Recording and Reasoning over Data Provenance in Web and Grid Services

Martin Szomszor and Luc Moreau
martinszomszor@yahoo.co.uk, L.Moreau@ecs.soton.ac.uk

School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ UK

Abstract. Large-scale, dynamic and open environments such as the Grid and Web Services build upon existing computing infrastructures to supply dependable and consistent large-scale computational systems. This kind of architecture has been adopted by the business and scientific communities allowing them to exploit extensive and diverse computing resources to perform complex data processing tasks. In such systems, results are often derived by composing multiple, geographically distributed, heterogeneous services as specified by intricate workflow management. This leads to the undesirable situation where the results are known, but the means by which they were achieved is not. With both scientific experiments and business transactions, the notion of lineage and dataset derivation is of paramount importance since without it, information is potentially worthless. We address the issue of *data provenance*, the description of the origin of a piece of data, in these environments showing the requirements, uses and implementation difficulties. We propose an infrastructure level support for a provenance recording capability for service-oriented architectures such as the Grid and Web Services. We also offer services to view and retrieve provenance and we provide a mechanism by which provenance is used to determine whether previous computed results are still up to date.

1 Introduction

Grids and Web Services are evolving into large-scale dynamic and open environments providing services owned and managed by multiple stakeholders [5]. The concept of virtual organisation (VO) is being anticipated by many [8, 5] as the computational model for coordinating the complex interactions of such services. A typical VO's lifecycle consists of the following steps: VO participants are discovered, the purpose and the terms of the VO are negotiated, the VO is created and then executed to deliver some result, before it is disbanded. At the end of a VO's lifetime, users potentially find themselves in the awkward situation in which they have access to the result of the computation without any information on why and how such a result has been obtained. The lack of information about the origin of results does not help users to trust such open environments, and therefore may hamper the deployment of advanced services [15].

Workflow enactment has become popular in the Web Services [14, 4] and Grid communities [11]. Workflow enactment can be seen as a mechanism offering a simpler form of virtual organisation: it is capable of composing Web Services, potentially discovered dynamically, according to data and control flows specified in a workflow language, such as WSFL [14] or BPEL4WS [4]. Similarly, users are confronted with the problem of determining the origin of a result produced by such enactment. Furthermore, deciding when results of computation, whether a scientific analysis or a business transaction, are no longer valid becomes an important concern.

Against this background, *provenance* is an annotation able to explain how a particular result has been derived; such a provenance information can be used to better identify the process that was used to reach a particular conclusion. Specifically, in a service-oriented architecture, provenance identifies what data is passed between services, what services are available, and what results are generated for particular sets of input values, etc. Using provenance, a user can trace the "process" that led to the aggregation of services producing a particular output.

It is our belief that provenance recording should be part of the infrastructure, so that users can elect to enable it when they execute their complex tasks over the Grid or in Web Services environments. Currently, the Web Services protocol stack and the Open Grid Services Architecture [7] do not provide any support for recording provenance, though the need has been acknowledged by that community, as illustrated by a recent workshop on provenance [18], and some Grid projects such as myGrid (`www.mygrid.org.uk`). Additionally, producing provenance data is of no use, if we do not provide the means of exploiting it — an activity referred to as *provenance reasoning* in this paper.

The purpose of this paper is to investigate the notion of provenance in services-oriented architecture, such as Grids and Web Services, and more specifically when computations are the result of workflow enactment. Our specific contributions are:

1. A service-oriented architecture for provenance support in Grid and Web Services environments, based on the idea of a provenance service;
2. A client-side API for recording provenance data for Web Service invocation;
3. A data model for storing provenance data;
4. A server-side interface for querying provenance data;
5. Two components making use of provenance: provenance browsing and provenance validation.

This paper is organised as follows. In Section 2, we discuss the notion of provenance and identify some of its requirements. In Section 3, we present the architecture for provenance support in a service-oriented environment. We then discuss some implementation aspects of our provenance system in Section 4. Finally, in Section 5, we analyse our design and conclude the paper by identifying issues that need further investigation.

2 Background

Provenance is primarily concerned with data derivation. It provides a proof of origin for a segment of data and a record of its history. This section looks at what provenance is, why it is useful and how it might be provided in a dynamic and open environment such as the Grid and Web Services.

2.1 Provenance

Provenance allows us to take a quantity of data and examine its lineage. Lineage shows each of the steps involved in sourcing, moving and processing the data [17]. In many of today's modern information systems, particularly those concerned with business and scientific data handling, data can be collated from a variety of distributed and diverse resources and processed to form new data. We can consider the sequence of taking a dataset, processing it and producing a dataset product as a dataset *transformation*. In order to provide provenance, all datasets and their transformations must be recorded. Saltz [19] suggests that we can achieve a sound lineage record by recording enough information to ensure that any dataset transformation is reproducible.

The storage and maintenance of provenance records is an important consideration. Frew and Bose [10] propose the following requirements for provenance collection:

1. A standard lineage representation is required so data lineage can be communicated reliably between systems (currently there is no standard lineage format).
2. Automated lineage recording is essential since humans are unlikely to record all the necessary information manually.
3. Unobtrusive information collecting is desirable so that current working practices are not disrupted.

The concept of providing provenance is relatively new and unexplored. It has attracted interest from both the academic and business communities where speculation about its benefits is widespread. Scientists are often interested in provenance because it allows them to view data in a derived view and make observations about its quality and reliability [3]. Goble [12] presents some notable uses for provenance:

- *Reliability and quality*
  Given a derived dataset we are able to cite its lineage and therefore measure its credibility. This is particularly important for data produced in scientific information systems.
- *Justification and audit*
  Provenance can be used to give a historical account of when and how data has been produced. In some situations, it will also show why certain derivations have been made.

– *Re-usability, reproducibility and repeatability*
  A provenance record not only shows how data has been produced, it provides all the necessary information to reproduce the results. In some cases the distinction between repeatability and reproducibility must be made. In scientific experiments results may be different due to observational error or processing may rely on external and volatile resources.
– *Change and evolution*
  Audit trails support the implementation of change management.
– *Ownership, security, credit and copyright*
  Provenance provides a trusted source from which we can procure who the information belongs to and precisely when and how it was created.

It could also be said that provenance significantly increases productivity by facilitating the discovery of existing derivations, which can save time, computational effort and storage. With a standard lineage representation, collaboration of resources is also made considerably easier [9,6].

2.2 Workflow Enactment and Provenance

The Grid [11] and Web Services communities [14,21,4] have identified workflow enactment as a key concept, essentially offering a scripted form of virtual organisation [8]. Workflow enactement is the automation of a process during which documents, information or tasks are passed from one participant to another for action, according to a set of declarative or procedural rules. In Grid applications, this task is often performed by a "workflow enactment engine". The enactment engine uses a workflow script, such as WSFL or BPEL4WS, to determine which services to call, the order to execute them in and how to pass datasets between them. Each of these services can be invoked by using their respective WSDL interface documents. This concept can be seen in Figure 1. This diagram shows a workflow enactment engine using a workflow script to control the flow of data between two Web Services. Each Web Service is passed an input and returns an output.

In terms of provenance, we can consider the input for a Web Service as a dataset, the Web Service itself as the transformation and the result as the data product. Therefore, provenance for a service-oriented architecture can be provided by logging all of the Web Services invoked by the enactment engine, the inputs given and the outputs returned. In order to meet the requirement that the provenance records must enable the re-creation of any dataset transformations, the WSDL documents for the services invoked must be recorded along with the workflow script describing their invocation context.

2.3 Accessing Provenance Records

Assuming that a system is capable of recording provenance, we should consider how these records are accessed and what they might be used for. A standard lineage representation should be formulated that is capable of recording the diverse

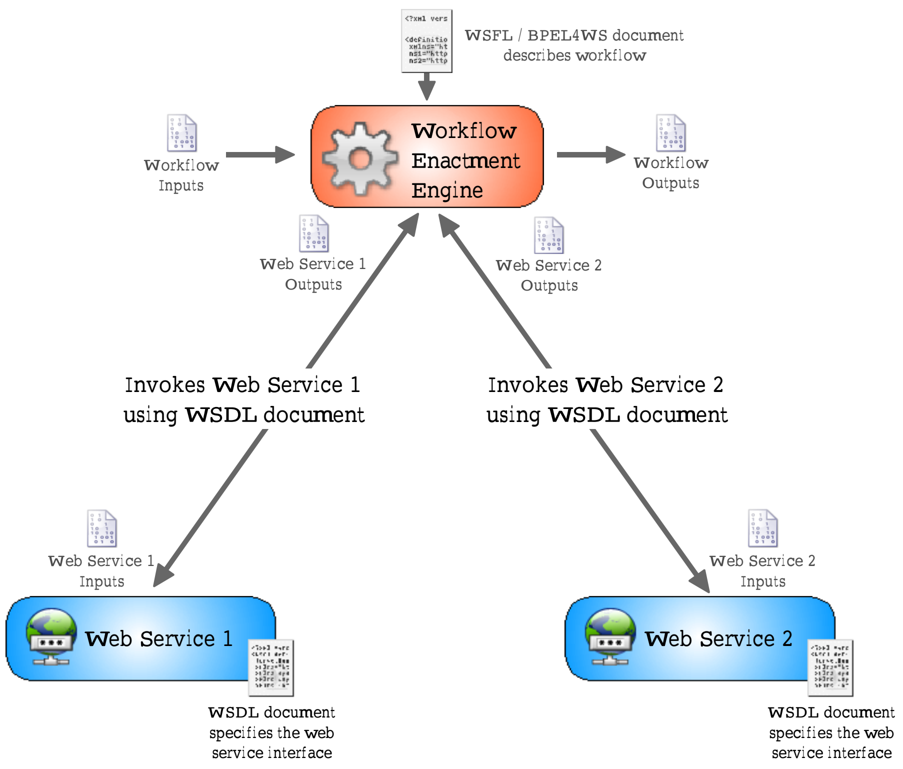

WSFL / BPEL4WS document describes workflow
Workflow Inputs
Workflow Enactment Engine
Workflow Outputs
Web Service 1 Outputs
Web Service 2 Outputs
Invokes Web Service 1 using WSDL document
Invokes Web Service 2 using WSDL document
Web Service 1 Inputs
Web Service 2 Inputs
Web Service 1
Web Service 2
WSDL document specifies the web service interface
WSDL document specifies the web service interface

Fig. 1. Workflow Enactment

range of services possible and the multifarious data structures used. Since the current workflow languages and service descriptions are represented using XML, the obvious syntactic framework for a provenance record is also XML. Some form of querying interface should be supplied that enables the retrieval of provenance records by both content and type. For example, it would be useful to query a provenance repository for records about a particular invoked Web Service, a specific workflow script used for enactment, or an exact observed output. It would also be desirable to provide an automated service to reproduce datasets and re-run dataset derivations.

2.4 Related Work

Other work [2] on provenance has concentrated on models grounded on database architectures in which queries are used to extract information. In general, if a query Q is applied to some database D, we can obtain the view V such that $V = Q(D)$. In this scenario, the provenance of a specific piece of data d in the output V shows which parts of the database D contributed to it. In [2], the important distinction between 'why' provenance (the set of tuples that contribute to the result) and 'where' provenance (the location(s) in the source database from which the result was extracted) is made. Using the explicit notions of location defined in the deterministic model [1] and a formal query language, a precise definition of provenance is formulated using a general data model that applies to both relational databases and hierarchical data structures such as XML.

MyGrid (`www.mygrid.org.uk`) aims at providing a personalised environment for bioinformaticians to perform *in silico* experiments [16]. In myGrid, provenance is stored in a user's personal repository and provenance generation is tightly integrated with the enactment engine [13]. In that context, the focus is not on the architecture and protocols required for supporting provenance in service-oriented architectures, but on personalising provenance information when presented to the scientist.

3 Architecture

A provenance capability in a Grid or Web Services environment has has two principal functions: to record provenance on dataset transformations executed, e.g. during workflow enactment, and to expose this provenance data in a consistent and logical format via a query interface. In this section, we discuss the design of a provenance architecture for Grid and Web Services, which is dictated primarily by where the data provenance is held, how this provenance is collected, what exactly constitutes a provenance record, and how provenance can be queried and reasoned over.

Provenance needs to be stored, and two possible solutions can be envisaged for storage. On the one hand, the data provenance is held alongside the data as metadata, whereas on the other hand, the data provenance can be stored in a dedicated repository, made accessible as a Grid or Web Service. The first solution requires the holders of any such data to maintain the integrity of the provenance records as transformations take place, and it imposes significant changes to any existing data storage structures. Such a kind of provenance can be useful for a user to remember how a result was derived, and what steps were involved. It is unclear that such provenance can be trusted by any third party, since the data and provenance owner has to be trusted to have recorded provenance properly.

Alternatively, the provenance service solution requires any party performing a dataset transformation to update the provenance service, with any new derivations and how they were performed. Many provenance services may exist over the Grid, some more trusted than others, and users may select which one to adopt when executing their workflows. By adopting a suitable protocol to submit provenance, some desirable properties can be enforced by the provenance service, such as non-repudiation (to be discussed in Section 5); the provenance service could therefore be used as a generalised auditing mechanisms, which can be trusted by any third party. While such protocols are interesting, they are beyond the scope of the current paper; and we will only focus on provenance being submitted by a workflow enactor.

Architecture Overview Our service-oriented provenance architecture is shown in Figure 2. As far as *provenance creation* is concerned, the architecture is composed of a server side and a client side. The Provenance Server holds the provenance records and provides methods to access and update them via a Web Service. The records themselves are held in a relational database. The Provenance Client Interface is responsible for submitting to the Provenance Server provenance records about any Web Service invocation executed by a client such as a workflow enactment engine. To this end, the Client Side Interface uses the Provenance Recording port provided by the Provenance Server.

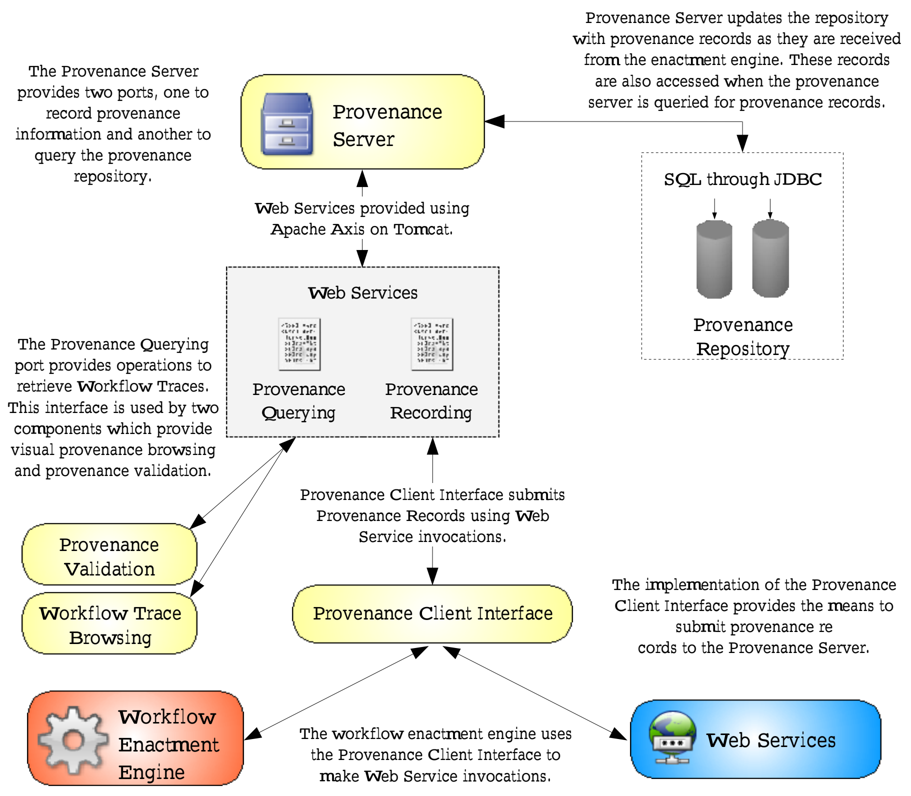

Provenance Server updates the repository with provenance records as they are received from the enactment engine. These records are also accessed when the provenance server is queried for provenance records.
The Provenance Server provides two ports, one to record provenance information and another to query the provenance repository.
Provenance Server
SQL through JDBC
Provenance Repository
Web Services provided using Apache Axis on Tomcat.
Web Services
Provenance Querying
Provenance Recording
The Provenance Querying port provides operations to retrieve Workflow Traces. This interface is used by two components which provide visual provenance browsing and provenance validation.
Provenance Client Interface submits Provenance Records using Web Service invocations.
Provenance Validation
Workflow Trace Browsing
Provenance Client Interface
The implementation of the Provenance Client Interface provides the means to submit provenance re cords to the Provenance Server.
Workflow Enactment Engine
The workflow enactment engine uses the Provenance Client Interface to make Web Service invocations.
Web Services

Fig. 2. Provenance Architecture

In order to provide data provenance on workflows, our approach is to record enough information to be able to re-create any dataset transformation (cf. Section 2.2). We are making here a simplifying assumption: we assume that all dataset transformations are carried out by invoking Web Services or workflows.

The order in which these services are invoked, the control flow and the data flow between services are governed by a workflow script, expressed in a workflow language such as WSFL or BPEL4WS. For any workflow execution, we store records in the provenance repository defining the workflow details, with precise information on each activity carried out. Any datasets passed to an activity will be logged along with the outputs that are received. We define a *workflow trace* as the conglomeration of provenance records for both the workflow itself and each of the activities it executed.

As far as *provenance retrieval and reasoning* is concerned, two facilities are provided. First, a browsing interface is offered so that users can navigate provenance traces. Second, a provenance validation capability is provided in order to decide if the results logged in a workflow trace are still up to date; such a capability is for instance beneficial to users who have run a workflow, but would like to decide if they need to re-enact the workflow because some of its services now produce different results. Both the browsing interface and the provenance validation capability make use of a common Provenance Querying Interface.

From a practical viewpoint, although a provenance service may result in the redundant storage of data, it does not demand any alterations to the services invoked by workflows; only those components which invoke these services require modification — namely the workflow enactment engine.

Provenance Recording Port The recording port provides the means to record a provenance trace in the provenance service. A set of key operations are provided, which we describe below. The interface was designed so as to support asynchronous and transactional submission of provenance data. Asynchronicity is required so that the performance of Web Services intensive workflow application is not penalised by frequent submission of data. Transactional capabilities provide for simple cancellation of trace recording.

The `beginTrace` operation is called by the Provenance Client Interface (PCI) when a new workflow is started. It requires the workflow script, its description, and workflow inputs as arguments. These are used to create a new trace in the provenance database, for which a unique workflow Trace ID is returned. This ID must be used by the PCI in subsequent activity registrations.

The `registerWSActivity` operation is called by the PCI whenever a Web Service invocation is about to be made. It requires the trace ID (returned by the previous method), the workflow activity name corresponding to this Web Service invocation, the URL of the WSDL interface specifying the Web Service, the operation that is going to be called and further information needed to invoke the Web Service (the service namespace, the service name, the port type namespace and the port type name). A unique activity ID is returned which is used later to register the activity inputs and outputs.

The `registerSubWorklowActivity` operation is called by the PCI whenever a sub-workflow activity is executed. The trace ID and activity name are passed as arguments along with the sub-Workflow Trace ID. A unique activity ID is returned.

The `registerWSInput` operation is called by the PCI just before a Web Service invocation is made. It takes as argument the activity ID (allocated when the Web Service is registered) the message parts and the operation increment. All message parts are supposed to be encoded in XML, like they would be for the message invocation, which allows the provenance service to be able to store any complex data type.

The `registerWSOutput` operation is called by the PCI just after the results of the Web Service invocation have been received. The arguments are the same as the `registerWSInput` operation.

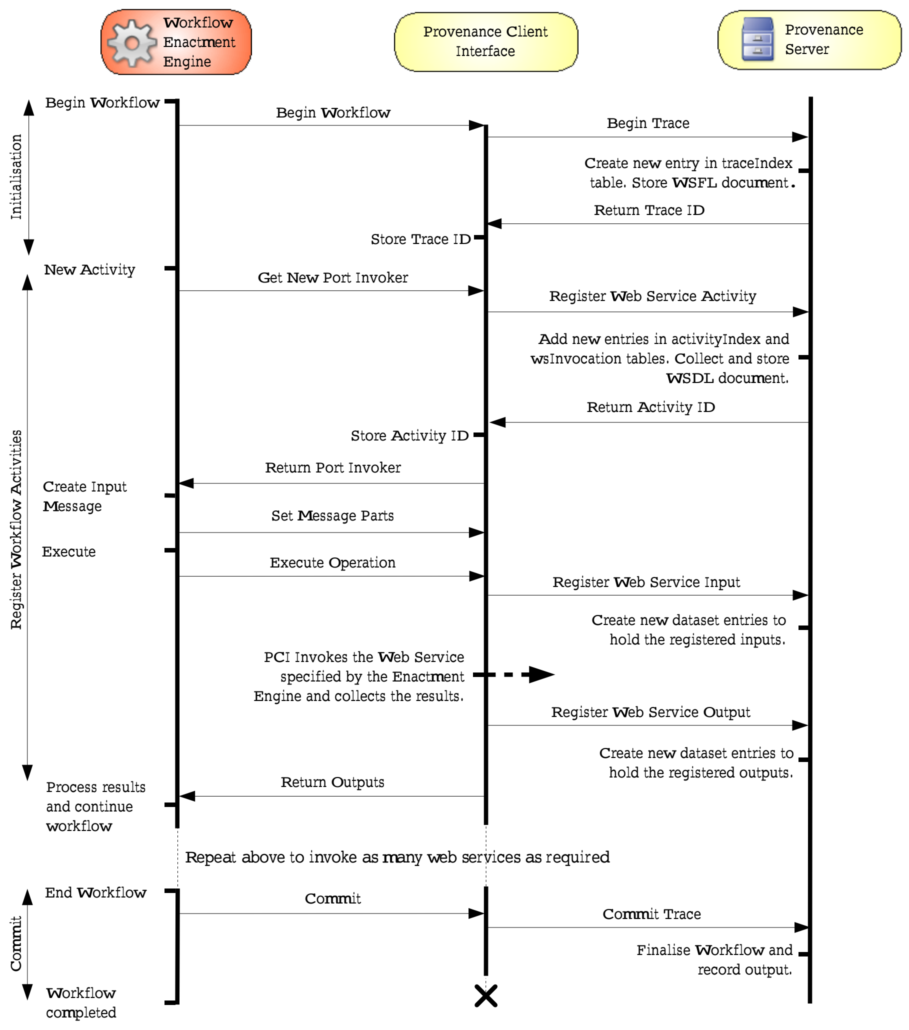
The `commitTrace` operation is called by the PCI when the workflow enactment has been completed. Once this operation is called, the workflow status is changed from 'active' to 'committed'. It takes the workflow outputs in XML format in the arguments along with the trace ID for the trace which is to be committed.

The `abandonTrace` operation may be called when a workflow status is 'active' after which it will be changed to 'abandoned'. Once this operation has been called, the trace is abandoned and no further logging can take place.

The interaction between the Provenance Server and the Provenance Client Interface during workflow execution is best shown using the sequence diagram in Figure 3. The interaction can be divided into three major phases: the first initialises the provenance recording, the second registers workflow activities and the third commits the workflow trace. From the perspective of the enactment engine, the interaction is simple. When a new workflow is started, the 'Begin Trace' method is called on the Provenance Client Interface. For every unique Web Service invocation, a new client-side stub is requested from the PCI by calling the 'Get New Port Invoker' method, passing the WSDL document location and desired operation as arguments. The input message can then be created in the usual way with the message parts being set using the 'Set Message Parts' method on the client-side stub. Consequently, the operation is executed using the 'Execute Operation' method and the result message is returned. On completion of the workflow, the enactment engine must notify the PCI that the provenance trace has to be committed by calling the 'Commit' method.

The Provenance Client Interface interacts with the Provenance Server by making Web Service invocations on the Provenance Recording Port. When the workflow is started, it calls the 'Begin Trace' operation and stores the unique trace ID allocated. When a new client-side stub is created, the PCI registers the new activity with the Provenance Server by using the 'Register WS Activity' operation, storing the unique activity ID alloted. When asked by the enactment engine to invoke a Web Service, the PCI first registers the inputs with the Provenance Server using the 'Register WS Input' operation. It then invokes the desired Web Service and registers the results using the 'Register WS Output' operation before passing them back to the enactment engine.

Provenance Query Port The Query port supplies a set of operations by which a client can retrieve workflow trace information. The operation `getTraceListXML` can be used to query the workflow repository for a list of trace ID numbers. These



Workflow Enactment Engine
Provenance Client Interface
Provenance Server
Initialisation
Begin Workflow
Begin Workflow
Begin Trace
Create new entry in traceIndex table. Store WSFL document.
Return Trace ID
Store Trace ID
Register Workflow Activities
New Activity
Get New Port Invoker
Register Web Service Activity
Add new entries in activityIndex and wsInvocation tables. Collect and store WSDL document.
Return Activity ID
Store Activity ID
Return Port Invoker
Create Input Message
Set Message Parts
Execute
Execute Operation
Register Web Service Input
Create new dataset entries to hold the registered inputs.
PCI Invokes the Web Service specified by the Enactment Engine and collects the results.
Register Web Service Output
Create new dataset entries to hold the registered outputs.
Process results and continue workflow
Return Outputs
Repeat above to invoke as many web services as required
Commit
End Workflow
Commit
Commit Trace
Finalise Workflow and record output.
Workflow completed

Fig. 3. A Sequence Diagram showing the interaction between the Enactment Engine, the Provenance Client Interface, and the Provenance Server

numbers identify traces that must satisfy some properties specified in the query interface, such as start and end time, etc.

It is possible to retrieve workflow trace information in two ways. The operation `getFullTraceXML` takes a trace ID as input and returns a complete XML representation of the workflow trace. For a complex workflow, this XML document could be very large, so a set of operations are also provided to request parts of a workflow trace. With these operations, it is possible to extract a list of activities composing a trace, details for each activies, including inputs and outputs. To this end, the ProvenanceQuery Port offers a querying API which we use in the deployment of the provenance reasoning components.

4 Implementation

The Provenance Server receives provenance data via the Provenance Recording port and stores it in the Provenance Repository. The repository is a relational database that uses the tables and relationships shown in Figure 4.

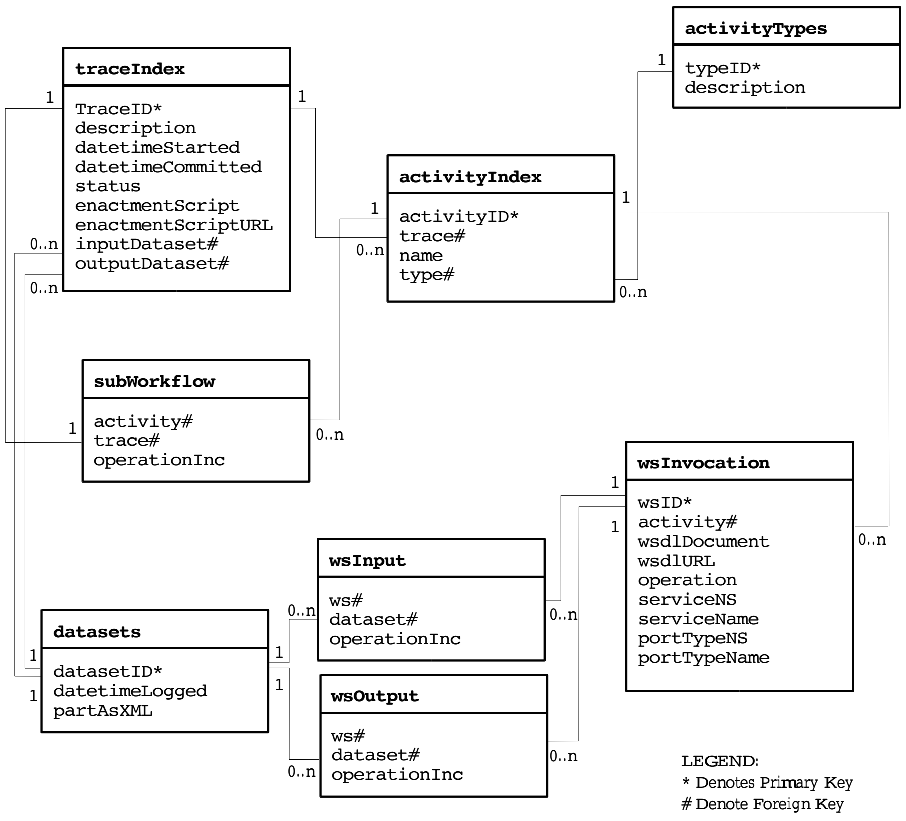


Fig. 4. Provenance Database

It is designed to store all the information necessary to provide provenance data on workflows, activities and all datasets passed between them. There follows a brief overview of the role of each table. The `traceIndex` table holds the workflow trace information for every workflow enacted. The trace ID is automatically allocated by the database incrementally and provides a unique handle to each workflow trace. The inputDataset and outputDataset fields are references to entries in the datasets table. The status field can take one of three values - active, committed or abandoned. The `activityIndex` table links activities executed to their given workflow by the trace field. The type field references an entry in the activityTypes table to specify the activity type. The `activityTypes` table holds descriptions of the possible activity types. In this implementation, there is only two types: Web Service invocations and sub-workflows. It would be feasible to add extra activity types such as JMS or RMI providing extra tables were created to hold the relevant provenance data. The `wsInvocation` table is used to store details on activities that invoke Web Services such as the WSDL document and operation name. The `wsInput` and `wsOutput` tables are used to link a Web Service invocation inputs and outputs to the corresponding entries in the datasets table. The purpose of operation increments (field operationInc) is to accommodate activities where multiple calls are made. The `subWorkflow` table is used to link activities which call sub-workflows to their respective entries in the traceIndex table. Again, an operation increment is recorded to allow multiple sub-workflow calls. The `dataset` table holds any datasets recorded. These might be datasets passed to and from Web Services or to workflow themselves. The partAsXML field stores the dataset in a serialised XML format.

As a workflow is executed and the Provenance Client Interface updates the Provenance Server with invocation details, the database is filled with provenance data. With this data in place, we are able to generate workflow traces in an XML format by querying the database using SQL statements.

An example workflow trace, as displayed by the trace browsing interface, can be seen in Figure 5. On the left-hand side, we see a pretty-printed hierarchical view of a trace with its inputs, outputs and invoked activities; on the right hand-side, the XML representation is shown, with explicit representation of data types.

4.1 Client Side Dataset Encoding

A challenging task was finding an appropriate way to submit the recorded datasets in a structured manner. In simple cases, the inputs and outputs to a Web Service are primitive data types such as integers and strings. However, WSDL also provides the ability to define and use complex types, which are formed of primitive or other complex types. As far as the client side library was concerned, we needed a mechanism to convert arbitrary data types into a serialized format that enabled easy re-creation and re-use. To this end, we adopted Java Record Object Model (JROM) from IBM's Alphaworks. This provides methods to serialize Java Objects used in Web Service invocations into XML format.

A typical Grid or Web Service client relies on a communication layer, such as WSIF or JAX-RPC, to invoke Web Services using using their respective WSDL documents. We have provided a wrapper class, making use of WSIF for invocation of Web Services, and at the same time implementing the PCI for submitting provenance data.

Using JROM to construct the XML representations of Java objects also has an added advantage: The JROM API also provides methods to convert XML representation back into JROMValues. This allows the datasets recorded to be easily re-used for other tasks such as provenance validation.

4.2 Logging Mechanism

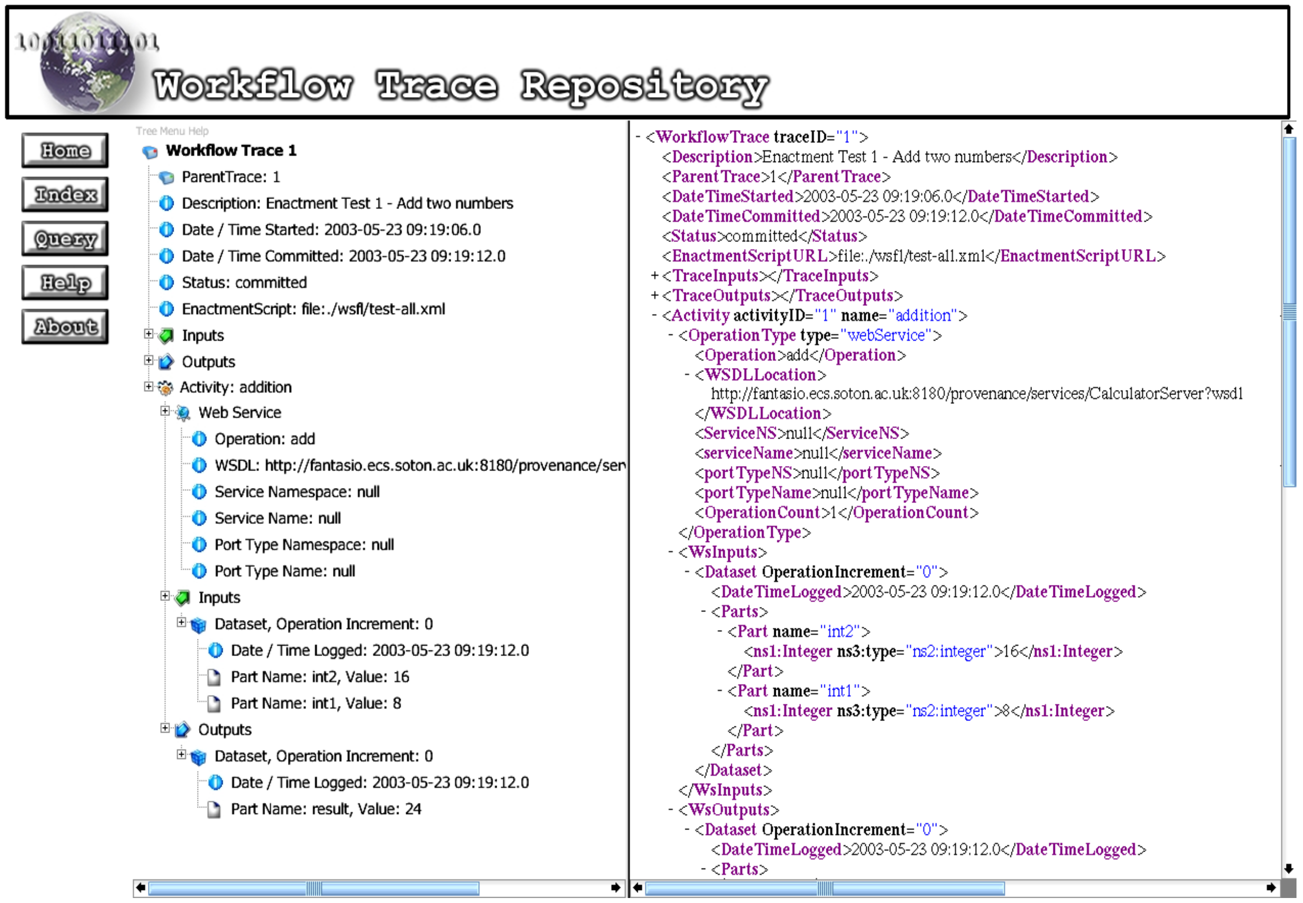
There are two possible ways in which the Provenance Client Interface can submit provenance trace to the the Provenance Server. *(i)* In a *synchronous* submission, the Provenance Client blocks workflow enactment, while updating the Provenance Server. This is the submission mode that underlies the sequence diagram of Figure 3. *(ii)* Alternatively, using an *asynchronous* submission mode, the enactment can continue in parallel with submission of data to the Provenance Server, which takes place in another execution thread.

Synchronous recording can have significant detrimental effect on the performance of the enactment engine, but it provides timeliness as the provenance information is recorded as enactment proceeds, and therefore can be seen as a detailed progress log. On the other hand, with the asynchronous mode, an extremely lazy manner of submitting provenance data would be only to schedule the transfer of such provenance data at enactment time. The provenance data to be submitted could be stored temporarily in a local database; when network traffic is reduced, the data could then be transferred, possibly well after completion of the workflow. Such an asynchronous mode of submission is particular appealing for large data sets.

4.3 Retrieving and Reasoning over Data Provenance

As an experiment into the uses of Data Provenance, we designed and implemented two additional system components; a set of web pages to provide a browse-able workflow trace interface and a provenance validation mechanism which checks each of the Web Service invocations still provides the same results. A screenshot of the web page interface is supplied in Figure 5, it shows the workflow trace for a simple workflow with only one activity that invokes a Web Service to add two integers together. The left frame shows the workflow trace in a familiar tree-view format, while the right frame shows the raw XML for the trace. It is possible to use this interface to browse workflows that are still in progress and therefore it also provides a way to monitor long and complex workflows.

Provenance Validation allows a user to check that each Web Service invocation made during a workflow still produces the same results. It iteratively re-executes each Web Service invocation made, using precisely the same inputs as those recorded. The output produced by the re-invocation is compared with the output stored in the provenance trace. A difference in the result is notified



```
Workflow Trace 1
  ParentTrace: 1
  Description: Enactment Test 1 - Add two numbers
  Date / Time Started: 2003-05-23 09:19:06.0
  Date / Time Committed: 2003-05-23 09:19:12.0
  Status: committed
  EnactmentScript: file:./wsfl/test-all.xml
⊞ Inputs
⊞ Outputs
⊞ Activity: addition
  ⊞ Web Service
      Operation: add
      WSDL: http://fantasio.ecs.soton.ac.uk:8180/provenance/ser
      Service Namespace: null
      Service Name: null
      Port Type Namespace: null
      Port Type Name: null
  ⊞ Inputs
    ⊞ Dataset, Operation Increment: 0
        Date / Time Logged: 2003-05-23 09:19:12.0
        Part Name: int2, Value: 16
        Part Name: int1, Value: 8
  ⊞ Outputs
    ⊞ Dataset, Operation Increment: 0
        Date / Time Logged: 2003-05-23 09:19:12.0
        Part Name: result, Value: 24
```

```
- <WorkflowTrace traceID="1">
    <Description>Enactment Test 1 - Add two numbers</Description>
    <ParentTrace>1</ParentTrace>
    <DateTimeStarted>2003-05-23 09:19:06.0</DateTimeStarted>
    <DateTimeCommitted>2003-05-23 09:19:12.0</DateTimeCommitted>
    <Status>committed</Status>
    <EnactmentScriptURL>file:./wsfl/test-all.xml</EnactmentScriptURL>
  + <TraceInputs></TraceInputs>
  + <TraceOutputs></TraceOutputs>
  - <Activity activityID="1" name="addition">
    - <OperationType type="webService">
        <Operation>add</Operation>
      - <WSDLLocation>
          http://fantasio.ecs.soton.ac.uk:8180/provenance/services/CalculatorServer?wsdl
        </WSDLLocation>
        <ServiceNS>null</ServiceNS>
        <serviceName>null</serviceName>
        <portTypeNS>null</portTypeNS>
        <portTypeName>null</portTypeName>
        <OperationCount>1</OperationCount>
      </OperationType>
    - <WsInputs>
      - <Dataset OperationIncrement="0">
          <DateTimeLogged>2003-05-23 09:19:12.0</DateTimeLogged>
        - <Parts>
          - <Part name="int2">
              <ns1:Integer ns3:type="ns2:integer">16</ns1:Integer>
            </Part>
          - <Part name="int1">
              <ns1:Integer ns3:type="ns2:integer">8</ns1:Integer>
            </Part>
          </Parts>
        </Dataset>
      </WsInputs>
    - <WsOutputs>
      - <Dataset OperationIncrement="0">
          <DateTimeLogged>2003-05-23 09:19:12.0</DateTimeLogged>
        - <Parts>
```

Fig. 5. A screen shot of the web interface showing a tree-view of a workflow trace

to the user. By re-using the existing Provenance Client Interface, we are able to record provenance on this re-execution and hence compare it with the original.

A typical example of the use of this facility is the following. A user having been informed that new data are available in a database (accessible as a Web Service), or that a new version of a service has been deployed, may decide to use the provenance validation facility to verify if the results produced by a previous execution of a workflow are still up to date. The workflow activities that produce results that differ from those stored in the provenance database indicate points in the workflow at which re-enactment should occur.

It is important to distinguish provenance validation from workflow re-enactment. The former only uses the information recorded in a provenance trace to detect which Web Service produces outputs that do not match the ones in the trace for the same inputs. Web Services can be invoked in any order, and this activity does not require knowledge of the workflow script and service dependencies. On the contrary, workflow re-enactment requires the understanding of a workflow script: it typically requires the enactment engine to be provided with a runtime state (i.e. its continuation) to resume the execution at a specific point of the workflow.

5 Discussion and Conclusion

In this section, we study how the proposed architecture meets the requirements identified in Section 2; we then summarise our contributions, and discuss how our work can be extended.

The principal requirement for a provenance recording facility is that data lineage must be recorded and stored. We adopted the position that a sound lineage record can be achieved by logging enough information to enable the re-creation any dataset transformation. For a service-oriented architecture, in which Web Service invocations are used to perform these transformations, this means collecting all the invocation details. Our Provenance Client Interface does this autonomously, supplying the user with the option to turn provenance recording on or off as desired. The data model put forward in Section 4 provides a means to store this provenance, accommodating complex workflow activities and arbitrary data types. With the client-server interaction described in Section 3, we are able to reliably record and store all Web Service invocations carried out by a client and hence describe all dataset transformations. We also designed our system so that it could be integrated with any existing components simply and unobtrusively.

The second major requirement is that the data provenance is made available through a query interface. This is provided through the Provenance Query Port; an API that allows a user to extract data provenance according to content and type via Web Services. The provenance records are made available in the form of a workflow trace, an XML formated data structure that shows precise details of all aspects of a complex data processing task such as workflow enactment. Although our workflow trace definition is by no means an attempt at a standard lineage representation, it does uphold the requirement that it allows data provenance to be communicated reliably and consistently between platforms.

The greatest testament that our architecture meets the requirements is demonstrated through the Provenance Validation component. By enabling the validation of any dataset transformation, we show that the records have been recorded correctly, stored in a reliable format that facilitates re-use, and exposed via a user friendly query interface.

In this paper, we have defined a notion of provenance for service-oriented architectures such as Web Services and the Open Grid Services Architecture. Based on this definition, we have proposed a provenance architecture, composed of two key components: *(i)* a provenance server capable of recording provenance information and offering a querying interface over the provenance information it stores; *(ii)* a client-side API that allows clients invoking Web Services to submit provenance records to the provenance server. We demonstrated the use of provenance data by providing two further components making use of the querying interface: *(i)* A browsing interface allows users to navigate provenance records as they are submitted; *(ii)* a provenance validation capability is able to decide if the results produced by previous workflow enactments are still up to date by re-executing all invoked services and comparing their outputs with the ones stored in the provenance trace.

Provenance is a rather new topic in service-oriented architectures, and a number of issues still need to be addressed. In our implementation, the provenance service is acting as a single point of contact, and therefore may become a bottleneck and a single point of failure. Provenance information need not be stored at a single location, and could be distributed. It makes however life easier for the implementor of the querying of interface if all provenance records are stored at a single location.

In Section 3, we motivated the existence of a provenance service by the possibility of auditing provenance information by third parties. The architecture includes a Provenance Client Interface, which allows a client, such as a workflow enactment engine, to submit provenance records. In such a context, provenance can only be trusted if the provenance service and the enactment engine are trusted. It is desirable to reduce the trust assumptions so that the provenance service can be accepted as properly auditable by third parties. In particular, not having to trust the enactment engine to conformly execute a workflow will make the system more attractive. To this end, we could request from the invoked services to contribute to the provenance record submission process, in order to certify the records submitted by the enactment engine. There is here an interesting similarity with Mobile Agents having to run on non-trusted platforms [20]: provenance protocols could be inspired by protocols designed to verify that the results returned by a mobile agent have not been corrupted by the environments in which it operated. Distributed systems properties are seeked such as mutual authentication of all parties involved in the computation, and non-repudiation, by which we can retain evidence of the fact that a service has committed to executing a particular invocation and has produced a given result.

Section 4.2 evoked the problem of large data sets, for which the transfer of data could be performed asynchronously. In some cases, it may not be desirable to duplicate the data either because it is too large, or because of copyright or intellectual property reasons. The data owner would have to commit to archive such data, and a unforgeable digest could be submitted instead. The protocol should be extended to accomodate such cases, and incorporate data owners in a network of trust.

We propose a provenance validation component capable of re-invoking Web Services and compare their results with the results stored in the provenance server. Currently, our comparison relies on a strict syntactic equality check, but this has some restrictions. Some services can be based on stochastic processes (e.g. Monte-Carlo simulation), and therefore generated results may not be syntactically equal, though they could be considered equivalent. To be generic, a provenance architecture must be able to support domain specific comparisons, so that from a domain's viewpoint, one can decide whether two results are similar.

6 Acknowledgments

Thanks to colleagues for discussions on provenance including Syd Chapman, Omer Rana, Carole Goble, Matthew Addis and Mark Greenwood. This research is funded in part by EPSRC myGrid project (reference GR/R67743/01).

References

[1] Peter Buneman, Alin Deutsch, and Wang-Chiew Tan. A deterministic model for semistructured data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1998.

[2] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory (ICDT)*, 2001.

[3] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Computing provenance and annotations for views, October 2002. Published at [18].

[4] Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, and Sanjiva Weerawarana. Business process execution language for web services (bpel4ws). http://www.ibm.com/developerworks/library/ws-bpel/, 2002.

[5] David de Roure, Nicholas R. Jennings, and Nigel Shadbolt. The semantic grid: A future e-science infrastructure. *International Journal of Concurrency and Computation: Practice and Experience*, 2003.

[6] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.

[7] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid — An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Argonne National Laboratory, 2002.

[8] Ian Foster, Carl Kesselman, and Steve Tuecke. The Anatomy of the Grid. Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 2001.

[9] Ian Foster, Jens Vockler, Michael Wilde, and Yong Zhao. The virtual data grid: A new model and architecture for data-intensive collaboration, October 2002. Published at [18].

[10] James Frew and Rajendra Bose. Lineage issues for scientific data and information, October 2002. Published at [18].

[11] Grid computing environments working group at the global grid forum. http://www.computingportals.org/, November 2002.

[12] Carole Goble. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics, October 2002. Published at [18].

[13] Mark Greenwood, Carole Goble, Robert Stevens, Jun Zhao, Matthew Addis, Darren Marvin, Luc Moreau, and Tom Oinn. Provenance of e-science experiments - experience from bioinformatics. In *Proceedings of the UK OST e-Science second All Hands Meeting 2003 (AHM'03)*, 4 pages, Nottingham, UK, September 2003.

[14] Frank Leyman. Web Services Flow Language (WSFL). Technical report, IBM, May 2001.

[15] Michael Luck, Peter McBurney, and Chris Preist. *Agent Technolgy: Enabling Next Generation Computing*. AgentLink, 2003.

[16] Luc Moreau, Simon Miles, Carole Goble, Mark Greenwood, Vijay Dialani, Matthew Addis, Nedim Alpdemir, Rich Cawley, David De Roure, Justin Ferris, Rob Gaizauskas, Kevin Glover, Chris Greenhalgh, Peter Li, Xiaojian Liu, Phillip Lord, Michael Luck, Darren Marvin, Tom Oinn, Norman Paton, Stephen Pettifer, Milena V Radenkovic, Angus Roberts, Alan Robinson, Tom Rodden, Martin Senger, Nick Sharman, Robert Stevens, Brian Warboys, Anil Wipat, and Chris Wroe. On the Use of Agents in a BioInformatics Grid. In Sangsan Lee, Satoshi Sekguchi, Satoshi Matsuoka, and Mitsuhisa Sato, editors, *Proceedings of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing*, pages 653–661, Tokyo, Japan, May 2003.

[17] Dave Pearson. Data requirements for the grid - scoping study report, February 2002. Status Draft.

[18] Data provenance/derivation workshop.
http://people.cs.uchicago.edu/ yongzh/position_papers.html, October 2002.

[19] Joel Saltz. Data provenance, October 2002. Published at [18].

[20] Hock Kim Tan and Luc Moreau. Extending Execution Tracing for Mobile Code Security. In Klaus Fischer and Dieter Hutter, editors, *Second International Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002)*, DFKI Research Report, RR-02-03, pages 51–59, Bologna, Italy, June 2002. DFKI Saarbrucken.

[21] Satish Thatte. Xlang, web services for business process design, 2001.