Towards Service Collaboration Model in Grid-based Zero Latency Data Stream Warehouse (GZLDSWH)

Tho Manh Nguyen, A Min Tjoa
Institute of Software Technology,
Vienna University of Technology
tho,tjoa@ifs.tuwien.ac.at

Guenter Kickinger,Peter Brezany
Institute for Software Science,
University of Vienna
kickinger,brezany@par.univie.ac.at

Abstract

A Grid-based Zero-Latency Data Stream Warehouse (GZLDSWH), built upon a set of OGSI-based grid services and GT3 Toolkit, overcomes the resource limitation issue for data stream processing without using traditional approximate approaches. However, due to its "automated event-based reaction" characteristic, the GZLDSWH requires a mechanism which allows the grid services to be able to work together to fulfil the common tasks. This paper describes the Collaboration Model for the Grid Services which enables the automation of the GZLDSWH in capturing and storing continuous data streams, making analytical processing, and reacting autonomously in near real time with some kinds of events based on well-established Knowledge Base.

Keywords: *Grids based Zero-Latency DWH, Data Streams processing, dynamic workflow execution*

1. Introduction

We are entering a new area of computing in today's incredibly complex world of computational power, very high speed machine processing capabilities, complex data storage methods, next generation telecommunications, new generation operating systems and services, and extremely advanced network services capabilities. At the same time, the number of emerging applications which handle various continuous data streams [1,10,12,17], such as sensor networks, networking flow analysis, telecommunication fraud detection, e-business and stock market online analysis, is growing. It is demanding to conduct advanced analysis over fast and huge data streams to capture the trends, patterns, and exceptions. Data streams arrive in high-volume, in un-predictable rapid bursts and need to be processed continuously. Processing data streams, due to the lack of resource, is challenging in the following two aspects. On the one hand, random access to fast and large data streams may be impossible. On the other hand, the exact answers from data streams are often too expensive. Therefore, the approximate answers [1,9,12,15] are acceptable because there is no existing computing capacity which is strong enough to produce exact analytical results on continuous data streams.

In the last few years we have witnessed the emergence of Grid Computing [6,8] as an important new technology accepted by a remarkable number of scientific and engineering fields and by many commercial and industrial enterprises. Grid Computing provides highly scalable, secure, and extremely high performance mechanisms for discovering and negotiating access to remote computing resources in a seamless manner. Our research effort is trying to build a Grid based Zero-Latency Data Stream Warehouse (GZLDSWH) that can capture continuous data stream to perform analytical processing and react automatically with some kinds of events based on well-established Knowledge. Instead of applying the approximate approach based on statistical estimations, we try to capture and store all data streams continuously while making the analytical processes within the Grids. The GZLDSWH is composed of several Grid services built on top of OGSI and GT3 toolkit. However, due to its "automated event-based reaction" characteristic, the GZLDSWH requires a mechanism which allows the grid services to be able to work together to fulfil the common tasks. During its runtime, a Grid service instance must be able to discover, create, bind, and invoke other relevant service instances within the Grids environment.

This paper describes the Automated Collaboration Model for Grid Services enabling the automation of a GZLDSWH in capturing and storing continuous data streams, making analytical processing, and reacting autonomously in near real time with some kind of events based on well-established Knowledge. An overview of the GZLDSWH will be described in Section 2. Section 3 introduces the operations of the GZLDSWH services to react against continuous data streams in the near real time. The Collaboration Model of the Grid services will be discussed in Section 4. In Section 5, we introduce our XML-based language namely DSCL. The Workflow management service will be described in Section 6. Section 7 concerns some Related Work. Finally, we give a conclusion and future work in Section 8.

2. GZLDSWH overview

A Zero-Latency Data Warehouse (ZDLWH) [3,11] aims to significantly decrease the time to react to business events. This allows the organizations to deliver relevant information as fast as possible to applications which need a near real-time action to new information captured by an organization's information system. In Data Stream applications, events take the form of continuous data streams. The exact analysis results on these data stream events are very expensive because they require a very high computing capacity which is capable of huge storage and computing resources. A Grid-based approach thus is applied in ZLDWH to tackle the issue of lacking resource for continuous data stream processing. Figure 1 depicts significant phases throughout the overall process of such GZLDSWH.

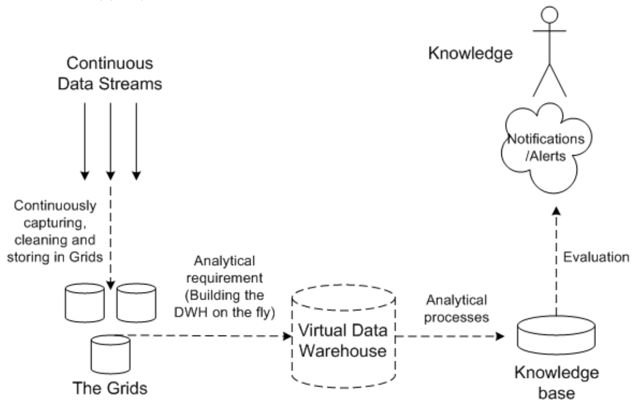

Figure 1. Overview of GZLDSWH

The continuous data streams will be captured, cleaned and stored within the Grids. The analytical processes can be executed immediately after the new data arrival or based upon some predefined schedule. Consequently, the virtual Data Warehouse will be built on the fly [7] from data sources stored within the Grid nodes. Obviously, this approach is not concerned with traditional incremental updating issues in Data Warehouse because the virtual DWH is built from scratch using the most current data. Analytical processes will then be executed on such virtual DWH and the results will be evaluated by the Knowledge Base. Finally, depends on specification of the Knowledge Base, the system sends notifications, alerts or recommendations to the users.

All of these activities will be executed automatically without user intervention. The activities at each phase compose of several tasks and the specific tasks will be accomplished by invoking relevant services within the Grid environment. Sharing the same approach with GridMiner [2,13,14], the structure of GZLDSWH is built on top of OGSI and GT3 Toolkit, including the following specific Grid services as described in figure 2.

- **Data Capturing Service (DCS)** captures data streams in the limited time without loosing the data.
- **Data Cleaning Service (DES)** is the optional service that cleans the data before storing into the Grids.
- **Data Storing Service (DSS)** resides in each Grid node and ready for storing the data streams without loss.
- **Data Mediation Service (DMS)** provides a single virtual data source having the same client interfaces as classical grid data sources but integrating data from multiple heterogeneous federated data sources.
- **Data Integration Service (DIS)** is responsible for secure, reliable, efficient operation and management of the necessary data transfer within the grid environment.
- **OLAP Cube Management Service (CMS)** is one of the major components of GZLDSWH. This service creates distributed OLAP cubes [1] from several data sources stored at specified Grids nodes. After the initial cube creation, the service can be used for cube interaction and life cycle management.
- **System Information Service (SIS)** The SIS is a specialized implementation enabling the system for specific decision making and monitoring.
- **Resource Broker Service (RBS)** is used to find best-fitting resources for resource allocation as a reference for the Workflow Engine in discovering, creating, binding and invoking other services instances.
- **Data Preprocessing Service (DPS)** performs several pre-processing activities such as data cleaning, normalization, selection, reduction, transformation etc.
- **Data Analysis Service (DAS)** is another major core component of the system. It works very close with the OLAP Engine and performs analytical process by sending commands such as "drill up", "drill down", "slide and dice", etc. which allows analyzing datasets at different abstraction levels. The outputs of the Data Analysis Service are the analysis results which will then be evaluated by the Knowledge Base for further actions.
- **Data Mining Service (MIS)** is created as an extensible framework providing necessary data mining algorithms making it convenient for the related application developers to easily plugin their algorithms and tools.
- **Knowledge Base Rule Design Service (RDS)** allows

[1] Online Analytical Processing (OLAP) is based on multi-dimensional data structures called cubes

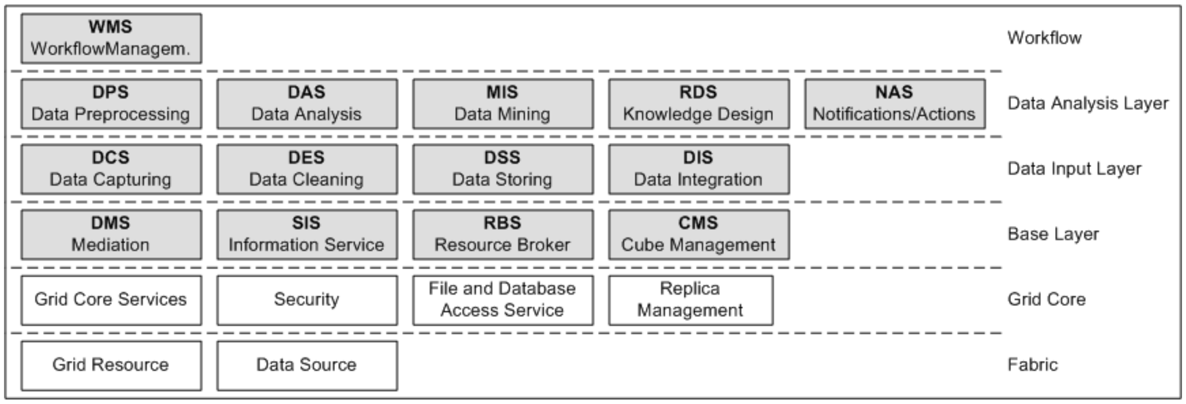

Figure 2. The Service Components of GZLDSWH

the users to specify the Knowledge Base for the system. The Knowledge Base embeds the ECA rules which consists of the event, condition and action part, but carries out complex OLAP analyzes on warehouse data instead of evaluating simple conditions as compared with ECA rules in OLTP system.

- **Notification/Action Service(NAS)** takes the analytical results from DAS, evaluates these results against the Knowledge Base rules, and finally takes suitable actions such as issuing notifications, alarms or recommendations to the users. It can also invoke the DAS to perform analytical process when receiving the data update events from Grids node data sources.
- **Workflow Management Service (WMS)** is used for services collaboration and cooperation. This service provides the execution of the complex, highly dynamic workflows for several heterogeneous grid services. The workflow supports service execution, service termination, service communications, etc.

3. The operation of GZLDSWH

In this section, we describe how the GZLDSWH system operates for processing and reacting to the continuous data streams, in the near real time. The system operations are based on the collaborative interaction between the services to fulfill the pre-defined reactive plans which are specified by the advanced knowledge user. Within the Grids environment as described in Fig. 3, there are one Master node and several child nodes (Node 1, 2…, Node N). The Master node controls other child nodes to fulfill system activities. These child nodes keep the role of storing data within the Grid environment. The Master node therefore includes most of the essential services while the child nodes only contain some data input services and local data update detection services. The Master node also keeps the Grid metadata for Grids management and the Knowledge Base for controlling event reaction behavior.

The operation of the GZLDSWH is as follows. The Data Capturing Service (DCS) receives continuous data streams from stream sources such as sensor systems, satellites, etc. Due to the huge amount of data arriving, the DCS must capture the data timely and invokes available Data Storing Services (DSS) resident at several child nodes for storing data. The DCS could invoke Data Cleaning Service (DES) to clean the data before storing.

After storing data at child nodes, the Analysis Service (DAS) at the Master node will be invoked immediately or after predefined timely schedule depending on application requirements and performance trade off. DAS execution will create the virtual Data Warehouse from scratch. For this purpose firstly, the DASs available at several local child nodes are invoked. Due to this, the Cube Management Service (CMS) gains the essential raw data at the child nodes to build the global cube. Each child node contains part of the cube namely "cube chunk". Data will then be integrated into the common format by the Data Integration Service (DIS). Before being stored into the virtual Data Warehouse, data can be passed to preprocessing phase via the Data Preprocessing Service (DPS). The DPS can perform several tasks such as data cleaning, data transformation, data normalization or data reduction. After the global cube is formed, the DAS will perform analysis queries or data mining algorithms (via the equivalent Mining Service - MIS) based on the data inside the virtual DWH.

The analysis results then will be sent to the Notification-Action Service (NAS) for evaluation. The NAS accesses the Knowledge Base and evaluates the rules. The Knowledge Base rules are provided by the user through the Rule Design Service (RDS). The NAS then will issue relevant notifications or alerts to the users. It can also send back the action commands to several grid child nodes for executing some actions at the local data sources such as insert, delete, update, etc. Besides, the analysis process can be executed to answer the analysis queries issued by other applications. Especially, the analysis process can also be executed in case the local update data happens at the grid child nodes. Whenever the local data update happens, the NAS at local child node sends the "local data updates" message events to the NAS of the Master node. The NAS then invokes the DAS and the Analysis process will execute.

The Grid services invocation process described above is strictly monitored by the Resource Broker Service (RRS) and the System Information Service (SIS). These services manage the resource available and finding the best-fitting resources for resource allocation and dispatch. The role of Workflow Management service (WMS) is to execute the complex, highly dynamic workflows involving different grid service instances. The workflows are constructed flexibly through the adaptive, architectural interaction framework.

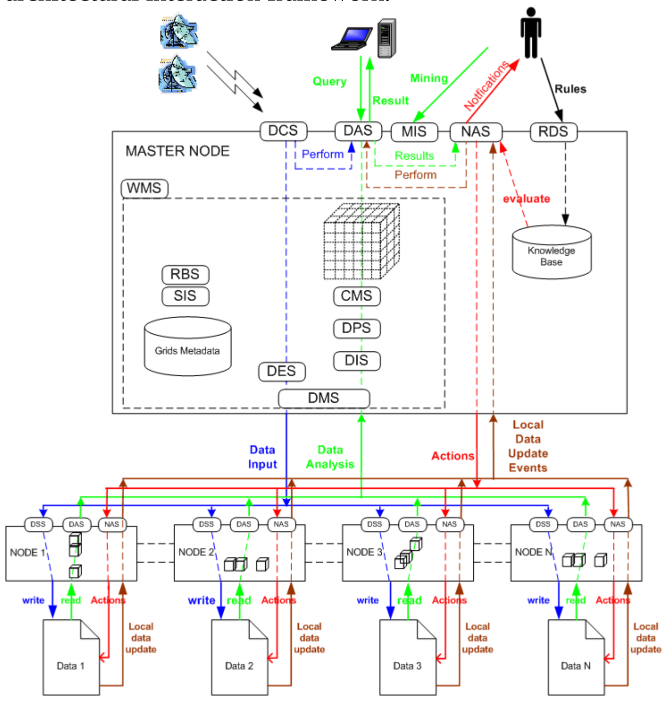

Figure 3. The operations of GZLDSWH

4. Automatic Grid service Collaboration

As previously mentioned, GZLDSWH is composed of many specific OGSI-based services. Each service is able to perform an individual task within the whole process. Obviously, these services must have the ability of collaborating with other services to fulfill the whole common purpose. In GridMiner [13,14], the services do not communicate with each other. The output of the first service serves as the input of the second service, the output of the second one serves as the input of the third one and so on. No service thus is aware of other existing services and each of the service is able to run completely independently.

However, in our system, due to the requirement of automated event-based reaction, a service must be able to discover, create, bind, and invoke relevant service instances within the Grids environment. The execution flows are specified by pre-defined workflow in which the services are arranged in the specified logical execution order to fulfill the common task. However, during the execution time, the services have their autonomy to invoke other relevant physical service instances depending on the context at that time. We therefore need the model that enables the automatic adaptive collaboration between the Grid services followed by the pre-defined plan which describes the logical service execution flows (as described in figure 4).

To the best of our knowledge, there are 2 possible approaches for the service flow execution i.e. centralized control and distributed control. In the former approach, there is a central service control engine which controls all service executions from the start node to the end node of the workflow. The engine itself is responsible in discovering, creating, binding, invoking, and destroying service instances to follow the logical workflow. The engine thus must keep the information of the whole workflow and should trace the information of the Grid environment such as grid nodes status, resource availability, etc. to coordinate the services execution. In the later approach, there is no such central engine but each service instance has its own "knowledge" to invoke the next service instances throughout the workflow. It is not necessary for each service to keep information of the whole workflow, instead, each service need to keep only part of the workflow metadata related to itself such as its immediate successor and predecessor services, the Grids environment context at its time of execution. That information is passed to the service as parameter at the time of its invocation. The service will use such information to invoke the next relevant service instances.

Both of the two approaches have advantages and disadvantages. In the centralized control approach, the central service control engine, which could also be realized as a service, copes with the coordination between other services. The other services thus only focus on their specific functionality without taking into account the workflow execution. However, it could be the heavy work-load for the engine service if it processes the high complexity workflow or if the number of service instances increases. The distributed control approach, on the other hand, does not have to deal with the bottle-neck issue. However, it is more complicated to develop the services because each service besides its specific functionality must be realized as an agent to adapt with flexible service instance invocation. Moreover, the service invocation would also become more complex due to the parameters

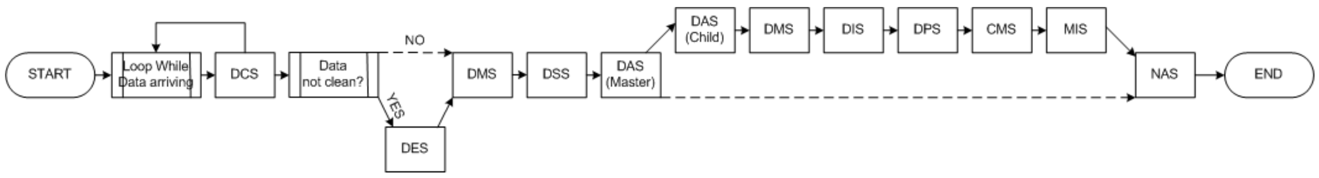

Figure 4. Predefine Workflow of Service Invocation

transferred between the service instances. Further investigation on distributed control approach is out of the scope of this paper. However, it will be one of our considerations in the future work.

In GZLDSWH, we use the central service control engine to coordinate the service execution via XML based workflow description language namely DSCL (Dynamic Service Control Language). The engine will extend the DSCE engine specified in GridMiner [14] to support the condition branches, loops as well as allow the references of the service instance handles (GSH) could be transferred as parameters in DSCL. The logical workflow could be specified with the service instance handles are not known in advanced (their handle references will be declared as variables). During the execution time, the Engine queries the Resource Broker Service to get the relevant dynamic service instance handle references. The Engine then will invoke these service instances via the reference variables. That operation will be repeated at each step of the workflow until the whole process is finished. We will describe in more detail about the DSCL in Section 5 and our Workflow Management service in Section 6.

5. Dynamic Service Control Language

DSCL is a XML based language allowing the users to specify the workflow of services activities. It contains exactly two sections:

- The <**variables**> section: all variables must be defined here. The variables could be either the parameters of service calls, or the results of service calls. XML Schema Simple Type, Complex Type and SOAP Arrays Type are supported as variable type.

```
<variables>
   <variable name="iAge">
        <value type="int" 25  </value>
   </variable>
</variables>
```

- The <**composition**> section contains the description of the workflow to be executed. A workflow composes of a set of activities which could be classified as "*control flow*" or "*operational*". The control flow activities controls the execution of the workflow and thus must contain other activities while the operational activities are the atomic activities which perform operations.

```
<composition>
        control flow activities
              other control flow activities or operational activities
        operational activities
</compostion>
```

5.1 Workflow Structure

Our DSCL supports 4 basic execution styles (Sequential execution, Parallel execution, Condition Branch and Loop) by providing several tags namely <**Sequence>**, <**Parallel>**, <**Condition>**, and <**Loop>** respectively. These tags could be nested to realize the complex workflow composition. Figure 5 states an example of a workflow including all control activities and the respective DSCL document

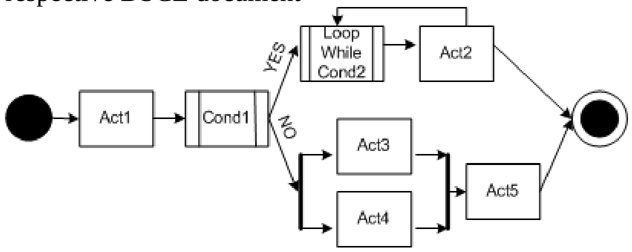

Figure 5. Composite Workflow Example

```
<composition>
  <sequence>
    activity1
    <condition>
      cond_var1 = TRUE
            <loop while cond_var2 = TRUE>
               activity2
            </loop>
      cond_var1 = FALSE
      <sequence>
            <parallel>
                        activity3
                        activity4
            </parallel>
            activity5
      </sequence>
    </condition>
  </sequence>
</compostion>
```

5.2 Workflow operations

Beside the control flow activities, DSCL supports other activities namely operational activities. Operational activities perform operations of interacting with the underlying Grid services such as creating new service instances, destroying instances, invoking operation of services, querying service data element. DSCL provides respective tags to specify these operational activities: <**createService**>, <**destroyService**>, <**invoke**>, and <**querySDE**>. The operational activity could not contain other activities and must have the mandatory attribute namely activityID which is of type DTD. This attribute is necessary for the workflow engine to identify the activity.

The great difference between Grid and common Web services is the fact that the Grid service could be either persistent or transient. The persistent service is created and available if its container is running. In contrast, the transient one is created and invoked when required and soon destroyed afterwards. The transient service is always created by its Factory service. The following information is necessary to create a new service instance

1. The location of the factory service
2. Additional service parameters
3. A virtual instance name of the newly created instance

```
…
<createService activityID ="Act1"
factory-gsh="http://url/serviceFactory"
instance name="newInstance1"/>
…
```

In GZLDSWH, there is the situation when we have more than equivalent factory services at different grid nodes at the same time e.g. the Data Storing services located at several child nodes when we need to store data stream. In such situation and in other cases when the Grid environment changes rapidly, the Resource Broker Service decide which Service Factory should be executed to create a new instance according to the availability and resource capacity of the different Grid nodes. DSCL provides the dynamic service creation by allowing Grid service handle references transferred via variables. It is also possible to create the service instance with user defined parameters via <**parameter**> tag.

```
…
<variables>
  <variable name="factgsh">
    <! Default value of the factory service handle>
    <value type="string" http:url/serviceFactory </value>
  </variable>
…
</variables>
…
<!other services set the value of factgsh, e.g. Resource Broker >
…
<! Create the service instance via reference to factory handle >
<createService activityID ="Act1"
factory-ref="factgsh"
instance name="newInstance1"/>
```

After a service instance is created, it is possible to destroy the instance, invoke its operations or query its data elements. These activities require the service instance reference to identify the relevant instance. We provide 3 optional attributes namely *instance-name*, *instance-gsh*, and *instance-ref* enabling to reference a service instance (1) via instance name (2) via Grid service handle and (3) via a variable reference to the instance. The usage of each attribute is optional, however exactly one of the three attributes must be used together with the activity.

```
…
<destroyService activityID ="Act1"
instance-gsh="http://url/SerInst01" (or instance-name =
“Instant1” or instance-ref = “varInstGSH”)
…
```

Invoking an operation of a Grid service is similar to invoking a method in common programming language, to invoke an operation of a Grid service, the following information is necessary:

1. The required Grid service instance – referenced by one of the attributes: *instance-name*, *instance-gsh*, *instance-ref*
2. Name of the operation – mandatory defined within attribute *operation* and optional attribute *portType*
3. The required parameters – specified in <**parameter**> tags, the parameter is simply a reference to a variable.
4. Result – storing the result of a Grid invocation via <**result**> tag

```
…
<Invoke activityID ="CleanData01"
instance-gsh="http://url/DES01" (or instance-name = “DES01”
or instance-ref = “varInstGSH”)
operation = “Clean_Data”
<parameter variable= “Data_Strore”>
<result variable= “Data_Result”>
</Invoke>
…
```

Some of operations do not return the results; instead store them into so called service data elements. To allow querying the content of these element, DSCL provides the tag <**querySDE**>, this operation requires the reference to Grid service instance and the name of the required service data element (stored in attribute *sdName*).

```
…
<querySDE activityID="act1"
  instance name="instance01"
  sdName="value"
  <result variable="var02"/>
</querySDE>
…
```

6. The Workflow Management Service

In GridMiner project [13,14], we have developed an engine service so called the Dynamic Service Control Engine (DSCE) which processes DSCL documents and controls the service execution in both interactive and batch modes. It provides some interesting features such as (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 for a user to stop, cancel or resume a workflow and (d) the possibility to change workflow at run time (by stopping the engine, changing the DSCL document, and restarting engine again).

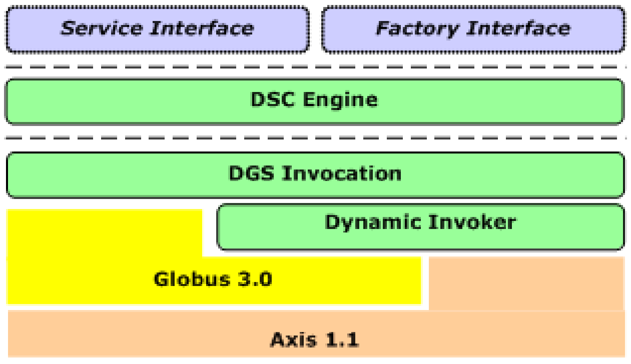

Figure 6. Conceptual Architecture of DSCE

Figure 6 describes the Conceptual Architecture of DSCE [14]. The engine is implemented as a stateful, transient OGSI Grid service and has several structured layers. The "top" layer is the Interface layer which provides essential operations to control the engine. The Factory interface allows users to create a new DSCE instance for a specific DSCL document via operation *CreateService (DSCLDocument dscl)*. The DSCE engine instance now will be created and its state will change within its life cycle according to user interactions and the activities execution results. The possible states could be *empty*, *initialised*, *running*, *stopping*, *waiting*, *finishing*, or *failure*. The Service interface provides interactive control operations such as *changeWorkflow()*, *start()*, *stop()*, *resume()* as well as several service data elements containing information about the DSCL document, Workflow state and activity state.

The "middle" DSC Engine layer covers the main functionality of DSCE. It controls the whole workflow execution by controlling the execution of activities specified by the DSCL document. First, the DSCL workflow description is parsed, then the "network of activities", an internal model of the workflow, is constructed before processing the activities. Such activities network describes the dependency between the activities. Each activity could have succeeding and preceding activities. Succeeding activities are executed right after the execution of actual activity is finished. If an activity has more than one successor, all of them will be executed in parallel after that actual activity is finished. Similarly, the activity could not be started until all of its proceeding ones are finished. This could happen in some situations like loop or parallel execution. Several internal operations are provided in this layer for managing the workflow such as *start()*, *stop()*, *resume()*, *reset()*, *setDSCLWorkflow()* etc. as well as some operations for controlling the activities such as *startActivity()*, *EndActivity()*, *CreateInstanceActivity()*, *DestroyInstance Activity()*, *InvokeActivity()*, *QuerySDE-Actvity()*, *startNext Activites()*, *wat-ForPrevious-Activities()* ,etc. The necess- ary parameters of all underlying services are also prepared at this layer.

Normally, when a Grid service is developed and implemented, additional stub and proxy classes are generated to hide the complexity of communication between the client and the service. This approach is very common and practically used in all distributed object systems like CORBA, Java RMI, and Web Service. To benefit from this approach, the required services or remote objects must be known at the compilation time. However, this requirement is not satisfied in DSCE because DSCE receives a DSCL workflow description document and shall be able to communicate with all services specified within that DSCL. The "lowest" layer namely Dynamic Grid Service Invocation (DGSI), is composed of the DGS Invocation and Dynamic Invoker. It provides classes which allow accessing Grid services and their operations without using common stubs/proxy approach. The Dynamic Invoker, the lowest layer, provides the possibility to invoke any operation on any underlying Grid service. It uses much of ApacheAxis [23], and SOAP engine which are based of GT3 toolkit. Dynamic Invoker translates an operation invocation into a SOAP1.1 message and sends it to the corresponding service to invoke specified operations. It provides all necessary marshalling and un-marshalling of arguments by firstly fetching the WSDL of the corresponding Grid service (via its handle GSH), then setting service port type via *setPortType(String port-TypeName)*, setting operation via *setOperation (String operation Name)* , setting parameters of the operation via *setParameters (Object[] params*). All of information is used to construct essential SOAP operation call. Finally *invoke()* executes the operation by sending that SOAP message to correspond services. At higher layer, the DGS Invocation provides the classes to use stub-less operation invocation and to access the functionality of the GT3. It provides three classes namely *DGSIService*, *DGSIFactory*, *DGSI-Listener* allowing the workflow engine to handle its underlying services such as creation and destruction of Grid service instance, invocation of operations, querying of service data element and synchronization of asynchronous service calls.

DSCE suits well in GridMiner where the interaction role of user is important. The engine operates based on the "*physical*" DSCL document specified by the user, i.e.

it only works with the DSCL that specifies exactly the service handles. It does not accept the "*logical*" workflow which only specifies the logical name of the required service. In GZLDSWH, we sometimes do not know in advance which service factory should be executed to create a new instance. Instead, the decision should depend on the runtime environment. Besides, because of the "*automated event-based reaction*" feature of GZLDSWH, a higher level of automation in service invocation engine is required. Therefore, the Dynamic Workflow Management Service in GZLDSWH extends the DSCE with the automatic Workflow Re-writer ability. Now, the WMS Service will accept the logical DSCL, parse it and find out logical services i.e. services which do not have the exact physical factory handle. It then queries the Resource Broker Service to have the relevant physical service factory handle and then re-write the DSCL with the new factory handle value. It finally passes the re-write DSCL to the DSCE engine to invoke the services. The architecture of Dynamic Workflow Management service is described in figure 7.

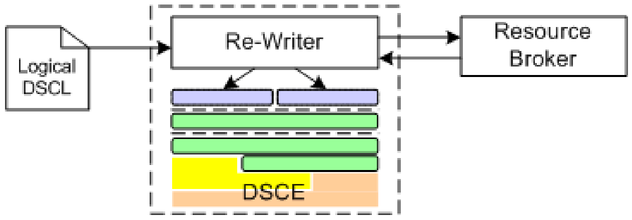

Figure 7.Dynamic Workflow Management service

7. Related Work

There has been a surge of interest recently in the area of query processing over continuous data streams [10,12,15], and other related problems such as resource management, approximately computation, architectures [9,10,16,17]. Furthermore, conventional OLAP and data mining models have been extended to tackle data streams, such as multi-dimensional analysis [5], clustering [18] and classification [19]. However, most of previous approaches on data stream processing focus on approximate methods based on statistical estimations due to the limitations of storage and computing resources. Our effort, instead, tries to store all data streams and processes them within the Grids as if they are stored in the super large databases.

So far, only a little attention was devoted to knowledge discovery on the Grid. An attempt to design an architecture for performing data mining on the Grid was presented in [4]. The authors present the design of a Knowledge Grid architecture based on the non-OGSI-based version of the Globus Toolkit, and do not consider any concrete application domain. R. Moore presents the concepts of Knowledge-based Grids in [20]. A lot of valuable data integration concepts have been developed in the project "Federated Database for Neuroscience" [27]. The WP4 of the OGSA-DAI project is concerned with the design of a distributed query processing service for the Grid. However, the above projects did not take into account the automatic collaboration between the Grids services.

Workflow, "the coordinated execution of multiple tasks or activities" [28], can be extended and applied virtually to other areas, from science and engineering to entertainment. Web services already provide mechanisms to handle complex workflows. Since every Grid service is a Web service with improved characteristics and services [29] (the converse of this statement is not true), it is possible to adapt the ideas for workflow compositions from Web services and apply them to Grid services. BPEL4WS 1.1 [24] is the actual standard, which can describe compositions of Web Services. The Grid Services Flow Language [26] intends to do the same for Grid Services. GSFL is based on the so called Web Services Flow Language [25], a predecessor of BPEL4WS, published by IBM. All of those flow language specifications have all the same target: describing a business process built up of various web services. This description then serves as input for a workflow engine like BPWS4J [24] (an engine for BPEL4WS developed by IBM). Such an engine works with the persistent Web services (not transient Grid services as in GridMiner or ZLGDSWH), and of course, it requires the specification documents of "physical" Web service URIs.

8. Conclusions and Future Work

The concept of Grid-based Zero-Latency Data Stream Warehousing system applied Grid technology for continuous data streams processing to tackle the resource limitation issues. In this paper, we have introduced a Grid service collaboration model which allows the Grid service in GZLDSWH to collaborate with each other automatically following the pre-defined logical workflow. We have extended the DSCL language and DSCE engine developed in GridMiner to allow the dynamic re-writing of the logical workflow to the physical one according to the Grid environment at runtime. Detailed technical references and some results of DSCL and DSCE could be found in our Technical Report [14].

Both Grid and Data Stream processing technologies are young and still evolving. The Semantic Grid [21] and Grids services have the role similar to Semantic Web and Web services. Recently, WS-Resource Framework &

WS-Notification [22] proposals have just been announced as an evolution of OGSI with the purpose of effective integration Grids and Web services standards. The work presented here is closely related to OGSA/OGSI so it has to adopt with the WS-Resource Framework with suitable modifications. The distributed control of service discovery, creation, invocation and destroying will be considered as an alternative of collaboration model. The orchestration of Grids or Web services, another approach for solving the workflow problem should also be further investigated.

Acknowledgement

This work is partly supported by grant P16095-N05 of the Austrian Science Fund (FWF).

The first author is supported by the Science and Technology grant of the Austrian Government and by the Student Travel Awards for Research of Microsoft Research (STAR Program).

Reference

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems", *Proc. of the 2002 ACM Symp on Principles of Database Systems*, June 2002.

[2] P. Brezany, J. Hofer, A. Tjoa, and A. Woehrer, "GridMiner: An Infrastructure for Data Mining on Computational Grids", *APAC Conf. and Exhibition on Advanced Computing, Grid Applications and eResearch*, October, 2003.

[3] R. Bruckner, *"Zero-Latency Data Warehousing: Toward an Integrated Analysis Environment with Minimized Latency for Data Propagations"*, Ph.D. Thesis, Vienna University of Technology, November 2002.

[4] M. Cannataro, and D. Talia, "The Knowledge grid: An architectture for distributed knowledge discovery", *Communications of the ACM, Vol. 46, No. 1*, January 2003.

[5] Y. Chen,, G. Dong, J. Han, B. Wah, and J. Wang, " Multi Dimensional Regression Analysis of Time-Series Data Streams", *Proc. of the 28th VLDB Conference*, Hong Kong, 2002.

[6] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations", *Intl. J. Supercomputer Applications*, Vol. 15, No. 3, 2001.

[7] I. Foster, and R. Grossman, "Data Integration in a Bandwidth-Rich World", *Communications of the ACM, Vol. 46, No. 11*, November 2003.

[8] J. Joseph, C. Fellenstein, "Grid Computing", *Prentice Hall PTR*, December 2003.

[9] S. Guha, and N. Koudas, "Approximating a data stream for querying and estimation: Algorithms and performance valuation", *Proc. of the 2002 Intl. Conf. on Data Engineering*, 2002.

[10] R. Motwani, and J. Widom et al., "Query processing, approximation, and resource management in a data stream management system", *Proc. First Biennial Conf. on InnovativeData Systems Research (CIDR)*, Jan. 2003.

[11] N. Tho, and A. Tjoa, "Zero-Latency Data Warehousing: Continuous Data Integration and Assembling Active Rules", *5th Intl. Conf. on Information Integration and Web-based Applications and Services*, Jakarta, Feb. 2003.

[12] P. Tucker, D. Maier, and T. Sheard, "Applying Punctuation Schemes to Queries Over Continuous Data Streams", *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, March 2003.

[13] G. Kickinger, J. Hofer, A. Tjoa, and P. Brezany, "Workflow Management in GridMiner", *3rd Cracow Grid Workshop, Cracow, Poland*, October 27-29, 2003

[14] G. Kickinger, and P. Brezany, "The Grid Knowledge Discovery Process and Corresponding Control Structures", *Technical Report*, March, 2004 (http://www.gridminer.org/ publications/gridminer2004-02.pdf)

[15] S. Chandrasekaran, and M. Franklin, "Streaming queries over streaming data", *Proc. 28th Intl. Conf. on Very Large Data Bases*, Aug. 2002.

[16] A. Dobra, M. Garofalakis, J. Gehrke and R. Rastogi, "Processing complex aggregate queries over data streams", *Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, 2002.

[17] A. Lerner, and D. Shasha, "The Virtues and Challenges of Ad Hoc + Streams Querying in Finance", *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, March 2003.

[18] S. Guha, N. Mishra, R. Motvani, and L. O'Callaghan, "Clustering Data Streams", *Proc. IEEE Symposium on Foundations of Computer Science (FOCS00)*, CA, 2000.

[19] G. Hulten, L. Spencer, and P. Domingos, "Mining Time-changing Data Streams", *Proc. of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD01)*, CA, 2001.

[20] R. Moore, "Knowledge-Based Grids", *Technical Report TR-2001-02*, San Diego Supercomputer Center, Jan. 2001.

[21] D. Roure, N .Jennings, and N. Shadbolt1, "The Semantic Grid: A Future e-Science Infrastructure", available at http://www.semanticgrid.org/documents/semgrid-journal /semgrid-journal.pdf

[22] Globus Alliance, IBM, and HP, "The WS-Resource Framework", at http://www-fp.globus.org/wsrf/default.asp

[23] The Axis Project, "Webservices – Axis 1.1", at http://ws.apache.org/axis

[24] IBM, "The IBM business process execution language for web services", at http://alphaworks.ibm.com/tech/bpws4j

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

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

[27] B. Ludaescher, A.Gupta, E. Martone, "Model-based mediation with domain maps", *17th Intl. Conference on Data Engineering (ICDE)*, Heidelberg, April 2001.

[28] D. Marinescu, "Internet-Based Workflow Management: Toward a Semantic Web", *John Wiley*, 2002.

[29] B. Sotomayor, "The Globus Toolkit 3 Programmer's Tutorial", at http://www.casa-sotomayor.net/gt3-tutorial/