INTEGRATING ENGINEERING WEB SERVICES WITH DISTRIBUTED DATA FLOWS AND MOBILE CLASSES

David Liu[1], Jun Peng[2], and Kincho H. Law[3]

ABSTRACT

This paper describes a software paradigm for composing engineering web services. Software modules have been designed and implemented to facilitate the construction and integration of web services. The paradigm employs a distributed data flow approach that supports direct data exchanges among web services, and thus avoid many performance bottlenecks attending centralized processing. The performance and flexibility are further improved by using mobile classes to distribute computations within the service framework and to reduce the amount of data traffic by moving computations closer to the data. A system has been prototyped to integrate several project management and scheduling software applications. It is shown that distributed data flow, combining with the utilization of mobile classes, is an effective approach for integrating large engineering software services.

KEY WORDS

Engineering web services; service integration; distributed data flow; mobile class; project management.

INTRODUCTION

The emergence of computer networks enables the integration of remote software services over the web to form composed applications. Two issues that need to be addressed in developing a service composition framework are interface incompatibility and performance. Web services are typically heterogeneous and consist of a variety of conventions for control and data. Even when data and communication protocol standards are promulgated, the precise meaning and scope of the output of a service will not necessarily match the expectations of another service. In a typical composed application, the results from one web service are transported to the application site, handled there, and then re-routed to the next web service. Since data are exchanged between distributed web services and central server, this centralized data-flow approach is inefficient for integrating large-scale software services, especially for engineering applications where large volume of data needs to be exchanged.

---

[1] Graduate Student, Electrical Engrg. Department, Stanford University, Stanford, CA 94305, Phone +1 650/725-1886, FAX 650/723-4806, davidliu@stanfordalumni.org

[2] Research Associate, Civil and Envir. Engrg. Department, Stanford University, Stanford, CA 94305, Phone +1 650/725-1886, FAX 650/723-4806, junpeng@stanford.edu

[3] Professor, Civil and Envir. Engrg. Department, Stanford University, Stanford, CA 94305, Phone +1 650/725-3154, FAX 650/723-7514, law@stanford.edu

This paper describes a Flow-based Infrastructure for Composing Autonomous Services (FICAS), which is designed and implemented to facilitate the construction and the integration of distributed web services (Liu 2003). FICAS addresses three design issues: (1) scalability – integration and management of large number of web services in the service composition infrastructure; (2) performance – high efficiency in the execution of composed applications; and (3) ease of composition – effective and convenient specification of service compositions by the application programmers. FICAS uses distributed data-flows to achieve better scalability and performance without sacrificing ease of composition. In addition, active mediation, supported by the utilization of mobile classes, is used in applications employing multiple web services that are not fully compatible in terms of data formats and contents. Active mediation increases the applicability of the services, reduces data communication among the services, and enables the application to control complex computations.

SERVICE INTEGRATION FRAMEWORK

Web services execute processes that involve one or more software applications along with their domain data. Web services are composed in a loosely-coupled fashion to allow flexible integration of heterogeneous applications and systems. There has been many research and development works in service composition, particularly in defining unified standards for describing, deploying, and accessing applications (Curbera et al. 2003). FICAS contributes a software composition paradigm that supports distributed data flow with active mediation and addresses the performance implications of service composition.

SOFTWARE SERVICE MODEL

To compose multiple web services into an integrated application consists of three basic steps. First, services that provide composable functionalities are catalogued. Existing services that decide to participate will expose their interfaces, and missing services are constructed. Second, the composed "mega" application is specified so that it will employ the most suitable combination of web services. Issues to be considered when composing web services include scalability of the services, robustness of the services, security of the service interaction, effective and convenient specification of the compositions, and performance of the composed applications. Third, the composed application is executed as needed and the autonomous services employed are often transparent to the users.

In FICAS, web services are specified as a homogeneous model that promotes communication and cooperation with each other. The service model in FICAS consists of a service core, an input event queue, an output event queue, an input data container, and an output data container:

- The service core represents the core functionality of the web service. It is responsible for performing computation on the input data elements and generating resultant data elements. Existing software applications can be wrapped into a service core.
- Events (messages) are exchanged between services to control the flow of web service executions. Asynchronous execution of web services is achieved by using queues for

event processing. The default queuing protocol in FICAS is FIFO (first in and first out), so event messages are processed in the order they arrive.

- The data containers are groupings of input and output data elements for the web service. Input data elements are fetched from the input data container and processed by the service core. The generated data elements are put into the output data container. The data containers enable web services to look up generated data elements.

MEDIATION

Services are usually built by leveraging existing software capabilities and information resources. These resources have had incompatibilities in many e-commerce and e-business applications. Mediators are intelligent middleware that sit between the information sources and the clients (Wiederhold 1992). Mediators reduce the complexity of information integration and minimize the cost of system maintenance. They provide integrated information, without the need to integrate the actual information sources. Mediators perform functions such as accessing and integrating domain-specific data from heterogeneous sources, restructuring the results into objects, and extracting appropriate information.

Figure 1(a) illustrates the mediation architecture which conceptually consists of three layers. The information source provides raw data through its source access interface. The mediation layer performs value-added processing by applying domain-specific knowledge processing. The information client accesses the integrated information via the client access interface. The architecture of the information-oriented service can be mapped to a processing-oriented architecture, as shown in Figure 1(b). The software is accessed through the application-specific interface. The service wrapper obtains specifications from the services and exposes its capabilities through the access protocol.

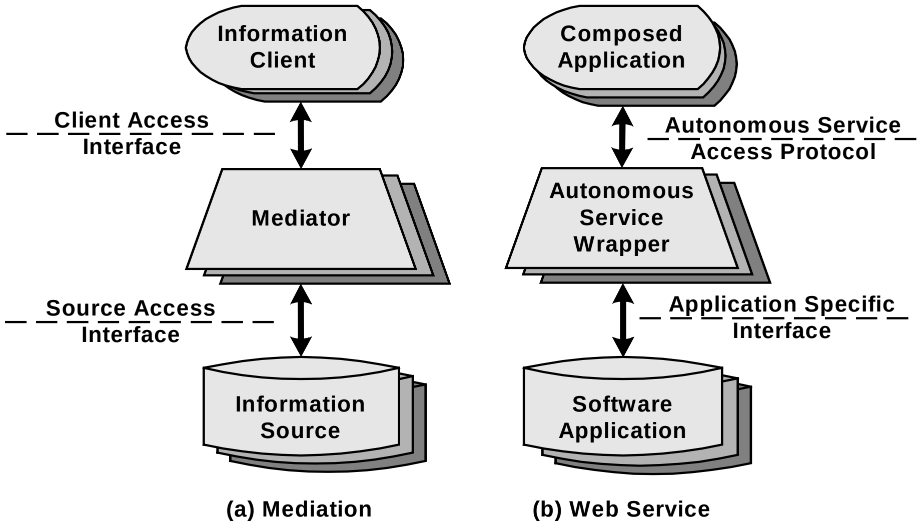


Figure 1: Conceptual Layers in Mediation and in Service Integration

In traditional mediators, code is written to handle information processing tasks at the time the mediators are constructed. Such mediators are static, and only modified when the application sources change their interfaces or behavior. Static mediators are appropriate when resource behavior is known at construction time. In contrast, the active mediators introduced in this paper allow clients to dynamically adapt to the services, in particular their interfaces.

DISTRIBUTED DATA-FLOW

In FICAS, the composed application controller has the responsibility for managing the execution of application services or the control-flows. Based on an execution plan, the controller executes and schedules web services by managing and coordinating the choice, timing, sequence, and dependencies of the events. Data communications among web services are handled separated from the control messages. One objective of the FICAS model is to minimize the aggregate data-communication cost among services.

SEPARATING CONTROL AND DATA FLOWS

A distinguishing characteristic of FICAS is its distributed data-flow model, which allows direct data-flow to occur among remote services. Commonly, in a web services management scheme, the site of the composed application is the central hub for handling all the control and data traffic. By distributing data and exploiting the network capacity among the web services, bottlenecks at the central hub where composed application resides can be avoided. Especially when the composed application resides on a mobile device, relying on centralized data-flow would severely stress its limited bandwidth. Due to the technical challenges in constructing and managing distributed operational code segments, control (i.e., the invocation of a remote service) remains centralized in the FICAS model.

The separation of control-flow and data-flow has been considered in several emerging service composition standards, such as BPEL4WS (Andrews et al. 2003), WSCL (Banerji et al. 2002), and DAML-S (Ankolekar et al. 2001), demonstrating that the importance of separating control-flow and data-flow is being recognized. The idea of separating data-flow from control-flow can also be seen in some distributed workflow environments. For instance, Exotica/FMQM adopts distributed workflow execution and data management for distributed workflow applications (Mohan et al. 1995). However, data flow in those environments is typically supported by a set of loosely synchronized replicated databases instead of managed direct data transfers.

Figure 2 shows a schematic composed application in FICAS where the data are directly exchanged among web services. By distributing data-flows, FICAS eliminates the focused, redundant, and heavy-duty data traffic caused by the forwarding of everything through the central, composed application. The distributed data-flow model utilizes the communication network among web services, and thus alleviates communication loads on the composed application. Furthermore, FICAS allows computations to be distributed efficiently to where data resides, so that the data can be processed without incurring communication traffic.

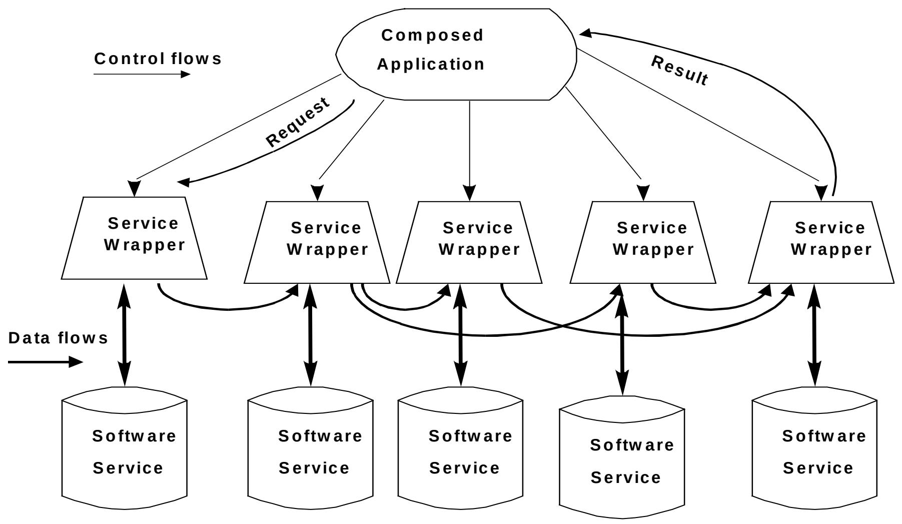


Figure 2: Sample Control Flows and Data Flows in FICAS

MOBILE CLASSES AND ACTIVE MEDIATION

While FICAS gains substantial performance from direct data communication among the services, it still requires the capability provided in mediator nodes to map incompatible sources or to integrate information from diverse sources. In the semantic web setting, where there are many and diverse providers, each service is not expected to deliver results that will be fully compatible and useful to further services. Active mediation applies the notion of mobile code (Fuggetta et al. 1998) to support unforeseen remote information processing. Specifically, matching, reformatting, rearranging, and mapping of data being sent or received among services can be embodied in mobile code, and shipped by the composed application to the remote service as needed. Remote services that can accept active mediation have the ability to adapt their behavior to the client requests. For instance, an information client can forward a compression routine to a service so that queried information is compressed before returned. In general, with active mediation to provide client-specific functionalities, services can be viewed as if they were intended for the specific needs of the client.

MOBILE CLASSES

A mobile class is an information-processing module that can be dynamically loaded and executed. Conceptually, a mobile class is a function that takes some input data elements, performs certain operations, and then outputs new data elements. The underlying programming support of mobile class in FICAS is similar to that of the mobile agent technology (Brewington et al. 1999; White 1997). They both utilize executable programs that can migrate during execution from machine to machine in a heterogeneous network. However, the mobile agents are self-governing in that they decide when and where to migrate on their own. On the other hand, the mobile class in FICAS is an integral part of the service composition framework. Since mobile classes are controlled by the composed application, their management and deployment becomes easier.

Mobile classes can be implemented in many general-purpose programming languages. In current implementation, Java is chosen as the specification language for mobile classes. First, Java is suitable for specifying computational intensive tasks. There are many available standard libraries that provide a wide range of computational functionalities. Second, Java has extensive support for portability. Java programs can be executed on any platform that incorporates a Java virtual machine. Third, Java supports dynamic linking and loading. Java class files are object files rather than executables in the traditional sense. Linking is performed when the Java class files are loaded onto the Java virtual machine. Compiled into a Java class, the mobile class can be dynamically loaded at runtime.

ACTIVE MEDIATION

To enable active mediation in FICAS, a composed application needs to be able to invoke mobile classes on a service, and the service needs to support the execution of the mobile classes. A service supports the execution of mobile classes through the incorporation of an active mediator.

Figure 3 illustrates the architecture of the active mediator. The Mobile Class Fetcher is responsible for loading the mobile class. The loaded Java class is stored into the Mobile Class Cache. The Mobile Class Cache is a temporary storage for mobile classes. The Mobile Class Cache is used to avoid the duplicate loading of a mobile class. It is looked up every time before any mobile class is loaded. The Mobile Class Fetcher is used to load the Java byte code only when the cache miss occurs. The Mobile Class Runtime is the execution engine for the mobile classes. To execute a mobile class, the Mobile Class Runtime loads the Java class from the Mobile Class Cache and invokes the execution of the mobile class. The Mobile Class API Library stores the utility classes that make the construction of mobile classes more convenient. For instance, the Java Development Kit (JDK) library is provided as part of the Mobile Class API Library. The Exception Handling module provides error handling for the loading and the execution of the mobile class.

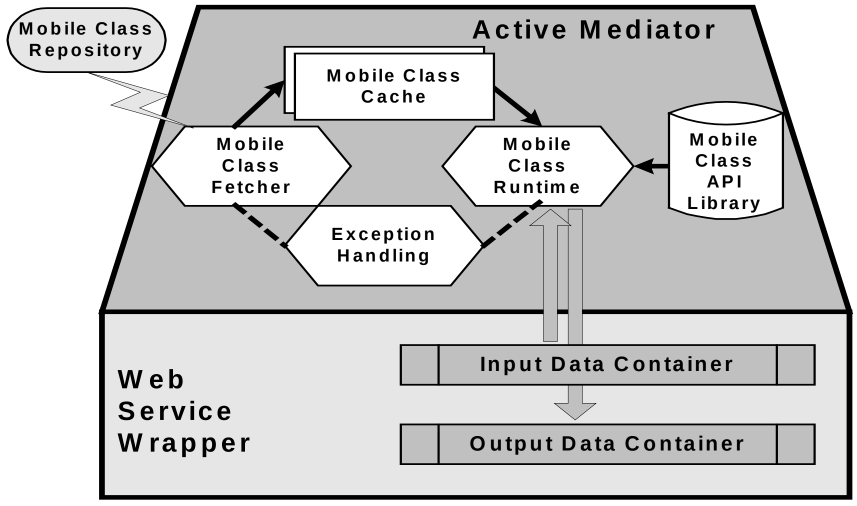


Figure 3: Architecture of the Active Mediation

Upon receiving a Mobile Class event, a service directs the Mobile Class Fetcher to load the mobile class into the Mobile Class Cache. The execution function of the mobile class is then invoked to process data local to the service. Since the service wrapper handles the interchange of data among services, the active mediator is only concerned with the data processing that is local to the service.

EXAMPLE APPLICATION

An example is presented in this section to demonstrate that FICAS is well suited for integrating web services that exchange large amount of data and for handling interface incompatibility (Liu et al. 2003). For a typical construction project scenario, different construction applications can reside at different locations. Some applications may reside on the site offices, and others may reside in the company headquarter. Furthermore, these applications may have different interfaces and require different data formats. Project information is not shared and accessible among all project participants at all time. It is difficult and time-consuming for project managers on the construction sites to get the latest project information from the company headquarter and other places. If there are some changes on the construction site, it is also hard for project managers to evaluate the impacts of the changes on the whole project immediately. Project managers cannot reschedule the project on the construction site right away using the latest information. To improve the proficiency, a ubiquitous environment is highly desired, so that project participants can access and manage the latest project information from various engineering services, using different software applications and at different locations.

An infrastructure shown in Figure 4 has been developed using the FICAS model. There are five types of clients and applications involved in the environment. PDA devices are used to access project information via wireless modems, and web browsers provide project information to users who have access to high-speed Internet connections and more powerful computing devices. Three engineering software applications are included to manage the design and scheduling aspects of the project. An Oracle 8i relational database serves as the backbone information storage for this distributed service infrastructure. The active mediator acts as an intelligent bridge that connects various applications and devices with the database. It captures the client requests in the form of active objects from devices such as Palm and desktop browsers. Source queries are constructed and sent to the Oracle 8i database. The active mediator retrieves the information from the Oracle 8i database and conduct client-specific information processing by invoking the mobile classes incorporated in the client requests. The processed information with desired abstraction and suitable format is returned to the clients for display. For example, the construction schedule can be reviewed by using Microsoft Project (Figure 5), Primavera P3 (Figure 6), or a PDA device (Figure 7).

In this application, the engineering software applications all have different proprietary data models for describing project schedule information. By applying a common data model and utilizing FICAS, different applications are able to communicate with each other. The data converters are implemented as mobile classes, which act as bridges to map between the propriety data models and the relational data model, enabling the Oracle 8i database to serve as the backbone information store.

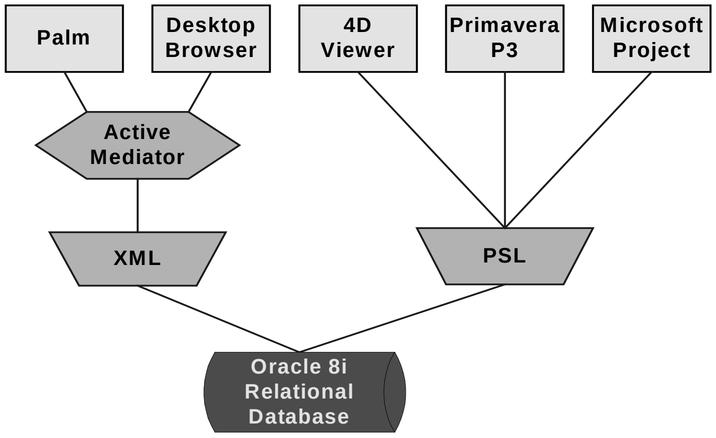
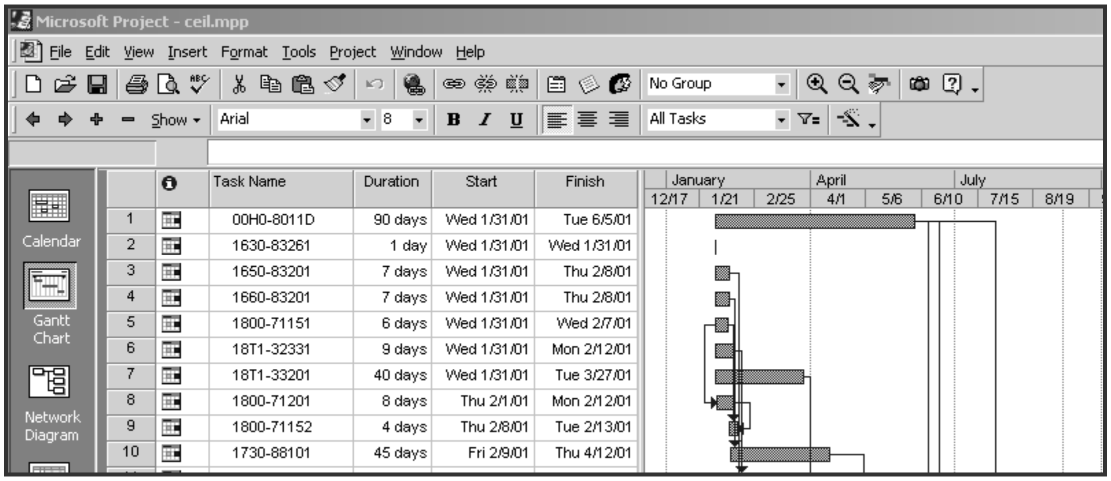
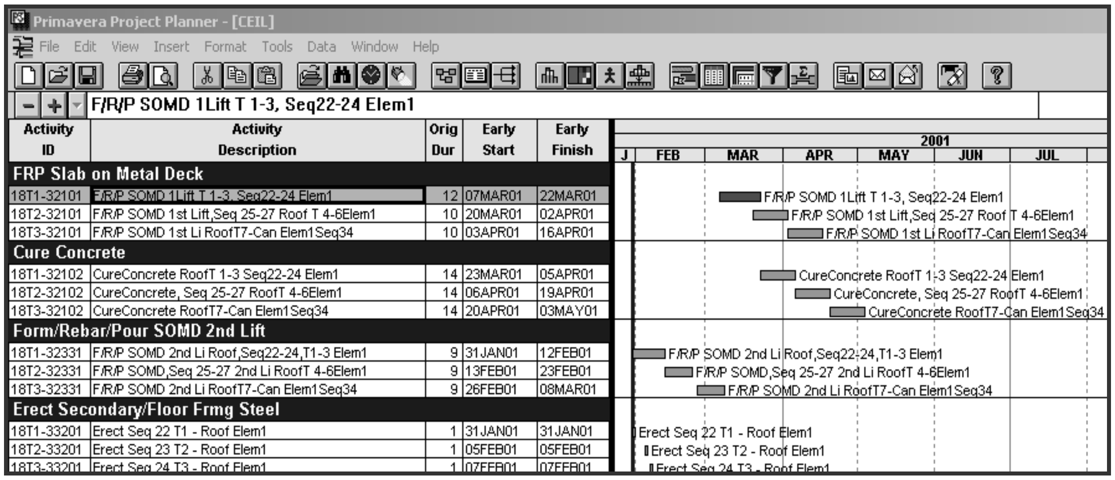


Figure 4: Integration of Various Engineering Services



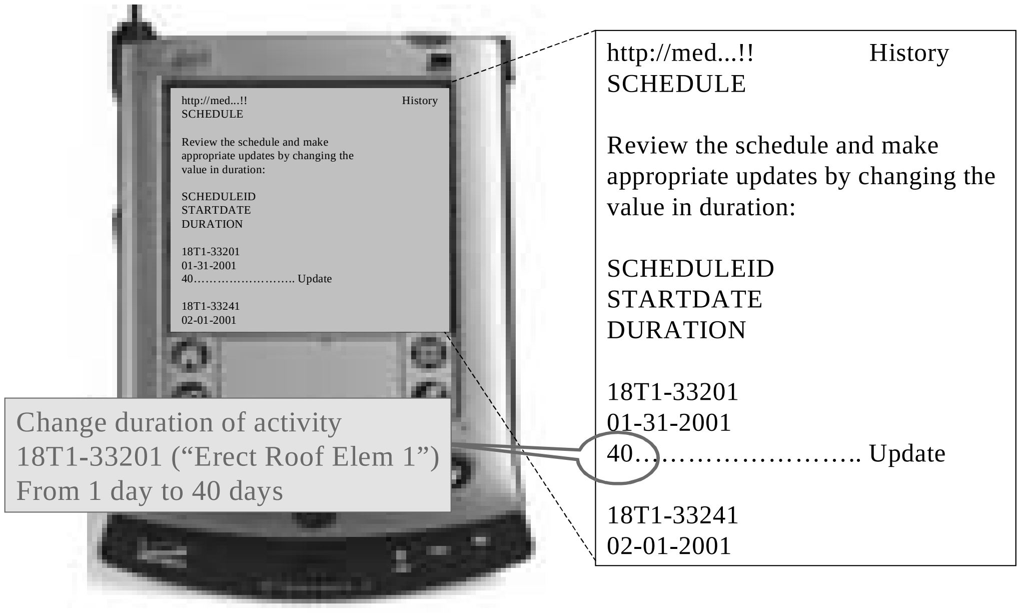
		Task Name	Duration	Start	Finish
1		00H0-8011D	90 days	Wed 1/31/01	Tue 6/5/01
2		1630-83261	1 day	Wed 1/31/01	Wed 1/31/01
3		1650-83201	7 days	Wed 1/31/01	Thu 2/8/01
4		1660-83201	7 days	Wed 1/31/01	Thu 2/8/01
5		1800-71151	6 days	Wed 1/31/01	Wed 2/7/01
6		18T1-32331	9 days	Wed 1/31/01	Mon 2/12/01
7		18T1-33201	40 days	Wed 1/31/01	Tue 3/27/01
8		1800-71201	8 days	Thu 2/1/01	Mon 2/12/01
9		1800-71152	4 days	Thu 2/8/01	Tue 2/13/01
10		1730-88101	45 days	Fri 2/9/01	Thu 4/12/01

Figure 5: Reviewing Construction Schedule in Microsoft Project


Primavera Project Planner - [CEIL]
File Edit View Insert Format Tools Data Window Help
F/R/P SOMD 1Lift T 1-3, Seq22-24 Elem1
2001
J
FEB
MAR
APR
MAY
JUN
JUL

Activity ID	Activity Description	Orig Dur	Early Start	Early Finish
FRP Slab on Metal Deck				
18T1-32101	F/R/P SOMD 1Lift T 1-3, Seq22-24 Elem1	12	07MAR01	22MAR01
18T2-32101	F/R/P SOMD 1st Lift,Seq 25-27 Roof T 4-6Elem1	10	20MAR01	02APR01
18T3-32101	F/R/P SOMD 1st Li RoofT7-Can Elem1Seq34	10	03APR01	16APR01
Cure Concrete				
18T1-32102	CureConcrete RoofT 1-3 Seq22-24 Elem1	14	23MAR01	05APR01
18T2-32102	CureConcrete, Seq 25-27 RoofT 4-6Elem1	14	06APR01	19APR01
18T3-32102	CureConcrete RoofT7-Can Elem1Seq34	14	20APR01	03MAY01
Form/Rebar/Pour SOMD 2nd Lift				
18T1-32331	F/R/P SOMD 2nd Li Roof,Seq22-24,T1-3 Elem1	9	31JAN01	12FEB01
18T2-32331	F/R/P SOMD,Seq 25-27 2nd Li RoofT 4-6Elem1	9	13FEB01	23FEB01
18T3-32331	F/R/P SOMD 2nd Li RoofT7-Can Elem1Seq34	9	26FEB01	08MAR01
Erect Secondary/Floor Frmg Steel				
18T1-33201	Erect Seq 22 T1 - Roof Elem1	1	31JAN01	31JAN01
18T2-33201	Erect Seq 23 T2 - Roof Elem1	1	05FEB01	05FEB01
18T3-33201	Erect Seq 24 T3 - Roof Elem1	1	07FEB01	07FEB01

Figure 6: Reviewing Construction Schedule in Primavera P3



http://med...!! History
SCHEDULE

Review the schedule and make appropriate updates by changing the value in duration:

SCHEDULEID
STARTDATE
DURATION

18T1-33201
01-31-2001
40……………………. Update

18T1-33241
02-01-2001

Figure 7: Reviewing the Project Schedule via a PDA Device

CONCLUSIONS

This paper investigates the integration of web services that communicate large volumes of data. Traditionally, a composed application resides at the central hub that handles all the data traffic, while each web service processes data supplied by the composed application. This centralized data flow is shown to be inefficient when data are substantial. To improve effective use, the distributed data flow approach is introduced which allows direct data exchange among the web services.

The FICAS architecture is defined to enable smooth adoption of distributed data-flow and active mediation in services composition. Active mediation increases the customizability and flexibility of web services. Specifically, it enables interoperation of web services without requiring that heterogeneous data be transmitted via central nodes. It utilizes code mobility to facilitate dynamic information processing in service composition. Delegating the maintenance of software is an important benefit of the services model (Wiederhold 2003). Active mediation allows data-processing tasks to be specified for composed applications, at the same time separating computation from composition.

Used appropriately, active mediation will greatly facilitate service composition, both in functionality and in performance. An application scenario is presented to demonstrate that distributed data flow, combining with mobile classes, is effective and more efficient than centralized processing when integrating large engineering software services. In the example engineering application, the controlling node can run on a low bandwidth device and thus has tremendous effects on performance. Typically, mobile devices are attractive to manage complex scenarios in dealing with governmental regulation (Lau et al. 2004), engineering (Liu et al. 2003), healthcare (Berners-Lee et al. 2001), and military situations (Sheng et al. 1992). These cases can be benefited significantly by distributed dataflow and active mediation model, such as FICAS.

ACKNOWLEDGMENTS

This work is partially sponsored by the Center for Integrated Facility Engineering at Stanford University, and the Product Engineering Program headed by Dr. Ram D. Sriram at NIST (National Institute of Standards and Technology). The authors would also like to acknowledge an equipment grant from Intel Corporation for the support of this research.

REFERENCES

Andrews, T., et al. (2003). "BPEL4WS Specification: Business Process Execution Language for Web Services." (available at http://www-106.ibm.com/developerworks/library/ws-bpel/)

Ankolekar, A., et al. (2001). "DAML-S: Semantic Markup for Web Services." *The International Semantic Web Working Symposium*, Stanford, CA, 411-430.

Banerji, A., et al. (2002). "Web Services Conversation Language (WSCL) 1.0." (available at http://www.w3.org/TR/2002/NOTE-wscl10-20020314/)

Berners-Lee, T., Hendler, J. and Lassila, O. (2001). "The Semantic Web." *Scientific American* 284 (5) 34-43.

Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G. and Rus, D. (1999). "Mobile Agents in Distributed Information Retrieval." *Intelligent Information Agents*, M. Klusch, ed., Springer-Verlag, 355-395.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S. and Weerawarana, S. (2003). "The Next Step in Web Services." *Communications of the ACM*, 46 (10) 29-34.

Fuggetta, A., Picco, G. P. and Vigna, G. (1998). "Understanding Code Mobility." *IEEE Transactions on Software Engineering*, 24 (5) 342-361.

Lau, G. T., Kerrigan, S., Law, K. H. and Wiederhold, G. (2004). "An E-Government Information Architecture for Regulation Analysis and Compliance Assistance." *ICEC'04: Sixth International Conference on Electronic Commerce*, Delft, The Netherlands.

Liu, D. (2003). *A Distributed Data Flow Model for Composing Software Services*. Ph.D. Thesis, Stanford University, Stanford, CA.

Liu, D., Cheng, J., Law, K. H., Wiederhold, G. and Sriram, R. D. (2003). "Engineering Information Service Infrastructure for Ubiquitous Computing." *Journal of Computing in Civil Engineering*, 17 (4) 219-229.

Mohan, C., Alonso, G., Gunthor, R. and Kamath, M. (1995). "Exotica: A Research Perspective on Workflow Management Systems." *Data Engineering*, 18 (1) 19-26.

Sheng, S., Chandrakasan, A. and Brodersen, R. W. (1992). "A Portable Multimedia Terminal." *IEEE Communications Magazine*, 30 (12) 64-75.

White, J. E. (1997). "Mobile Agents." *Software Agent*, J. M. Bradshaw, ed., MIT Press, 437-472.

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

Wiederhold, G. (2003). "The Product Flow Model." *15th Conference on Software Engineering and Knowledge Engineering (SEKE)*, Skokie, IL, 183-186.