Towards A Grid Collaborative Framework

Binh Thanh Nguyen, Duc Huu Nguyen, and Doan Bang Hoang

***Abstract*—In Grid Computing, the grid collaborative frameworks that support easy and effective collabaration and coordination between many remote users have emerged as an important research topic recently. In our previous paper [1], we have proposed a grid collaborative framework that is both general purpose and plan supported. With the theoretical foundation based on the activity theory and designed on top of existing OGSA infrastructure, our proposed framework aims at accelerating the development of grid collaborative systems that consider working plans as central role. To support plans, our framework needs including a workflow language that not only can invoke Web services, but also Grid services. Among current workflow languages, the BPEL seems to be most suitabe for our framework, but it still lacks of Grid service invocation capability. Therefore, in our other work [2], a clean solution for this problem has been proposed by using ODE engine. This paper aims to combine these results into a more complete picture of our framework.**

***Index Terms*— Grid collaborative framework, Grid services, BPEL, BPEL engine, ODE, Grid computing.**

I. Introduction

Over the last decade, Grid Computing has played an important role in resolving a real and specific problem of coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [3]. Grid research has progressed to its third generation [4], which focuses on resolving problems that occur when large scale and autonomic grid systems need to be built. This generation has seen an increase in the adoption of service-oriented architecture and the development of a comprehensible architecture for large scale grid applications. The Open Grid Service Architecture (OGSA) was developed to support the creation, maintenance and application of ensembles of services in Virtual Organizations (VO). OGSA adopted the OASIS Web Service Resource Framework to bring Grid services closer to Web services community, allowing them to share and reuse tools that have been well developed for Web services.

In our previous paper [1], we have proposed a grid collaborative framework that is both general purpose and plan supported. With the theoretical foundation based on the activity theory and designed on top of existing OGSA infrastructure, our proposed framework aims at accelerating the development of grid collaborative systems that consider working plans as central role. Our framework has a component called **Activity Planning** that is responsible for creating a new working plan or updating existing ones. Among the current workflow tools and languages that can support in creating working plans so far, the BPEL seems to be the best choice for our framework.

BPEL (Business Process Execution Language) has been deployed successfully in composing workflows of Web services for business applications. It is not surprised to see efforts to use BPEL for composing Grid services into higher level and structured tasks as Gridflows. However, due to the stateful nature of Grid services, BPEL and its engines cannot be deployed without additional features.

Much effort in recent researches attempts to overcome this problem. Some proposals have been suggested to invoke Grid services from BPEL [5, 6], but they are only for the ActiveBPEL engine. For the open source BPEL engine, ODE (Orchestration Director Engine), so far we are not aware of any concrete solution. Therefore, a clean solution for this problem has been proposed in our other work [2].

For deployment of BPEL processes in the ODE engine, an another issue still exists in the deployment stage, that requires manual preparation of a specific deployment file. This procedure is so time consuming and error prone that makes it very hard to let the BPEL processes run by this engine. This issue has also been solved by our solution in another work [7].

From these results, we have tried to integrate the engine into our framework in order to make it work, and this paper aims to describe this integration process.

The structure of the paper is as follows. Section 2 will present some background. Section 3 will present our collaborative framework. In section 4, our solution for invoking Grid services within the ODE engine will be presented. And then in section 5, a solution allowing automation of the deployment stage of BPEL processes in the ODE engine will be described. Section 6 will discuss about related work and make comparison with ours. The last section 7 summarizes the contributions and concludes the paper.

II. Background

A. WSRF

Before Grid services, a Web service refers to a stand-alone service with each instance is completely independent of any other instances of the same service as they do not keep any state information about themselves once they deliver their output to the requested Web client. This stateless makes Web services client-server model simple, however, developing transactional services based on Web services requires complicated manipulation of the persistent states at the server

Manuscript received February 12, 2012, revised February 27, 2012.

Binh T. Nguyen and Duc Huu Nguyen are with the school of Electronics and Telecommunication of the Hanoi University of Science and Technology, Vietnam (e-mail: ntbinh1974@gmail.com; ducnh-fit@mail.hut.edu.vn).

Doan B. Hoang is with Computing and Communications, Faculty of Engineering and Information Technology, the University of Technology, Sydney (UTS), Australia (e-mail: Doan.Hoang@uts.edu.au).

end that is not consistent with the nature of stateless Web services. For this reason, OASIS has adopted *Web Service Resource Framework (WSRF)*, a standard that allows Web services to access their persistent states in a consistent and interoperable manner. In this framework, a state is called *stateful resources*. WSRF aims to model and manage stateful resources based on a construct called WS-Resource, which is composed of a Web service and its associated stateful resources [8]. WSRF defines means by which:

- WS-Resources can be created and removed.
- A stateful resource is used when message exchanges of Web service are executed.
- A stateful resource can be queried and modified via message exchanges of Web service.

Each stateful resource usually has many independent instances that may be created and destroyed. When a new instance of a stateful resource is created, normally by a Web service referred to as a *resource factory*, it may be assigned an *identity* (also called *resource key*).

WSRF defines a special kind of relationship, called *implied resource pattern*, between a Web service and its stateful resources. This relationship is a mechanism to associate a stateful resource with execution of message exchanges of a Web service. The term *implied* means that the stateful resource associated with a given message exchange is considered as an implicit input for the execution of the message request. *Implicit input* means that the stateful resource is not provided as an explicit parameter in the body of the message request. Therefore, the association occurs mostly in a dynamic manner, which is at the time of the execution of the message exchange [8].

To represent the address of a Web service deployed at a given network endpoint, WSRF uses the *Endpoint Reference* construct from WS-Addressing. The main part of an endpoint reference is an *Endpoint Address* of the Web service. The endpoint reference may also contain a metadata associated with the Web service such as service description information and *reference properties* (this name is used in WSRF version 1.1. In version 1.2 it has been changed to *reference parameters*). The reference properties play an important role in the *implied resource pattern*, as it is used to keep the *resource key* of the instance of stateful resource.

Grid services are the Web services that follow WSRF standard. They are also called WSRF-compatible Web services.

B. Axis2

Axis2 is the new version of Axis (Apache eXtensible Interaction System), a SOAP engine and a Web Service middleware tool. It is a SOAP messaging system with modular architectural design. The Axis2 Framework is built up of 7 core modules. Non-core/other modules are layered on top of these core modules. Among the seven core modules, XML Processing and SOAP Processing modules are relevant to our research.

- **XML Processing Module**: Processing SOAP Messages is the most important and most complex task in Axis2, and its efficiency is the single most important factor that decides the performance. Axis2 makes use of AXIOM (AXis Object Model) to provide a simple API for improved SOAP and XML handling performance over Axis.
- **SOAP Processing Module**: This module controls the execution order of the processing. Besides defining different built-in phases of the execution, the model supports extensible capability by permitting users to extend the Processing Model at specific plug-in places. The SOAP Processing Model is shown in Fig. 1.

Two basic actions a SOAP processor are sending and receiving SOAP messages. To support these, the architecture provides two pipes (or messages processing flows) called ***In Pipe*** and ***Out Pipe***. The implementation of these two pipes is via definition of two methods, send() and receive() in the Axis2 Engine.

Extensible capability of the SOAP processing model is provided by handlers. When processing a SOAP message, only the handlers that are registered will be executed. Axis2 supports three scopes that the handlers can be registered in, global, service, or operation. The final handler chain will be calculated by combining the handlers from all the scopes.

Acting as interceptors, the handlers process parts of the SOAP message and provide add-on services. The different stages of the pipes are called *phase***,** which provides a mechanism to specify the ordering of handlers. A handler always runs inside a specific phase. Both Pipes have built-in phases, as well as the places for 'User Phases' which can be defined by users.

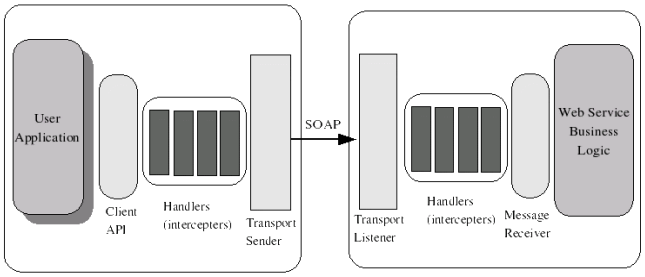

Fig. 1. SOAP processing model of Axis2.

C. ODE Engine

ODE is an open source BPEL engine of Apache. The latest stable version 1.2 offers many interesting features such as:

- ODE supports for both the WS-BPEL 2.0 OASIS standard and the BPEL4WS 1.1.
- ODE supports two communication layers: Web Services http transport of Axis2 and ServiceMix on the JBI standard.
- ODE can be easily integrated with virtually any communication layer thank to the high level API to the engine.
- ODE allows hot-deployment of processes. This means that one only needs to copy all the necessary files to a specific directory (a deployment directory regulated by the engine), and the running engine will automatically detect these files, compile them and prepare them ready for use.

The current ODE does not recognize all necessary information sent from a BPEL process and is not able to support the invocation of Grid services. The session 4 will

investigate this issue further and provide the basis of our solution.

III. Our Plan-Supported Grid Collaborative Framework

A. Objectives

The main objective of the architecture is to serve as a plan-supported Grid framework for a wide range of collaborative applications. The characteristic of plan supported of the framework can be explained in more details as follows.

Each collaborative work needs to have two related parts, a working plan and a work script. The working plan which corresponds to the activity level, consists of sequence of actions. Each action aims to achieve a goal among all goals of that plan/activity. The plan only takes care of what actions of work need to be done, and not of who will do those actions and how they can be done. In contrast, a work script needs to define clearly who will do what actions and how the actions can be implemented. Therefore, it composes of a sequence of operations and control structures. The operations are executable components such as programs, functions, services (Web and Grid services), etc.

Our framework aims to allow many users in concurrence to edit the working plans and scripts, as well as to run and monitor the status of the running scripts. During the process of editing a script, each user may try and select the best resources by his or her own experience, so that the script could be run most effectively. Fig. 2 shows the overview of our framework.

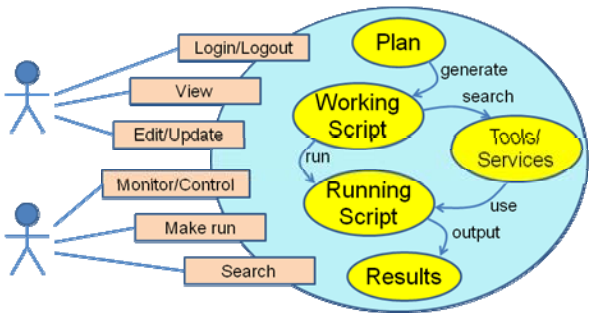

Fig. 2. Overview of the framework

B. Approaches

Activity Theory: There are two reasons for us to choose the theory as a theoretical foundation for our framework. Firstly, under the light of this theory, the role of plan of work and its relationship with the work itself can be understood more clearly. As stated in [9], "plans as socially constructed and used artifacts," this means that on one hand, a plan is the object of a collective activity. On the other hand, when the plan has been completed, it again becomes an artifact for achieving future work. Then after having finished, the work in turn may become a plan for the next work. This understanding is crucial for the development of plan-supported collaborative framework. Secondly, the activity theory implies and suggests a comprehensive set of collaborative tools of a general-purpose collaborative framework.

OGSA and grid infrastructure: recently, with the rapid development of both standardization and infrastructure, grid computing seems to become the most appropriate candidate for building geographically distributed and highly heterogeneous environments.

C. Architecture

Our framework composes of two layers:

Collective Activity Layer: this layer allows different users to collaborate in order to build working plans, edit work scripts, then run and monitor the edited work. Major components of this layer are described below (see Fig. 3):

(1) VO and Group Management: this component is responsible for updating of Virtual Organizations (VOs), groups in the VOs, users in the groups. It also needs to manage access rights and roles of the users in VOs.

(2) Activity Planning: this component is responsible for creating a new working plan or updating existing ones.

(3) Action Assigning: this component is responsible for assigning the action(s) in the working plan to each user.

(4) Selecting Resources and Artifacts: this component allows users to find and select suitable resources used by actions in the working plans as well as necessary collaborative artifacts for collaboration of the actions. The final result of the selection will be a work script

(5) Running and Monitoring: this component is responsible for launching, running, monitoring, and terminating activities.

(6) Resource Directory: this component contains resources needed for running operations of work.

(7) Collaborative Artifact Store: this store contains all collaborative artifacts.

Resource Coordination Layer: The main task of this layer is to manage distributed resources and make them ready for usage of the upper layer. The existing grid infrastructure (Globus Toolkit 4) will be used for this layer.

IV. A Solution for Invoking Grid Services within the ODE Engine

As mentioned above, the current ODE engine can only support the invocation of Web services and not Grid services.This problem needs to be investigated in detail, before providing a suitable solution. In [2], we developed a test determine how ODE engine invokes Web services and to locate exactly the problem when the ODE has to deal with invocation of an external grid service from a BPEL process. From understanding of the issue, we proposed a clean solution to solve it. To make it convenient for our readers, here we will describe briefly what have been done in our last research [2].

A. The Test

Our test comprises of two parts:

The main part is a BPEL process called Test-FactoryGS-V2. Diagram of the process is illustrated in

Figure 4. The process will be run by ODE engine which is deployed in Apache Tomcat 6.

The secondary part is a grid service called MathService, which has been adapted from the example in [10]. This grid service will be invoked from the above BPEL process. The service already follows the factory/instance pattern [11], which requires each grid service having its factory service called MathFactoryService, a normal Web service whose main role is to create instances of the Grid service. Besides creating new instances, this operation (called *createResource* in our example) also returns *ResourceKeys* for the instances. The Globus Toolkit version 4.2 has been used to deploy this grid service in its web service container.

In order to locate the problem, our BPEL process is designed to inspect three activities:

The first activity is to invoke the Grid service in order to get its *ResourceKey*. It invokes the operation *createResource* of *MathFactoryService* and stores the returned *ResourceKey* in a variable called *CreateResourceOut* by the following code:

```
<invoke name="Invoke1" partnerLink="MathFactoryServicePL"
operation="createResource"
xmlns:tns="http://www.globus.org/namespaces/examples/core/FactoryService"
      portType="tns:FactoryPortType"
   inputVariable="CreateResourceIn"
outputVariable="CreateResourceOut"/>
```

The second activity is to assign this *ResourceKey* (represented by a variable called *CreateResourceOut*) to a *PartnerLink*, allowing the *PartnerLink* to carry the *ResourceKey*. The code for this activity as follows:

```
<assign name="Assign5">
<copy>
 <from variable="CreateResourceOut" part = "response"/>
   <to partnerLink="MathServicePL"/>
</copy>
</assign>
```

The third activity is to use the *PartnerLink* to invoke another operation called *getValueRP* of the Grid service.

The assignment of *CreateResourceOut* to the PartnerLink *MathServicePL* was performed successfully. However, the invocation of the third activity failed. Through checking the in/out messages, the exact cause of the problem was found. In the message sent to invoke operation *getValueRP* of the grid service, the necessary *MathResourceKey* was not found.

It is clear from our investigation; two steps are required for getting *endpoint reference* (EPR). The first step is to invoke the operation *CreateResource* that returns the EPR from the Grid service, and then the returned EPR needs to be stored in a *PartnerLink* for later use. Surprisingly, even though the EPR contains the *ResourceKey*, the ODE still supports these two steps very well (please remember that in a Web service, its EPR contains no *ResourceKey,* and the ODE only supports Web services, not Grid services).

The issue of ODE in the Grid service invocation occurs exactly when the *PartnerLink* carrying the EPR is used to invoke operations in the Grid service. The reason is that the message used in this invocation lacks the *ResourceKey*.

B. The Solution

It is clear from our investigation that in order to enable the ODE engine to support Grid services invocation, we have to develop a mechanism to allow adding suitable ResourceKey into the message sent to invoke an operation of the Grid service. Because ODE engine is built on Axis2, which uses handlers (interceptors) to proceed messages, we need to develop new handlers whose roles are to find out necessary ResourceKeys and add them into the suitable messages. Furthermore, these handlers have to be positioned in proper positions in the series of existing handlers. In ODE, each handler is implemented by a Java class.

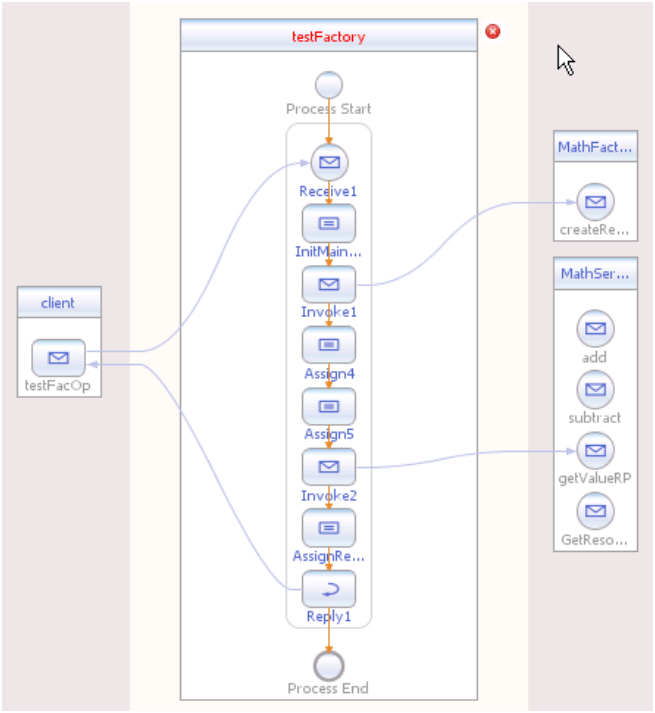

Fig. 3. Diagram of BPEL process Test-FactoryGS-V2

Inherited from Axis2, the ODE engine also uses the file axis2.xml to setup its configuration parameters such as TransportSender, TransportReceiver, MessageReceiver, Handlers and Handlers Processing Order, etc. Following is the phaseOrder part, which setups the Handlers Processing Order, extracted from configuration file axis2.xml of the ODE engine. In this extraction, the bold part shows our handlers that would be inserted after they have been implemented:

```
      <phaseOrder type="InFlow">
      …
      <!--  System pre defined phases      -->
      <!--  After Postdispatch phase module author or service author
can add any phase he want      -->
      <phase name="ProcessHeader">
         <handler name="SessionInHandler"

class="org.apache.ode.axis2.hooks.SessionInHandler">
```

```
            <order phase="PostDispatch"/>
        </handler>
        <handler name="ResourceKeyInHandler"
              class="our class here">
            <order phase="PostDispatch"/>
        </handler>

    </phase>
    <phase name="OperationInPhase"/>
    <phase name="soapmonitorPhase"/>
  </phaseOrder>

    <phaseOrder type="OutFlow">
    <!--    user can add his own phases to this area  -->
    <phase name="ProcessHeader">
      <handler name="SessionOutHandler"

class="org.apache.ode.axis2.hooks.SessionOutHandler">
            <order phase="PreDispatch"/>
        </handler>
        <handler name="ResourceKeyOutHandler"
              class="our class here">
            <order phase="PostDispatch"/>
        </handler>

    </phase>
    <phase name="soapmonitorPhase"/>
   …
  </phaseOrder>
```

C. Discussion on the Solution

We are not aware of any existing solutions that enable the invocation of Grid services with ODE engine.

We can only comment on solutions that use different BPEL engines.

i) Compared to the solution in [6] that extends the standard BPEL by adding new activities, our solution is much simpler on two points. First, with the solution that extends BPEL, the users will not only have to learn newly added activities, but also have to determine whether the service is a Grid service or a Web service when they invoke the service. With our solution invoking a Grid service is just like invoking a normal Web service. Second, the solution that extends BPEL requires many changes, from the module for compilation of BPEL process, to the runtime module for the compiled process. In contrast, our solution does not make any change to these modules. It only adds some independent handlers (each one is a Java class), and makes suitable changes only in the configuration file to register the new handlers.

ii) The solution proposed in [5] is similar to ours but for the ActiveBPEL engine, not for the ODE engine. Similar to the ODE, this BPEL engine has not supported Grid services invocation, but only Web services invocation. This solution, however, has not mentioned this issue in the ActiveBPEL engine. Only using tricks in a BPEL process and without overcoming the issue of the BPEL engine, it seems very hard to invoke successfully Grid services from the process.

In brief, our solution is the only one available so far for the ODE BPEL engine for invoking Grid services. It is simple, does not require modifications of the BPEL engine, and requires little effort from BPEL process developers.

V. A Solution for Automation of Deployment Stage of BPEL Processes in ODE Engine

A. The Existing Issue in Deployment Stage of BPEL Processes in ODE Engine

In order to deploy a BPEL process in ODE, a sequence of steps has to be performed: Firstly, Partner Links have to be created from the WSDL files and from the BPEL process itself. Except one partner link that presents entities that will use the BPEL process, other partner links represent external web services that will be invoked in the BPEL process. After that, a specific deployment file (*deploy.xml*) has to be created according to the ODE's structural specification of ODE. Lastly, all necessary files related to the ODE-BPEL process are copied to a specified directory to be compiled. If some errors occur, which is often the case, the log file of the Web server hosting the ODE has to be inspected manually for the error message and determine the reason for the error. Even if the compilation is successful, a runtime error can still exist and the inspecting and guessing process has to repeat again and again until successful.

It is clear from the above description that the pre-deployment process is complicated, time consuming and error prone. Even when one is familiar with the deployment of Web services, it is still an unnecessary complicated manual process that requires the understanding of BPEL, the intimate knowledge of WSDL structures and syntaxes, and the specific ODE's specific structure of *deploy.xml* files. It is also a tedious and time consuming process in manually creating the Partner Link component in various WDSL and *deploy.xml* files. The process is certainly prone to error. The most tedious part of the process is to find errors from the log file, which is extremely long and not well-structured as it contains all kinds of messages from the Web server, not just error messages relevant to one's BPEL deployment preparation process. This necessitates the development of an automated tool for the process.

B. Our Solution

In the paper [7], we present a tool that supports the deployment of BPEL processes in ODE. This tool will resolve the problems mentioned above by providing following capabilities:

- Automatically checking the validation of BPEL process file and related WSDL files. If they are all valid then a necessary *deploy.xml* file will be created.
- Automatically collecting all necessary files to deployment directory of ODE, and initiating the compilation process.
- Filtering and returning only relevant and useful information about the potential errors associated with the preparing and compilation process.
- Suggesting the possible solutions and reasons of the errors.
- Facilitating the integration with BPEL editors. In the next stage, the tool will be developed as an Eclipse plug-in, so that it can be easily plugged into our collaborative workflow editor that is currently being

developed in our research centre.

Our tool has been developed in form of a JAVA program. In order to run the tool, it requires two arguments, one input is the directory that contains all necessary files such as BPEL and WSDL files; and one output is a *deploy.xml* file if all necessary files on the input directory are valid. Otherwise, it will display an error that helps find the reason why the *deploy.xml* cannot be produced. In this tool, in order to read and write BPEL and WSDL files that are all XML files, we use DOM (Document Object Model) API of JAVA.

Our tool has been tested with several different projects of BPEL. With the projects that all files are valid, our tool also produces valid appropriate *deploy.xml* files that are ready for the deployment of these projects in deployment directory of ODE. In this case, our tool also supports two kinds of WSDL files:

- The normal WSDL file: with this type of files, the file contains all information about partner links, services and ports. Therefore, it is quite easy to get all this information by reading only one file.
- The wrap WSDL file: This type is an extension of a normal WSDL file. This file extracts the partner links from the appropriate normal WSDL file and wrap them in a new WSDL file. This wrap file has to import the normal file. Once the wrap WSDL file is created, the partner link information will be extracted, and then following the import link, the information about the service and port can also be extracted.

VI. Related Work

A. Existing Approaches in Invoking Grid Services from BPEL

A Grid service is a stateful web service: a Web service plus persistent *resource*s. Each resource has its own ID called *Resource Key*. The combination of grid service address and its resource defines an *endpoint reference* (EPR). Normally, the value of grid service address is static and known before its running time. However, the value of resource key is dynamic and this presents a problem for the ODE to deal with Grid services. Possible approaches for resolving this problem include:

- Extending BPEL by adding new activities [6] into the standard BPEL. Among these activities, one (called *gridCreateResourceInvoke*) is used to get EPR, and other one (called *GridInvoke*) is for calling operations of the Grid service.
- Adding an operation in the Grid service that needs to be invoked, as suggested in [5, 11]. This operation will return EPR of an instance of that Grid service (therefore the operation is normally called *CreateResource*).

Clearly, extending BPEL is not a simple task, as it requires numerous changes in the standard BPEL engine, from the compilation of BPEL processes to the implementation of new activities.

Our effort focuses on the adding the operation to Grid services as it potentially provides cleaner and better solutions.

B. Existing GCFs

1) The Patient Scheduler [12]

It is a prototype developed in the project SAIK whose objective is to investigate how network-based computers could improve cooperation and coordination of patient treatment. This objective shares some similarities with our project.

The development of PATIENT SCHEDULER aims at illustrating how the coordination and collaboration of healthcare work can be supported by computers. However, this product is only a prototype, and is only applied into the healthcare domain.

2) GridCole

GridCole [13] is a collaborative E-learning system that supports the realization of scripted learning situations which each of them consists of sequence of activities. In addition, with the desirable feature of tailorability, end-users of this system (educators and students) can integrate external tools into the learning situations. By using the grid services approach, this integration enables different kinds of tools; even those require supercomputer capabilities and specific hardware. In this system, IMS Learning Design (IMS-LD) specification has been used to describe learning situations. There are two kinds of external tools in GridCole, individual and collaborative. Among two of them, collaborative tools will be used to coordinate activities within each learning situation.

The description of the collaborative learning situations has been provided by means of a unit of learning which is according to IMS-LD specification. Two types of unit of learning can be used: complete and incomplete. Complete units of learning are those that contain all necessary information for integration of actual tools in the stage of realization of the learning situations. Otherwise incomplete units of learning do not have such information, but only a generic description of needed tools. Therefore, incomplete units of learning cannot be realized until they have been transformed to complete ones.

The main limitation of this system is not plan-supported. Even though that the incomplete units of learning seems to play the role of plans, but actually they are not independent plans.

3) Collaborative Design

The Collaborative Design Grid (CDG) [14] is a framework that aims to resolve two main problems in collaborative design: resource sharing and geographically distributed collaboration. The architecture of this framework bases mostly on OGSA, implemented Grid services on Globus Toolkit 3. This framework, however, has neither focused on supporting scripted work nor working plan.

4) Grid-enabled large scale

A framework called Grid-based Cooperative Framework [15] has been developed aiming to build Grid-enabled Large-scale Collaboration Environment. This environment aims to support users to create large-scale and real/natural collaborations with some main features:

- Large scale collaboration (deeper and wider collaboration, hierarchical structures).

- Various cooperative modes (syn/asyn, intra-group/inter-group).
- Various coordination mechanisms (explicit/implicit/improvise).
- Integration of several coordination mechanisms into a single one

Even though this framework aims to develop large-scale collaboration environment, but it has not supported scripted learning situations which play an important role as working plans for learning processes. Without these plans, it is very difficult to manage the sequence of activities in learning processes, and this may lead to ineffective and uncontrollable learning processes.

5) Open Collaborative Grid Service Architecture (OCGSA) [16]

This architecture aims to provide a common framework for collaborative applications. In this architecture, the Grid service concept in OGSA (as low level service) is extended to Collaborative Grid service (high level service), by the extension of Grid service portType with metadata for group management and security. In parallel, the notification mechanism is also extended with the ability of predefinition of notification topics. Another new component in OCGSA compared to OGSA is the Event Archiving service that is responsible for managing the logs/messages exchanged between users/groups. However, this architecture only offers a basic level that does not include adequate concrete mechanisms for supporting realistic collaboration. This makes it very hard to be applied in development of real grid collaborative frameworks or applications.

VII. Conclusions

The contribution of the paper is summarized as follows.

Firstly, this paper connects main ideas of the Activity Theory to Grid by proposing a Plan-supported Collaborative Grid Framework. This framework allows three levels of collaboration from coordinating activity, cooperating activity to coconstructive activity. Interactions between these levels and components of the framework allow collaborative plans to be created and dynamically modified; objectives to be shared and co-optimised; and actions to be distributed and optimally executed by participants. The aim is to provide a generic Grid Framework for supporting collaborative work applicable to a wide range of application domains. The significance of such a solution is that it allows various Grid services to be composed into structured Grid/Workflows using BPEL and to be invoked through the Grid infrastructure underneath. This implies complex sequences of tasks can be composed and automated over the Grid computing environment rather than simple Grid requests.

Secondly, we thoroughly investigated the problem with the ODE BPEL engine to pin point the reason for it not being able to invoke stateful Grid services. It offered a simple and practical solution that is not known to exist.

Lastly, the issue of manual preparation of deployment stage of BPEL processes by the ODE engine has also been solved completely by our tool. This simplified greatly this stage and allowed the BPEL process creators focusing only on the process creation step, not on following deployment step any more.

Our next step is to implement and deploy the solution for Gridflows applications.

References

[1] Binh Thanh Nguyen and D. B. Hoang, "Building a Plan-Supported Grid Collaborative Framework" in *2nd International Cenference on Communications and Electronics (ICCE 2008)* Golden Sand Resort, Hoi An City, Vietnam, 2008.

[2] Binh Thanh Nguyen, Doan B. Hoang, and T. T. Nguyen, "Enabling Grid Services from BPEL process using ODE engine," in *IEEE ICCSIT 2011* Chengdu, China, 2011.

[3] C. K. Ian Foster , Steven Tuecke, "The Anatomy of the Grid," *International Journal of High Performance Computing Applications,* 2001.

[4] M. A. B. David De Roure , Nicholas R. Jennings , Nigel R. Shadbolt, "The evolution of the Grid," *John Wiley & Sons,* 2003.

[5] Onyeka Ezenwoye , S. Masoud Sadjadi , Ariel Cary , and M. Robinson, "Grid Service Composition in BPEL for Scientific Applications," *Spinger-Verlag,* 2007.

[6] T. Dornemann , Thomas Friese , S. Herdt , and B. Freisleben, "Grid Workflow Modelling Using Grid-Specific BPEL Extensions," *German e-Science,* 2007.

[7] Binh Thanh Nguyen and D. B. Hoang, "An automatic tool for deployment of BPEL processes in ODE Apache," in *EEE'09 - The 2009 International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government* Monte Carlo Resort, Las Vegas, Nevada, USA 2009.

[8] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, and S. Weerawarana, "Modeling Stateful Resources with Web Services version 1.1," 2004.

[9] J. E. Bardram, "Plans as situated actions: An activity theory approach to workflow systems," in *Proceedings of the 5th European Conference on Computer Supported Collaborative Work*, Lancaster, UK, 1997, pp. 17–32.

[10] B. Sotomayor, "The Globus Toolkit 4 Programmer's Tutorial."

[11] Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Cary, and M. Robinson, "Orchestrating WSRF-based Grid Services," School of Computing and Information Sciences Florida International University 2007.

[12] J. E. Bardram, "Collaboration, Coordination and Computer Support - An Activity Theoretical Approach to the Design of CSCW," in *Aarhus*, 1998.

[13] E. G.-S. Miguel L. Bote-Lorenzo, Guillermo Vega-Gorgojo , Yannis A. Dimitriadis, Juan I. Asensio-Perez, Ivan M. Jorrin-Abellan, "Gridcole: A tailorable grid service based system that supports scripted collaborative learning," *Computers and Education,* 2007.

[14] X. J. Zhi Li , Yuan Cao , Xiaoyun Zhang , Yuanyin Li, "Architecture of collaborative design grid and its application based on LAN," *Advances in Engineering Software,* 2007.

[15] S. Y. Y. Li , Jinlie Jiang , Meilin Shi, "Build Grid-enalbled large-scale Collaboration Environment in e-Learning Grid," *ScienceDirect,* 2006.

[16] G. v. L. K. Amin , S. Nijsure, "Open Collaborative Grid Service Architecture," *Euroweb,* 2002.

Msc Binh T. Nguyen, a lecturer at the school of Electronics and Telecommunication of the Hanoi University of Science and Technology, received his Master in computer science at the Francophonie Institute for Information (IFI).

His research interests include Grid Computing, Cloud Computing, Collaborative Systems and Workflow Languages.

Huu-Duc Nguyen, a lecturer at the school of Information Technology anf Communication of the Hanoi University of Science and Technology, received his Ph.D. in computer science from the Japan Advanced Institute of Science and Technology.

His primary research interest is in the area of foundation for programming languages, specifically

design and implementation of parallel programming languages for multicore/manicore architechtures. He also has interests in problems relating to high performance computing, grid computing and cloud computing.

Doan B. Hoang is a Professor in the School of Computing and Communications, Faculty of Engineering and Information Technology, the University of Technology, Sydney (UTS). He is a Director of iNEXT - UTS Centre for Innovation in IT Services and Applications, a research centre at the University of Technology, Sydney for developing and nurturing innovation for the NEXT generation IT services and applications, including Internet-enabled business applications, mobile health services, high-end visualization technologies, novel image processing architectures, and advanced video surveillance systems.

His research interests include Next Generation Networks (Security, Quality of Service, Mobility, Service-Oriented Architecture, Peer-to-Peer), Broadband Service Architecture, Collaborative Grid and Cloud Computing, Wireless Sensor Networks and e-Health. He is currently leading research into establishing an innovation culture, reducing the cost of healthcare system through advanced technologies and assistive health Grid/Cloud, and generating wealth through innovative use of the Broadband Internet.