Workflow Management in GridMiner*

Günter Kickinger[1], Jürgen Hofer[1], Peter Brezany[1], and A Min Tjoa[2]

[1] Institute for Software Science
University of Vienna, Liechtensteinstrasse 22, A-1090 Vienna, Austria
{kickinger,hofer,brezany}@par.univie.ac.at
[2] Institute of Software Technology and Interactive Systems
Vienna University of Technology, Favoritenstrasse 9-11/E188, A-1040 Vienna, Austria
tjoa@ifs.tuwien.ac.at

Abstract
In the past five years 'The Grid' emerged as one of the most important new developments in building the computing and data management infrastructure for science, business, health, and society in the 21st century. This paper deals with the development and organization of the knowledge discovery processes in Grid environments, especially, with the collaboration and coordination of a set of OGSA based Grid services developed within our Grid knowledge discovery system called *GridMiner*. One of the kernel parts of GridMiner is an advanced workflow component, which can be also included into other Grid systems implementing applications having non-trivial workflow requirements.

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-the-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 we can manage the resulting workflow. Our approach is based on the Open Grid Service Architecture (OGSA) [2]. 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 at www.gridminer.org.

*This research is being carried out as part of the research projects "Aurora" and "Modern Data Analysis on Computational Grids" supported by the Austrian Research Foundation.

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 as synonyms. But principally, as described below, data mining is only one step within the whole KDD process (see Fig. 1), which concerns with applying appropriate algorithms to extract knowledge from a prepared dataset.

Cleaning and Integration. 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 dataset.

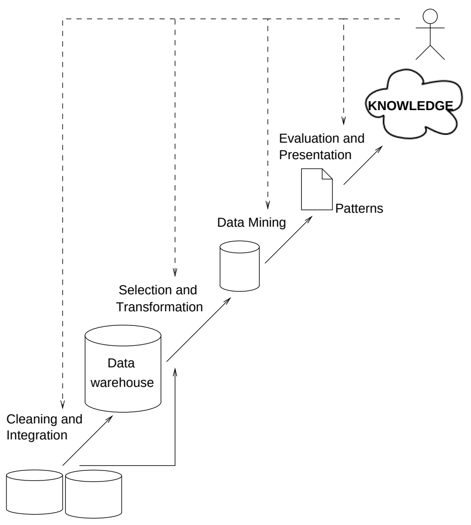

Fig. 1: The KDD Process [4]

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 specially transformed data. One example is a neural network algorithm which needs the input in a distinct interval like *[0,1]*.

Data Mining. This step involves 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. In the last step, the results of data mining are evaluated by distinct interestingness measures. Data mining algorithms deliver a set of patterns. Often these patterns are not interesting to a 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 ones.

3 Architecture of the GridMiner System

GridMiner 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 and are supported or used by services below and

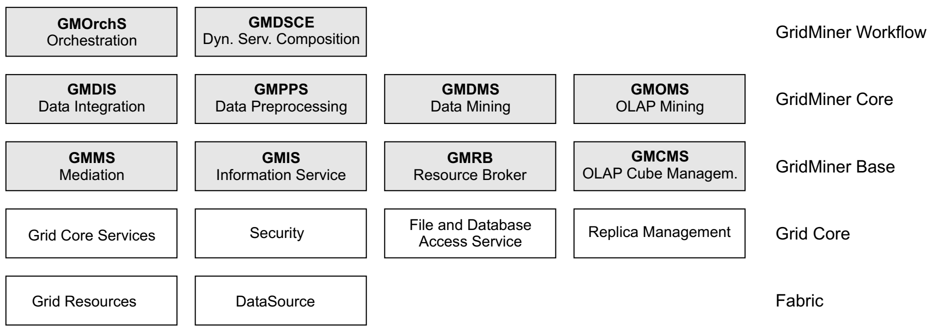

Fig. 2: GridMiner components

above them - see Fig. 2. The services of the GridMiner Core level support the KDD process directly, and hence they are used by the client.

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 sub-summarize under this 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 by 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 that all data mining algorithms usually need, and data mining application developers can easily plug their algorithms and tools in.

GridMiner OLAP Mining Service (GMOMS). It provides algorithms to perform data mining on OLAP cubes. This is a service which works very closely together with an OLAP engine providing, for example, ’drill up’ and ’drill down’ operations, which allow to analyze datasets at different abstraction levels.

GridMiner Dynamic Service Composition Engine (GMDSCE). This service allows the execution of complex, dynamic workflows for distinct knowledge discovery scenarios. The workflow itself is specified by an XML-based language designed within our project. GMDSCE is described in more details in Section 5.

4 Collaboration of the GridMiner Services

The previous section showed the base services provided by the 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 of its parts.

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 individual steps of KDD process, the output of the first service serves as input for the second service. The independence of the various services also allows a parallel execution without any communication overhead. This results in an improvement of performance.

The data themselves of course, are not sent via the service parameters, since they can get very large. The input parameters and the service results contain only references to files or other resources (like references to data services). So data is not transmitted to the services. Every service can choose itself how it would handle the data access (either copying data by GridFTP or remote access by intelligent selection of the needed data rows). Data is not only physical data stored in files or databases. Input or output data sources can also be represented as access services. So some middleware service can provide the data.

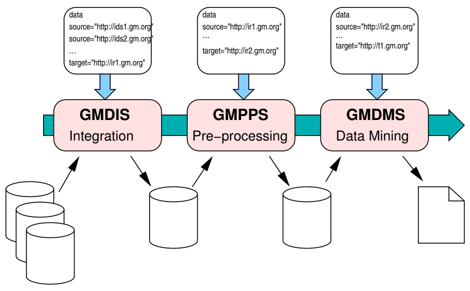
data
source="http://ids1.gm.org"
source="http://ids2.gm.org"
...
target="http://ir1.gm.org"
data
source="http://ir1.gm.org"
...
target="http://ir2.gm.org"
data
source="http://ir2.gm.org"
...
target="http://t1.gm.org"
GMDIS
Integration
GMPPS
Pre–processing
GMDMS
Data Mining

Fig. 3: Example of one KDD process with GridMiner

Fig. 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. Every service has a set of parameters as input. These parameters tell the service for example where to get the input data from, where to store the results and so on and what distinct task to do. The data sources depicted in this figure does not need to be data from a database. So it is possible, that a service uses a common file as input or output data, or even that this data can be an own service. However the service parameters contain information, about the location and type of data.

In this scenario, the client wants to do, for example, clustering on a specific data set. The data is distributed so it would be first necessary to integrate the different data sources into one data set. The parameters tell GMDIS what to do, so they specify where to find the input data sets and where to write the results. The resulting data set is used as input for the GridMiner Pre-Processing Service, which does for example some data cleaning. Next, GMDMS is started, which tries to find clusters. Again the service is controlled via its parameters. So the parameters include the references to data, information about the algorithm and some interestingness measures for the discovered patterns.

The example above illustrates 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 a user wants to mine on more than one data source, it is better to apply sampling before data integration is performed. Or consider that incomplete raw data records shall be ignored. Obviously it is better to execute a preprocessing service for each data source before integration. So the data integration service may need to process less data. It is obviously, that those pre-processing services are executed in parallel and not sequentially. For data mining on top of OLAP we need to replace the GMDMS by an other service called GMOMS (Grid-Miner OLAP Mining Service) and to add a service which creates the cubes (GridMiner Cube Management Service). Sometimes the user wants to apply different algorithms for the same methods, to look which one best fits to the problem solved and compare the results. Often it is useful to call more than one pre-processing service (E.g., the first one does cleaning, the second one data transformation, etc.). So we have many possible scenarios for constructing a workflow which supports knowledge discovery.

5 Workflows

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 an *interactive workflow engine* which *instantiates*, *executes* and *coordinates* a set of *OGSA Grid services* according to a predefined *execution plan*.

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 in a workflow engine. Since every knowledge discovery process requires a unique specific workflow, the workflow engine does not "construct" an own orchestrated service.

Dynamic Service Control Language (DSCL). *Dynamic Service Composition Language* (DSCL) is an XML notation, which has been specially designed to support 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 script specification to control the execution of the described workflow by an engine. A DSCL document is produced directly by a possible client, which prescribes the engine the required execution scenario.

A workflow described by DSCL consists of two important sections: the <**variables**> section and the <**composition**> section. <**variables**> pre-defines all parameters later used by the involved services. These include both, input parameters and output parameters.[1] A concrete parameter is defined in scope of the <**variable**> tag. If the parameter

[1]A parameter is corresponding to a *message* defined within the GWSDL document of an OGSA grid

content is known at built time of the workflow, it is necessary to initialize it, that means to fill in concrete parameter values. The initialization of a parameter follows the SOAP [5] RPC style. Sometimes, not all of the parameter values are known in advance. This happens, if the values are created by a service which will be called earlier in the workflow. In this case it is only necessary to declare the variable without any initialization.

The **<composition>** section describes the workflow to be executed. We define our workflow as a set of *activities*. We distinguish two kinds of the activities. First of all, we have *control activities*. These are activities like *"start parallel execution"* (**<parallel>**) or *"start sequential execution"* (**<sequence>**). These control activities consist of other activities. An additional activity is the so called **<assign>** activity, which is used to copy the content from one variable to another one using XPath [6]. This functionality is needed, if the structure of the output parameter of one service and the input parameter of an other one do not fit together. This is the case when they are using different data structures. *Service activities* are used to control underlying services. DSCL defines activities for creating a new service by a factory or destroying it (**<createService>,<destroyService>**), calling an operation (**<invoke>** and querying the service data elements of a grid service instance (**<querySDE>**). All of these activities make use of parameters described within the **<variables>** section. The example below shows the description of a very simple workflow by DSCL.

```
<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:outputParam="dscl:..."/>
        </dscl:sequence>
    </dscl:composition>
</dscl:dscl>
```

Dynamic Service Control 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 Control 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 use DSCE.

DSCE is implemented as a stateful, transient OGSA Grid service. That means DSCE first of all provides service data elements which are describing the state, and the results of all involved activities. In data mining the user is not only interested in final results. If the results are not satisfying his expectations, the user has to take a look at the intermediate results. Hence he or she must be able to analyze the whole workflow.

service.

Second, DSCE is transient. So every new workflow results in a new service instance. The workflow itself is described by DSCL.

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 [2], and *(d)* the possibility for a user to stop, cancel or resume a workflow.

Fig. 4 shows the architecture of the Dynamic Service Control Engine. As already mentioned, DSCE is implemented as an OGSA grid service. So it provides a factory interface and a service interface. The factory interface is responsible for creating new DSCE instances. It provides only one operation: **CreateService(DSCLDocument dscl)**. If a client calls this operation a new DSCE instance will be created and the new service will be initialized with a DSCL document. Additionally the client will receive the Grid Service Handle (GSH) of the newly created service.

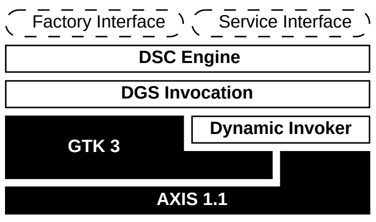

Fig. 4: Architecture of DSC Engine

The service interface provides operations (**updateDSCL()**, **start()**, **stop()**, and **resume()**) to enable a client interactive control. The next architecture layer – *DSC Engine* – is the implementation of the services. It processes the DSCL document and manages the parameters of all underlying services. *DGS Invocation* provides the functionality for doing dynamic invocation in grid services. To do so, it combines the base concepts provided by Globus Toolkit 3 [7] – like accessing service data elements, creating grid service instances and so on – and the *Dynamic Invoker* layer, which provides the dynamic invocation of operations by making use of Axis 1.1 [8] – a SOAP engine.

DSCE also provides a simple caching mechanism, which can result in a performance gain. Consider the following situation. A client sends a workflow described by DSCL to the engine and the engine processes the workflow. The client feels that the results are not satisfying. So it changes some service parameters of one service, somewhere in the workflow and sends the workflow again to the engine (by use of **updateDSCL()**). DSCE recognizes that all services which do not depend on the changed service do not have to be executed again. So a lot of processing capacity can be saved.

6 Related Work

The first proposal of an architecture for performing data mining on the Grid was published in [9]. 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,

[2] If the results of a “run” does not satisfy the user, he analyses 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 these in the workflow description document and start the workflow again.

which can describe compositions of Web Services. The Grid Services Flow Language [11] intends to do the same for Grid Services. GSFL is based on the so called Web Services Flow Language [12], 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 [13] (an engine for BPEL4WS developed by IBM).

7 Conclusions and Future Work

Our concept of Dynamic Service Control presented in this paper is a general approach, which is not only restricted to GridMiner. Other systems, which are based on OGSA Grid services and need components for workflow handling or job control can benefit from this approach. It is even possible to use the workflow component to combine services provided by different systems, or just to pick out some useful services from an other system. To achieve this goal, the Dynamic Service Control Language must provide some additional features. DSCL has to support a notification model to allow direct communication between services, it has to introduce error handling, and there is also the need for an additional security layer.

References

1. Brezany, P., Hofer, J., Tjoa, A.M., Wöhrer, A.: GridMiner: An Infrastructure for Data Mining on Computational Grids (2003)
2. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
3. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P.: From Data Mining to Knowledge Discovery: An Overview. In Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery and Data Mining, AAAI Press/ The MIT Press (1996) 1–43
4. Han, J., Kamber, M.: Data Mining. Concepts and Techniques. Morgan Kaufmann (2001)
5. W3C: Simple Object Access Protocol (SOAP) 1.1. http://www.w3c.org/TR/SOAP (2000)
6. W3C: XML Path Language (XPath) Version 1.1. http://www.w3c.org/TR/XPATH (1999)
7. Project, T.G.: Globus Toolkit 3.0. http://www.globus.org
8. Project, T.A.: WebServices - Axis 1.1. http://ws.apache.org/axis
9. Cannataro, M., Talia, D.: Knowledge grid: An architecture for distributed knowledge discovery. Communications of the ACM (2003)
10. Systems, B., IBM, Microsoft, SAP, Systems, S.: Business Process Execution Language for Web Services. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ (2003)
11. Krishnan, S., Wagstrom, P., Laszewski, G.: GSFL: A Workflow Framework for Grid Services. http://www.globus.org/cog/papers/gsfl-paper.pdf (2002)
12. Leymann, F.: Web Services Flow Language (WSFL 1.0). http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf (2001)
13. IBM: The IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J). http://alphaworks.ibm.com/tech/bpws4j