GRID KNOWLEDGE DISCOVERY PROCESSES AND AN ARCHITECTURE FOR THEIR COMPOSITION

G. Kickinger, J. Hofer, P. Brezany
Institute for Software Science
University of Vienna
Liechtensteinstrasse 22
A-1090 Vienna, Austria
email:{kickinger,hofer,brezany}@par.univie.ac.at

A. M. Tjoa
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstrasse 9-11/E188
A-1040 Vienna, Austria
email:tjoa@ifs.tuwien.ac.at

ABSTRACT
The Grid is the computing and data management infrastructure, which is transforming science, business, health and society. This paper deals with a challenging task addressing knowledge discovery in data repositories within the Grid. It gives an overview about the architecture of a novel Grid knowledge discovery system, the collaboration of the designed services and the concepts of controlling the whole knowledge discovery process by the deployment of a workflow component.

KEY WORDS
Grid, OGSA, Data Mining, Workflow, Job Control

1 Introduction

Grid computing has been identified as an important new technology by a remarkable breadth of scientific and engineering fields as by many commercial and industrial enterprises. This paper describes our research effort, which aims to extend the state-of-art Grid technology to a completely new and societally important category of applications. It will develop and thoroughly evaluate the novel concepts of knowledge discovery in databases and other large data sets attached to the Grid. We focus our effort on data mining and On-Line Analytical Processing (OLAP), two complementary technologies, which, if applied in conjunction, can provide a highly efficient and powerful data analysis and knowledge discovery solution on the Grid. These two technologies are being investigated and experimentally implemented within a novel infrastructure called GridMiner [1]. In this article, we describe the structure and components of the knowledge discovery processes and their mapping onto appropriate Grid services. Further, we address the challenge of integrating these services and describing the various interaction between them and how to dynamically compose new services out of existing ones. Our approach is based on the Open Grid Service Architecture (OGSA; to our best knowledge, GridMiner is the the first prototype of a Grid knowledge discovery system developed on top of OGSA.

The topics presented here are only a small and general excerpt of our work. A more detailed description can be found in [7].

2 The Knowledge Discovery Process

Knowledge discovery in databases (KDD) can be defined as *the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* [3]. Often the terms *data mining* and KDD are used interchangeably. But principally, as described below, data mining is only one step within the whole KDD process (see Figure 1), which concerns with applying appropriate algorithms to extract knowledge from a prepared dataset.

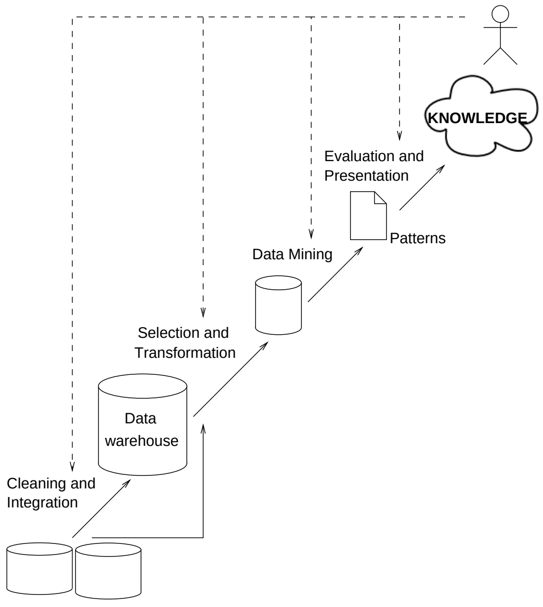

Figure 1. The KDD Process [5]

Cleaning and Integration. In most organizations nearly every department has its own so called *operational database*. These isolated applications are customized to the individual needs of each department. In virtual organizations (VOs) [4], which are enabled by grids, the situation is often similar. But even if there exists one common dataset, it is possible that data is distributed over many locations. Data in operational databases are typically not "clean". This means that those databases contain errors, due to wrong inputs from users or application failures. Besides, this data may be incomplete, and essential information may not be available for some attributes. Hence, for data mining and other analysis tasks, it is necessary to "clean" data and to integrate the data into one common data set.

Selection and Transformation. The next step of KDD will be to choose an appropriate subset of the integrated data and perform some necessary transformations. Many data mining algorithms work only on special transformed data. One example is a neural network algorithm which needs the input in a distinct interval like *[0,1]*.

Data Mining. This step describes the application and parameterization of a concrete data mining algorithm, to search for structures and patterns within the dataset, like a *classification, clustering, association, characterization*, or *comparison* algorithm.

Evaluation and Presentation. The last step will be to evaluate the results of data mining with distinct measures. Data mining algorithms deliver a set of patterns. Often these patterns are not interesting to the user. Hence, measures like support and confidence discriminate interesting from not interesting patterns. Of course the results of data mining have to be prepared in some appropriate way to present them to a user. Various presentation forms exist, like tables, charts or more sophisticated presentation mechanisms.

3 Architecture of the GridMiner System

3.1 GridMiner Services

GridMiner [1] is a set of OGSA conformous services with well-defined interfaces, that are used as needed to assemble applications on top of them. These services cover the steps of the KDD process horizontally as explained in Section 2 and are supported or used by services below and above them - see Figure 2.

GridMiner Mediation Service (GMMS). Data Mediation is an approach to simplify the work with multiple, federated, usually geographically distributed data sources. Our service creates a single virtual data source providing the same client interface as classical grid data sources but integrating data from multiple federated data sources. It currently supports the `join` operation for vertical distribution and the `union` operation for horizontal distribution.

GridMiner Information Service (GMIS). Information Services are a vital service within every grid infrastructure providing static and dynamic information on available grid resources. The GMIS is a specialized implementation enabling GridMiner specific decision making and monitoring.

GridMiner Resource Broker (GMRB). Resource Broker are used for match-making of requests and available resources. Within GridMiner the Resource Broker might be used to to find best-fitting resources, e.g. a cluster with efficient network access to the dataset.

GridMiner Cube Management Service (GMCMS). This service creates distributed OLAP cubes from a specified input data source that might even not be persistent. After initial creation GMCMS can be used for interacting and lifecycle management of the cube.

GridMiner Data Integration Service (GMDIS). Data integration refers to transferring data subsets from multiple data sources into usually one other data source that is either more efficiently accessible or more analysis dedicated. GMDIS is responsible for secure, reliable, efficient management and operation of the necessary data transfers within grid environments.

GridMiner Preprocessing Service (GMPPS). We subsummarize under the term preprocessing activities besides data integration that are performed before the data mining step like data cleaning, normalization, selection, reduction etc. The Preprocessing Service is a service that can easily be extended with additional algorithms.

GridMiner DataMining Service (GMDMS). This service is one of the major core components of the GridMiner system. It is an extensible framework providing facilities all data mining algorithms usually need and data mining application developer can easily plug their algorithms and tools in. In our prototype we ship a centralized and a distributed technique that creates classification decision trees.

GridMiner OLAP Mining Service (GMOMS). It provides algorithms to perform data mining on cubes. This is a service which works very close together with an OLAP engine sending for example 'drill up' commands to work with the dataset at different abstraction levels.

GridMiner Orchestration Service (GMOrchS). This service allows the execution of simple, recurring data mining tasks as a static orchestration of GridMiner services. So it provides the ability to use static workflows. Hence it is used for often used routine data mining jobs.

GridMiner Dynamic Service Composition Engine (GMDSCE). This service provides the execution of complex, highly dynamic workflows for one distinct knowledge discovery task as a composition of different grid services. The workflow itself is described by a specially designed XML language extension.

3.2 Usage Modes

As already described in earlier sections, a KDD process is composed from individual steps, including various preprocessing activities (e.g. integrating various data sources, data cleaning, de-normalizing complex data structures to flat tables), the central data mining step that searches the dataset for structures and patterns and finally the representation of the knowledge found in the preceding data mining step. In the GridMiner project, three important different system us-

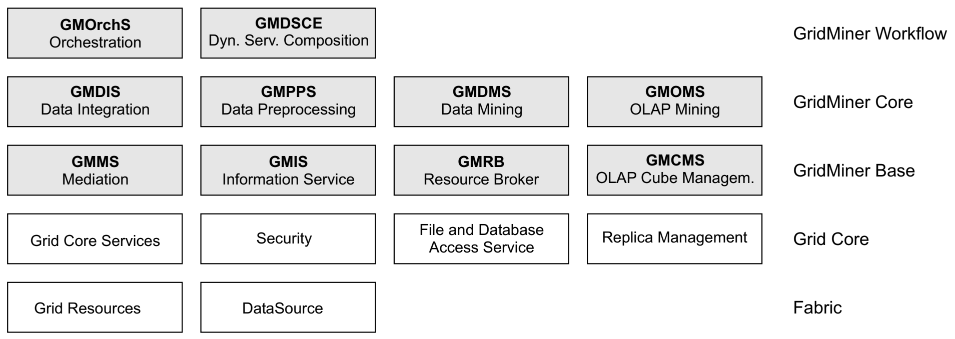

Figure 2. GridMiner components

age modes - approaches to control the system - have been identified, as described in the following paragraphs.

Interactive Mode. Humans using a data mining system often have no a-priori idea of what to search for. They are given a database holding records out of a certain domain and want to raise the value of these data by extracting valuable knowledge. User behavior sometimes can be described as "try and look what comes out". Though this is not a very analytical and scientific way to solve problems, it is a quite common way. What we really need to establish GridMiner as an application is a high dynamic intelligent interface fulfilling the requirements of workflow and interactive processing. In such cases especially the data mining step is an iterative process. Different algorithms with different parameterizations are used multiple times, perhaps with different taxonomies as background knowledge. If too many results are produced stronger restrictions on the interestingness measures (e.g. confidence, support) or additional constraints on the source dataset (e.g. additional `where`-conditions) to reduce the number of examples to a smaller domain have to be passed. We refer to this usage mode as *interactive mode*.

If data mining systems are used interactively, the user interface plays an important role. It has to ease instruction and control of the components involved and present the results in a user-friendly way. The user interface should hide the complexity of the underlying control commands and languages, perhaps by providing intuitive (graphical) editors for creating and using the services.

Batch Mode. In our opinion the second usage mode in grid environments is as important as the first one is. The so called *batch mode* allows to specify all details of the complete or parts of the knowledge discovery process in advance in terms of an appropriate workflow description language.

The characteristic of an a-priori full job description has several advantages over the interactive approach. In grid environments, where the users have to deal with wide-area resource distribution and very large multidimensional databases, jobs are are typically long-running. Once the user requirements have been expressed by a workflow language and submitted to the workflow engine, the user is free to do anything else, while his job is reliably executed. Other advantages are that participating resources can be initialized, reserved, Quality-of-Service requirements verified, and performance predictions calculated. Further, an execution plan and schedule can be generated and optimized to improve performance.

Hybrid Mode. The previous approaches may be useful in cases, where the users either have detailed a-priori knowledge of the complete workflow or want to use the system fully interactively.

If the users want to use the system interactively, but this is not feasible due to the huge amount of data to process, as often found in grid environments, a combination of the previous approaches has to be available. So GridMiner must allow both: e.g. to specify long-running preprocessing jobs (perhaps multiple integration, cleaning and transformation processes) in terms of a workflow language in advance and afterwards to analyze the resulting dataset interactively.

As a conclusion, we state that there is an additional need to a hybrid approach that combines the advantages of the interactive and batch mode by allowing to execute parts of the complete knowledge discovery process automatically, or directly by the user.

4 Collaboration of the GridMiner Services

The previous section showed the base services provided by GridMiner. Each of these services is able to perform one step within the knowledge discovery process. Now we investigate how the different services work together to support the whole knowledge discovery process and not only one part.

The services themselves do not communicate with each other. No service is aware of other existing services. Hence each of the services is able to run completely independently. To support the steps of KDD process the output of the first service serves as input for the second service. The independence of the various services allows also to execute them in parallel and not only in sequence, which results in an improvement of performance.

4.1 Data Mining on top of "flat tables"

The most robust data mining algorithms require propositional (attributional) representation of data in the form of a single table ("flat table"). However, some algorithms need additional knowledge. Taxonomies are one example for additional knowledge. A taxonomy describes additional conceptual hierarchies of a dimension relevant to a data mining algorithm. This additional knowledge is provided either by a *knowledge base* or directly as a parameter of a GMDMS operation call.

Figure 3 shows an example how various GridMiner services can be connected to perform one KDD process. The services do not interact with each other. The only existing connection are the different data sources. As illustrated in this figure, the type of the data sources can be different.

The target data source of one service serves as input data source to the next service. In this scenario, a client wants to perform e.g. clustering on a specific data set. The data has some inconsistency, so first of all it will be necessary to clean the data (*GMPPS instance 0*). The clustering algorithm needs the values in a distinct range (maybe from 0 to 1), thus the client starts a service which can transform this attributes (*GMPPS instance 1*) to the desired values. GMDMS clusters the preprocessed data. GMPRS prepares the results for presentation, for example it creates a 3D plot, where each cluster is represented by a sphere.

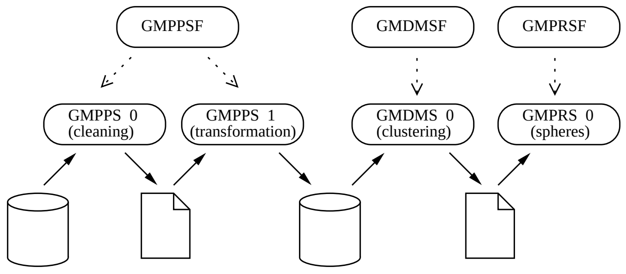

Figure 3. Example of one KDD process with GridMiner

The example illustrated above represents only one, simple scenario. The sequence of the service calls can be far more complex. It is very likely that distinct services are executed in parallel. One example are sampling methods before integration of data. Those sampling methods draw random samples from huge data amounts. If the user wants to mine on more than one data source it is better to apply sampling before data integration is performed. It is obvious, that those preprocessing services are executed in parallel and not sequentially.

4.2 Data Mining on top of OLAP

The GridMiner functionality also includes *On–Line Analytical Mining* (OLAM). OLAM algorithms are able to perform data mining on cubes.

The knowledge discovery process on cubes is very similar to KDD on "flat tables". Figure 4 shows an example of a workflow, which allows data mining on cubes. Cubes always contain preprocessed data. A cube is constructed for a distinct task. Therefore it is necessary to start preprocessing services before the cube is constructed. The cube management service (GMCMS) creates an OLAP cube. During this operation, it reads the preprocessed data and the knowledge base for creating concept hierarchies to structure the dimensions of the cube. After the cube is constructed a data mining algorithm can be started. That is the task of the OLAM service (GMOMS).

The results of the OLAM step are serving as input for a presentation service, which prepares the results for presentation.

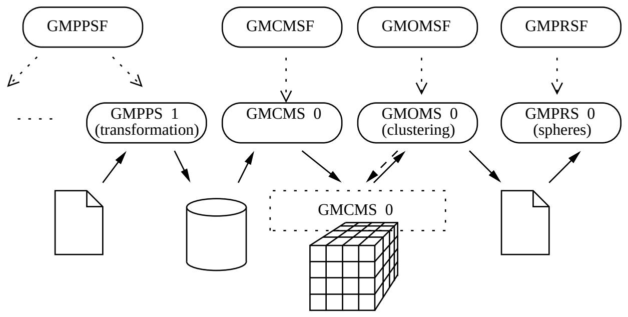

Figure 4. Example of the whole OLAM process

5 Workflow Engine

In the previous sections we investigated how the underlying service can be deployed to cover the whole knowledge discovery process, both on top of flat tables and on top of OLAP. The execution of this knowledge discovery process can take from a few minutes to several hours or even longer. If a client application, located for example on a user's workstation, controls the job and the job would be terminated, following tasks would never be executed. The results of the already executed services would have been lost and the job would be aborted. KDD can include several concurrent activities, e.g. several classification models can be created in parallel and then sequentially evaluated. If a client implements all this functionality, it will become very complex and very "thick". Thus we need a service which is able to handle those long running, complex jobs and executes them on behalf of an execution plan.

Our goal is to develop a *workflow engine* which *instantiates*, *executes* and *coordinates* a set of *OGSA Grid services* according to a predefined *execution plan*.

Two different approaches for implementing a workflow engine can be identified (Figure 5). These approaches and their differences are discussed in more detail in the following sections.

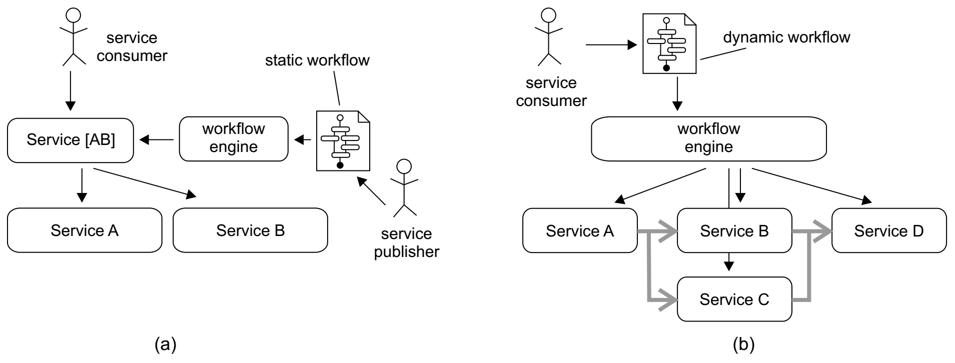

Figure 5. A static workflow specified by the service provider that is executed each time in the same way is shown in (a). Contrarily a service consumer specified workflow with a more complex scenario is shown in (b).

5.1 Static Service Composition by Service Publisher

Figure 5 (a) gives a schematic overview of *static service composition*. Service A and Service B are two already existing services. Now a service publisher recognizes that it would be useful to combine these two services and to provide an orchestrated service (Service [AB]). He does so by writing a workflow file, which describes the interaction of the two services involved. Additionally he creates a new WSDL file, which describes the ports and operations of the orchestrated service. Now he installs this service within a workflow engine. The result of this process is a new service, which can be used by the client like a common Web service. Workflow languages for Web Services like BPEL4WS [10] are using this strategy.

5.2 Dynamic Service Composition by Service Consumer

GridMiner requires dynamic workflows, which are not reusable in the sense that a workflow is pre-described by a workflow description language once and installed to a workflow engine. Since every knowledge discovery process requires a unique specific workflow, the workflow engine does not “construct” an own orchestrated service.

There is also another reason why the composition to a new service is not useful. The interface for accessing the workflow service must be static. If a service publisher orchestrates a new service, the interfaces and operations of the new orchestrated service can be different from those of an old one. A client written to use the old service possible cannot operate on a new composed service or the implementation of a client would result in enormous complexity, because it cannot make use of proxy classes since the service interfaces would change permanently. So, whereas the workflows are dynamic the interface to access the service instance has to be static.

Figure 5 (b) gives a schematic overview of *dynamic service composition by a service consumer*. Services A, B, C, and D are already existing services. Now the service consumer specifies in a workflow file in which order he wants to execute the services, i.e. the number of the instances to create from each service, which services can be executed in parallel. In the example depicted in Figure 5 (b), the user specifies that service A has to be executed first, then service B and C have to be executed concurrently, and as the last step service D has to be called. He also specifies the concrete values of the arguments for each of the involved services. Than the service consumer sends this workflow description to a workflow engine, which executes the workflow.

Dynamic Service Composition Engine (DSCE). Since in most publications, a workflow engine is a service, which allows the composition of services to a new orchestrated service, we use the term *Dynamic Service Composition Engine* (DSCE) to make our dynamic, service consumer related approach more expressive. DSCE is developed for GridMiner, but does not necessarily depend on it. Indeed we plan to construct an application–independent service, which is able to execute a dynamic process, consisting of a set of OGSA grid services. So every system consisting of different grid services, which are executed in different orders, and/or make use of parallel processing can make use of DSCE.

DSCE is implemented as a stateful, transient OGSA Grid service. A special XML language, referred as *Dynamic Service Composition Language* (DSCL) serves as input for our engine.

To support both batch and interactive processing, DSCE provides *(a)* independent processing (without any interaction of the user) of a workflow described in DSCL, *(b)* the provision of all intermediate results from the services involved, *(c)* the possibility to change workflow at run time [1], and *(d)* the possibility for a user to stop, cancel

[1]If the results of a “run” did not satisfy the user, he analysis the intermediate results (see also item *(b) above*). If he recognizes, for example, that he applied a wrong algorithm or wrong parameters, he can change

or resume a workflow.

Dynamic Service Composition Language (DSCL). *Dynamic Service Composition Language* (DSCL) is an XML-based document, which is specially designed to support highly dynamic and complex workflows. Unlike common languages like BPEL4WS, which aim to provide a new orchestrated service composed out of other services, DSCL only serves as an input document to control the execution of the described workflow by an engine. A DSCL document is produced directly by a possible client, which tells the engine what it has to do in the next step. DSCL allows to specify the order of the different activities and the parameters used, as outlined in the DSCL code below. An activity can be: the start of a sequence of activities; the start of parallel execution of activities; the creation of a new grid service instance; the invocation of an operation; the querying of service data elements; and the access and modification of operation parameters.

```
<dscl:dscl xmlns:dscl="..." xmlns:xsi ...>
  <dscl:variables>
    <dscl:variable dscl:name="param1">
       <dscl:init>
          <ns1:value xsi:type="xsd:int" ...>
             4711
          </ns1:value>
       </dscl:init>
    </dscl:variable>
  </dscl:variables>
<dscl:composition>
  <dscl:sequence>
    <dscl:createService dscl:factory-name="..."
                        dscl:instance-name="test"/>
    <dscl:invoke dscl:instance-name="test"
                 dscl:portType=""
                 dscl:operation="setIntValue"
                 dscl:inputParam="dscl:param1"/>
    </dscl:sequence>
  </dscl:composition>
</dscl:dscl>
```

6 Related Work

The first proposal of an architecture for performing data mining on the Grid was published in [2]. M. Cannataro and D. Talia, present design of a Knowledge Grid architecture based on the non-OGSA-based version of the Globus Toolkit. BPEL4WS 1.1 (Business Process Execution Language for Web Services) [10] is the actual standard, which can describe compositions of Web Services. This specification represents a convergence between the ideas in XLANG [11] and WSFL [9]. The Grid Services Flow Language [8] wants to do the same for Grid Services. GSFL is based on the so called Web Services Flow Language [9] a predecessor of BPEL4WS published by IBM. Those flow language specifications have all the same target: they are describing a business process built up of various web services. This description then serves as input for a workflow engine like BPWS4J [6] (an engine for BPEL4WS developed by IBM).

these in the workflow description document and start the workflow again. DSCE knows only one service at the end of the sequence has changed, so it is not necessary to execute the previous activities again.

7 Conclusions

In this paper, we briefly analyzed components of Grid knowledge discovery processes and outlined how they are composed into a novel open service architecture called GridMiner. We also presented our approach to the development of a mechanism for a dynamical composition of existing services to a more complex functionality based on the Dynamic Secvice Composition Language and the appropriate Workflow Engine; these concepts have already been implemented. Other systems, which are based on OGSA Grid services and need components for workflow handling or job control can also benefit from this approach.

References

[1] P. Brezany, J. Hofer, A M. Tjoa, and A. Wöhrer. Gridminer: An Infrastructure for Data Mining on Computational Grids, APAC'03 Conference, Gold Coast, Australia, October 2003.

[2] M. Cannataro and D. Talia. Knowledge grid: An architecture for distributed knowledge discovery. Communications of the ACM, January 2003.

[3] Usama M. Fayyad et al. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–43. AAAI Press/ The MIT Press, 1996.

[4] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. Intl. J. Supercomputer Applications, 15(3), 2001.

[5] J. Han and M. Kamber. *Data Mining. Concepts and Techniques*. Morgan Kaufmann, 2001.

[6] IBM. The IBM business process execution language for web services java run time (BPWS4J). http://alphaworks.ibm.com/tech/bpws4j.

[7] G. Kickinger, J. Hofer, and P. Brezany. Anatomy of grid knowledge discovery processes and architecture for their composition, 2003.

[8] S. Krishnan, P. Wagstrom, and G. Laszewski. GSFL: A workflow framework for grid services. www.globus.org/cog/papers/gsfl-paper.pdf, 2002.

[9] F. Leymann. Web services flow language (WSFL 1.0). http://www-4.ibm.com/software/solutions/ webservices/pdf/WSFL.pdf, May 2001.

[10] BEA Systems. Business process execution language for web services. http://www-106.ibm.com/ developerworks/webservices/library/ws-bpel/, May 2003.

[11] S. Thatte. XLANG: Web services for business process design. http://www.gotdotnet.com, 2001.