Engineering Process Coordination based on A Service Event Notification Model

Jian Cao[1], Jie Wang[2], Shensheng Zhang[1], Minglu Li[1], Kincho Law[2]

[1] Department of Computer Science, Shanghai Jiaotong University,
200030, Shanghai, P. R. China
{cao-jian, sszhang, li-ml}@cs.sjtu.edu.cn
[2] Department of Civil and Environment Engineering, Stanford University,
Stanford, CA 94305, U.S.A
{jiewang, law}@stanford.edu

Abstract. Due to the complexity and uncertainties, the engineering process requires dynamic collaborations among the heterogeneous systems and human interactions. In this paper, we propose a service event notification model based on grid service to coordinate different applications and human. The model takes advantage of the grid infrastructure and reduces the need for ad hoc development of middleware for supporting process coordination. In this model, an event notification server composed of a group of grid services can capture events from other grid services and generate process events. When an event occurs, event notification server decides to whom it should send the event according to an awareness model that keeps the states of the underlying coordination policies and business rules. The awareness model and the methodology for building an event notification system, together with the infrastructure of the notification server are presented in the paper. An example in applying the model to an engineering project coordination scenario is presented to illustrate the potential application of the event notification model.

1 Introduction

For complex and knowledge intensive projects such as engineering/construction and design/manufacturing, multiple participants residing in different locations often need to work together throughout the lifecycle of the project. The dynamic nature of project requirements and the inevitable collaboration among multiple organizations and participants pose many challenges from both technological and management perspectives.

The first challenge is the diverse heterogeneous software and hardware environments that are used in the engineering processes. A fully integration solution imposing on homogeneous software and hardware platforms is infeasible. We need an alter-

native approach that can coordinate heterogeneous applications during the project lifecycle process.

Grid-based engineering service is a potentially useful technology for process coordination. Grid concepts and technologies were first developed to enable resource sharing and to support far-flung scientific collaborations [1]. The Open Grid Services Architecture (OGSA) [2] integrates key Grid technologies with Web services mechanisms [3] to create a distributed system framework.

Another challenge is the coordination of the participants in an engineering project. Because engineering process is often highly dynamic in nature, it is difficult to lay out an exact plan detailing all the necessary tasks, their interdependency and interactions.

To address these two challenges, a platform that can support both system coordination and human coordination is important. Current grid service technology itself does not support human coordination. However, it does provide a notification mechanism to publish the status of changes in events, which can be forwarded to interested parties to support human coordination. Thus we propose a Grid service based event notification model to support engineering process.

This paper is organized as follows. Section 2 defines an awareness model for engineering process. Section 3 discusses how to capture and transform events based on business requirements for a complex process. Section 4 introduces the event notification mechanism. In Section 5, the structure of an event notification server supporting engineering process coordination is proposed. In Section 6 we provide an example in building design and construction management to demonstrate the grid-based event notification approach described in the paper. Section 7 discusses related works and Section 8 concludes the research and points out some future works.

2 An Awareness Model for Engineering Process

Dourish and Bellotti [4] define awareness as "an understanding of the activities of others, which provides a context for your own activity." There are many types of awareness information that can be provided to a user about other users' activities [5]. We focus and categorize the awareness information of engineering process into two main types: (1) Awareness Information related to Project Artifact Sharing, and, (2) Awareness Information related to Process Logic

We propose to build the awareness model based on the technique of integrated cooperative process modeling.

2.1 Artifact Structure Model

The artifact itself can be used as a "shared representation" through which people can communicate with one another and coordinate their activities.

An engineering artifact can be represented by a hierarchical tree, which is called an artifact tree. An artifact tree is a triple $<C, r, R>$, where C is a finite set of components,

$r \in C$ is the root component, $R \subseteq C \times C$ such that $(c_1, c_2) \in R$ if $c_1 \in sub(c_2)$, where *sub* denotes the "sub-components of " relationship)

An artifact type can be defined as $<P, A>$, where P is a set of parameters that can characterize this artifact, A is a set of operations to manipulate this artifact. The artifacts produced during an engineering process are not independent, and are often interdependent. That is, if an artifact a depends on another artifact b, whenever b is changed, a must at least be checked to ensure consistency and if necessary, need to be changed accordingly. Moreover, the dependency is transitive.

To model an artifact structure, two relationship types are defined among the components: they are the *sub* relationship and, the *depend* relationship, as shown in Figure 1.

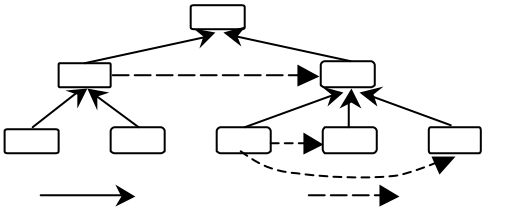


Fig.1 Artifact Structure Model

2.2 Process Structure Model

To model a dynamic engineering process, we partition the process model into two layers. The top layer is a project plan model and the bottom layer represents a library of workflow models with a set of tasks.

A project is denoted as $P=<T_p, R_p, SP, R_{dp}>$, where T_p is the anticipated time schedule of the project, $R_p$is the organizational property that will be defined in section 2.3, SP is a set of activities, R_{dp} are ordering relationships among activities which are defined according to a general project model such as CPM (Critical Path Method).

An activity can be complex or simple. The definition of a complex activity is similar to a project. A simple activity is denoted as $a_p=<T_a, R_a, TA>$, where T_a is the time scope defined for the activity, R_a is the organizational property. $TA=\{ta_{p1}, ta_{p2}, \ldots, ta_{ph}\}$, in which ta_{pi} is a task of activity a_p and a_p is also called the parent of ta_{pi} (denoted as $a_p = \uparrow(ta_{pi})$). A task can be expressed as $ta=<a_t, c_t, I, O, R_t>$, where a_t is an operation (defined for artifact type of c_t), c_t is an artifact, I is the input artifact set of this task, O is the output set of this task and R_t is the organization property of this task. There are no ordering relationships defined among the tasks of an activity.

In an artifact structure model, for each operation defined for an artifact type, we should specify it's content as:

(1) applications or services that are possibly conducted by a person,
(2) invoking a service by the system automatically, or
(3) workflow models that can fulfill the operation.

The process structure model is shown in Figure 2. A project model consists of activities and their relationships and it provides high level ordering constraints for the entire engineering process. While at the lower level description, tasks are fulfilled by invoking applications and services by a person or by a structured workflow.

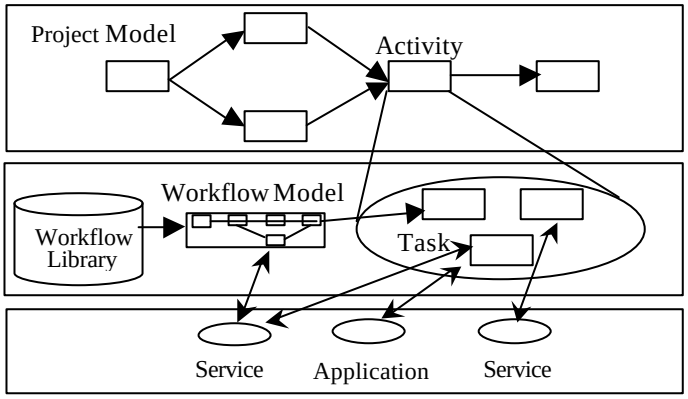


Fig.2 Process Structure Model

2.3 Organizational Structure Model

A project, activity or task is usually allocated to an organizational unit for execution. Organizational structure reflects the project process structure and management structure.

Each participant can be assigned several roles. Team *te*=<*TR, TM, TF*> is defined as a triple, in which *TR* represents a role set, *TM* is an actor set, and $TF \subseteq TR \times TM$ represents the enabled roles of the members *TM* in team *te*.

2.4 Resource Model

The resource managed by a notification server includes applications and services. Applications and services are registered in the notification server. For a public service in an engineering process, its address and the embedded methods should be registered so that other applications can find this service.

3 Event Capturing and Transforming

Notification mechanism has been defined in OGSA [2]. An important aspect of this notification model is the tight integration with service data: a subscription operation is a request for subsequent "push" delivery of service data.

In addition to capture the notification of grid service, we should capture the context of the notification, i.e., project name, the operations and the artifact affected. A service can specify the context information by adding a special service data type as follows:

<complexType name="*EventDataType*"><sequence>
<element name="*ProjectName*" type="string"/>

<element name="*ArtifactName*" type="string"/>
<element name="*OperationName*" type="string"/>
</sequence></complexType>

When a method of this service is invoked, the values of the related service data elements (EventData) should be set and pushed to the notification sink.

Activities and tasks have different states. When an activity changes from one state to another, events will be triggered. These events are generated by a process engine and distributed by the notification server.

The events captured can be transformed into other events according to the business requirements. We provide a transformation rule in the following form:

On *Event Expression* **If** *Condition* **Then** *RaiseEvent* (*e*)

An *event expression* is composed of a set of event filters. A *Condition* is a conjunction of the constraints, which define the relationships among parameters of different event types. Action *RaiseEvent* will produce a new event.

4 Event Notification

After an event has been captured, the concerned individuals or applications should be notified to deal with the event. We assume that each event is related to at least an artifact or an activity, i.e., in the definitions of an event type, there is an attribute that indicates at least an artifact or an activity to which this event targets upon. If the event is related to an artifact c_0, then we can find other related artifacts through the dependency relationships among them. Suppose they compose a set C. To receive the necessary notifications for application services of a task, they should be registered at the notification server regarding the locations where these applications reside. Events related to the artifact c_0 will be broadcasted to these applications if their related artifacts are in C.

In order to notify the individuals, who are the task owner of particular tasks, when events related to artifacts happen, we should find the tasks directly related to these artifacts and notify the task owners. Because we have defined the input for each task, it is quite straightforward to find those tasks that are waiting for the events.

5 System Structure of Notification Server

The system architecture based on grid service platform for a notification server is shown in Figure 3. A user joins an engineering cooperative process through a personal workspace. In the personal workspace, a user can invoke different applications. These applications in turn can invoke services that are running in different grid service containers. A notification server consists of a set of grid services. An event receiver service is running to gather events from distributed services and it will store all the events gathered and recorded the events into an event history. Once an application starts running, it will create an application broker within the notification server. An

application broker monitors the event history and determines whether the server should notify the application based on the awareness model. Similarly, a personal broker is created by each personal workspace. A personal broker will also monitor the event history and notify the personal workspace based on the awareness model. As for each project, a project service instance will be created. The project service instance provides methods to be invoked by the personal workspaces, generates the events and coordinates the tasks according to the process logics. A set of facility services is also provided by a notification server. These services include modeling service and management service. A user can revise the project model, initiate a project and manage the project process through these facility services.

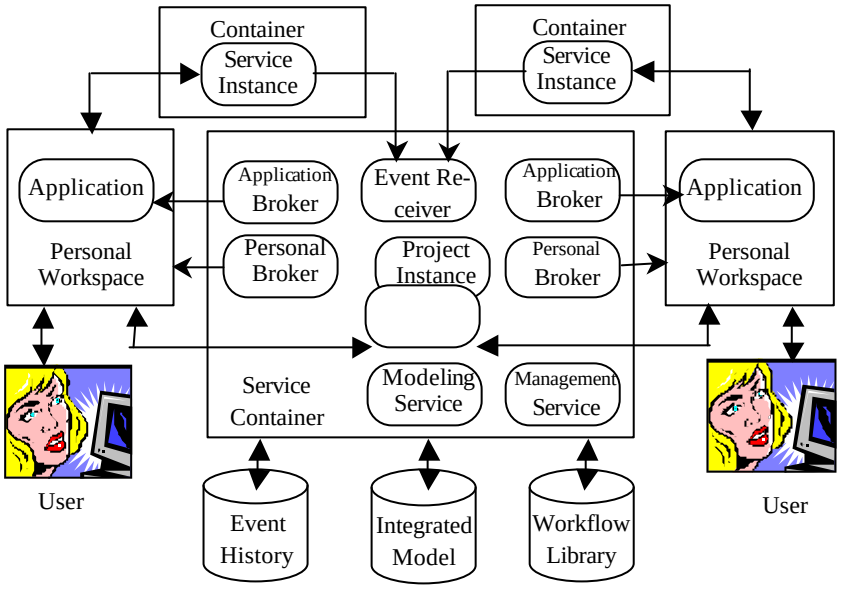


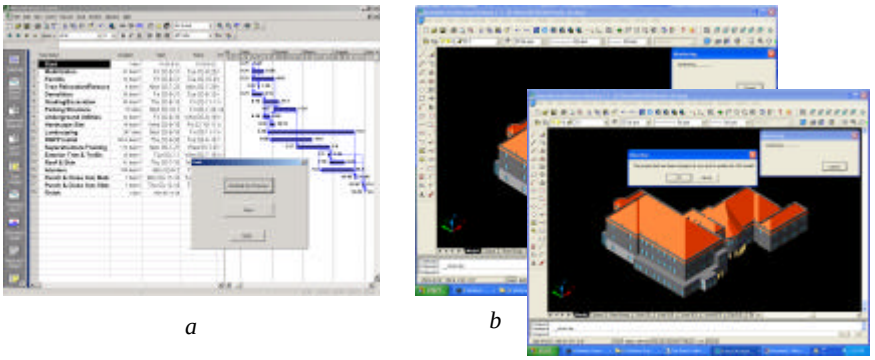
Fig.3 Notification Server Structure based on Grid Service Platform

6 A Case Study

We have built a simple demonstration to illustrate the engineering service notification model. This case example demonstrates a grid based coordination system for facilitating a building design process. The project manager can define or update process plan through MS Project software. The process plan can be checked in to a grid repository service (Figure 4 a). The repository service will generate an event to the notification server. Notification server sends this event to AutoCAD software, which can be used to design and display different phases of the facility (Figure 4 b, c).

The distributed coordination framework for this example demonstration is developed based on Globus Toolkit 3.0. Specific gateways are developed to connect the

software applications into a grid service. For example, we developed gateway plug-ins for MS Project and AutoCAD using Microsoft's .NET framework.



a *b* *c*

Fig.4 A Case Study

7 Related Works

There are many event notification servers [6]. Three representative examples are CASSIUS, Elvin and Siena [7]. CASSIUS was tailored to provide awareness information for groups. Elvin was originally developed to gather events for visualizing distributed systems, but it evolved later into a multi-purpose event notification system. Siena emphasized on event notification scalability in distributed environments. Our notification server model aims to coordinate and support engineering process in which an awareness model is designed with built-in knowledge. Another characteristic of our model is that the notification server is based on service computing paradigm that is not investigated by other approaches.

A product awareness model, Gossip, was intended to support the development process [8]. Gossip included a shared composite product model with rules of awareness information. Product object and awareness relation are registered in Gossip. Our artifact model is similar except our awareness model also includes process sub-model, organizational sub-model and resource sub-model.

Recently, a white paper describing a method based on a notification mechanism of a web service has been proposed [9]. The NotificationBroker in that model is conceptually quite similar to our notification server. However, no detailed descriptions on their model, nor any implementations guidelines, are provided.

8 Conclusions and Future Work

This paper proposes a solution based on an event notification model of grid services. Current grid service technology only emphasizes on cooperative computation. The proposed model extends the grid service to support cooperative work of individual

human participants during a complex engineering process. Based on the awareness model, a notification server can not only broadcast events to the interested parties to support cooperative work, but also support process control that is important for an engineering process. Our future work includes developing a notification server to support more expressive event transformation rules in a dynamic engineering process.

Acknowledgement

This work was performed while the first author was visiting Stanford University. The authors would like to thank Mr. Jim Cheng for providing the data and the building model.

References

1. Foster, I., Kesselman, C., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann (1999)
2. Foster, I., Kesselman, C., Nick, J., Tuecke, S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, http://www.globus.org/research /papers/ogsa.pdf, (2002)
3. Graham, S., Simeonov, S., Boubez, T., et. al. , Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, SAMS (2001)
4. Dourish, P., Bellotti, V., Awareness and Coordination in Shared Workspaces, Conference Proceedings of Conference on Computer Supported Cooperative Work, Toronto, Ontario, Canada, (1992) 107-114
5. Steinfield, C., Jang, C. Y., Pfaff, B., Supporting Virtual Team Collaboration: The TeamSCOPE System. Proceedings of GROUP Conference, Stockholm, Sweden, (1999) 81-90
6. Cugola, G., Nitto, E., Fuggetta, A., The JEDI Event Based Infrastructure and Its Application to the Development of the OPSS WFMS. IEEE Transactions on Software Engineering, Vol. 27(9) (2001) 827–850
7. Cleidson, R. B., Souza, D., Santhoshi, D., et. al., Supporting Global Software Development with Event Notification Servers, Proceedings of the ICSE 2002, http://citeseer.ist.psu.edu /desouza02 supporting.html, (2002)
8. Farshchia B. A., Gossip: An Awareness Engine for Increasing Product Awareness in Distributed Development Projects, http:// www.idi.ntnu.no/~ice/publications/caise00.pdf, (2000)
9. IBM, Publish-Subscribe Notification for Web services, http://www-106.ibm.com/developerworks/ library/ws-pubsub/ WS-PubSub.pdf, (2004)