Distributed Design Process Coordination based on a Service Event Notification Model

Jian Cao, Shensheng Zhang, Minglu Li, Jie Wang

CONCURRENT ENGINEERING: Research and Applications

Distributed Design Process Coordination based on a Service Event Notification Model

Jian Cao,[1,*] Shensheng Zhang,[1] Minglu Li[1] and Jie Wang[2]

[1]*Department of Computer Science & Technology, School of Electronic, Information and Electrical Engineering, Shanghai Jiaotong University, 200240, 800 Dongchuan Road, Min Hang, Shanghai, P.R. China*
[2]*Department of Civil and Environment Engineering, Stanford University, Stanford, CA 94305, USA*

Abstract: Due to the complexity and uncertainties, the distributed design process requires dynamic collaborations among the heterogeneous systems and human interactions. In order to coordinate different system applications and humans, an event notification model is required. In this article, we propose a service event notification model based on grid service. 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 article. An example in applying the model to a vehicle design project coordination scenario is presented to illustrate the potential application of the event notification model.

Key Words: distributed design process, grid, awareness model, notification model.

1. Introduction

For distributed design projects such as product development [1] and building design [2], 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 distributed design processes. A fully integrated solution imposing on homogeneous software and hardware platforms is not feasible. An alternative approach is needed 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 [3]. The open grid services architecture (OGSA) [4] integrates key grid technologies with web services mechanisms [5] to create a distributed system framework.

Another challenge is the coordination of the participants in a distributed design project. Because design 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, the authors propose a grid service based event notification model to support distributed design process.

This article is organized as follows. Section 2 defines an awareness model for design 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 distributed design process coordination is proposed. In Section 6, an example in product design is provided to demonstrate the grid-based event notification approach described in the article. Section 7 discusses related works and

*Author to whom correspondence should be addressed.
E-mail: cao-jian@cs.sjtu.edu.cn
Figures 4 and 5 appear in color online: http://cer.sagepub.com

1063-293X/05/04 0301–10 $10.00/0 DOI: 10.1177/1063293X05060134

Section 8 concludes the research and points out some future works.

2. An Awareness Model for Design Process

Dourish and Bellotti [6] 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 [7]. The authors focus and categorize the awareness information of distributed design process into two main types:

1. awareness information related to project artifact sharing and
2. awareness information related to process logic.

The authors propose herein an awareness model based on the technique of integrated cooperative process modeling, which makes the implementation of process logics and a rule based notification-awareness system for process coordination.

2.1 Artifact Structure Model

A design process, designed to produce an artifact, typically consists of several components and each component may have many sub-parts. In other words, an engineering artifact can be represented by a hierarchical tree, 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. The connections between the components C form a tree. The end-product is the root component r.

In a distributed design process, different participants have interests in different parts of the artifact tree. For example, a project manager may organize several teams for the whole project and monitor the progress according to the upper portion of the artifact tree. The team leader assigns participants to be responsible for specific components and monitors the progress of sub-processes corresponding to a certain portion of the artifact tree. In other words, each project participant is only concerned with a partial artifact tree.

An artifact type can be defined as $<P, A>$, where P is a set of parameters that can characterize this artifact, and A is a set of operations to manipulate this artifact. The artifacts produced during a distributed design process are not independent, and are often inter-dependent. 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: if a depends on b, and b depends on c, then, a depends on c. In this case, we call a directly depends on b and a indirectly depends on c.

The degrees of dependency among the artifacts are different. They can be strongly dependent or weakly dependent. If artifact a is produced based on information of artifacts $b_1, b_2, \ldots, b_m$, then the degree of dependency of a on b_i is defined by w_i $(1 \geq w_i > 0)$, where w_i is a normalized weighted measure. A direct dependent relationship is denoted as *Direct_Depend* (a, b, w), where a is directly dependent on b with degree w. For example, if a direct depends on b with weight w_1 and b directly depends on c with weight w_2, then a is indirectly dependent on c with weight $w_1 * w_2$.

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.

2.2 Process Structure Model

To model a dynamic distributed design 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. $TA = \{ta_{p1}, ta_{p2}, \ldots, ta_{pi}\}$, 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

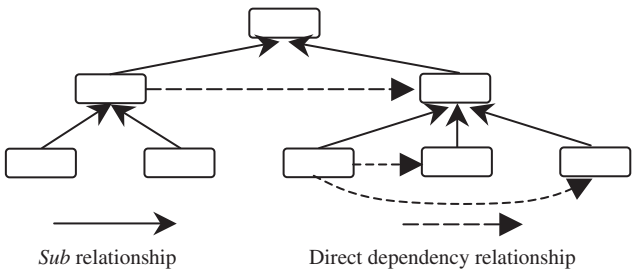


Figure 1. Artifact structure model.

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, the content should be specified 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 design process. While at the lower level description, tasks are fulfilled by invoking applications and services by a person or by a structured workflow. The time to invoke an application or a service is not pre-determined. New tasks can be added to an activity at runtime. To coordinate these tasks, team members should be aware of each other's work.

2.3 Organizational Structure Model

A project, an activity, or a 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 based on the roles' hierarchies. If a participant plays a superior role, it is assumed that this participant can do any task requiring inferior 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. At least, TR should include the role 'manager'. In the definition of a team, each role does not need to be bound with an actor. Actors can be assigned at run time. An activity or task can be assigned to a specific participant, a role, or a team.

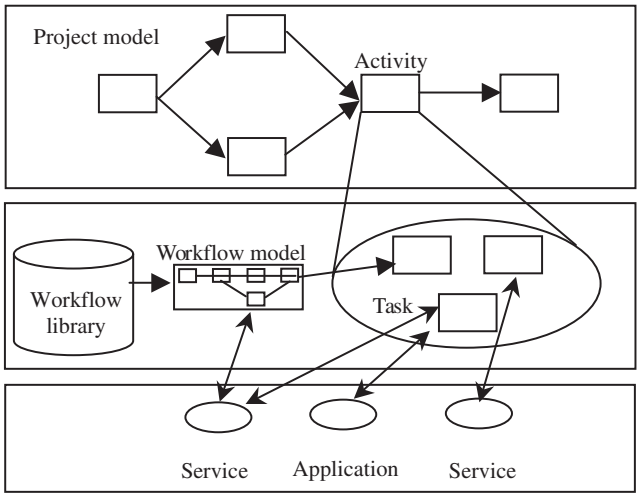


Figure 2. Process structure model.

2.4 Resource Model

The resource managed by a notification server includes applications and services. Application is a program that has an interface that a user can interact with. Service (in our example, service means grid service) always runs in the background and it provides a set of methods that can be invoked on the Internet. Applications and services are registered in the notification server. For a public service in a distributed design process, its address and embedded methods should be registered so that other applications can find this service.

3. Event Capturing and Transforming

3.1 Service Event Capturing

Notification mechanism has been defined in OGSA [4]. 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 capturing the notification of grid service, the context of the notification should also be captured, 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. Thus, the service event to be captured can be denoted as:

$$e_s = <ProjectName, ServiceName, ArtifactName, OperationName, TimeStamp> .$$

3.2 Process Events Capturing

Activities and tasks have different states, including *waiting* (W), *ready* (R), *executing* (E), *completed* (C), *overtime* (T), and *aborted* (A). When an activity changes from one state to another, events will be triggered, as illustrated in Figure 3. These events are generated by a process engine and distributed by the notification server. If the state of an activity or task is '*waiting*', then they become inactive. If their states are '*ready*' or '*executing*', then they are activated.

Project participants can access activity or task information through personal workspace. When an event indicating that a task's status has been changed from '*waiting*' to '*ready*' is received, the participants can accept the task assigned. He can change the state of the task by issuing a set of operations, which will in turn generate new activity events.

The process engine will monitor the whole process and change the state of the task and activity. Changing states may bring a set of process events captured as follows:

$$e_a = < ProjectName,\ ActivityName\ (or\ task\ name),\ FromState,\ CurrentState,\ TimeStamp >$$

3.3 Event Transforming

The events captured can be transformed into other events according to the business requirements. The authors 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. Event filters are connected using the composition operators, such as:

1. AND: e_1 AND e_2 means e_1 and e_2 both have happened.
2. OR: e_1 OR e_2 means at least one of e_1 and e_2 has happened.

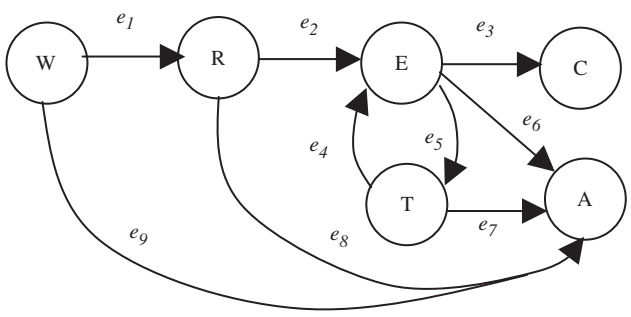


Figure 3. State diagram of an activity or a task.

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. For example, a new event type called operational event type indicating that an artifact is changed by an operation is denoted as:

$$e_a = < ProjectName,\ ArtifactName,\ OperationName,\ TimeStamp >$$

During a distributed design process, one may also have service event that requires services from other participants or systems. Since the authors have defined the relationships of operations and service method, a service event can be transformed into an operational event. As an example, the artifact type '*project plan*' may have operations '*build new plan*', '*update plan*', which are mapped into a method '*check in*' of service '*Plan Repository Service*'. As an example, if an event is captured: $e_s = <$'*ATV Project*', '*Plan Repository Service*', '*Project Plan*', '*Check in*', '2005-3-15 15:30'$>$, this event can be transformed into: $e_o = <$'*ATV Project*', '*Project Plan*', '*create*', '2005-3-15 15:30'$>$. This statement can be written as a transformation rule:

On $e_s < OperationName =''\ Check\ in'' >$
Then $RaiseEvent(e_o(e_s.ProjectName, e_s.ArtifactName, ''Create'', e_s.TimeStamp)$

If one receives another event: $e_s = <$'*ATV Project*', '*Plan Repository Service*', '*Project Plan*', '*Check in*', '2004-4-13 12:20'$>$, the event should be transformed into: $e_o' = <$'*ATV Project*', '*Project Plan*', '*update*', '2004-4-13 12:20'$>$. The transformation rule can be written as:

On $e_s < OperationName =''\ Check\ in'' >$ AND
$e_o(OperationName =' Create')$
If $e_s.ProjectName = e_o.ProjectName$ &
$e_s.ArtifactName = e_o.ArtifactName$
Then $RaiseEvent(e_o(e_s.ProjectName, e_s.ArtifactName, ''Update'', e_s.TimeStamp))$

4. Event Notification

After an event has been captured, the concerned individuals or applications should be notified to deal with the event. One can 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. Furthermore, when an event relates to a composite artifact or complex activity, some dedicated analysis tools can be provided to determine which part is actually changed. For example, when the project plan is changed, an analysis tool is provided to compare the original plan and current plan to determine which part has been modified.

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, for each $c_i \in C$, we can calculate the dependency weight among c_0 and c_i. Since a task includes a set of application services, activating a task also activates a set of application services and their related artifacts. To receive the necessary notifications for these application services, 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 owners of particular tasks, when events related to artifacts happen, one should find the tasks directly related to these artifacts and notify the task owners. Because the input for each task has been defined, it is quite straightforward to find those tasks that are waiting for the events. It should be emphasized that only the activated tasks need to be notified.

All events will be stored as an event history. When an event related to an artifact happens, notifications (with a dependency weight) to the tasks are also recorded. Notification server will keep track on the last time that applications or users are online. When applications or users connect to the server, those events whose timestamps are more recent than their last logon timestamps will be sent to the connecting applications or users. In order to prevent the events being sent to the individuals not related, one can also define event filters in the personal workspace. For example, for an event related to an artifact, one can assign a threshold on the dependency weight of the notification. Then the event notification related to an artifact whose dependency weight is less than the threshold value will not be sent to the workspace.

5. System Structure of Notification Server

The system architecture based on grid service platform for a notification server is shown in Figure 4. 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, record the events into an event history and determine how to dispatch events to the applications and humans. 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.

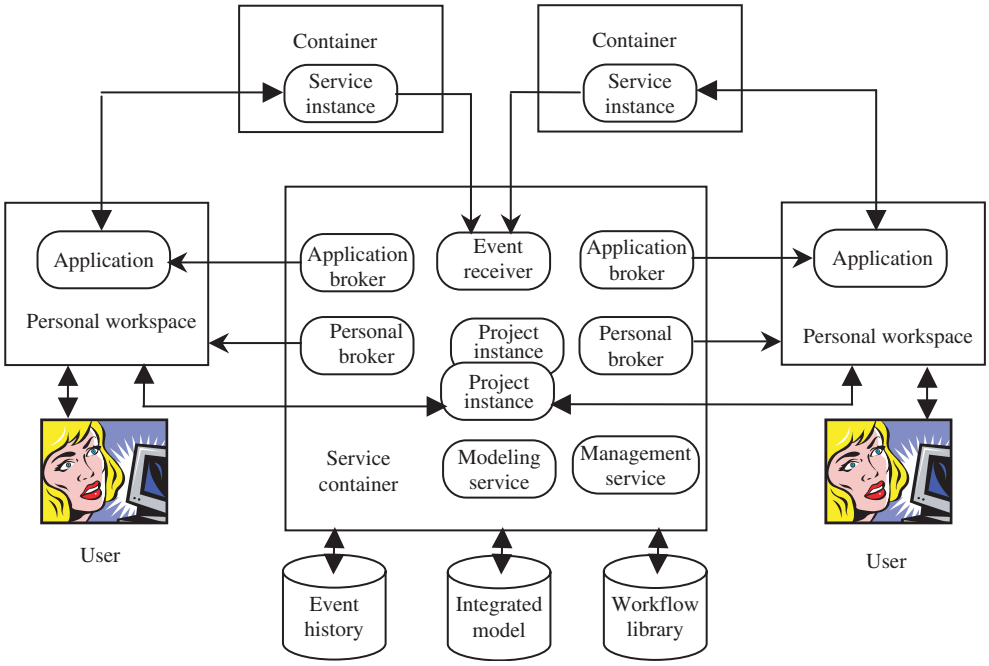
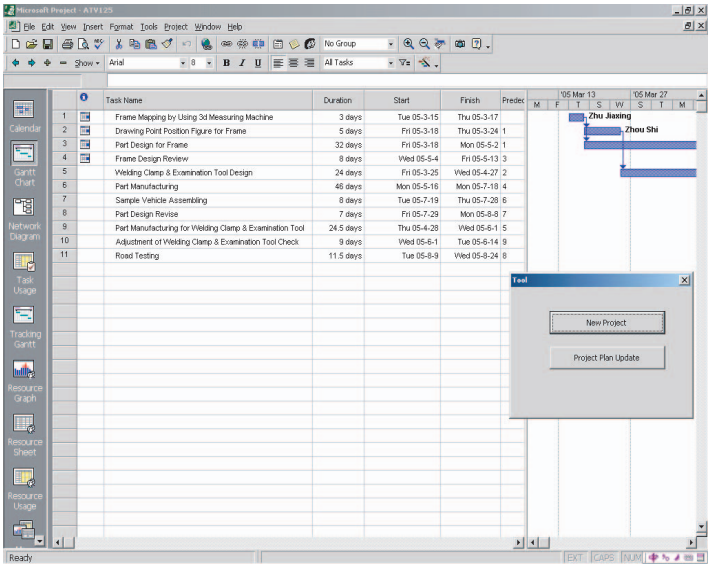
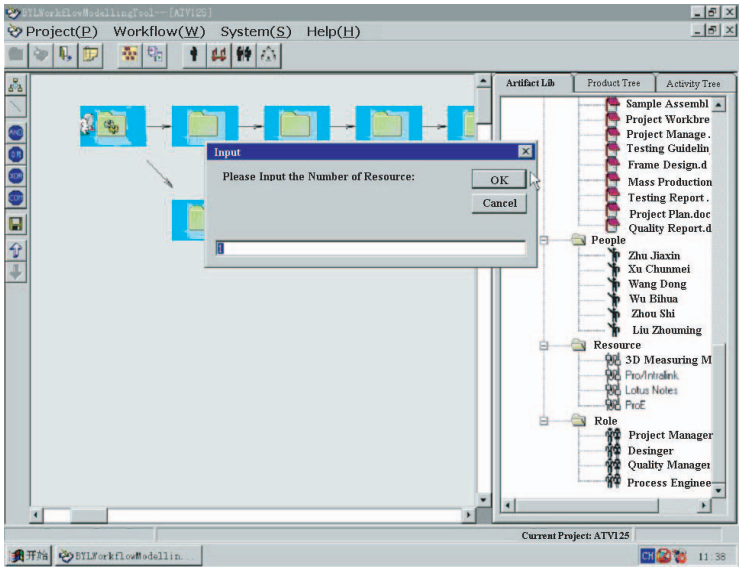


Figure 4. Notification server structure based on grid service platform.

(a)



(b)



(c)

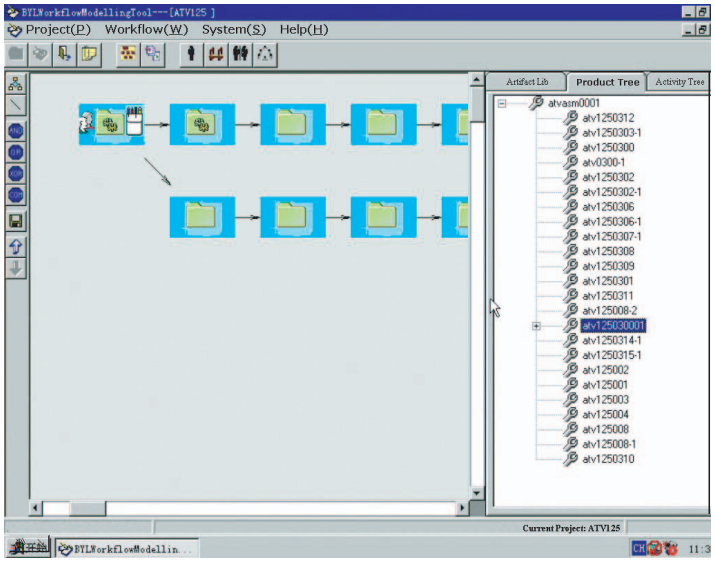
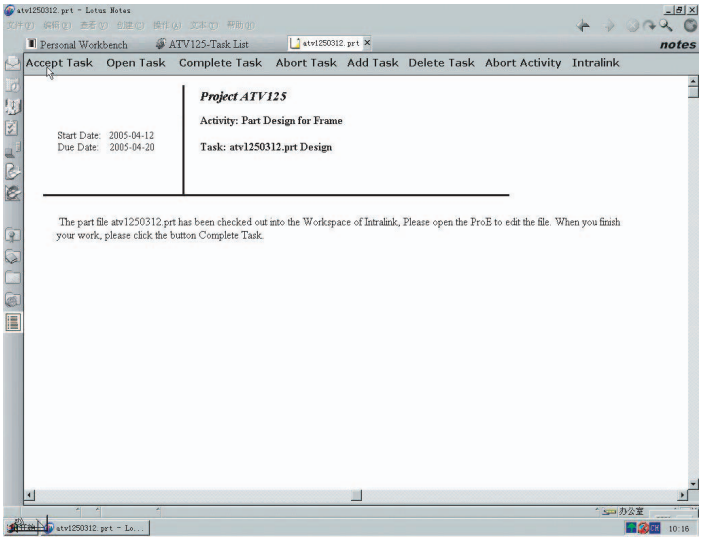
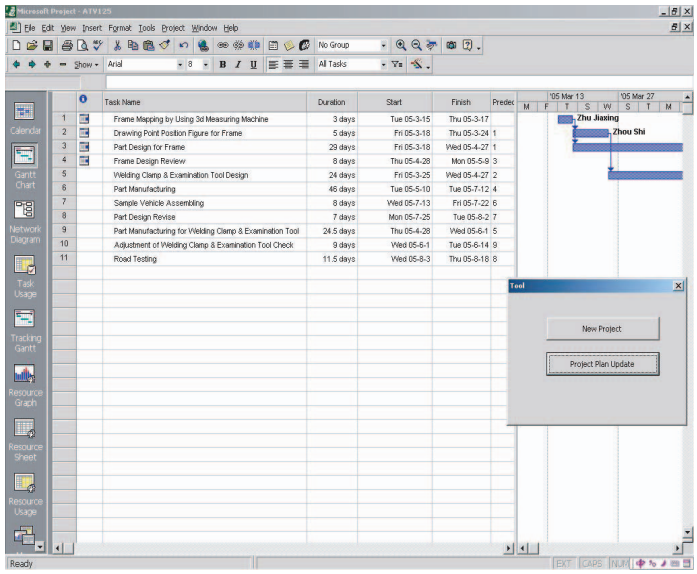


Figure 5. A case study.

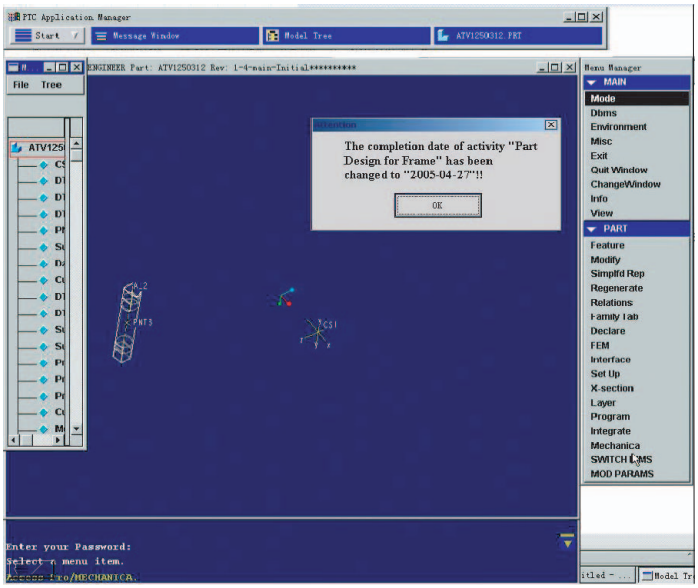
(d)


Personal Workbench
ATV125-Task List
notes
Accept Task
Open Task
Complete Task
Abort Task
Add Task
Delete Task
Abort Activity
Intralink
Project ATV125
Activity: Part Design for Frame
Start Date: 2005-04-12
Due Date: 2005-04-20
Task: atv1250312.prt Design
The part file atv1250312.prt has been checked out into the Workspace of Intralink, Please open the ProE to edit the file. When you finish your work, please click the button Complete Task.

(e)


Task Name
Duration
Start
Finish
Frame Mapping by Using 3d Measuring Machine
Drawing Point Position Figure for Frame
Part Design for Frame
Frame Design Review
Welding Clamp & Examination Tool Design
Part Manufacturing
Sample Vehicle Assembling
Part Design Revise
Part Manufacturing for Welding Clamp & Examination Tool
Adjustment of Welding Clamp & Examination Tool Check
Road Testing
Zhu Jiaxing
Zhou Shi
Tool
New Project
Project Plan Update
Ready

(f)


PTC Application Manager
Start
Message Window
Model Tree
ATV1250312.PRT
File
Tree
Menu Manager
MAIN
Mode
Dbms
Environment
Misc
Exit
Quit Window
ChangeWindow
Info
View
PART
Feature
Modify
Simplfd Rep
Regenerate
Relations
Family Tab
Declare
FEM
Interface
Set Up
X-section
Layer
Program
Integrate
Mechanica
SWITCH DIMS
MOD PARAMS
The completion date of activity "Part Design for Frame" has been changed to "2005-04-27"!!
OK
Enter your Password:
Select a menu item.

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. The notification server also provides a set of facility services. 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.

6. A Case Study

The authors will introduce a demonstration to illustrate the service notification model. This case example demonstrates a grid based coordination system for facilitating a vehicle design process.

The project manager can build a new project plan through MS Project software, which is connected to the notification server through an application broker. The project plan can be checked into a repository through a grid service (Figure 5(a)) and a corresponding project instance is created in the notification server. With the help of modeling service in the notification server, we can add more information into the project model, such as organization, resources, and artifact sub-models (Figure 5(b) and (c)). Figure 5(c) shows a product tree defined for this project and specific tasks will be generated for those parts according to the pre-defined workflow model. Each designer involved in this project will be allocated some tasks through the personal workspace. When he opens a task, some guidelines are provided to help him finish this task. Figure 5(d) shows a task of design part atv1250312, which should be done in the ProE.

Due to the customer's request, the project manager decides to change the completion date of activity 'Part Design for Frame' from '2005-05-02' to '2005-04-27'. When the new project plan is submitted, the repository service generates an event, and then the event receiver determines who should be notified. According to the awareness model, all frame part designers should be notified. Figure 5(f) shows the designer who is drawing part atv1250312 in ProE is notified. This information is sent from event receiver to the application broker of the ProE.

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, the authors developed gateway plug-ins for MS Project and ProE using VB applications.

7. Related Works

There are many event notification servers [8]. Three representative examples are CASSIUS, Elvin, and Siena [9]. 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. The authors notification server model aims to coordinate and support distributed design process in which an awareness model is designed with built-in knowledge. Another characteristic of the 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 [10]. Gossip included a shared composite product model with rules of awareness information. Product object and awareness relation are registered in Gossip. The 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 grid service has been proposed [11]. By employing the notification mechanism of a web-based grid service, the web service can be composed of other web services, without the need of prior knowledge of how these web services are developed. The NotificationBroker in that model is conceptually quite similar to the authors notification server. A NotificationBroker is an intermediary, which, among other things, allows publication of messages from entities that are not themselves service providers. It includes standard message exchanges to be implemented by NotificationBroker service providers along with operational requirements expected of the service providers and requestors participating in the brokered notifications. However, no detailed descriptions on their model, nor any implementation guidelines, are provided.

With the aim of reducing the development time, design process coordination is one aspect of concurrent engineering. Under the umbrella of concurrent engineering, various philosophies or methods such as integrated product development (IPD) and knowledge based engineering (KBE) are proposed. So far, many works are done to implement the IT environment to support these new ideas [12]. Comparing with these systems, we define a new infrastructure, which makes use of up-to-date grid technologies to provide both application and human coordination based on an awareness model.

8. Conclusions and Future Work

This article 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. The future work of the authors includes developing a notification server to support more expressive event transformation rules in a dynamic engineering process.

Acknowledgments

This work is partly supported by 'SEC E-Institute: Shanghai High Institutions Grid', Chinese high technology development plan (No. 2004AA104340), Chinese Semantic Grid project and Chinese NSF projects (No. 60503041, No. 60473092). This work was partially conducted while the first author was visiting Stanford University. The authors would like to thank Mr. Jim Cheng and Prof. Kincho Law for their help.

References

1. Gorton, I. and Motwani, S. (1996). Issues in Co-operative Software Engineering using Globally Distributed Teams, *Information and Software Technology*, **38**(10): 647–655.
2. Kalay, Y.E., Khemlani, L. and JinWon, C. (1998). An Integrated Model to Support Distributed Collaborative Design of Buildings, *Automation in Construction*, **7**(2–3): 177–188.
3. Foster, I. and Kesselman, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Fransisco.
4. Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, http://www.globus.org/research/papers/ogsa.pdf
5. Graham, S., Simeonov, S., Boubez, T., Daniels, G., Davis, D., Nakamura, Y. and Neyama, R. (2001). *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*, SAMS, Indiana.
6. Dourish, P. and Bellotti, V. (1992). Awareness and Coordination in Shared Workspaces, In: *Proceedings of Conference on Computer Supported Cooperative Work*, Toronto, Ontario, Canada, pp. 107–114.
7. Steinfield, C., Jang, C.Y. and Pfaff, B. (1999). Supporting Virtual Team Collaboration: The TeamSCOPE System, In: *Proceedings of GROUP Conference*, Stockholm, Sweden, pp. 81–90.
8. Cugola, G., Nitto, E. and Fuggetta, A. (2001). The JEDI Event Based Infrastructure and Its Application to the Development of the OPSS WFMS, *IEEE Transactions on Software Engineering*, **27**(9): 827–850.
9. Cleidson, R.B., Santhoshi, D.B. and David, F.R. (2002). Supporting Global Software Development with Event Notification Servers, In: *Proceedings of the ICSE 2002*, http://citeseer.ist.psu.edu/desouza02 supporting.html.
10. Farshchia, B.A. (2000). Gossip: An Awareness Engine for Increasing Product Awareness in Distributed Development Projects, http://www.idi.ntnu.no/~ice/publications/caise 00.pdf.
11. IBM (2004). Publish-Subscribe Notification for Web services, http://www-106.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf.
12. Prasad, B. (1996). *CE Fundamentals*, Concurrent Engineering Fundamentals, Volume II: Integrated Product Development, New Jersey: Prentice Hall.

Jian Cao

Dr Cao is an Associate Professor with the Department of Computer Science and Technology at Shanghai Jiaotong University (SJTU), China, and the deputy director of the Grid Center of the University. He received his BSc and PhD in Automatic Control Theory and Control Engineering from Nanjing University of Science and Technology (P.R. China) in 1997 and 2000 respectively. His research interests include Advanced Manufacturing Theory and System, Collaborative Information System, Grid and Service Computing, and Software Engineering. His main areas of expertise are the developments of software and models to support coordination and cooperation among humans, systems, and components. He has authored or co-authored over 80 journal and conference papers in the above areas.

Shensheng Zhang

Dr Zhang is a full Professor with the Department of Computer Science and Technology at Shanghai Jiaotong University (SJTU), China, and the Vice Dean of the Electronics and Information School in Shanghai Jiao Tong University. Shensheng Zhang graduated from Department of Mechanical Engineering of Shanxi Mining Institute in 1982. He holds MS and PhD degrees from the

Mechanical Engineering Department of Stanford University during the period of 1983–1988. Dr Zhang's current research interests include information integration, pervasive computing and distributed virtual reality. He has published more than 200 papers in journals and conferences.

Jie Wang

Dr Wang received his BS from Shanghai Jiao Tong University, his MS from both the University of Miami and Stanford University, and his PhD in engineering and informatics from Stanford University. He conducts researches in decision making and knowledge management for enterprise sustainable development and global competitiveness, engineering and environmental informatics, E-commerce, and E-government; IT as a modern tool for business competitive strategy, he currently directs a research program on knowledge management and environmental informatics at Stanford University and an executive training program, Stanford-China Leadership for sustainable development and global competitiveness, between Stanford and Development Research Center of The State Council in China.

Minglu Li

Dr Li graduated from the School of Electronic Technology at the University of Information Engineering in 1985. He received his PhD in Computer Software from Shanghai Jiao Tong University (SJTU) in 1996. Dr Minglu Li is a full Professor at the Department of Computer Science and Technology of SJTU. Now, he is the Vice Dean of the Department of Computer Science and Technology and the Director of the Grid Computing Center. Currently, his research interest includes Web services, grid computing, and multimedia computing. He has published over 100 papers in important academic journals and international conferences.