sort of automation can be achieved using Semantic technologies, such as those underlying the Semantic Web.

Building on WSDL, Semantic Annotations for Web Service Description Language (SAWSDL) [21] adds hooks that let WSDL components point to their semantics (see Figure 8). The SAWSDL specifications do not provide any specific semantics; rather, it allows the annotation of syntactic WSDL descriptions with pointers to semantic concepts. These concepts can be consumed by software systems to (partially or fully) automate tasks such as service discovery, composition, and invocation.

Technically, SAWSDL is a set of extensions for WSDL. WSDL uses XML as a common data-exchange format and apply XML Schema for data typing. It describes a Web Service on three levels:

Reusable abstract interface defines a set of operations, each representing a simple exchange of messages described with XML Schema element declarations.

Binding describes message serialization; it follows the structure of an interface and fills in the necessary networking details (for instance SOAP or HTTP).

Service represents a single physical Web Service that implements a single inteface; the Web Service can be accessed at multiple network *endpoints*.

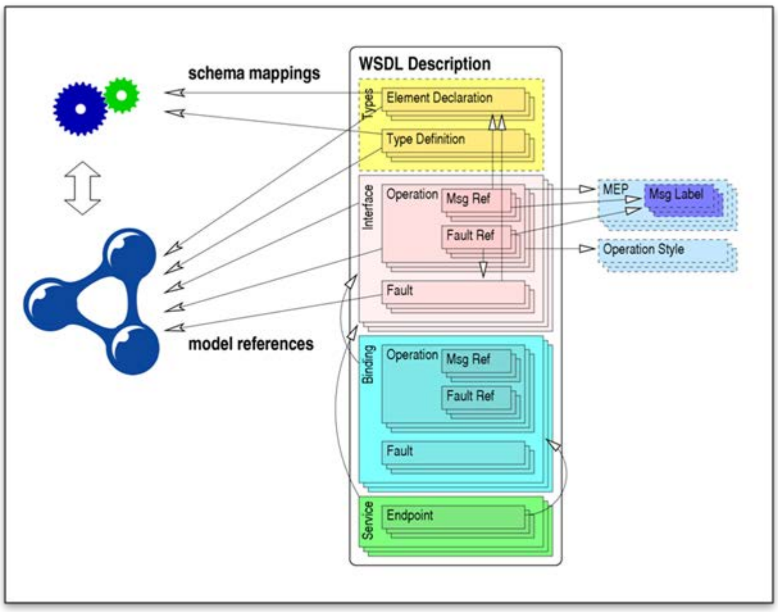

Figure 8 - WSDL with SAWSDL Annotations

WSDL describes the Web Service on a syntactic level, whereas SAWSDL specifies WSDL components semantic by extending WSDL with a semantic layer. Specifically, SAWSDL defines extension attributes that can be applied both in WSDL and in XML Schema to annotate WSDL interfaces, operations, and their input and output messages. These extensions take two forms: *model references* that point to semantic concepts and *schema mapping* that specify data transformation between messages' XML data structure and the associated semantic model. The table in Figure 9 summarises the complete syntax introduced by SAWSDL.

Several tested implementations have developed for the SAWSDL specifications [22]. Direct implementations are parser APIs that make the annotation available to applications and tools that let users annotate WSDL documents with semantic annotations. The Woden API [23] for WSDL 2.0 and the WSDL4J API [24] for WSDL 1.1 were both extended to handle SAWSDL. Two GUI tools exist to help annotate WSDL documents with semantics: Radiant from the University of Georgia and the Web Service Modelling Ontology (WSMO) Studio from Ontotext.

Name	**Description**
`modelReference`	A list of references to concepts in some semantic models (XML attribute)
`liftingSchemaMapping`	A list of pointers to alternative data-lifting transformations (XML attribute)
`loweringSchemaMapping`	A list of pointers to alternative data-lifting transformations (XML attribute)
`attrExtensions`	Attaches attribute extensions where only element extensibility is allowed (XML attribute)

Figure 9 - SAWSDL syntax summary.

Model References

A model reference is an extension attribute, `sawsdl:modelReference`, which can be applied to any WSDL or XML Schema element. However, SAWSDL defines its meaning only for `wsdl:interface`, `wsdl:operation`, `wsdl:fault`, `xs:element`, `xs:complexType`, `xs:simpleType`, and `xs:attribute`. This attribute allows multiple annotations to be associated with a given WSDL or XML Schema component via a set of URIs, each one identifying concepts expressed in different semantic representation languages. Model references generically refer to semantic concepts, serving as hooks for attaching semantics. They are used to describe the meaning of data or to specify the function of a Web Service operation.

Schema Mapping

SAWSDL provides two attributes for attaching schema mappings: `sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping`. Lifting mappings transform XML data from a Web Service message into a semantic model (for instance, into RDF data that follows some ontology), whereas lowering mappings transform data from a semantic model into an XML message. Lifting and lowering transformations address post-discovery issues in using Web Services. Mismatches between the semantic model and the structure of the inputs and outputs can exist between matched Web Services. In XML Schema, an XML elements' content is described by *type definitions* and the name is added as an *element declaration*. SAWSDL model reference and schema mapping annotations can be both on types and on elements.

WSDL 1.1 Support

The SAWSDL specifications are built primarily for WSDL 2.0, but it also supports WSDL 1.1. Both model references and schema mappings apply without modification to WSDL 1.1. However, the XML Schema for WSDL 1.1 allows only element extensions on operations, so a WSDL 1.1 document with the SAWSDL `modelReference` attribute on an operation would not be valid. To overcome this obstacle, SAWSDL defines the element `attrExtensions` to carry extension attributes in places where only element extensibility is allowed. Instead of putting the model reference directly on the `operation` element, SAWSDL can put it on the `attrExtensions` element, and then insert that into the operation element.

Annotating WSDL Documents

The different semantic annotation constructs in SAWSDL serve to describe semantically an aspect of the Web Service. Annotating element declarations and type definitions in XML Schema with model references accompanied by lifting and lowering schema mappings provide an information model. This model is needed when performing data mediation when it is exchanged between the semantic client and the XML-based Web Service. The description of service capabilities advertizes what the service offers to users, and thus it enables the service to be discovered, composed, and eventually invoked. Pointing to the appropriate description of a Web Service's capability is achieved by annotating the service and the interface constructs by model reference annotations. Apart from describing the service (or the interface) as a whole, capabilities can be ascribed to the operations using model reference pointers. The latter type of annotations might be needed by semantic clients to perform a more fine-grained operation discovery, whereas annotations of service and interface constructs serve to categorise the different services; this is useful for general service discovery.

```
<wsdl:definitions...>
<wsdl:message name='getFASTA_DDBJEntry'>
 <wsdl:part name='Result' type='xsd:string'
sawsdl:modelReference="http://www.mygrid.org.uk/ontology#DNA_sequence"/>
</wsdl:message>
...
<wsdl:portType name='GetEntry'>
 <wsdl:operation name='getFASTA_DDBJEntry'>
  <wsdl:input name='getFASTA_DDBJEntryIn'
   message='tns:getFASTA_DDBJEntryIn'/>
  <wsdl:output name='getFASTA_DDBJEntryOut'
   message='tns:getFASTA_DDBJEntryOut'/>
 </wsdl:operation>
...
</wsdl:portType>
</wsdl:definitions>
```

Figure 10 - A WSDL 1.1 document annotated with SAWSDL

Figure 10 shows an example of a WSDL 1.1 document describing a Bioinformatics Web Service that fetches a DNA sequence in the FASTA format from the DNA Data Bank of Japan (DDBJ) using an access number. The service description of the input and the output of the operation in the Web Service ambiguously name the input and the output *getFasta_DDBJEntryIn* and *getFasta_DDBJEntryOut* respectively. The message component specifies that part of the output message is of type string, but says nothing about the semantic meaning of the data returned from the operation. By adding the SAWSDL model reference annotation, we can point to a semantic concept that semantically describes the data retrieved from the service. In our example we annotated the output with the *DNA_sequence* concept from the *myGrid* ontology, which is an OWL Domain Ontology for Bioinformatics.

Discussion

OWL-S, WSMO, and SAWSDL share the vision that ontologies are essential to support automatic discovery, composition and interoperation of Web Services. OWL-S defines a set of ontologies that support reasoning about Web Services, following the chronological order of SWS framework tasks – discovery uses descriptions from the profile and process ontology and invocation needs grounding descriptions in the grounding ontology. WSMO on the other hand define a conceptual framework within which ontologies are created. WSMO makes clear distinction between the types of Web Services i.e. requesters and providers, and outline the role of mediators as a solution to the interoperation problem. Both efforts define a formal framework that is highly expressive and could be too complex for some domains.

Heavy approaches like OWL-S and WSMO can be impractical for manual annotation for data and tasks in scientific domains. Tasks in Bioinformatics are rather lightweight and often stateless since the state of information does not change. This omits the need for preconditions and most importantly effects. Even though SAWSDL itself does not provide actual SWS modelling capabilities but by embedding annotations directly in WSDL documents, existing WSDL repositories can be used for semantic discovery of services. Furthermore, developing applications based on SAWSDL is relatively easy since it is reduced to upgrading existing tools for Web Services. We therefore have a strong belief that SAWSDL is the right SWS technology for annotating scientific data and services.

3.3 Semantic Web Services in the Grid: The Semantic Grid

Both the Grid and the Semantic Web communities started as two distinct research efforts. The need to develop new Grid applications and make reuse of data and workflows led to the proposition of the Semantic Grid. It is a joint effort that aims to enable building scientific solutions for scientific problems. Realizing this vision is achieved by applying Semantic Web technologies to Grid developments, from Grid services to Grid applications.

InteliGrid [25] proposes an architecture based on three layers: conceptual, software and basic resource. The conceptual layer represents descriptions of resources in the form of ontologies, graphs, etc. The software layer consists of software that consumes descriptions defined in the conceptual layer. The basic resource layer includes the low level infrastructure. Service discovery and other functionalities are supported by ontology services provided by the software layer.

S-OGSA [26] is a proposed architecture that extends OGSA by providing support to semantic content. The approach proposes the use of semantic services that can manage knowledge about resources in the Grid. The proposed model identifies resources on the Grid such as services and data, knowledge about these resources in the form of ontologies, graphs, etc, and the actual association between knowledge and resources. The architecture introduces specialised services that can create, manage, and consume ontologies and metadata.

Although WSMO and OWL-S were not developed in the Grid context, they do provide a methodology and language to describe relevant aspects of services and information resources in order to enable the automation of tasks such as selection, composition and monitoring of complex services. Resources discovery on the Grid can be facilitated by using Semantic Web languages including RDF, OWL, and WSMO. The expressivity of these languages allows sophisticated reasoning in order to discover and select required resources. Complex tasks can be realized by aggregating and composing multiple resources on the Grid. This is facilitated by supporting workflow description and enactment. Existing languages such as OWL-S define process (workflow) using the OWL-S process model ontology. WSMO defines the process model and execution semantics for workflow description and execution using abstract state machines. Aside from solving the composition problem, developers are concerned with data and control flow compatibility. Annotating data and workflows facilitates matching and supports any necessary conflict detection.

OWL-S and WSMO are two initiatives that aim to describe requests and Web Service functionality in a way that can help in the automation of service discovery and composition. They also proved to be good candidates in realizing the vision of the Semantic Grid, and could be key components when building Grid applications. However, the vision of the Semantic Grid has yet to be realized. Several architectures and prototypes have been proposed for the Semantic Grid [27, 28], however none of them cope properly with the current requirements of the Grid such as scalability, security and performance. Many challenges face the uptake of the Semantic Grid. The

Semantic Grid needs to demonstrate the added value of semantics in Grids, facilitate the task of gathering, managing, and maintaining data, improve the performance of creating and retrieving semantic metadata, and last but not least securing exposed metadata and automated reasoning.

Chapter 4 Scientific Workflow Systems

4.1 Introduction to Workflows

Web Services standards provide solutions to the interoperability problems. However, existing methods for creating business processes are not designed to work with cross-organizational components. Orchestration describes an aspect of creating business processes from composite Web Services. Microsoft's XLANG and IBM's Web Services Flow Language (WSFL) were the early standards proposed for designing business processes. These efforts were later combined to form the Business Process Execution Language for Web Services (BPEL4WS) [29] or BPEL for short. BPEL allows enables a user to specify how different Web Services can be composed together in various ways to design an executable workflow. Designed workflows can also presented as new services, thus enabling recursive composition of workflows. Another way of describing workflows is as choreography. Choreography describes the observable interactions between services from a global point of view rather than a service perspective. The Web Services Choreography Description Language (WS-CDL) is a choreography language that can be used to describe workflows. WS-CDL as a workflow solution may provide better flexibility because choreography descriptions can be changed independently of the services. However, a few unresolved issues have an impact on wider adoption of WS-CDL in particular and choreography in general. Choreography languages' lack of a concrete syntax definition requires developers to use orchestration languages in order to render workflows executable.

Scientists face many of the same challenges that are found in enterprise computing, namely integrating distributed and heterogeneous resources. Collaborations are becoming more geographically dispersed and use machines distributed across several institutions. Scientists are increasingly relying on Web technology to perform in silico experiments. The task of running and coordinating scientific applications across several domains, however, remains complex.

4.2 Scientific Workflow Systems

Several research efforts [30, 31] have investigated the suitability of BPEL and its implementation for scientific workflows. Some approaches involved the identification of the requirements of scientific workflows and assessing to what extent the BPEL specifications satisfy these requirements. Other approaches followed an experimental methodology by implementing scientific workflows that solve some scientific problems. The research work demonstrated that BPEL could be successfully used to combine Grid services to develop scientific workflows, and to deploy these workflows using an enactment engine.

The research community produced various specialized workflow systems designed specifically to aid the development of scientific workflows. Globus [32] is an open source toolkit that implements many Grid related standards. It is the paradigmatic example of a heavy-weight Grid system. Globus provides a low-level toolkit that enables the construction of Grid based applications. The toolkit is composed of several software components. These components are divided into five categories.

- *Security* components are based on the Grid Security Infrastructure (GSI).
- *Data Management* components such as Open Grid Services Architecture Data Access and Integration (OGSA-DAI) and GridFTP allow large data management.
- *Execution Management* components such as Grid Resource Allocation and Management (GRAM) deal with the initiation, monitoring, management, and scheduling of executable programs.
- *Information Services* refer to the Monitoring and Discovery Services (MDS). It includes components such as WebMDS, Index, and Trigger to discover and monitor resources.
- *Common Runtime* components provide libraries and tools to build WS and non-WS services.

Discovery mechanisms in Globus are concerned with obtaining, indexing, and processing information about the state of services and resources. The Globus toolkit provides services such as GRAM that defines resource properties to enable service discovery. Aggregator resources collect state information from registered information sources, which can be queried using command line, web based, and Web Service interfaces. The information collected by these aggregator services is maintained as XML, and can be queried via Xpath queries (as well as other Web Services mechanisms).

Different workflow systems have been proposed in order to support developing Grid applications with the Globus toolkit. GridAnt [33] is an XML/Java-based tool for representing and executing workflows of computational codes and Web Services. GridAnt contains a control construct for expressing parallel and sequential tasks. Data is propagated between the different tasks in the workflow using a simple copy command. The framework does not provide any mechanism to check for data type mismatch or heterogeneity.

UNICORE [34] is a Grid middleware that allows users to access Grid resources. The UNICORE Grid system consists of the Client, Gateway, Network Job Supervisor (NJS), and Target System Interface (TSI) software Components. The UNICORE Client allows end-users to connect to a UNICORE gateway. The UNICORE Gateway is the entry point for all UNICORE connections. The UNICORE NJS manages submitted UNICORE jobs, it also realises Abstract Job Objects (AJO) into concrete execution commands and hands them over to the TSI. The UNICORE TSI accepts the submitted job components and passes them to the local system for execution. A UNICORE AJO used to be modelled as a directed acyclic graph (DAG) of tasks or other jobs. It has been extended to include conditionals and loops, available via the client GUI. DAGs define dependencies in job submission and dictates the order of execution. However, job descriptions provide no mechanism to check for semantic compatibility of pipelined data between different jobs.

Kepler [35] is scientific workflow system that has a graphical user interface, thus enabling users to design and execute workflows. Kepler workflows can be exchanged in XML using Ptolemy Modelling Markup Language (MoML). Ptolemy is the underlying system of Kepler, making the system actor-oriented. Scientific workflows in Kepler are viewed as a composition of components called actors. Using the extensibility feature of actors, support for Web Services is provided through generic Web Service actors. The Kepler system benefits from an extension that implements what is called *smart semantic links* [36]. The system identifies structural and semantic data types, where ports on actors (input, output) are associated with OWL-DL ontology based semantic type [37]. The proposed approach generates mappings in XQuery and XSLT to transform data from a source structure to a target structure. Parameter mapping is a work in progress not yet supported by Kepler. While the system provides mapping between structural data types, grounding of semantic data types to structural data types does not exist. The use of semantics is reduced to symbolically check data compatibility. In our approach, we propose a safe type system, where semantics are grounded to concrete data and are part of the data transformation process. SPARQL [38] is used to transform semantic data from one structure to another, while semantic reasoning is used to check for data compatibility.

Triana [39] is a workflow system that has a graphical user interface allowing users to add services to the workflow. The Grid Application Protocol (GAP) Interface allows Triana to communicate with composed services, including Web Services. WServe is the API that implements the GAP binding for We b Services. Using this API, services are queried from a UDDI server and are invoked through a WS Gateway. Using the graphical interface, services are composed and connected with pipes. Resulting compositions can be written in a proprietary format or BPEL [40]. To our knowledge uses information about input and output data-type objects to perform design-time type checking i.e. ensuring data compatibility between components [41] . This approach does not capture any semantic information about the exchanged data, nor does it deal with structural mismatches. It is as good as the type checking mechanism used in Windows Workflow Foundation.

The Taverna Workbench [42] is a tool targeted at developing workflows in bioinformatics. Taverna provides a graphical tool for creating and executing workflows. Workflows are taken to be a graph of processors represented in the Simple

Conceptual Unified flow Language (Scufl), an XML-based language. Workflows in Scufl consist of three main components:

- *Processors* are transformations that take input data and produce output data. Types of processors include WSDL types, nested types, local processor types, and string constant types.
- *Data links* are data bindings between sources and sinks. Data sources can be a processor output and the data sink can be a processor input.
- *Coordination constrains* link two processors and control their execution. These constraints specify the order execution of processors where no direct dependency exists.

Services and workflows in Taverna are annotated using Feta descriptions in RDF(S), which are queried through reasoning using Jena [43]. The Feta engine uses the annotations to discover Web Services and Workflows using a semantic approach. Users can add discovered services without checking for their compatibility. Taverna proposes using specialised services called shims [44] that are similar to WSMO mediators. Shims are services that transform data that are compatible from one format to another. These services do not perform a structural transformation of data as it is concerned with format only. Moreover, the mismatches between connected services have to be detected by the workflow designed, and shim services are manually added as required.

Chapter 5 Semantic Annotations in Windows Workflow Foundation

5.1 Microsoft's Windows Workflow Foundation

In the business domain, human-intensive and machine-intensive processes are combined to express the required business processes. Workflow is a mechanism that expresses business processes as a collection of activities. Workflows can be created, executed, and managed using Business Process Management (BPM) systems. Many approaches have emerged to provide solutions to workflow problems: Web Service Flow Language (WSFL), Web Services for Business Process Design (XLANG), and Business Process Execution Language (BPEL), to name a few. BPEL is the one with most traction in part due to its backing by major industry vendors. BPEL allows the orchestration of Web Services into business workflows. However, it restricts the developer to creating workflows from services only. This limits the scope of BPEL when it comes to the integration of non-serviceable legacy applications.

Windows Workflow Foundation (WF) solves the integration problem and allows the creation of workflows that compose Web Services with legacy systems. WF is the latest addition from Microsoft to workflow management systems. It is released as part of the .NET Framework 3.0 and 3.5. The technology provides developers with a group of workflow-related components, thus allowing the creation, execution, monitoring and tacking of workflows.

WF workflows can be developed using Visual Studio. The WF extension to Visual Studio provides a visual designer, a set of workflow templates, and visual debugging capabilities, easing the workflow development task. The Extensible Application

Markup Language (XAML) is a new XML-based language commonly used to develop WF workflows. Workflows can also be developed using code in any CLR language, or using markup with code separation. WF workflows are expressed as a collection of composed activities. Activities are used to represent business specific activities. WF provides a set of general-purpose activities and allows developers to create their own domain-specific activities. WF supports two types of workflows: structured and state machine. A sequential workflow is procedural in nature; the composed activities are executed in sequence resulting in a predictable execution path. State machine workflows, in the other hand, are event driven and workflow execution relies on external events.

In addition to the activity library, WF provides a runtime engine and runtime services components that executes workflows and provides monitoring and tracking services. WF workflows can be hosted on different host applications varying from console applications to windows services. The runtime services manage workflow instances, transactions, tracking, and state management.

WF workflows can be composed of Web Services, desktop applications and legacy systems. The WF runtime provides a backbone to execute and coordinate workflow instructions. It would be analogous to a BPEL engine, but it differs in its deployment strategy. The BPEL engine forms part of a server-tier deployment, whereas WF runtime is deployable classically on the server side, as well as any other application that can be linked to the .NET framework. This architecture makes WF a more lightweight and faster framework than BPEL.

5.2 Web Service in Windows Workflow Foundation

With the success of Web Services in the business domain, the scientific community started migrating their Grid resources and applications to follow SOA. In order to standardise this new Grid service based architecture, the Global Grid Forum (GGF) developed the Open Grid Services Architecture (OGSA) specifications. OGSA is based on other Web Services technologies, notably WSDL and SOAP. Due to the wide use of Web Services both in business and scientific domains, workflow systems in both domains had to provide support for invoking Web Services as well as publishing developed workflows as Web Services.

OGSA provides a common architecture for developing grid-based applications where Web Services are the underlying middleware. Web Services can, in principle, be stateless or stateful, however they are usually stateless and there is no standard way of making them standard. The Open Grid Services Infrastructure (OGSI) was a GGF proposal that intended to provide an infrastructure layer for OGSA. OGSI [45] addressed the statelessness issues by extending Web Services to accommodate statefull Grid resources. It essentially defined a mechanism for creating, managing and exchanging information among Grid Services by extending WSDL and XML Schema. OGSI evolved into the Web Services Resources Framework (WSRF) specifications [46]. The specifications constitute WS-Resource, WS-Resource Properties, WS-Resource Lifetime, WS-Service Group, and WS-Base Fault. WSRF provides support for implementing stateful Web Services. WSRF competes for wider industry adoption with similar specifications. The Web Services Interoperability Organization (WS-I) is an industrial body that aims to achieve interoperability amongst the stack of Web Services specifications (WS-*). These specifications include WS-Security which provides means for applying security to Web Services by, for example, attaching signature and encryption headers to SOAP. WS-Addressing is another specification that defines mechanism allowing communicating addressing information between Web Services. Due to competing specifications, interoperability issues arise in the Web Services world. For instance, WS-Transfer, WS-Eventing and WS-Management standards proposed by Microsoft, IBM, Sun, and Intel are functionally similar to WSRF.

The Web Services functionalities are supported in WF through the basic activity library. The *InvokeWebServiceActivity* is used to invoke a Web Service from within a workflow. A reference to the Web Service is added to the workflow using its WSDL description file. This results in the generation of a proxy class to be used to invoke the Web Service once the activity is configured properly. WF workflows can also be published as Web Services, thus different workflows can communicate with each other if their instances are exposed as Web Services. The activities *WebServiceInputActivity* and *WebServiceOutputActivity* enable the workflow to be used as Web Service end points. The first activity enables a workflow to receive a Web Service request, and the second activity pairs with the first to respond to a Web Service request. The *WebServiceFaultActivity* pairs with *WebServiceInputActivity* to raise an exception packaged into a SOAP exception. Workflows published as Web Services are invoked from other workflows using the *InvokeWebServiceActivity*.

5.3 Scientific Workflows in Windows Workflow Foundation

Although WF is presented as a solution to business problems, the work in [47] presented an implementation of scientific workflows using WF in wind tunnel applications. The implementation demonstrated that WF is interoperable with Grid services, specifically the Globus grid services. The evaluation of BPEL for scientific workflows pushed researchers to identify the differences between business and scientific workflows, and the requirements for the latter. In [30], different tasks were identified when managing scientific workflows. The tasks included defining the workflows, deploying them and finally the enactment of the workflows. For WF, the basic activity library shipped with the framework provides the necessary support to invoke Web Services and to send and receive message content in and out of the workflow. Orchestrating different Web Services is enabled through control and data flow constructs, such as sequencing, repetitive and conditional execution of activities. These simple and complex constructs are supported in WF through activities like *IfElse, Parallel,* and *While*. WF also enables sub-workflows to be combined to define workflows that are more complex. The framework allows workflows to be published as Web Services, and invoked from other workflows using the *InvokeWorkflow* activity. WF makes a distinction between exceptions, transactions, and compensation. Consequently different handlers are defined for each type of failure. WF workflows can be deployed by the runtime engine provided by the framework. The WF framework provides a set of runtime services that enable the monitoring of workflow execution such as tracking, persistence, and transactions. The experimental implementation in [47] successfully orchestrated Globus Grid services in WF using MyCoG. We believe that WF is a good candidate for Grid service orchestration and scientific workflows development and deployment due to its lightweight and performance. A thorough analysis of the WF framework is required to prove that it satisfies the needs of orchestrating Grid services into scientific workflows.

Bioinformatics refer to the creation of algorithms and computational techniques to solve biological problems arising from analysing biological data. Performing *in-silico* experiments frequently requires bioinformaticians to use a combination of local applications and most importantly remote services owned by various organisations.

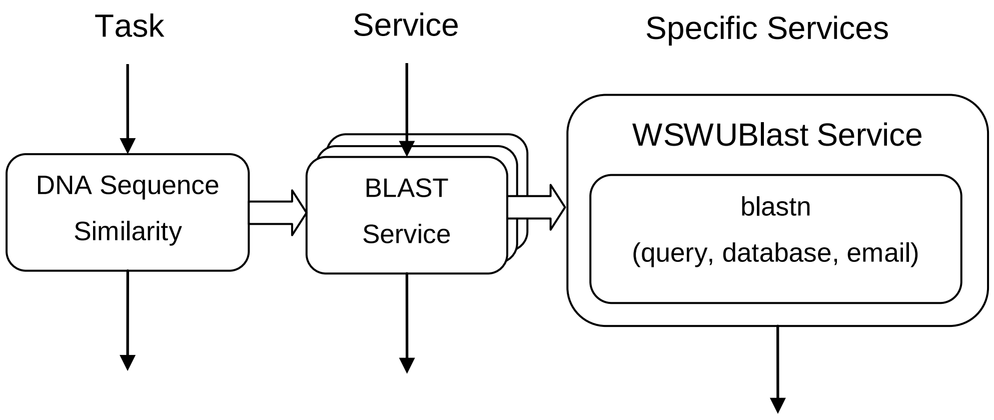

Figure 11 - An *in silico* experimental design: Seuqence Similarity Search

Figure 11 shows an example of an in silico experiment for the task of searching for similar sequences to a given DNA sequence. The bioinformatician identifies several services that implement sequence alignment methods. The user chooses to use an implementation of the BLAST algorithm. Finally, the specific WSWUBlast service is chosen. The user invokes the *blastn* method and supplies the corresponding parameters i.e. the DNA sequence to be queried, the database to search, and an email to receive the results. More complex bioinformatics tasks involve the execution of more services and most often the manual handling and management of generated data. Documenting the experiments into workflows and automating the process is what researchers in the field currently are trying to achieve [48-50]. The vision lies in developing workflows in an automatic or at least a semi-automatic way, aiming to minimize the efforts required by the user in conducting their experiments, by simplifying the task to a “drag and drop” process. By producing self documenting workflows and automating the execution of specified tasks, the complexity is hidden from scientific users as well as supporting collaboration by sharing data and experiments.

Chapter 6 Semantically Resolving Type Mismatches

6.1 Semantic Annotations in Windows Workflow Foundation

6.1.1 Semantic Parameter Binding in Scientific Workflows

One common approach in modelling scientific workflows is directed acyclic graphs (DAGs), where arcs denote scheduling dependencies between computation tasks called jobs [51, 52]. Alternatively, scientific workflow systems adopt expressive languages for modelling scientific workflows based on dataflow process networks [53, 54].

Dataflow is a natural paradigm for data-driven and data intensive scientific workflows. Workflows expressed using dataflow process networks can be efficiently analysed and scheduled, and are also a simple and intuitive model for workflow designers [55]. In addition to building workflow using the dataflow model, it is necessary to use control flow constructs such as branching, iteration, and concurrency in order to engineer robust and adaptive workflows. Constructs help build complex workflows that connect different Web Services and applications requiring the alignment of input and output data structures (schemas).

WF supports Web Services through the Web Service activity library. The framework provides dependency properties on activities as a mean to store their values or the workflow’s state. Activity binding binds a property on an activity to a property on another activity or on the workflow itself. Binding properties ensures data propagation between activities in the workflow. When composing activities to build a workflow, the user needs to bind the properties of the activities as they are added. At design time, WF validates the bindings between activities using a mechanism that checks the assignability of the runtime type of one property to another. Web Services activities expose their parameters (inputs, and outputs) as properties, which are linked to other properties i.e. parameters using the activity binding mechanism. A binding between two parameters from two composed activities is valid if their types are exactly the same, implement the same interface, or have an inheritance relationship. Syntactic matching is the key to successfully validating the compatibility of two types.

Web Services are usually owned and provided by different organisations. Developers of these Web Services do not necessarily agree on the naming or the representation of data in their implementations, which is very essential in syntactic matching. This mechanism, however, has two flaws, first it omits equivalent types with different names, second it omits equivalent types that have different internal data representation. To overcome this problem, a new level of type description needs to be introduced.

In order to convey the semantic information about data passed between activities in a workflow, we proposed the annotation of these data with semantic concepts. This allowed us to use semantic matching technique to validate data bindings on the syntactic level as well as the semantic level. Among the different annotation mechanisms we chose SAWSDL. SAWSDL builds on existing Web Services standard, so the implementation of SAWSDL-based applications is more efficient. By using model references to point to semantic concepts in existing ontologies gives the developer access to a wide and rich range of ontologies in different domains. Finally, it enables semantic interoperability by supporting rich mapping mechanism between Web Services XML Schema types and ontologies.

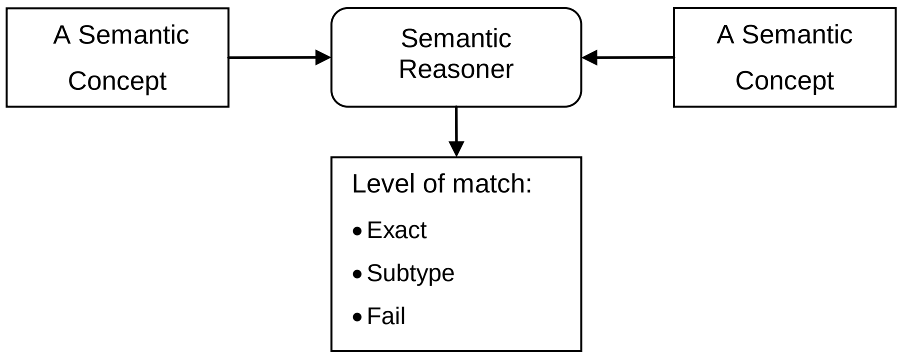

Figure 12 - Semantic Similarity Matching

We rely on semantic matching techniques automatically to connect semantically compatible between composed Web Services. Figure 12 summarises the semantic matching technique employed to find the level of match between two semantic concepts in some ontology representation language. When a service is added to the workflow, WF attempts to match automatically the input parameters of that service to the output parameters of the service it is connected to in the workflow. This ensures that all the data bindings between composed Web Services are semantically valid at design time. We identify three level of match, exact, subtype, and fail. The exact match denotes a semantic equivalence between the two semantic concepts. The two parameters can be safely connected. Subsumption means that a semantic concept is a subconcept or a superconcept of another semantic concept. In our matching engine, we consider a subtype match in one direction, i.e. it is safe to connect an input to an output if the input parameter's type is a subtype of the output parameter's type. If the reasoner fails to find a semantic match between two parameters, it is said that the match failed and the two parameters cannot be connected together due to the lack of sufficient semantic information to bind them automatically bind.

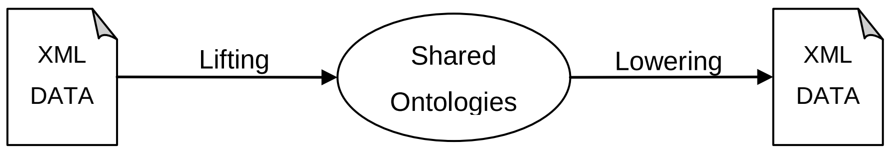

Figure 13 - XML Data Mediation

6.1.2 Parameter Mapping at Design Time

Model references operate at the semantic level and provide a safe type system where compatible parameters can be connected disregarding their syntactic differences in the case where they are semantically similar. However, as mentioned above even if compatible types are semantically similar they could structurally different.

Figure 13 illustrates how ontologies can act as mediators that can lift the data in XML format to data in the shared ontology and then lower it to another XML format using the lifting annotation from the first schema and the lowering one from the second schema. Using the combination of shared ontologies and schema mappings, resolving structural conflicts between compatible parameters is straightforward. As well as transforming data from one form to another, the schema mappings are essential to pass the necessary data from a supertype to its subtype. It is not until execution time that these mappings are executed.

6.1.3 Integration and Implementation

In order to support the SAWSDL annotations, we exploited WF’s extensibility feature and developed a custom activity to represent Semantic Web Services. The Semantic Web Service (SWS) activity extends the existing Web Service activity by supporting the model reference and both types of schema mappings. The SWS activity consumes SAWSDL documents and applies the necessary mechanisms in order to bind automatically compatible parameters between composed Web Services. The activity can act as a conventional Web Service activity and consume WSDL documents to generate the Web Service and execute it. Figure 14 illustrates the architecture of the tool, its components and their interactions.

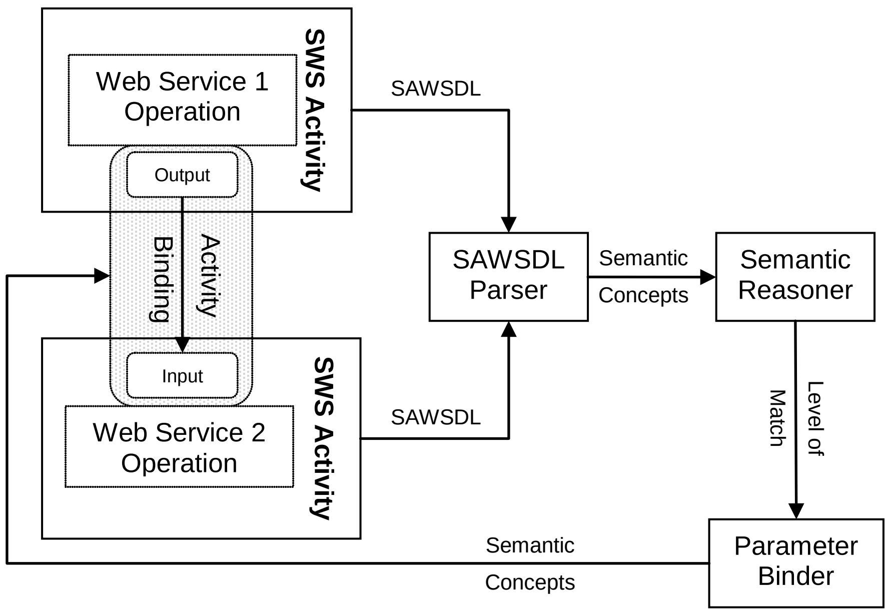

Figure 14 - The Architecture of the Automatic Binding System

The .NET framework provides standard libraries for developing WSDL 1.1 based applications, but no current support for WSDL 2.0 specifications. Furthermore, most of the Web Services in Bioinformatics and other scientific domains provide WSDL 1.1 description files. Due to the aforementioned reasons, we opted to provide an implementation for the WSDL 1.1 semantic annotations rather than WSDL 2.0. Supporting WSDL 2.0 semantic annotations can be providing by implementing translations in XSLT since both specifications are XML based. Our API extends .NET's WSDL 1.1 API by providing full support for all SAWSDL annotations including model reference and schema mappings.

The SAWSDL specifications do not restrict the annotation mechanism to a specific ontology representation language. However, for the sake of our implementation we selected OWL and RDF, being two W3C recommendations and widely used for developing ontologies. By adopting OWL and RDF we gained access to a wide range of existing domain models e.g. life sciences and healthcare. What's more, OWL and RDF are well supported by the research community. Part of implementing our tool required us to integrate the reasoning capability in order to perform the semantic matching between services' parameters. There are a few .NET libraries that provide read and write support of OWL and RDF, including SemWeb[1] and Euler[2]. However, None of these libraries, however, provide full, robust support and inferencing capabilities for OWL and RDF.

Jena [56] is an open source Semantic Web framework for Java. It provides a well supported API for OWL and RDF. The framework includes a few generic reasoners, but also supports the use of external reasoners such as Pellet [57]. To gain access to these features we had to make Java and C# interoperable. At this stage, two options were available: first expose the necessary semantic reasoning capabilities as a Web Service and invoke it whenever needed or second convert the Jena libraries to .NET. We opted for the second option since executing native .NET code is faster than exchange XML messages, as well as being more reliable. IKVM [3]is an implementation of Java for the .NET framework. It includes a Java Virtual Machine implemented in .NET, a .NET implementation of the Java class libraries, and tools that enable Java and .NET interoperability. IKVM provides a static compiler that converts Java API to .NET Common Intermediate Language (CIL), producing .NET Dynamic-Link Libraries (DLL). Using IKVM we recompiled the Jena libraries into a .NET library and used it to integrate the semantic reasoner into the SWS activity.

When the developed workflow is executed, the schema mappings associated with the data types are executed. No restriction exists on the choice of the mapping language, so we opt to use XSLT and SPARQL combination to support the bidirectional mapping. XSLT and XQuery are supported by a set of .NET library natively. We provide support for SPARQL using Jena's .NET libraries. At runtime, XML data is lifted to semantic data using XSLT and XQuery translations, and then lowered back to XML data using SPARQL queries and XSLT transformations.

[1] http://razor.occams.info/code/semweb/

[2] http://www.agfa.com/w3c/euler/

[3] http://www.ikvm.net/

6.2 Applying the Semantics to Scientific Workflows

We present a case study in the Bioinformatics domain in order to show how our tool will automatically bind the parameters of two Web Services. Several scientific organisations provide different public bioinformatics Web Services. The European Bioinformatics Institute (EBI) is one such organisation that provides access to different services, for example database retrieval and similarity searches. Most of the key data types in bioinformatics have multiple data representation. Most of the operations in bioinformatics services have weakly types parameters. In most cases, parameters are defined either as strings or as arrays of strings. The use of strings becomes ambiguous and inefficient when it comes to composing Web Services safely, thus the need for a strong typing system in the developing environment becomes necessary. We proposed introducing semantic annotations to the workflow environment. For the sake of our work, we suggested applying the SAWSDL annotations to the Windows Workflow Foundation environment. We claim that such a workflow system provides a strong typed system that ensures composing Web Services safely.

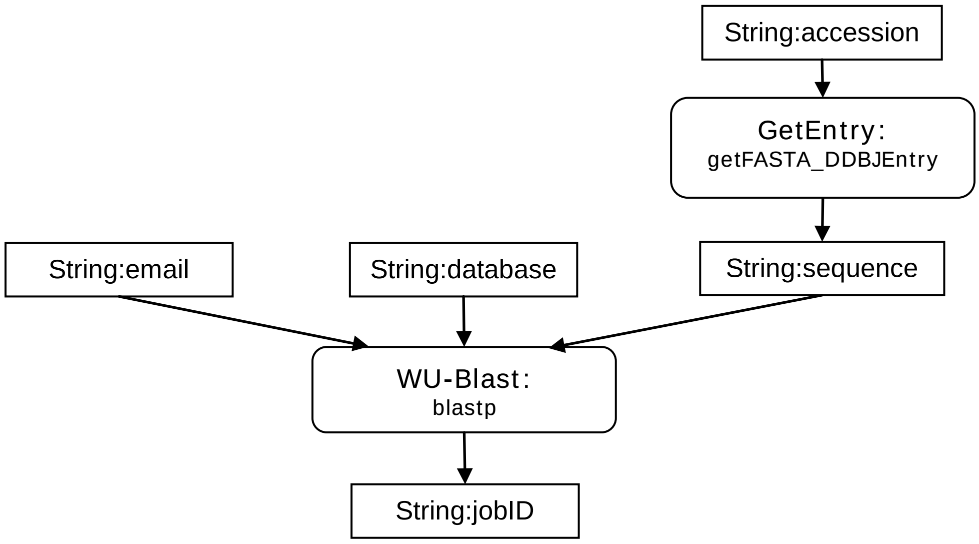

Figure 15 - A Bioinformatics Workflow Case Study

Figure 15 illustrates a simple typical workflow in bioinformatics. The task here is to find all the sequences that are similar to a given biological sequence. The workflow is composed of two Web Services. The first Web Service is *GetEntry* and it provides operations to retrieve entries from DNA and Protein databases in several formats using unique accession numbers. The *getFASTA_DDBIEntry* specifically retrieves a DNA sequence from the DNA Data Bank of Japan (DDBJ)[4] in the FASTA format. The input of this operation is *accession* which is of type string, and the output is *sequence*, also of type string. The second Web Service is *WSWUBlast*, standing for Washington University Basic Local Alignment Search Tool. It is used to compare a sequence with those contained in nucleotide and protein databases. The *blastp* operation takes a *protein sequence* and compares it against a protein database. In addition to the sequence query, the user needs to specify the *database* to use and an *email* for retrieving the results. The sequence parameter is of type string, but it does not convey the nature of the sequence, in other words whether it is a DNA sequence or a protein sequence. The operation returns a *jobID* to retrieve the search results. The example above just gives a simple scenario where data is not well annotated and maintained. Working with a few Web Services could be manageable. However, as the tasks get more complex, and the workflows grows larger keeping a track of what services do and what kind of data is required becomes more difficult. The workflow above is successfully validated when built using WF and the conventional Web Service activity, where in fact we have a conflict of retrieving a DNA sequence and using the wrong algorithm to find similar matches.

We demonstrated how, by applying semantic annotations to WF, we can automatically dynamically bind parameters of composed Web Services at design time. This approach is also used to detect mismatches and conflicts between connected Web Services, thus becoming a debugging tool as well as a building tool. Figure 16 shows a simple example where the mismatch between the two parameters is detected. When the operation from GetEntry is invoked, it results in retrieving a DNA sequence form the DDBJ database. This output parameter is of type string. The similarity search operation from WU-Blast takes a sequence of type string and finds all the similar sequences. WF successfully validates this workflow at design time by using syntactic techniques. The semantics of the data passed from the first service to the second is not validated to verify that it is safe to execute the composition of the two services.

[4] http:www.ddbj.nig.ac.jp

The myGrid project provides an OWL version of a Domain Ontology[5] for bioinformatics concepts, such us genes, proteins, and enzymes. We annotated the sequence output parameter of getFasta_DDBJEntry with the semantic concept *DNA_sequence*, and the sequence input parameter of blastp with *protein_sequence*. *Protein_sequence* is a subclass of the *biological_sequence* concept, and *DNA_sequence* is a subclass of the *nucleotide_sequence*, which is in itself a subclass of the concept *biological_sequence*. When the second service is added to the workflow, WF tries to

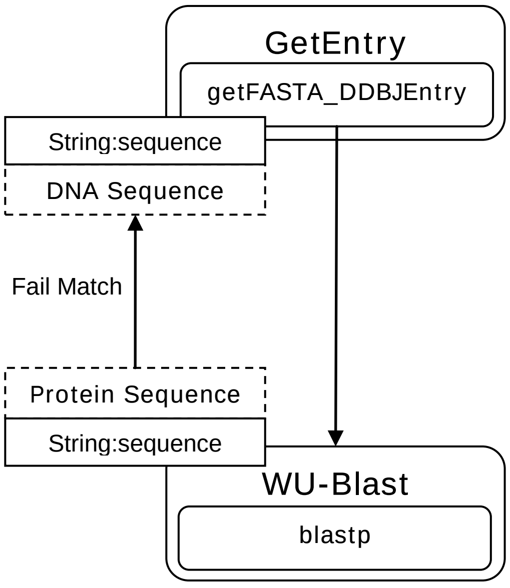

Figure 16 - The Semantic Annotations Applied to WF

bind the two parameters by trying to match between the two parameters. The two semantic concepts are not equivalent nor are they subsumed by one another. The match fails, and WF detects the mismatch and reports it back to the user.

[5] http://www.mygrid.co.uk/ontology

Chapter 7 Workflow Compensations Mechanisms

7.1 Workflow Failure Handling

Our approach already ensures type safety during workflow composition. However, abnormal situations such as system failures and deviations (exceptions) are unavoidable. Proper exception handling mechanisms are needed to deal with those deviations. Validating workflows is important to users such as scientists as it ensures the correctness and the reliability of their experiments. Different approaches have been proposed to validate workflows.

Some efforts use provenance in their validation techniques [58]. The mechanism stores metadata about processes, operations, and data types after workflow execution. Validation uses semantic reasoning over provenance data such as XML data and some specified properties such as XML Schemas. The use of provenance verifies the correctness of a workflow after its execution. However, it does not handle exceptions at execution time, which ensures terminating faulting workflows to a correct and stable state.

Another approach applies atomicity rules used in database transactions on activities in workflows [59]. This notion of atomicity is supported by an event log presented as a provenance system that handles system failures. However, as discussed in [60], the use of traditional ACID transactions to deal with errors is not useful in Web Services due to differences from closely coupled systems. The ACID properties are not present in Web Services. Since cancelling atomic activities is not feasible, compensation needs to be associated with a scope, which groups related transactions to be cancelled.

Besides capturing data semantics, a different approach [61] proposes logging data dependencies in order to recover from failures. The recovery mechanism restarts faulting workflows, and reconstructs them by tracking the execution logs. This approach does not verify the reliability of the workflow before its execution, and ignores the importance of defining explicit exceptions and compensations to handle errors and faults.

In the following sections, we will be analysing compensations, and the way they differ from exceptions. We will be reviewing the compensations mechanisms introduced in the de facto workflow composition language BPEL, and identifying the main issues with the proposed recovery mechanisms. In light the of our BPEL compensations analysis we will be reviewing the WF compensations mechanisms, and our proposed approach to solve the issues associated with deviant workflows.

7.2 Workflow Compensation Analysis

Workflows involve hierarchies of activities whose execution needs to be orchestrated. These activities typically involve interactions and coordination between multiple partners. Faults may happen at any stage during the execution of the activities. Standard atomic transactions, such as database transactions, use rollback mechanisms to handle faults, thus maintaining the atomicity property. However, in long running transactions (LRT), rollback is not always possible because parts of the transaction will have been committed, or cannot be undone using automatic techniques. Compensations can partially solve this issue by providing mechanisms that semantically undo the effects of an executed activity.

Workflows (or any orchestration language) can provide constructs through which compensations for actions can be declared. In the context of BPEL and WF, these constructs are called *compensation handlers*. Compensation handlers are associated with scopes of activities in workflows, and they can be nested arbitrarily. Compensations are intended to be a backward recovery mechanism since they can only be invoked on successfully completed scopes. Compensations can only be defined as fault (exception) handlers. Unlike simple exception handlers, compensations attempt to restore the workflow to a consistent state rather than just abort or terminate the execution. Once a workflow is restored, forward handling mechanisms can be applied in order to restart and resume the workflow execution by either retrying the same execution path or trying alternatives. An important aspect of Web Services workflows is that not all activities are automated and most resources cannot be locked. In these scenarios, reversing the effect of completed activities cannot be accomplished, and the use of forward and backward handling mechanisms is more difficult. Compensations can be nested and applied to scopes at different levels of the workflow. It is necessary to define clearly how such complex compensations are executed in order to guarantee a consistent recovery process. For example, concurrent activities might have compensations associated to the scope of each of them as well as the scope of their composition. The history of workflow execution is necessary here to define the backward execution path.

7.2.1 BPEL Compensations

Since traditional ACID techniques may not be used with LRTs, the BPEL specification defines mechanisms to deal with unforeseeable faults, i.e. events that occur contrary to the expected behaviour. These fault-handling mechanisms were inherited from XLANG. XLANG defined constructs to handle and raise exceptions, as well as specify compensation blocks that compensate long running transactions. These constructs can also be used in BPEL to handle faults and deal with LRTs. Compensation constructs in BPEL attempt to undo the effect of executed activities before a fault. However, how many activities should or could be compensated depends on the situation. As workflows get more complex, the task of designing compensation and fault solutions becomes more difficult [62].

Activities in BPEL can be associated with *scopes*, which provide a context for their execution behaviour. Each scope requires a primary activity that defines its normal behaviour. The primary activity can be a complex activity, with many nested activities that all share the context provided by their enclosing scope. Scopes themselves can be nested to construct complex hierarchies.

In order to handle faults and errors in LRTs, BPEL provides *compensation* constructs. Compensations in BPEL provide a mechanism to reverse the effect of committed transactions as best as possible. The logic of a compensation is defined within a *compensationHandler*. Compensations can be associated with a particular scope, and are only installed after its successful execution. An unhandled fault causes the invocation of all the compensations in the workflow. This is defined in BPEL as *default compensation*.

Default compensations simply attempt to terminate the workflow after trying to restore it to a stable state. However, this approach is not fault tolerant as it does not attempt to minimize the effect of the fault and relies completely on the designer to define the fault handling logic. Furthermore, the termination of the whole workflow will cost the user any results acquired during the execution of the workflow, as well as having the possibility of causing inconsistency across the non-isolated transactions.

The recovery mechanism provided by BPEL offers limited capabilities that are not enough to define the handling logic of complex scenarios. To alleviate the complexity of designing strong fault handling solutions, some approaches proposed enhancing the design capabilities through improving various aspects of the language [63]. Using an XML annotation mechanism, a designer can provide meta-descriptions that can be used to generate the appropriate BPEL constructs. In an effort to simplify the construction of compensation handlers, their approach allows the specification of *safe points*. Any faults occurring beyond a certain safe point will be propagated up to that point, causing the invocation of any installed compensations in reverse order. This approach relies on the designer to specify points in the workflow where he thinks it is safe to restart from, with the assumption that the state of the workflow is stable enough to resume its execution.

An alternative approach is to define a fault handling logic that produces fault tolerant BPEL workflows [64]. This approach separates the business logic of the workflow and its fault handling logic. Specifically, the fault handling logic is specified by a set of Event-Condition-Action (ECA) rules that build on fault-tolerant patterns. These ECA rules are consumed at runtime with the business logic to generate business processes. Some of the patterns used include *Ignore, Skip,* and *Retry*. These patterns represent the action section of the ECA rules, and can be specific to the various types of faults emitted by the faulting scope. This leaves the task of specifying the fault types, and the different actions to be taken in different cases. In both proposed approaches, the design of the fault logic is not verified for soundness and completeness. Practically speaking, the workflow is not validated against a set of clear semantics that guides the designer while specifying the fault handling logic.

We look next at Windows Workflow Foundation (WF) and the different fault handling mechanisms it provides, and highlight the main differences between the two standards.

7.2.2 WF Compensations

WF provides a rollback mechanism for conventional short lived transactions, as well as compensating mechanism to handle long running transactions. The ACID properties are applied when developing and executing short lived transactions since resources can be locked and changes are not committed until the complete successful execution of transaction. The activity *CompensatableTransaction* in WF provides a way to define the logic of a short lived transaction. This type of transaction can handle faults occurring before and after committing. Roll back techniques are used to handle faults occurring while the transaction is being executed and, since the ACID properties are enforced, it is safe to just restore the state of the workflow. When a transaction is successfully executed, all the changes are committed to the workflow and the locks on resources used are released. A compensating activity can be associated with the transaction so that fault occurring later can be used to attempt to compensate the effect of the transaction.

Long running transactions cannot lock resources for an extended period of time. They do not, therefore, possess atomicity and isolation. Since a long running transaction is defined by the nesting and composition of activities within its scope, it is considered committed when the last statement in it has completed. Long running transactions can be defined in WF through the *CompensatableSequence* activity. Since the ACID properties cannot be maintained, compensation serve as a fault handling mechanism that can help mitigate the effect of a committed transaction in a way.

Fault Scope	CompensatableSequence	CompensatableTransaction
Inside	Fault can be caught by fault handler but compensation handler cannot be invoked	The transaction can be rolled back using the persistence service
Outside	The fault triggers the compensation for the transaction	

Table 1 - Fault Handling

Above is a simple table showing the similarity and difference between the fault handling mechanisms of short lived and long running transactions.

Compensations can be associated with short lived and long running transactions, and they can be invoked explicitly or implicitly. Explicit invocation of compensations can be made through the *compensate* call from fault handlers or compensation handlers. Through explicit compensations developers can define different fault handling mechanisms. In the absence of such constructs WF's runtime engine will, however, implicitly invoke all the compensations in the workflow and attempt to bring it to the initial state. The latter mechanism is rather abrupt and it does not provide or guarantees a sound fault handling mechanism. Furthermore, faults occurring within a long running transactions are not properly handled since committed changes within the scope of the transaction cannot be reversed using rollbacks or compensations. This leaves the workflow in an unstable state.

7.2.3 Evaluation of fault handling mechanisms

Although the BPEL specifications define how default compensations are implicitly invoked, this mechanism fails to properly handle specific scenarios where faults causes the workflow to invoke all the installed compensations. We show the particular issues that we believe are unforeseen yet important to minimize the effect of unexpected faults. We have deployed our BPEL test cases on two deployment engines Oracle's BPEL Process Manager [65] and Sun's BPEL Service Engine [66]. Figure 17 illustrates how the workflow behaves when a fault occurs within a scope associated with a compensation handler. In the example below, when a fault occurs within the scope S2, its compensation handler CH(S2) is not invoked. If the fault is not caught by a fault handler in S2, then any effects that resulted from executing the activities of S2 are not compensated; this may leave the workflow in a incorrect state. Consequently, since the workflow cannot recover from its faulting state, all the installed compensations will be invoked, i.e. CH(S1), in attempt to undo the workflow execution. However, this approach does not guarantee that the workflow has been brought to a stable state, and it might even affect further attempts to execute the workflow again.

WF does not provide formal specifications to workflow definitions, but its deployment

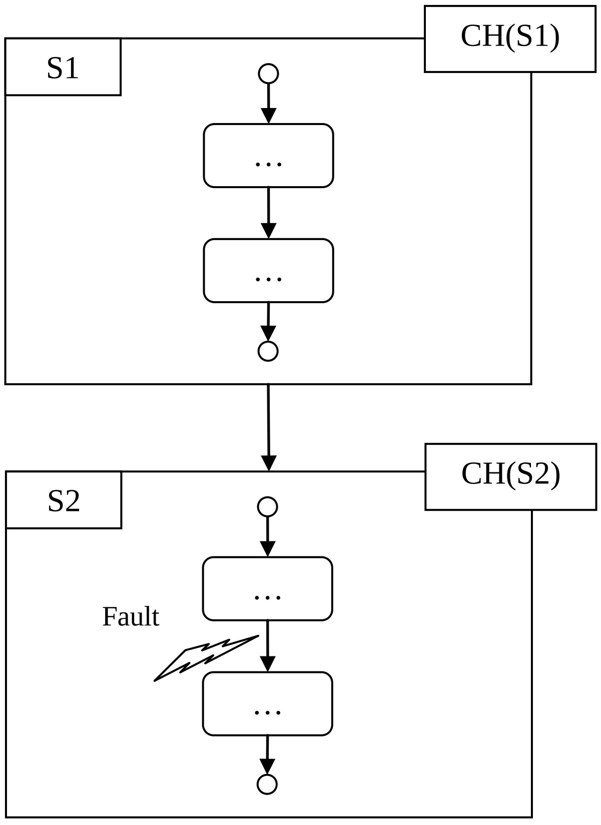

Figure 17 – Fault within scope

engine behaves in a similar way to its BPEL counterpart. We further illustrate further how fault propagation affects the compensation mechanism in WF. Through fault unwinding, compensation invocation can be carried out in a controlled manner. As the fault is propagated through the workflow, specific compensation handlers are explicitly invoked. By rethrowing a fault from one scope to an outer one, fault handlers are supposed to contain the effect of the fault. However, in WF rethrowing a fault will trigger the compensation handler of any successfully executed scope within the throwing scope. This mechanism makes more difficult to define robust fault handling mechanism as WF takes over.

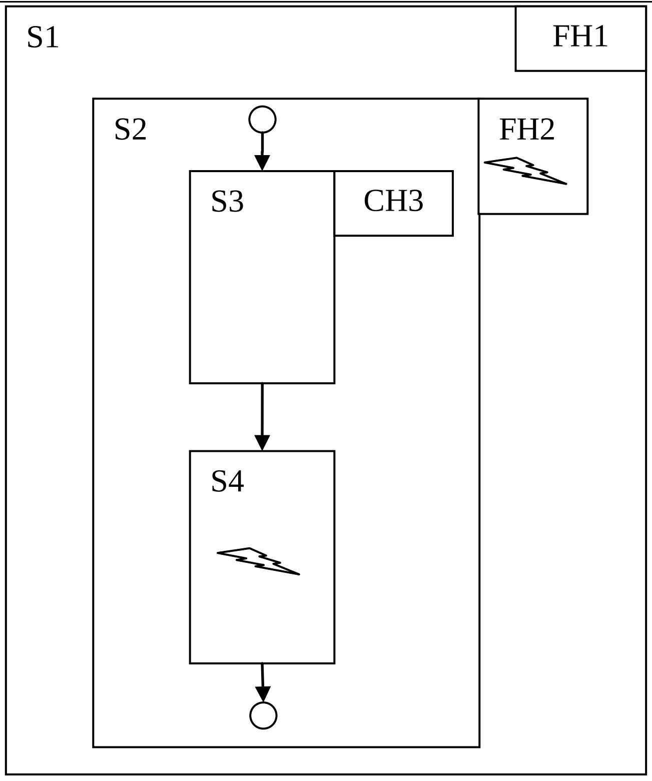

Figure 18 - Fault Propagation

Figure 18 provides a scenario where rethrowing a fault will trigger the default compensations of the workflow. Scope S4 throws a fault that is caught by S2's fault handler FH2. FH2 decides not to compensate S3, and rethrows the fault to S1. Before the fault is caught by S1's fault handler FH1, WF will flag S2 as faulting and will trigger its default compensation, which will invoke all the installed compensations of its inner scopes, i.e. the compensation CH3 of S3. The WF runtime engine does not deliver the expected behaviour when invoking default compensations; we therefore propose keeping track of all the installed compensations during the workflow execution and instead push the control to the compensation handlers. This will enable us to activate and deactivate the compensations depending on the annotations of the different scopes. The user will still be able to use the compensation construct. The compensating logic will, however, be wrapped in a semantically controlled construct, so that we verify defined explicit compensations, and we make implicit compensations explicit like.

In the following section, we will investigate how semantic enriched workflows might guide developers into defining semantically reliable workflows. We will demonstrate our approach by implementing a prototype tool for Microsoft’s Windows Workflow Foundation.

7.3 Semantics of Reliable Workflows

The Semantic Web technology allows the annotation of concepts from a specific knowledge domain to content, so that information can be derived from these data by employing semantic reasoning techniques. If a sound ontology could be engineered to describe a specific domain, it can be used to enrich and validate content in a compositional way. We have already enforced type safety in workflow systems through the annotation of types in Web Services. Our approach allows the semantic augmentation of workflows so that runtime type mismatches are handled at design time [67]. A few approaches tackled this issue and introduced fault handling and recovery systems.

7.3.1 Workflow Verification and Validation

Recovery approaches are usually based on a standard notion of explicit fault handlers known from programming languages such as Java or C++, and compensation actions for undoing effects of unsuccessfully finished activities. Some efforts relied on the classification of faults within a hierarchy of events [68, 69]; in these approaches, as the workflow is executed, various events are emitted and structured to be used by what the authors define as Constraint violation handlers (CV-Handlers). CV-Handlers are essentially event handlers that are triggered by specific events defined in the recovery ontology that are emitted by the system. Recovery actions are defined in these CV-Handlers to handle workflow faults properly. The events ontology does define different types of events depending on the emitting action, however it does not add to the existing compensation mechanism. The proposed compensation approach is easily comparable to the one found in BPEL or WF, and does not exploit the semantic annotation accumulated during the workflow execution.

Compensations are designed to undo the effects of executing an activity. However, a scope activity can only be associated with a single compensation handler. This implies that compensation handlers do not distinguish between different fault types, i.e. if a compensation is invoked it can compensate the activity in a generic way; and proper handling semantics cannot be specified. Although the BPEL compensation semantics have not been extended to allow multiple compensations, a transaction language called Structured Activity Compensation (StAC) [70] which defines compensation constructs comparable to BPEL has been extended to allow an activity to have more than one compensation handler [71]. This approach might alleviate the ambiguity of default compensations, but it does not address it directly.

Semantically BPEL or WF cannot be extended to accommodate multiple compensations, but it can be pushed to the compensation itself. Multiple compensations allow the indexing of different compensating logics, then the selective invocation of these compensations. We briefly give an example of how multiple compensations can be defined and invoked.

We aim to develop an ontology specific to the compensation concept, where the user can annotate the workflow component with a set of concept, which should enable the verification of the compensation constructs and how they should be defined to handle various faults. One approach to realise this system is through controlling fault unwinding. As previously explained, default compensations have the defect of unreliably propagating a fault through a workflow. This approach overlooks the effect of executing a compensation on the workflow. Furthermore, complete fault unwinding neglects the possibility that the workflow could be partially compensated, hence bringing it to a state where some results could be retained, or where some components could be restarted.

7.3.2 Semantics of WF Compensation

The concept of default compensation is ambiguous, and it can be regarded as an emergency recovery mechanism where unexpected behaviour may trigger handled faults and cause the whole workflow to terminate. We aim to assess the semantics of compensatable scopes and assist the developer in defining compensation handlers where necessary. This is to avoid the invocation of default compensation and make sure that the workflow always terminates in a stable state.

We provide here an initial insight into the defined semantics of compensatable scopes in WF and the associated compensation handlers. For the sake of completeness we also provide annotation data to scopes not requiring compensations. Activities that do not change global state such as searching for a flight do not require compensating actions since they do not have an effect on the state of the workflow. These scopes can be annotated as “non-compensatable” scopes since they do not require compensation, which will help the algorithm decide how to handle fault within them.

Due to the distributed nature of long running activities, it is not always possible to recover from faults. We might refer to activities with such semantics as “non-recoverable” activities. These activities do not have compensations associated to them since they cannot be compensated. Faults emitted from these activities might be referred to as “Fatal faults”.

Compensatable scopes can be categorized as fully and partially compensatable. Fully compensatable scopes are associated with compensating actions that semantically undoes the effect of the scope. For example, getting a full refund for a flight ticket can be regarded as a compensating action for a fully compensatable scope that is the purchase of a flight ticket. Partially compensatable scopes are associated with compensating actions that is not equal to the effect of the scope. For example, getting a 90% refund of flight cancellation can be regarded as a partial compensation for a partially compensatable scope.

The above classification can be used as a guideline to develop an ontology that categorizes scopes, and their semantic compensatability. We also propose defining an ontology to describe compensations themselves. The semantics can be used to annotate compensating actions in a workflow. We believe that providing information on the compensability of a scope and the available compensating actions will optimize the reasoning capabilities when devising recovery strategies.

The first evident action of a compensation is actually to compensate. This is obviously dependent on the compensatability of the scope. A fully compensatable scope infers that the compensating action will fully compensate the scope, and likewise for partially compensatable scopes. A compensation may rethrow a fault if the error needs to be propagated further. A skip action can be regarded as the negating action of a rethrow action. A scope can be skipped after it was compensated or if it was deemed as non-compensatable, in either case the workflow will resume execution. A terminate action is a workflow level action that will trigger all the installed compensations in a workflow. Such action can be invoked to handle fatal faults emitted from a faulting critical scope. A further compensating action is retrying a compensated scope. This can also be regarded as a recovery mechanism, since the workflow attempts to contain the fault and resume workflow execution at a certain checkpoint. These compensating actions are not exclusive. For an example, a compensating action can compensate a scope then rethrow a fault or decide to skip.

We give below a case study on the usage of the proposed semantic annotations and the implementation details of the recovery actions.

In Figure 19, we present a simple scenario outlining how our semantics can be applied to a workflow. In the presented example, a user can attempt to book a flight, then a hotel, and finally a car.

Each activity has an associated cancellation handler, where a business logic can defined to appropriately cancel the effect of the activity and propagate the failure in the workflow. This provides the ability to handle failures in workflows at various levels of granularity.

We assume all the activities defined in the proposed scenario can be compensated, whether fully or partially. A fully compensatable booking activity is a fully refundable one upon cancellation. If a cancelled booking incurs a charge, we define is a partially compensatable one. Now we can classify the flight and car booking services as fully compensatable activities, assuming that they are fully refundable. We also classify the hotel booking service as a partially compensatable, activity, since cancellation incurs a charge.

The workflow to handle the cancellation of these bookings is defined the activity level, and explicitly in the cancellation handler of each activity or scope.

We can also annotate the compensations with our semantics as follows. The compensations for booking the flight and hotel can be annotated as “compensate” and “rethrow”. This means that if any of these scopes fail, their compensations are invoked, and the fault is propagated to the enclosing scope. The compensation for the car booking service can be annotated as “compensate” and “skip”. Failing in booking a car should not affect the execution of the workflow. The effect of executing the service should be compensated, and the workflow should resume it execution resulting in a successful booking of a flight and a hotel.

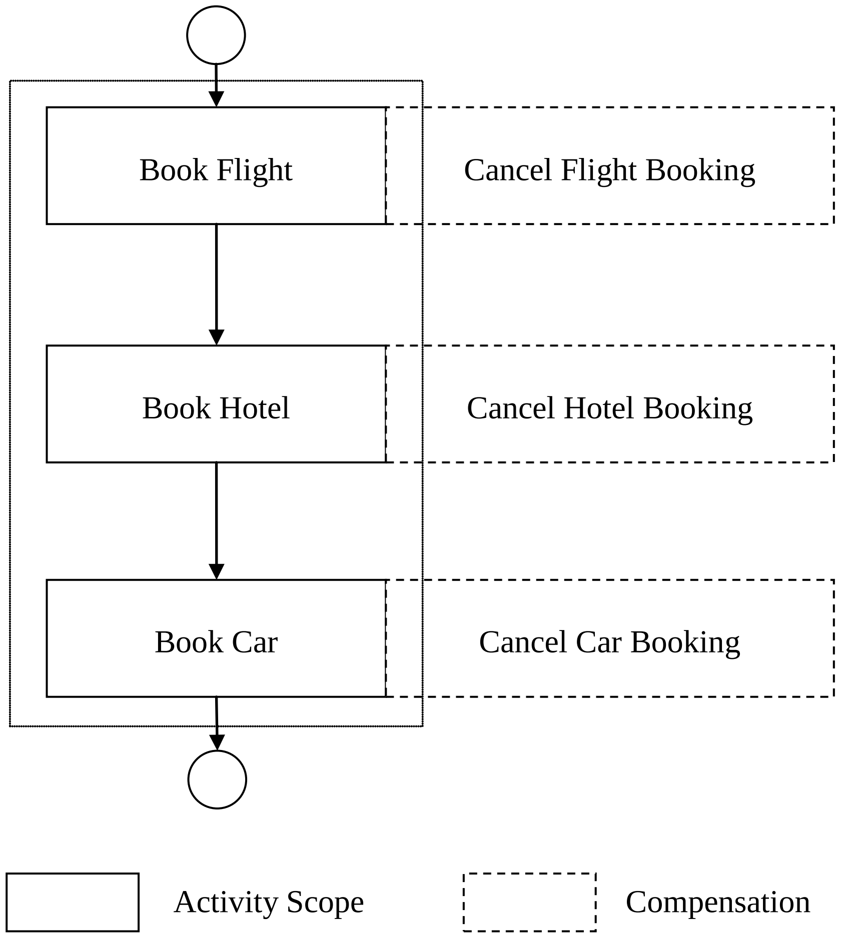

Figure 19 - Compensation Sematnics

In the light of this scenario, we can also use the annotations to devise various recovery strategies where the state of the workflow execution can be easily manipulated. Using semantic reasoning capabilities, we use the semantic annotations of activities and associated compensations to validate the defined handling logic of the compensating actions. For example, a non-recoverable scope would require the invocation of the terminate action in order to invoke all the compensations and halt the execution of the workflow.

Chapter 8 Conclusions and Future Work

8.1 Conclusion

In this thesis we discussed the open problem of structural and semantic mismatches associated with data in data-driven workflows, and we presented our ongoing approach that augments services with annotations in order to ensure type safety and workflow correctness through grounding type semantics to concrete data structures. We identified that mismatches can occur at two levels during workflow composition. Existing approaches like Taverna deal with structural mismatches between data and ignore the compatibility at the conceptual level. Other workflow systems like Kepler identify and separate semantic data from its concrete grounding, thus do not provide mapping solutions for structural mismatches. The rest of the workflow systems do not address either issue, delegating the task of ensuring workflow correctness to developers. Our approach provides a strongly typed workflow system, where mismatches are detected at the conceptual level as well as the concrete level. To this purpose we developed a prototype that implements a collection of semantic technologies to realise both approaches. We argued the reliability of using Windows Workflow Foundation framework as scientific workflow development system. The framework provides robust support for Web Services and allows us to build complex data-driven and control-driven workflows. However, it is not enabled for semantic type verification, so no mechanism exists in order to track the consistency of semantic information in propagated data. Annotating data with metadata captures the semantic information required to carry out type verification at the semantic level. Current Grid services are based on Web Services standards as defined by the OGSA specifications. We made use of Web Services annotation technologies to achieve our goal. Our tool relies on

Semantic Web techniques such as semantic reasoning to assist the workflow composition task. We extended the WF framework to make it compliant with SAWSDL specifications. We demonstrated the effectiveness of our approach using an in silico experiment in Bioinformatics as a test case. Our proposed approach provides a safe type system for a sound workflow development environment, as well as a reliable grounding mechanism for semantic data to enable workflow execution.

Another aspect of workflow development that we investigated in this report is workflow reliability. Current workflow systems provide mechanisms to handle and recover from failures. Activities in workflows can be associated with fault, event and compensation handlers. Fault handlers deal with faults emitted from an activity. They also implicitly invoke rollbacks in the case of short lived transactional activities. Long running transactional activities cannot lock resources for long periods, and thus the effect of its execution cannot be isolated. Since LRTs typically cannot be undone, compensations provide a mechanism to define a recovery logic for the effect of these activities.

Workflow composition languages such as BPEL and WF define compensations; their specifications, however, are ambiguous and may get complicated when dealing with complex workflows. Two types of compensations are identified, explicit and implicit. While explicit compensations have to be invoked in order to compensate a successfully completed activity, implicit compensations are usually invoked when an unhandled fault may cause the workflow to terminate. Implicit compensations can be considered emergency mechanisms where the workflow will try to invoke all the installed compensations then terminate the execution. This approach suffers from inflexibility and several efforts have attempted to alleviate its effect by introducing recovery mechanisms.

In our research, we have showed how a semantic approach can be used in order to develop an ontology to annotate activities and compensations in a workflow. These semantic annotations can be consumed in order to validate the compensations defined in the workflow. We attempt to eliminate the use of implicit compensations by invoking runtime recovery mechanisms that will make planned invocation of compensations rather than a complete workflow termination. This approach should leverage the task and guide workflow development. It should also tackle the shortcoming of current compensation approaches by ensuring workflow reliability and sensible fault recovery.

We have introduced an initial attempt at compensation and activity classification, and outlined how these semantics can be used to annotate workflows. We have also explained how WF can be extended to recover from workflow faults.

8.2 Future Work

We believe that our framework is realisable. With the proper definition of an activity and compensation ontology, we can extend WF to integrate the semantic annotation of workflow, as well as validation and recovery functionalities. Using reasoning mechanisms semantic information can be used to infer the necessary constraints to handle the failure of the workflow. Using our proposed approach semantics can be integrated to programming languages in order to provide a robust development environment. The C# programming language allows adding metadata through attributes. These attributes provide a method of associating information with C# code including types, methods, properties and so forth. The attributes can be queried at runtime by using reflections. Semantic workflow information can be used at execution time to monitor workflows for faults and trigger the correct handling mechanism, or infer a suitable one attempting to terminate the workflow and maintain a correct state.

Chapter 9 Bibliography

1. J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec, volume. 34, issue 3**,** pages 44-49, 2005.

2. Y. Gil et al. Artificial intelligence and grids: workflow planning and beyond. Intelligent Systems, IEEE, volume 19, issue 1, pages 26-33, 2004.

3. C. Kesselman and I. Foster. The Grid: Blueprint for a New Computing Infrastructure. 1998: Morgan Kaufmann Publishers, ISBN: 1558604758.

4. T. Berners-Lee, J. Hendler, and O. Lassila (May 17, 2001). "The Semantic Web". In *Scientific American Magazine*. [cited 2008 12].

5. I. Polikoff and D. Allemang. TopQuadrant Technology Briefing v1.2, Semantic Technology. 2004.

6. N.F. Noy, R.W. Fergerson, and M.A. Musen. The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 17-32, 2000. Springer-Verlag.

7. D. Quan. Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data. 2005.

8. N. Shadbolt et al. CS AKTive Space, or how we learned to stop worrying and love the semantic Web. Intelligent Systems, IEEE, volume 19, issue3, pages 41-47, 2004.

9. T. Berners-Lee. "Semantic Web - XML2000, slide 10". *W3C.* http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html [cited 2008 12]

10. F. Manola and E. Miller. RDF Primer: W3C Recommendation. 2004 [cited 2008 12].

11. D.L. McGuinness and F.V. Harmelen. OWL Web Ontology Language Overview W3C Recommendation. 2004 [cited 2008 12].

12. T. Scallan. A CORBA Primer. http://www.omg.org/news/whitepapers/seguecorba.pdf [cited 2008 12].

13. DCOM Technical Overview, Microsoft Coroporation, November 1996. http://msdn.microsoft.com/en-us/library/ms809340.aspx [cited 2008 12].

14. D. Box et al. Simple Object Access Protocol (SOAP) 1.1. 2000 [cited 2008 12].

15. M. Bell. Introduction to Service-Oriented Modeling. Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. ISBN 978-0-470-14111-3.

16. R. Chinnici et al. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation. 2007 [cited 2008 12].

17. T. Bellwood et al. UDDI Technical White Paper. www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, 2002.

18. D. Martin et al. OWL-S: Semantic Markup for Web Services: W3C Member Submission. 2004 [cited 2008 12].

19. R. Masuoka et al. Semantic Web and Ubiquitous Computing - Task Computing as an Example. In *AIS SIGSEMIS Bulletin*, volume 1, issue 3, pages 21-24, 2004.

20. H. Lausen, A. Polleres, and D. Roman. Web Service Modeling Ontology (WSMO): W3C Member Submission. 2005 [cited 2008 12].

21. J. Kopecky et al. SAWSDL: Semantic Annotations for WSDL and XML Schema. Internet Computing, IEEE, volume 11, issue 6, pages 60-67, 2007.

22. SAWSDL Candidate Recommendation Implementation Report. W3C. http://www.w3.org/2002/ws/sawsdl/CR/ [cited 2008 12]

23. Woden4SAWSDL. http://lsdis.cs.uga.edu/projects/meteor-s/opensource/woden4sawsdl/index.html [cited 2008 12]

24. Web Services Description Language for Java Toolkit (WSDL4J). http://sourceforge.net/projects/wsdl4j/ [cited 2008 12]

25. Ž. Turk et al. Towards Engineering on the Grid. In *proceedings of the 5th European conference on product and process modelling in the building and construction industry - ECPPM*. 2004. Istanbul, Turkey.

26. O. Corcho et al. An overview of S-OGSA: A Reference Semantic Grid Architecture. Journal of Web Semantics, volume 4, issue 2, pages 102-115, 2006.

27. M. Bubak and S. Unger. K-WfGrid - The Knowledge-based Workflow System for Grid Applications. In *Proceedings of CGW'06, Vol. II*. 2007: ACC CYFRONET AGH.

28. H. Zhuge. China's E-Science Knowledge Grid Environment. IEEE Intelligent Systems, volume 19, issue 1, pages 13-17, 2004.

29. T. Andrews et al. Business Process Execution Language for Web Services Verison 1.1. 2003 [cited 2008 12].

30. W. Emmerich et al. Grid Service Orchestration using the Business Process Execution Language (BPEL). Journal of Grid Computing, volume 3, issue 3. pages 283-304, 2005.

31. A. Akram, D. Meredith, and R. Allan. Evaluation of BPEL to Scientific Workflows. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*. 2006.

32. I. Foster. A Globus Primer. http://globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf, August 2005 [cited 2008 12].

33. G.V. Laszewski et al. GridAnt - Client Side Grid Workflow Management with Ant. 2003 [cited 2008 12].

34. D.W. Erwin and D.F. Snelling. UNICORE: A Grid Computing Environment. In *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*. 2001, Springer-Verlag.

35. B. Ludscher et al. Scientific workflow management and the Kepler system: Research Articles. Concurr. Comput. : Pract. Exper., volume 18, issue 10, pages 1039-1065, 2006.

36. S. Bowers and B. Ludäscher. Actor-Oriented Design of Scientific Workflows. In *Conceptual Modeling – ER 2005*. Pages 369-384, 2005.

37. J. Zhang. Ontology-Driven Composition and Validation of Scientific Grid Workflows in Kepler: a Case Study of Hyperspectral Image Processing. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops*. 2006, IEEE Computer Society.

38. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF: W3C Recommendation. 2008 [cited 2008 12].

39. D. Churches et al. Programming scientific and distributed workflow with Triana services: Research Articles. Concurr. Comput. : Pract. Exper., volume 18, issue 10, pages 1021-1037, 2006.

40. S. Majithia et al. Triana: a graphical Web service composition and execution toolkit. In *Web Services, 2004. Proceedings. IEEE International Conference on*. 2004.

41. I. Taylor et al. Triana Applications within Grid Computing and Peer to Peer Environments. Journal of Grid Computing, volume 1, issue 2, pages 199-217, 2003.

42. T. Oinn et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, volume 20, issue 17, pages 3045-3054, 2004.

43. P. Lord et al. Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery. In *The Semantic Web: Research and Applications*. Pages 17-31, 2005.

44. D. Hull et al. Treating shimantic web syndrome with ontologies. In *Proc. of AKT-SWS04*, 2004, ISSN: 1613-0073.

45. S. Tuecke et al. Open Grid Services Infrastructure (OGSI) version 1.0. 2003 [cited 2008 12].

46. T. Banks. Web Services Resource Framework (WSRF) - Primer v1.2. 2006 [cited 2008 12].

47. A. Paventhan et al. Leveraging Windows Workflow Foundation for Scientific Workflows in Wind Tunnel Applications. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*. 2006.

48. L.A. Digiampietri, C.B. Medeiros, and J.C. Setubal. A framework based on Web service orchestration for bioinformatics workflow management. Genet Mol Res, volume 4, issue 3**,** pages 535-54, 2005.

49. S. Bowers et al. Enabling ScientificWorkflow Reuse through Structured Composition of Dataflow and Control-Flow. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*. 2006, IEEE Computer Society.

50. M. Peleg, I. Yeh, and R.B. Altman. Modelling biological processes using workflow and Petri Net models. Bioinformatics, volume 18, issue 6, pages 825-837, 2002.

51. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience: Research Articles. Concurr. Comput. : Pract. Exper., volume 17, issues 2-4, pages 323-356, 2005.

52. E. Deelman et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Sci. Program., volume 13, issue 3, pages 219-237, 2005.

53. E.A. Lee and T.M. Parks. Dataflow process networks. In *Readings in hardware/software co-design,* pages 59-85, 2002. Kluwer Academic Publishers.

54. G. Kahn and D. Macqueen. Coroutines and Networks of Parallel Processes. In *Information Processing 77*. 1977: North Holland Publishing Company.

55. E.A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. In *Tutorial: hard real-time systems*. 1989, IEEE Computer Society Press. p. 237-248.

56. J.J. Carroll et al. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. 2004, ACM: New York, NY, USA.

57. B. Parsia and E. Sirin. Pellet: An OWL DL Reasoner. In *3rd International Semantic Web Conference (ISWC2004)*. 2004.

58. S. Miles et al. Provenance-based validation of e-science experiments. Web Semant., volume 5, issue 1, pages 28-38, 2007.

59. L. Wang et al. A Dataflow-Oriented Atomicity and Provenance System for Pipelined Scientific Workflows. In *Computational Science – ICCS 2007*. 2007. pages 244-252.

60. R. Lara et al. Semantic Web Services: Description Requirements and Current Technologies. 2003.

61. T. Tavares et al. An Efficient and Reliable Scientific Workflow System. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*. 2007.

62. H. Yi and P.A. Watters. On the Complexity of Compensation Handling in WS-BPEL 2.0 for 3rd Party Logistics. In *Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES*. 2007.

63. S. Modafferi and E. Conforti. Methods for Enabling Recovery Actions in Ws-BPEL. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*. 2006. pages 219-236.

64. L. An et al. A Declarative Approach to Enhancing the Reliability of BPEL Processes. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*. 2007.

65. M.J.C.S. Reis, G.M.M.C. Santos, and P.J.S.G. Ferreira. Promoting the educative use of the internet in Portuguese primary schools: a case study. Aslib Proceedings, volume 60, issue 2, pages 111-129, 2008.

66. B.C. Neves and H.F. Gomes. Digital inclusion in Brazil: an experience inside the university. BiD, issue 21, pages 5-5, 2008.

67. K. Derouiche and D.A. Nicole. Semantically Resolving Type Mismatches in Scientific Workflows. Lecture Notes in Computer Science On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops. Issue 4805, pages 125-135, 2007.

68. R. Vaculin and K. Sycara. Semantic Web Services Monitoring: An OWL-S Based Approach. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*. 2008.

69. R. Vaculin, K. Wiesner, and K. Sycara. Exception Handling and Recovery of Semantic Web Services. In *Networking and Services, 2008. ICNS 2008. Fourth International Conference on*. 2008.

70. M.J. Butler and C. Ferreira. A Process Compensation Language. In *Proceedings of the Second International Conference on Integrated Formal Methods*. 2000, Springer-Verlag.

71. M. Chessell et al. Extending the concept of transaction compensation. IBM Syst. J., volume 41, issue 4, pages 743-758, 2002.