Ubiquitous Computing Environment for Project Management Services

David Liu[1], Jinxing Cheng[2], Kincho H. Law[3] and Gio Wiederhold[4]

Abstract

This paper describes a mediation based software framework for the development of a ubiquitous computing environment for distributed engineering services. Two fundamental issues are addressed: (1) universal accessibility from devices to services, and (2) coordination and interaction among the parties accessing the services. A prototype infrastructure has been developed and illustrated using a variety of project management application software as well as different devices ranging from PDA, web browsers and desktop computing.

Introduction

A shift to distributed computing is underway. Rapid proliferation of Internet protocols, fast expanding computing power, coupled with broadband and mobile communication technologies make ubiquitous computing possible. We will soon have an interconnected web of small devices that provide valuable information to people regardless of their locations. However, ubiquitous computing is more than simply tying many wired or wireless gadgets together. Everything from client devices, communication networks to software applications needs to work together to enable two main characteristics of ubiquitous computing: (1) universal accessibility from devices to services, and (2) effective coordination and interaction among the parties accessing the services.

Ubiquitous computing can find many applications in the design and construction industries. Such applications range from field inspection, to site procurement of materials, to interactive on-site project planning. For instance, with mobile devices, one could readily compare the as-built site condition with the planned design information, enquire availability of materials and receive immediate response to change orders, and gain dynamic interactions with Internet-based services.

[1] PhD Student, Electrical Engineering Department, Stanford University, Stanford, CA 94305-4020, email: davidliu@stanford.edu

[2] PhD Student, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: cjx@stanford.edu

[3] Professor, Civil and Environmental Engineering Department, Stanford University, Stanford, CA 94305-4020, email: law@cive.stanford.edu

[4] Professor, Computer Science Department, Stanford University, Stanford, CA 94305, email: gio@db.stanford.edu

In order to provide a ubiquitous computing environment for engineers, project managers, and on-site personnel to more effectively communicate with each other, the first challenge is to provide project personnel easy access to the engineering software services. The issue we are addressing lies in the software layer rather than the physical access layer. Making the assumption that the communication channel and the network protocol are in place for client devices to gain access to software services, we have to construct the software services in such a way that a wide range of devices with drastically different characteristics can be supported. For example, the output of a CAD design tool on a high-speed graphics workstation would likely be different from the output display on a handheld device. Information needs to be filtered and presented in different granularity depending on the types of client devices. In this paper, we investigate the design and implementation of a mediation-based framework that would allow the incorporation of computing devices, such as PDA and web-browsers, into a distributed engineering service environment and support a wide range of clients. Secondly, coordination and interaction are of significant importance when information is shared among different application services. We demonstrate using a combination of information modeling standard technologies, including relational database, XML and PSL, to build intermediary data model for a variety of project management service applications.

Mediation-Based Framework for Distributed Service Integration

As software getting more complex and services becoming more powerful, it becomes essential to define a framework by which software can be constructed to serve clients with dramatically different computation and communication power. The key challenges are to lower the complexity of software design and to minimize the software maintenance cost. To cope with these issues in dynamic collaborative computing environments, mediators are introduced (Wiederhold 1992, Wiederhold and Genesereth 1997). Mediators are intelligent middleware that sit between information system clients and sources. They provide integrated information, without the need to integrate the data sources. Specifically, mediators perform functions such as accessing and integrating domain-specific data from heterogeneous sources, restructuring the results into object structures, and extracting appropriate information to be transmitted.

Mediation architecture is conceptually comprised of three layers, as shown in Figure 1. The mediation layer resides between the base resource access interface and the service interface, incorporating value-added processing by applying domain-specific knowledge. A major task for the mediation service is to reduce the data volume to be shipped to information clients, while maintaining the desired information content. The principal tool for data reduction is abstraction. Techniques differ, though, on how the abstraction is obtained and on how the information granularity is controlled. Active mediation is a variation of mediation technology, in which a mediator is designed to have the ability to adapt its behavior to the client request or the source data stream, hence providing the ability to dynamically change the granularity of information abstraction (Liu et.al. 2000). We apply active mediation for constructing device-independent software services.

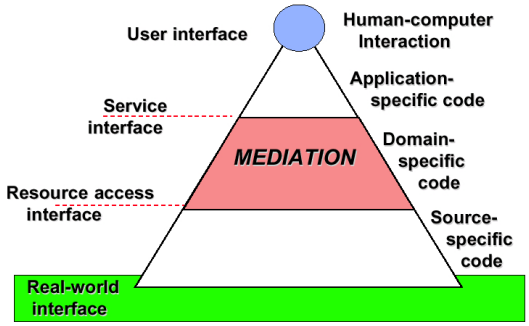

Figure 1: Mediation Architecture

Active Object

Objects are used as the basic model to describe data. Most clients are best served by information in object form that may integrate multiple heterogeneous sources. We choose XML as the object representation format based on its extendibility, structure, and validation as a language. XML is a meta-markup language that consists of a set of rules for creating semantic tags used to describe data (Young 2001). An XML element is made up of a start tag, an end tag, and content in between. The start and end tags describe the content within the tags, which is considered the value of the element. In addition to tags and values, attributes are provided to annotate elements.

Active object is a special type of XML object. In active objects, two types of elements are defined: data elements and active elements. A data element describes the content of an object and an active element contains code segment to filter the source information. The code segment is called active node, a serialized byte code string of executables. Figure 2 shows a sample active object and the Java source code for the contained active element. The active object specifies a query request for activity information from a software service. The active element, **PalmTrimmer**, contains a Java byte-code segment that specifies how the result object should be filtered before returning to the client. **PalmTrimmer** will be appended to the result XML object retrieved from the information source, and interpreted by the active mediator run-time. All active nodes are derived classes of **ActiveNode**, and they overload the execute function to provide specific functionalities. The execute function takes three parameters: the current active element handle, the root element handle, and the client environment information. The active mediator runtime environment fills in these three parameters when the mediator loads the active nodes. The example, **PalmTrimmer**, keeps only the activity identifiers and activity descriptions as relevant components when enacted.

There are many methods in which active nodes can be created. We have developed templates where active node source code can be generated to perform simple schema-based information filtering. The Java source code can then be compiled and automatically inserted into active object. The responsibility to provide client-agnostic information content remains with source information service, while the responsibility to define client-specific information abstraction and reduction are shifted to individual clients.

```
<REQUEST>
  <QUERY>
    ACTIVITY
  </QUERY>
  <PalmTrimmer active-node="yes">MROZZO@`#`"T`,P$`!$-O9&
  4!``I3;W5R8V5&:6QE`0`)4V]U<F-E1&ER`0`-M0V]N<W1A;G1686Q
  U90$`"D5X8V5P=&EO;G,!``],:6YE3G5M8F5R5&%B;&4!M`!),;V;%
  9A<FEA8FQE5&%B;&4!``=4<FEM;65R!P`(`0`08VAA;W,O06-T
  </PalmTrimmer>
</REQUEST>

public class PalmTrimmer extends ActiveNode
{
  public String execute(Element current,
                        Element root,
                        ClientEnv env) {
        Vector tags = new Vector();
        tags.addElement(new String("ACTIVITYID"));
        tags.addElement(new String("DESCRIPTION"));
        keepOnlyNodes(root, tags);
        return "Done";
    }
}
```

Figure 2: A Sample Request Active Object and Active Node Source Code

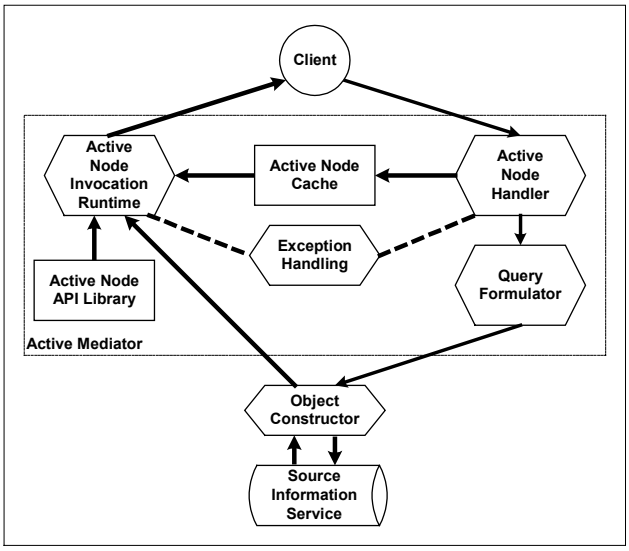

Figure 3: Active Mediation Architecture

Active Mediator Architecture

Mediators reside between source information service and client application interface. They are used to mediate the queried content obtained from the source information service. Figure 3 illustrates the architecture of an active mediator. The active mediator conceptually consists of four functional units and two code segment repositories. Client requests in the form of active objects are initially processed by the active node handler. The active object will be divided, with source requests being forwarded to query formulator and active nodes being stored in the active node cache. Query formulator will form a source query and forward it to the information service. Upon completion of the source information service, a result data object is returned to

the active node invocation runtime. The runtime environment loads relevant active nodes into the result data object, during which code segments are dynamically loaded from the active node cache and the active node API library. Active nodes are invoked by calling their "execute" function, after which the active object is transformed and the content is filtered. The resulted objects will then be returned back to the client.

It is also important for the system to have a comprehensive exception handling policy. Our current implementation prohibits any results from getting through the mediator in the case of exception. In addition, the conditions are logged for future maintenance. The active mediation system can be deployed without changes to either the information client or the source information service, enabling a smooth transition from legacy information service infrastructure to one using active mediation infrastructure.

Device Independent Software Services

Information services are responsible for providing information content and normally lack the ability to provide client-specific granularity for the content. Retrofitting existing services to be device-aware is an expensive exercise and does not follow the good software design principle of separating client specification and server functionality. Moreover, it is infeasible to cover all existing and future client device types. The key in constructing device-independent software services, thus, lies in separating the content from the abstraction of the information that a service provides. Different granularity of information content can be acquired by applying different level of abstractions.

Active mediation is a natural solution for such problem by giving the information client the ability to specify how information should be abstracted and filtered. As shown in Figure 4, the source information service maintains its responsibility of providing information content to clients based on the query of a client, regardless of the device type of the client. The content of information is composed of its presentation style and its data. The active mediator, situating between the source information service and the information clients, has the responsibility of reducing information volume through abstraction. As described earlier, the active mediation architecture requires the information clients to provide the specific filtering routines that are called upon to conduct information abstraction. The active mediator itself is not aware of the client types, hence client-independent.

Given the active mediation infrastructure, we can divide the process of constructing device-independent information services into two inter-related components. The first component focuses on constructing source services to provide modular and object-oriented information content. The second component focuses on developing information filtering routines for the information content, also known as active node routines, for each client device type. One benefit of active mediation infrastructure is the separation of information clients and information services. New client device types can be added into the existing computing environment by developing new information filtering routines. No modifications are necessary in either the source information service, the active mediator, or the other client devices.

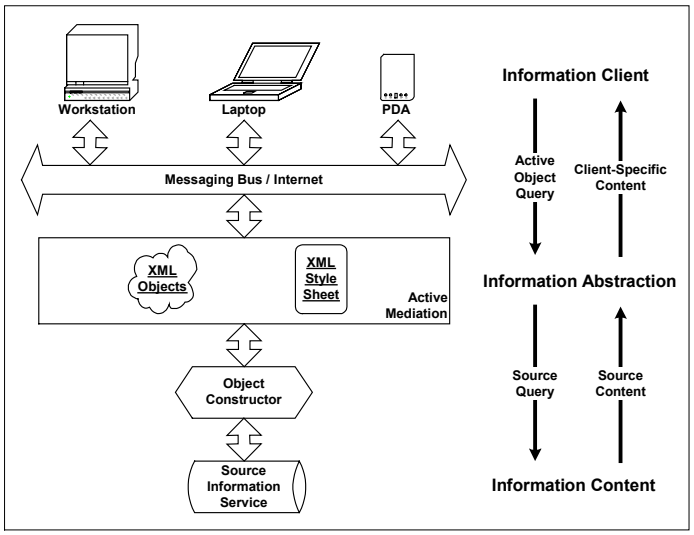

Figure 4: Active Mediation in Software Service Construction

Information Modeling

Information modeling plays an important role in ubiquitos computing and distributed service integration. Information in different applications usually has different representations. Even for the same type of application, the internal representations of the information are also different. To cope with the issue of different representations among applications, we need an ontology standard to model information. There have been many efforts to develop product data standards for data exchange, such as STEP (ISO 1994), IFC (IAI 1997), ifcXML (Liebich 2001), aecXML (IAI 2002), etc.. Most of the current ontology standards however focus mainly on product data and do not provide extensive information about process and task specifications which are important data attributes for project management applications.

PSL (Process Specification Language) was initiated by NIST (National Institute of Standards and Technology) and is emerging as an international standard for the manufacturing industry (Schlenoff et.al. 2000). The goal is to create a language for the exchange of process information among different applications. The development of PSL is motivated by two basic reasons. First, there are not many existing standards for process information exchange. Second, current ontology standards lack a formal logic to define relationships and constraints. PSL is based on first order logic and situation calculus, which make it an ideal candidate standard for representation of process information and for project and workflow management.

Process Specification Language

PSL is based on KIF (Knowledge Interchange Format), which is designed for knowledge interchange among disparate computer systems. KIF has declarative semantics, and is logically comprehensive (Genesereth and Fikes 1992). Figure 5 shows the overall organization of PSL, which includes the PSL core, the PSL outer core and PSL Extensions (Schlenoff et al. 2000).

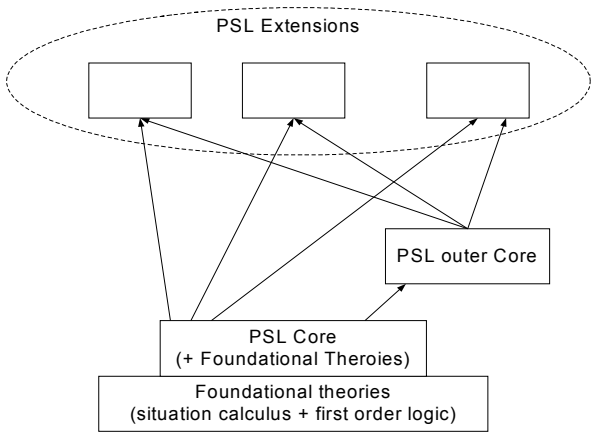

Figure 5: PSL Ontology

- The PSL core is a set of axioms based on KIF. The PSL core includes four basics classes: Object, Activity, Activity_Occurrence and Timepoint. Relations are defined among the classes, for example:
  (occurrence-of activity-occurrence activity)
  (before timepoint timepoint)

- PSL outer core consists of a small set of extensions, which are generic and pervasive in their applicability. The extensions in the PSL outer core include Subactivity Extension, Activity-Occurrence Extension and States Extension. Relations can be defined using the PSL outer core extensions, for example:
  (subactivity-occurrence activity-occurrence activity-occurrence)
  (subactivity activity activity)

- PSL extensions include ontology modules such as generic activities, ordering relations and schedules. Each module is motivated by a set of applications and covers concepts in a specific application domain. Below are some example relations in the PSL extensions:
  (before-start activity-occurrence activity-occurrence activity-occurrence)
  (before-start-delay activity-occurrence activity-occurrence activity-occurrence duration)

We have extended the PSL core by including extensions that model the essential information related to project management applications (Cheng and Law 2002).

Implementation of PSL Wrappers

Once the PSL ontology for a specific application domain is defined, software wrappers, which act as a bridge between common (PSL) representation and proprietary representations (for each application), need to be built. PSL wrappers are used to retrieve project information from the applications, and are also used to update project information in those applications. The basic process of using PSL for project information exchange can be illustrated in Figure 6 and consists of three major steps -- ontology mapping, communicating with applications, outputting or parsing PSL files. It is not unusual that the same term is often associated with different meanings

in different applications. To exchange project information, first we need to map the concepts in different applications into PSL ontology, so that they are PSL compliant.

Different wrappers are developed to transfer and retrieve information to and from different applications. The application software considered in our current prototype infrastructure includes Primavera P3™, MS Project™, Vite™ and 4D Viewer (McKinney and Fischer 1998). The applications can exchange information using PSL as the ontology standard. To enhance the accessibility of the project information from those applications, we also build a translator between PSL and database. We have designed a database schema according to the PSL ontology and developed a translator in Java to convert information from database to PSL file and vice versa.

An Example Engineering Scenario

An infrastructure shown in Figure 7 has been developed to illustrate the ubiquitous computing environment developed for distributed project management services. In this distributed service infrastructure, the active mediator acts as an intelligent bridge that connects various devices with the Oracle 8i database, while PSL acts as a common data model through which various engineering services can communicate with each other. The active mediator captures the inquiry request from any device, constructs XML object and sends the request to the Oracle 8i database. The active mediator can also respond to the user query by retrieving the latest information from the Oracle 8i database and converting the information to suitable format for displaying at users' devices. PSL wrappers are used to retrieve information from various project management applications. Project information can also be translated between PSL files and the Oracle 8i database.

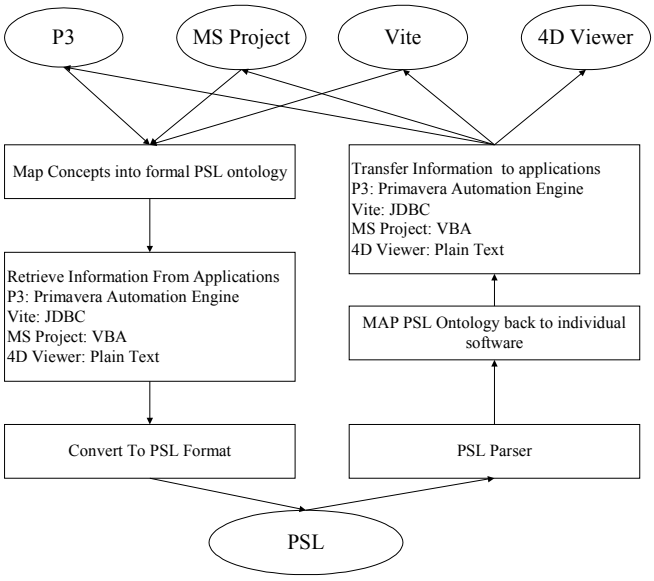

Figure 6: PSL Wrappers

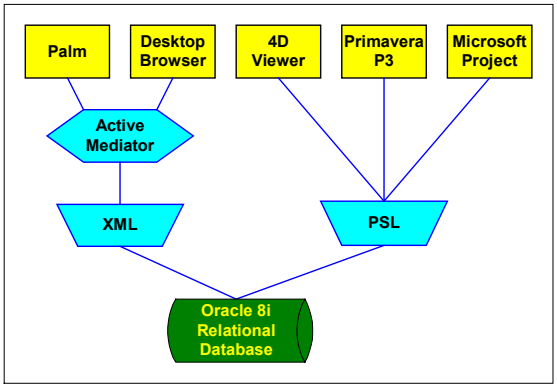

Figure 7: A Ubiquitous Computing Environment for Engineering Services

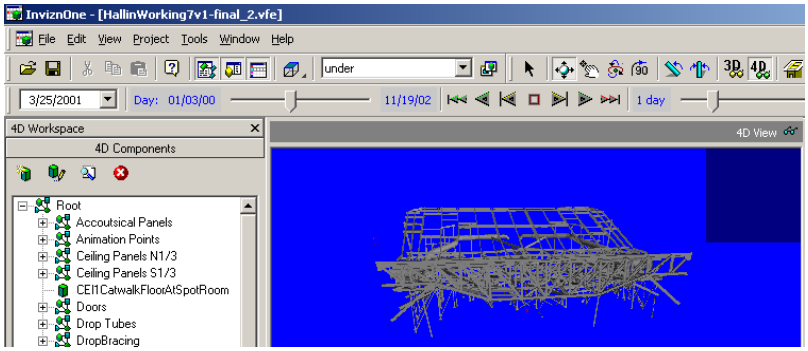

Figure 8: Reviewing Sample Project on 4D Viewer

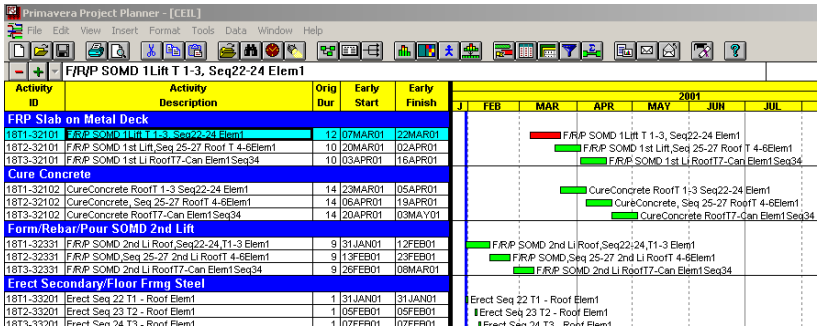

Figure 9: Reviewing Sample Project on Primavera

Let's look at an example scenario and demonstrate how the ubiquitous computing environment may help facilitate personnel from different functional groups conduct collaborations. We use the project model of the Disney Concert Hall as the test case example. Figure 8 shows a snapshot of construction progress using the 4D Viewer which is a very effective tool for analyzing and visualizing 3D architectural designs and their relationships to project schedules (Koo and Fischer 2000). Figure 9 is the view of the scheduling information using Primavera P3, a specialized tool that focuses on the scheduling aspect of the project. Using PSL as the intermediate data model, the information is shared between the relational data model and the proprietary Primavera data model. The scheduling information can also

be reviewed using a handheld Palm device, for example by an on site personnel, as shown in Figure 10. The information is first converted into XML model, and then the active mediator filters the information and adapts the content for the handheld device that is reviewing the information.

Suppose, as an hypothetical example, that the duration for the activity, 18T1-33201, for erecting a roof element is to be changed from 1 day to 40 days. The change can be made using the Palm device by an on site personnel. The update will be stored into the relational database and trigger Primavera P3 to reschedule the project. The revised schedule can be viewed using MS Project as shown in Figure 11 and the project model can be displayed and viewed using the 4D Viewer as shown in Figure 12. The project status can also be viewed using a simple web browser as shown in Figure 13. The web browser adopts the same information path as in the case of the Palm device. An XML model constructor and an active mediation are used to generate appropriate information content for different information clients. Comparing to the Palm device, the web browser can display much more detailed scheduling information, for example, with the altered activity and the affected activities highlighted -- the information that project managers may find helpful to diagnose the impact of the updated schedule.

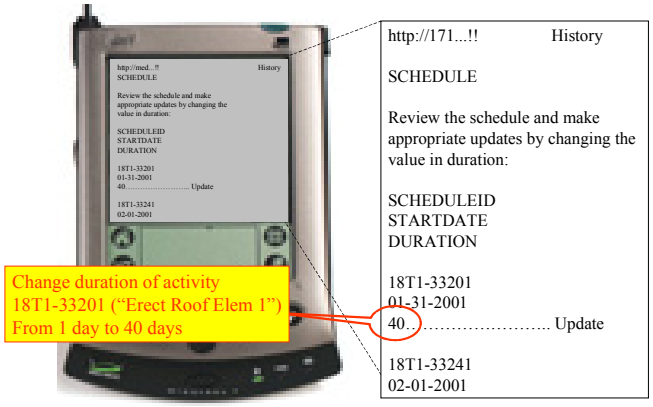

Figure 10: Revising Project Schedule Via a Palm Device

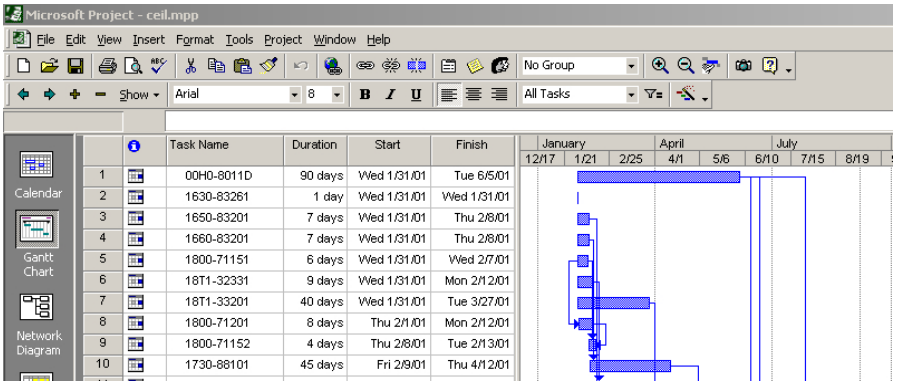

Figure 11: Regenerated Gantt Chart in Microsoft Project

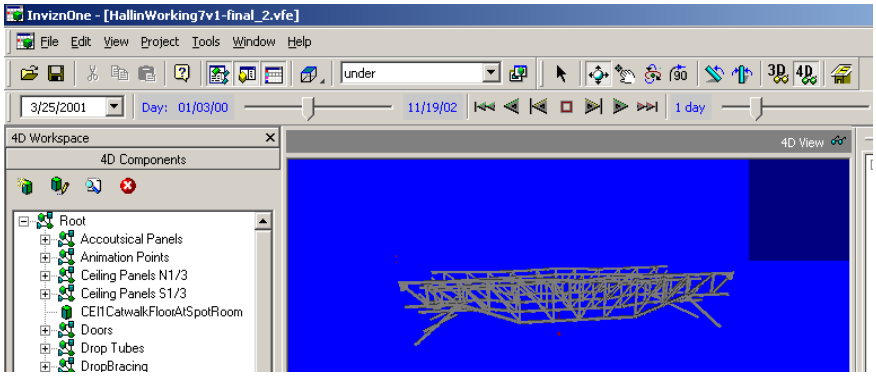

Figure 12: Reviewing Updated Project on 4D Viewer

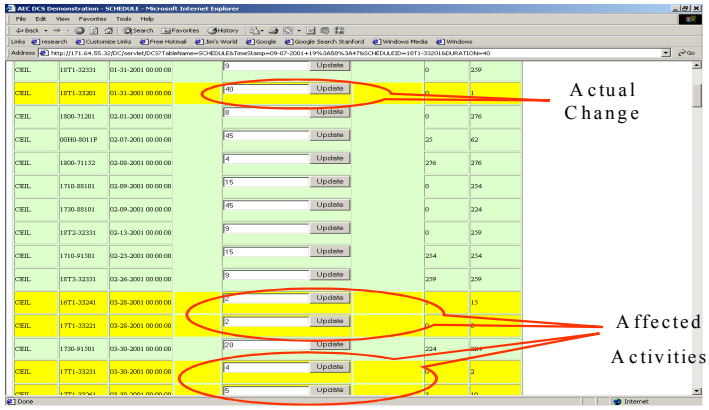

Figure 13: Reviewing Updated Schedule on Web Browser

Summary

In this paper, we have presented a mediation-based framework, which enhances information accessibility and user interactivity, for ubiquitous computing and distributed engineering services. Specifically, active mediation is introduced as a value-added service layer that resides between source information service and information client. Its main functionality is to provide universally accessible service to any information client according to its characteristics, hence making source information service device-independent. We have shown that PSL, XML and relational models can be used together to effectively model data required by the various engineering service tools. Data can be exchanged by transforming different data models. We have also demonstrated that PSL can potentially be an effective ontology standard for project management applications. By building a PSL wrapper for each application, we have successfully exchanged project information using PSL. By building a translator between PSL and Oracle database, we greatly improve the accessibility of the project information by various engineering services.

In addition to effective data integration among different software tools, ubiquitous computing requires truly integrated engineering services. Our current investigation is to automate such process by developing a service composition environment (Liu et.al. 2002).

Acknowledgement

This work is partially sponsored by the Center for Integrated Facility Engineering at Stanford University, a Stanford Graduate Fellowship, and the Product Engineering Program headed by Dr. Ram D. Sriram at NIST. The Product Engineering Program gets its support from the NIST's SIMA (Systems Integration for manufacturing Applications) program and the DARPA's Radeo Program. The 4D Viewer and the 4D model of the Mortenson Ceiling Project are provided by Professor Martin Fischer and his research group at Stanford University.

Acknowledgement

Cheng, J. and Law, K.H (2002), "Using Process Specification Language for Project Information Exchange," *3rd International Conference on Concurrent Engineering in Construction,* Berkeley, CA, pp. 63-74.

Genesereth, M.R. and Fikes, R. (1992), "Knowledge Interchange Format Version 3.0 Reference Manual," Computer Science Department, Stanford University.

IAI (1997), "Industry Foundation Classes," Specification Volumes 1-4, International Alliance for Interoperability, Washington, DC.

IAI (2002), "AecXML," International Alliance for Interoperability, http://www.aecxml.org.

ISO (1994), "Product Data Representation and Exchange: Part 1: Overview and Fundamental principles," 10303-1:1994, ISO.

Koo, B. and Fischer, M. (2000), "Feasibility Study of 4D CAD in Commercial Construction," *Journal of Construction Engineering and Management, ASCE*, 126(4):251-260.

Liebich T. (2001), "XML Schema Language Binding of EXPRESS for ifcXML," MSG-01-001(Rev 4), International Alliance of Interoperability.

Liu, D., Law, K.H. and Wiederhold, G. (2000), "CHAOS: An Active Security Mediation System," Proceedings of International Conference on Advanced Information Systems Engineering, LNCS, Vol.1789, B. Wangler and L. Bergman (eds.), Springer-Verlag, pp. 232-246.

Liu, D., Law, K.H. and Wiederhold, G. (2002), "Analysis of Integration Models for Service Composition," To appear in *Third International Workshop on Software and Performance*, Rome, Italy.

McKinney, K. and Fischer, M. (1998) "Generating, Evaluating and Visualizing Construction Schedules with 4D-CAD Tools," *Automation in Construction*, 7(6): 433-447.

Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., and Lee, J. (2000), "The Process Specification Language (PSL): Overview and Version 1.0 Specification." NISTIR 6459, National Institute of Standards and Technology.

Wiederhold, G. (1992), "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, 25(3):38-49.

Wiederhold, G. and Genesereth, M. (1997), "The Conceptual Basis for Mediation Services," *IEEE Expert, Intelligent Systems and Their Applications*, 12(5):38-47.

Young, M.J. (2001), *Step by Step XML*, Microsoft Press.