Peer-To-Peer architectures using Asynchronous Web services

Francisco Ferreira **Carlos José Pereira de Lucena** **Daniel Schwabe**
e-mail: {chico,lucena,schwabe}@inf.puc-rio.br
PUC-RioInf.MCC02/03 January, 2003

Resumo: Neste artigo, descrevemos Everyware, uma plataforma para aplicações peer-to-peer baseada nos padrões de Web Services. Estar plataforma foi concebida para dar suporte ao compartilhamento de informações pessoais entre parceiros usando diferentes dispositivos e estruturas de hardware. Através do uso de padrões abertos e de uma abordagem totalmente distribuída, Everyware permite que os usuários finais possam disponibilizar seus serviços pessoais sem necessidade de configuração ou administração. Uma comparação com iniciativas similares é feita, levando em consideração os principais problemas recorrentes em arquiteturas peer-to-peer e na composição de Web services. Uma aplicação real baseada na plataforma Everyware é apresentada e os principais aspectos de implementação são discutidos.

Palavras-chave: Arquiteturas Peer-to-Peer, Web Services, Components, Composição

Abstract: In this paper, we describe Everyware, a platform for peer-to-peer applications based on Web services standards. It is conceived to support personal information sharing among peers using different devices and hardware infrastructure. Using open standards and a completely distributed approach, Everyware allows end users to deploy personal Web services easily without requiring configuration or administration efforts. A comparison with similar initiatives is made, centered on major issues of peer-to-peer architectures and Web services orchestration. A real application based on Everyware is presented and related implementation issues are discussed.

Keywords: Peer-To-Peer Architectures, Web Services, Component–based Software Engineering and Component Orchestration.

1. INTRODUCTION

Enterprise application integration is one of the most challenging issues of Internet computing. Several technologies have been developed to face this problem such as OMG's CORBA, Microsoft's COM/COM+ and Sun's Enterprise JavaBeans [6,20].

However, with the increasing computing power of different new devices for accessing the Internet, information is more distributed across heterogeneous sources, requiring new approaches to application integration.

Web services and its set of related standards are a promise for a completely platform-independent, open standards-based way of integrating enterprise components using existing software and hardware infrastructure [24].

Most platforms for Web services development and deployment (i.e. Microsoft's .Net and IBM's Websphere) provide full integration of Web Services technologies with existing Web Servers and Application Servers. This allows developers and IT managers to make their components available as services that reuse the same hardware and software infrastructure.

On the other hand, these platforms rely on application servers that are hard to administrate and usually require specialized professionals for corrective maintenance. They also focus on the use of synchronous protocols (mainly HTTP), limiting scalability and computing power.

Ordinary users who want to expose their personal data as Web services need simpler platforms with low memory and processor fingerprints and self-tuning, easy to configure capabilities.

In this paper we present Everyware, a platform for deploying and orchestrating distributed Web services that meets the above requirements, leveraging enterprise information integration by using existing software infrastructure and both synchronous and asynchronous protocols. Everyware has been implemented based on open standards and it is fully platform and programming language-independent.

Another important feature of Everyware is that it reuses existing software infrastructure to access devices with low availability by using asynchronous protocols.

The remainder of the paper is presented in four sections. Section 2 describes how asynchronous protocols are fundamental to assure scalability and failover, demonstrating how they fit to provide access to distributed data sources. Section 3 discusses some related activities and Section 4 presents our proposed platform for both synchronous and asynchronous web services orchestration. Finally, Section 5 discusses our conclusions and future work.

2. THE NEED FOR DISTRIBUTED AND ASYNCHRONOUS WEB SERVICES

Nowadays, there are several tools to support Web services development and deployment. These tools enable service providers to expose code written in several programming languages (such as Java, C++, Visual Basic and C# among others) as a Web service accessible through an invocation protocol (typically SOAP) and publish it in UDDI registries [16].

A common feature is to allow service requesters to invoke Web service by generating proxies [9] which also accelerates service integration. Although most of them provide some kind of support for asynchronous protocols (generally SMTP [23]), there are few examples of real applications that fully explore the advantages of asynchronous protocols over synchronous approaches (which are usually based on HTTP [13]).

Some tools do not offer asynchronous execution models as they share the same software infrastructure with synchronous protocols, which limits scalability and computing power.

2.1 Distributed Information Sources

On the Web, data is naturally distributed across different information sources. Each user has personal application data that could be available to other users and applications.

Common shareable data includes calendar information, contacts, files, address books, documents and e-mail systems. Traditional approaches to knowledge sharing and management usually create repositories of information provided by end users. After some processing and formatting phases, data is published and can be accessed by specific software clients, which in most knowledge management systems are Web browsers.

Although easy to implement, this approach introduces some extra activities to end users, since each one must somehow upload his/her personal data to the shared repository. Davenport [4] points out that information sharing tools should not demand additional work from users.

Personal data should be available to other users and applications in a natural way, requiring minimum user interaction and setup. Also, its representation should not be tied to any application specific protocol, allowing composition of information regardless of syntax and format. For example, calendar information from Microsoft Outlook™ or IBM Lotus Notes™ must be accessed the same way. Personal information must be kept in each user's device.

Centralized approaches raise several digital rights, privacy and security issues. Users do not feel comfortable when they leave important, personal information under third party custody.

Information sharing systems must search for information in real-time, dealing with unavailable sources, network failures and delays.

2.2 Multiple Devices

Computing is becoming more pervasive and the Web tends to be device-centric, allowing users to be connected from anywhere through different devices ranging from normal PC-like computers to handhelds and mobile phones.

These devices have different forms of connection to the Internet. Some of them are far away from 100%-availability, mainly due to disconnections from the network. Mobile phones may cross uncovered and shadow areas, PCs tend to be booted from time to time and handhelds might be turned off.

Synchronous approaches tend to limit access power to these information sources. They create a bottleneck that can be extremely inefficient in the context of a distributed environment. Asynchronicity can be seen as a generalization of synchronicity, since the later can be derived as a special case of the former [1].

The main issue in connecting multiple heterogeneous devices is how to integrate these machines into a peer network taking into account their limitations of memory usage, processing power and network availability. For example, high-performance enterprise services that have support teams to manage configurations and guarantee service-levels should perform more significant tasks than a simple mobile phone that has less than 1Mb of memory and disconnects from the network more than five times a day [19].

To fully explore the potential of peer-to-peer architectures, it is important to deliver personalized information and infrastructure that takes into account the profile of each device and its associated hardware features and limitations. Different profiles lead to architectures with some sort of hierarchy, where devices with higher computing power (i.e. enterprise servers) have more authority and coordination tasks than ordinary peers (i.e. mobile phones or normal PCs).

2.3 Peer-to-Peer Architectures

Bandwidth growth has allowed devices to exchange information on a larger scale. Peer-to-Peer architectures are an evolution of the traditional client-server model where every component (peer) can request (as a client) a service or serve (as a server) in the same way.

Peer-to-peer architectures enable more systems distribution, providing scalability and flexibility. By using a peer-to-peer approach, developers can distribute computational effort among different machines connected by network facilities, thus providing better use of available computational resources.

"Pure" peer-to-peer architectures enable peers to share resources directly with each other, without any central authority or server [21].

However, most real peer-to-peer systems (i.e. ICQ™, Napster™ and Groove™ [10, 14, 18]) use some sort of central authority to coordinate peers, at least partially. P2P servers enable caching of data and robustness, avoiding unnecessary message exchange and enabling fast-executing processes.

Hewitt and Agha [12, 1] have proposed the concept of Actors as a conceptual model for concurrent programming. Actors are asynchronous objects that execute concurrently in distributed systems and communicate among themselves by sending messages.

Web service-based peers can be seen as actors that can send and receive messages in order to accomplish a specific task. As they can be accessed asynchronously, the order of message arrival is nondeterministic and must be coordinated somehow.

This introduces the need for some kind of coordination and synchronization mechanism that assures the correct processing order in complex processes.

2.4 BPEL4WS – Using a process execution language to orchestrate peer services

The Business Process execution for Web Services Language (BPEL) [3] is designed to define business process behavior based on web services.

Using workflow approaches to coordinate several services is natural way to orchestrate services [8, 24] since most of them perform simple tasks that must be combined with others to produce valuable results to end users.

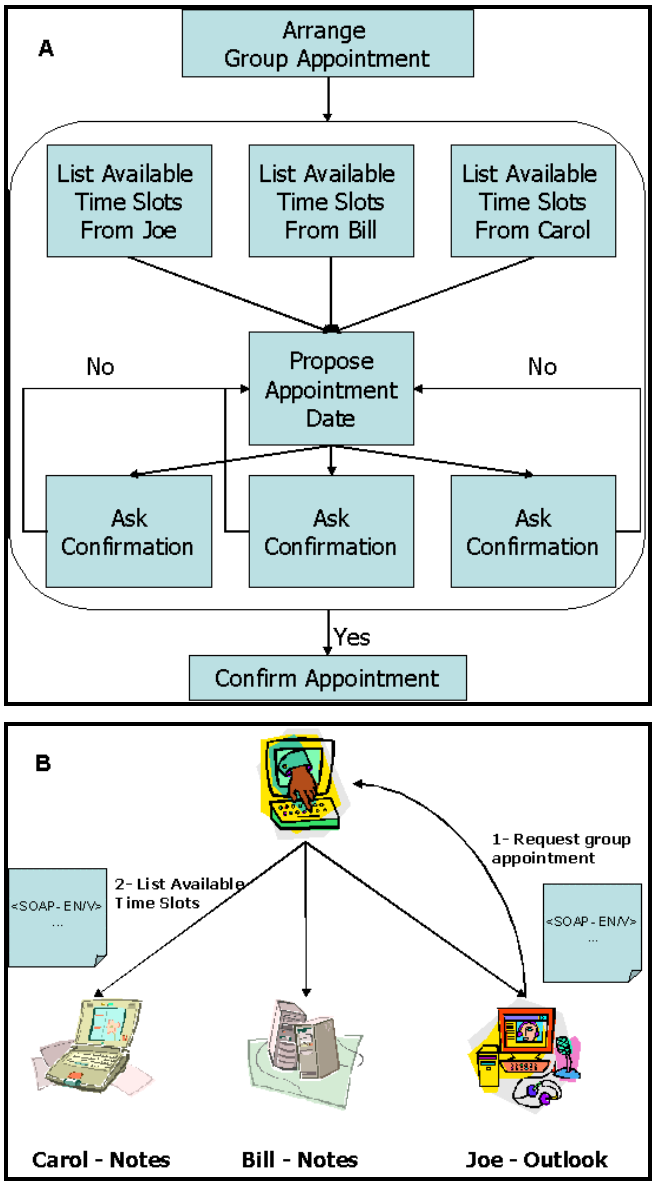

Figure 1 – (a) A sample process specification by workflow (b) Joe has started a process with the aid of a process coordinator.

For example, to setup an appointment, a calendar service must access the agenda of each guest. Considering that each one will be available as a service accessible on the network, a calendar process will have to exchange several messages with other peers in order to find out the time slot that best fits the desired appointment.

Process specification will define a set of coordination constraints that will guide the entire process and will enable correct process orchestration. This task is usually done by one single entity of the system that is responsible for keeping its consistency. A process specification can be seen as a set of constraints that define a clear order of actions and messages that must be followed to assure correct process execution. Fr0lund [7] has introduced the concept of a synchronizer, which is responsible for assuring that messages exchanged by Actors will follow synchronization constraints.

Similar to synchronizers, Web Services coordinators must guarantee correct service invocation and response order by following rules specified by a process specification.

The main goal of Web Services coordination is to reduce code needed to orchestrate multiple Web Services. Synchronizers have proved themselves a good mechanism to reduce code size by an order of

magnitude [7]. The use of coordination services seems to be an interesting approach to allow direct service invocation without the need for specific wrapping code.

3. RELATED WORK

Although Everyware can be used to orchestrate any kind of service-oriented process, it is focused on connecting personal information services that might be hosted anywhere by any sort of device. Some other initiatives address the same problem and are discussed here in order to put the Everyware design principles and decisions into perspective. A comparison of these initiatives is provided in section 4.4.

3.1.1 Microsoft's My Services

As part of its .NET™ initiative, Microsoft™ has announced the development of My Services, a set of basic web services available 24 hours/day.

My Services platform would host the personal information of users, allowing application developers to access data to personalize their applications.

My Services' major shortcoming was its Application Service Provider (ASP) approach. Users would have to "leave" personal information under Microsoft's custody, something that raises several privacy questions besides other access limitations. Any network disconnection of Microsoft's My Services servers would leave users "isolated" from their personal data and services.

Initial services provided by Microsoft with .Net My Services include: profile, calendar, inbox, application settings, documents and presence. A simple .Net My Services-based application could allow users to get personal documents from a mobile phone using their document service and send them to friends or colleagues.

Authentication to .Net My Services should be done with Microsoft Passport. Users needed to be accredited to access a specific service.

.Net My Services introduced HSDL (My Services Data Manipulation Language). HSDL primarily is focused on transporting data in and out of .NET My Services service documents, but its design and syntax reflect two other underpinnings of .NET My Services: security and data structure [17].

Each service is made up of key XML nodes that hold more relevance than others. My Services interactions involve exchanging SOAP messages containing XML data based on HSDL.

XML-based languages allow easy representation of data provided by the service. Nevertheless, it implies extra XML-processing on service requesters.

3.1.2 Apple .Mac

Apple .Mac is a service provided by Apple™ that allows Macintosh™ users to share data over the Internet. Some Mac applications like iCal and iDisk are able to publish data on .Mac automatically.

Users can send URLs containing personal information through e-mail. These URLs are generated by .Mac when the user publishes his/her personal data on the web.

One severe limitation of the system is the interface to personal information. .Mac services are presented in HTML pages and must be accessed using HTTP. It does not allow any other application to access resources using any kind of middleware infrastructure. Information is just presented to end users on .Mac's Web site. Other peers do not have access to application components, being unable to update information or coordinate process among different machines.

3.1.3 JXTA

WWW 2003, May 20-24, 2003, Budapest, Hungary.
ACM xxx.

JXTA is a research project led by Sun Microsystems that aims to develop a Peer-to-Peer language-independent platform Architecture. It aims to create a common platform that makes it simple and easy to build a wide range of distributed services and applications in which every device is addressable as a peer, and where peers can bridge from one domain into another [21].

It does not provide any application; it is just the infrastructure for building P2P networks.

The JXTA core consists of a set of components that allows the creation, management and monitoring of peer groups and communication channels. Using this basic set of components, application developers can create peer-to-peer environments without having to worry with security, network protocols, message exchange mechanism and other basic P2P features. It is based on a set of protocols that are completely asynchronous [15]. It comprises a complete peer-to-peer infrastructure that is able to find and share resources among JXTA peers.

Information sharing over JXTA networks is made possible by several protocols for message exchange among peers. Although every JXTA protocol is made public, they are not considered industry standards as Web Services standards are. This limits interoperability with software provided by other vendors, which is a major drawback.

One important feature of JXTA is its security layer. It isolates security implementation details, providing secure communications without introducing additional effort to application developers.

4. EVERYWARE: A PLATFORM FOR PEER-TO-PEER WEB SERVICES

Everyware is a platform for distributed peer-to-peer Web services coordination. It allows the definition and composition of standard services available over the Internet.

Based on open standards, Everyware encompasses all of the parts needed to set up a peer-to-peer Web services network. There are two basic components: a lightweight Web services provider (Everyware LW) and a coordination infrastructure built on top of a J2EE™ Application Server. Users can deploy personal Web services using the lightweight provider, with very limited configuration or administration effort.

The coordinator infrastructure is responsible for managing the execution of a process by coordinating several distributed web services. Although it could be installed on any peer, it is more straightforward to host it on a separate machine. It provides high-availability and computing power that may act as a coordinator peer.

Ordinary peers (i.e. normal PCs or mobile phones) may provide services using only asynchronous protocols. Everyware lightweight server implements a transport layer based on SMTP. It works as a mail proxy for the actual mail server of the user, allowing requests to be sent to the same mail account used for common e-mail messages. Everyware processes service requests while ordinary mail is forwarded to the mail client program (i.e. Microsoft Outlook™ or Eudora™). Synchronous support also is possible with the HTTP transport. Figure 2 depicts the information flow of a simple service request to an ordinary peer.

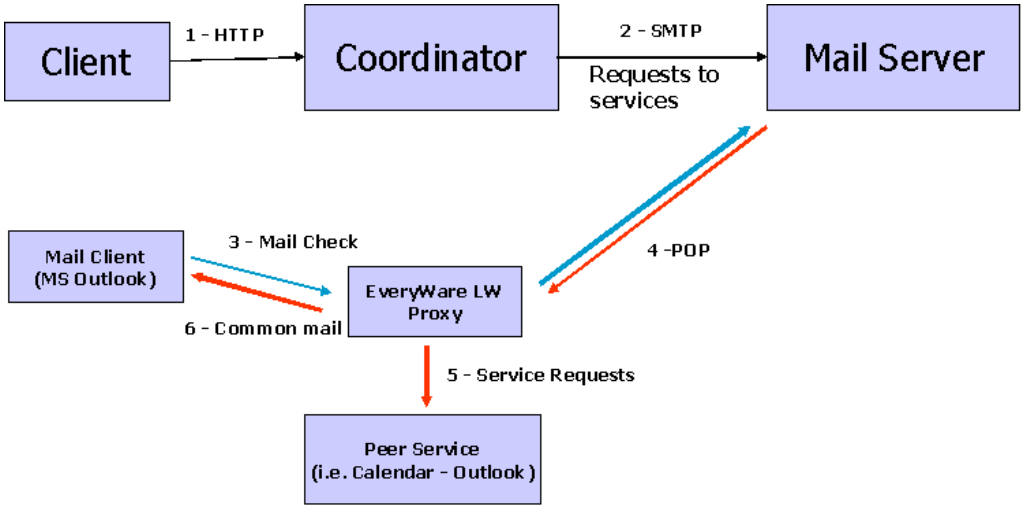

Figure 2 - Everyware LW information flow. A service client issues a request to a coordinator that invoke services to execute a process. Everyware LW stands as a proxy to the mail service, processing service requests when the service owner checks for e-mail using an ordinary mail client.

4.1 Architecture

Everyware uses a central coordinator to execute a business process. This means that applications based on Everyware must implement a coordinator service that must be invoked in order to execute more complex business logic.

The main goal of this approach is to provide a centralized component that has more computing power and can act as a manager of the entire process. Any peer on the Everyware network can act as a coordinator, as it is implemented as a Web service that could be deployed using Everyware LW. Coordinators can be seen as a variation of synchronizers [7].

However, practical experience during the implementation of the calendar service showed that having a "coordinator peer" installed on a more powerful machine enhances system performance and failover. This is contrary to the more traditional P2P vision, but seems to be more reasonable since there are considerable performance differences among the devices involved.

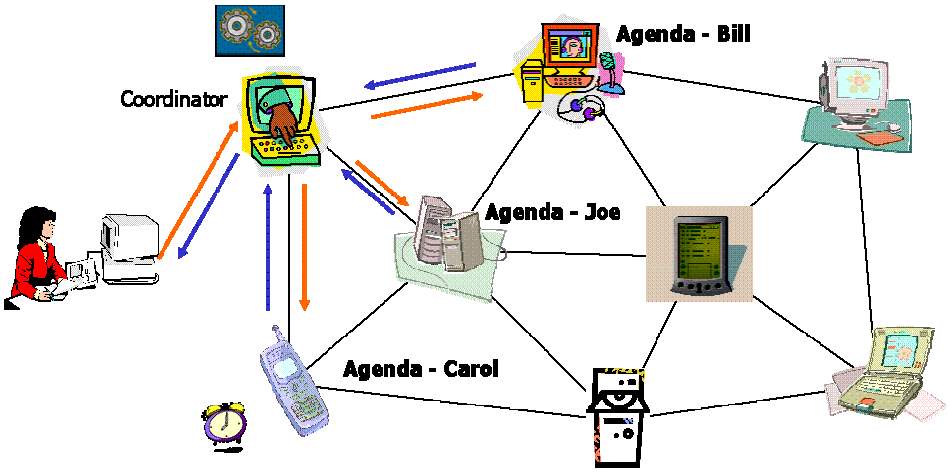

Figure 3 – A simple execution of the calendar service. After being requested, it invokes each individual agenda and recovers appointment information. The clock near Carol's mobile phone illustrates the asynchronous behavior of that service during the entire coordination process.

Every request is received through some transport protocol. A transport layer processes the request, extracting the SOAP call following the different transport protocols. The call is then forwarded to the

Service Management layer, which parses XML content and then creates a context for the request that will be used to coordinate the process. Context information is then passed to the coordination layer, which establishes a transaction identification number that will be used to follow subsequent request within the same process context. Figure 4 depicts XML data processed in each application layer.

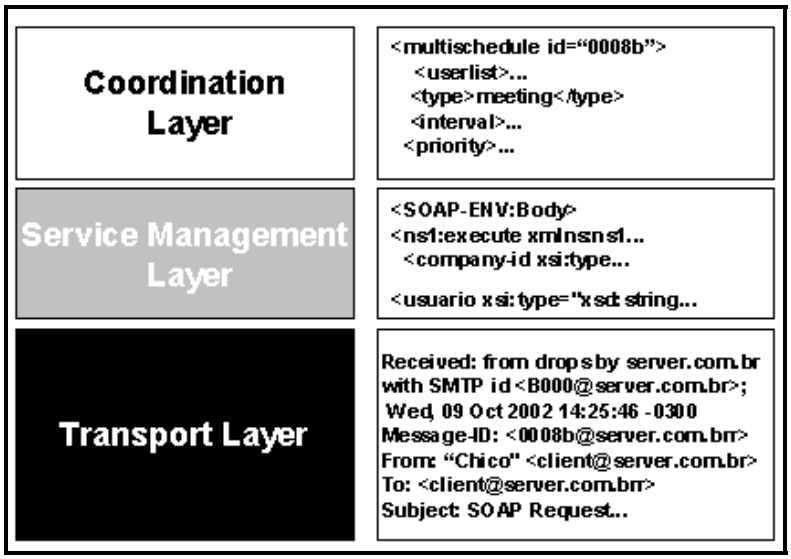

Figure 4 – Everyware layers and associated XML content.

4.2 Implementation and Design Issues

Everyware was entirely developed in Java™. However, each component of the architecture can be developed in a different language, since communication is conducted using standard Web services protocols. Two components have been developed and are necessary to establish an Everyware peer network:

- A Lightweight stand-alone Web services provider. It can work with both synchronous (HTTP) and asynchronous (SMTP) modes. The second can check-up for mail requests in a fixed time interval or when a user checks for his/her mail (see Figure 2 in Section 4). Everyware LW allows users to deploy any kind of Web service. Service can be Java classes, Enterprise JavaBeans or COM components.
- A Coordination API that can manage complex business processes. It allows the execution and coordination of long-running processes using both synchronous and asynchronous calls to services.

Several development and design issues have been raised during the development of Everyware.

The most important challenge of peer-to-peer Web service orchestration was service heterogeneity. Besides integration of different software platforms using standard protocols (functional integration), there is also a need for data integration. For instance, each source may represent data in a different way, so it is important to define a standard way of representing knowledge.

For example, in our case study, most calendar applications define an appointment with a start-end date tuple, while others use start date-duration to define appointment interval. Another common problem is the granularity of an appointment. Some calendar systems only allow appointments that start and finish the same day while others allow appointments spanning several days. In this case, a request for a single long-duration appointment must be converted into a series of requests for short appointments that span the same time period. These differences had to be wrapped inside the services' implementations. Since

Web services standards lack support for semantic representation of data, such problem seems to be recurrent.

The lack of support for session information also presented a challenge to the calendar service application. There are few standard mechanisms to hold and transfer session information across web service calls. Since web services are naturally loosely coupled, services providers need to persist-and-restore session and authentication information in every call. This represents an extra processing need in each service call. As a consequence, clients had to provide login information every time they invoke a specific service. To overcome this, a caching mechanism has been created, storing in memory recently requested information. Since process execution tends to have highly interactive message exchange, user information does not need to be retrieved from a file or database every time a request arrives. This mechanism also has the advantage of improving performance of database accesses.

From a performance point of view, it was necessary to create a caching mechanism to avoid unneeded database accesses from application clients during highly interactive Web services-based processes.Security support was not developed in the first version of Everyware. Every peer is trusted by default.

Limiting unnecessary peer activities also was an important design issue. SMTP-based peers were implemented as mail server proxies. Requests are processed with normal mail messages, without requiring any extra mail accounts or mail checks. Since peers tend to have lower traffic than normal Web Service-enabled Application Servers, Everyware LW is much simpler than other Web Service architectures. Common HTTP-based Web service providers require a Web Server to be installed, configured and managed. SMTP ones check for mail messages on a regular time interval. Everware LW processes requests when the user checks for e-mails. This normally happens when he or she accesses the Internet to perform some task. If a service needs any kind of human interaction to be performed, there is a high probability that the user will be able to do it just in time.

4.3 Case study: The EveryCal Calendar Service

Scheduling meetings and appointments still is a surprisingly difficult task. Integration of different calendar systems is weak and it is virtually impossible to automatically schedule meetings with guests that use different calendar systems except for the most trivial cases.

Suppose John, who works at company A and uses Outlook™ as his mail client, wants to set up a meeting with Carol and Bill from company B. He wants a two-hour meeting that must be held on the week of May 24th 2003. Unfortunately, company B has adopted Lotus Notes™ as its calendar service. Besides having to integrate calendar databases from both systems, it is necessary to exchange several messages to find the best time slot for the meeting. This implies the existence of Web services that provide a series of calendar information for the different calendar systems. The EveryCal Calendar system uses Everyware platform and coordination mechanisms to perform this task with minimum human interaction.

To allow simple group calendar management, EveryCal has as a primary requirement the need to integrate different calendar systems such as Microsoft Outlook™, IBM Lotus Notes™ and Macintosh iCal™.

After defining a standard Calendar service that was able to setup appointments and to list already defined appointments it was necessary to expose the standard interfaces provided by software vendors as calendar services. Each service was then deployed using the Everyware Lightweight SOAP Server, which is triggered by requests sent by SMTP or HTTP.

Finally, a coordination calendar service was written to allow group calendar management. A majority consensus algorithm was used to set up group appointments. Future versions may mix different

coordination services that use different algorithms to arrange appointments. Applications may use the coordination framework to instantiate different coordination strategies, varying the algorithm used to setup appointments.

When a user wants to setup appointments with a group of users, he or she invokes the calendar coordinator service to start a multi-user appointment scheduling. The coordinator is responsible for orchestrating different distributed calendar service implementations.

A Web-based client of the calendar coordinator service is also implemented, allowing users to orchestrate multiple calendar services from a single application. Users can check their timetables and request group appointments using EveryCal's web site. Figure 5 shows some screens from EveryCal web-based client.

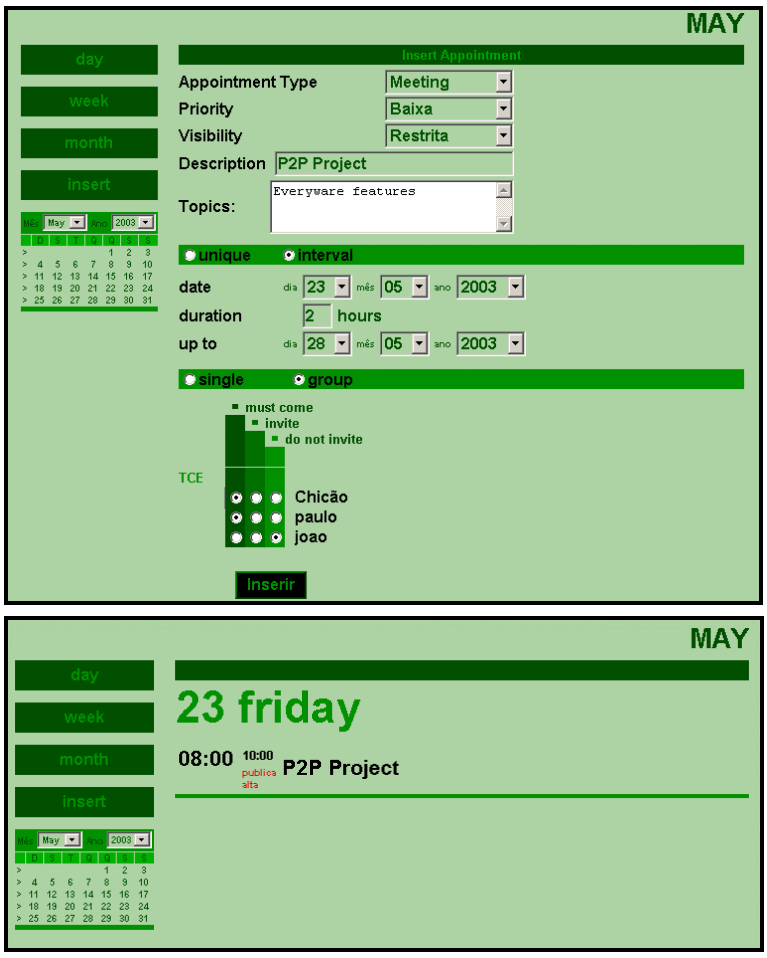

Figure 5 – EveryCal in action. A user arranges an appointment with a work group. Information is updated on each peer automatically using calendar services.

Using the Everyware platform, any peer could host the calendar coordinator service. However, using high-performance servers to orchestrate different services enables more fast-executing processes, since peers with low availability may take a very long time to process responses to service requests and proceed with process execution due to disconnections from the network.

4.4 Comparison

Everyware proposes a new approach to information sharing and can be compared to other personal services initiatives. Good information sharing infrastructure must be scalable, easy-to-use and configure, portable and distributable. These properties guarantee that end users will be able to share information among themselves regardless of the type of software or hardware they use daily.

To evaluate Everyware's main features, other personal information sharing systems have been analyzed and tested. Four features have been verified:

- variable distribution of personal data
- reuse of existing software and hardware infrastructure
- interoperability with other platforms
- provision of mechanisms for developing clients of personal services.

Some design principles must be followed to provide these properties. Support for distribution assures that developers can access personal data from anywhere. Use of open standards and platform independence allow integration with existing software and hardware and interoperability. Asynchronous support and composition enables easy development of personal service clients.

Table 1 presents a comparison of four different personal data sharing initiatives. It compares them in terms of distribution, compliance with open standards, support for asynchronous communication and platform independence by presenting some design principles of each approach.

Table 1. Comparison of related initiatives

Criteria	Apple .Mac	JXTA	MS My Services	Everyware
Based on open standards	No	No	Yes	Yes
Standardized Composition of different services	No	No	Yes	Yes
Asynchronous support	No	Yes	Yes	Yes
Platform-Independent	No	Yes	No	Yes
Support for distribution	No	Yes	No	Yes
Security Support	Yes	Yes	Yes	No

Apple's .Mac is not oriented towards service integration. Although it offers some nice mechanisms for other users to view personal information from web sites, it does not contain mechanisms that integrates them with other applications.

In contrast, JXTA provides most of the features needed to integrate applications and peer information. Its major drawback is the absence of a coordinating mechanism, something that makes developers write extensive code to create process execution mechanisms. Non-compliance with de facto industry wide standards also is a problem.

On the other hand, Microsoft My Services is based on Web Services standards, which eases integration with other web services products. However, its centralized approach has major drawbacks as previously discussed.

Everyware aims to combine the best of each initiative, being compliant with open standards, platform-independent and providing support for composition and execution of complex distributed processes.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented Everyware, a platform for peer-to-peer Web services orchestration. It overcomes some limitations of most Web services platforms through the use of asynchronous protocols and lightweight components.

Everyware enables end users to share their personal data as web wervices without having to configure and manage complex application servers.

The implementation of Everyware has raised several questions about the Web services architecture. Service-oriented applications still need more standardization efforts to allow the development of real-world scalable applications.

Our experience shows that most Web service standards lack support for session management and data representation. The need for asynchronous protocols also was clear since most peers do not offer 100%-availability.

Support for semantic description and integration of data is envisaged in future versions of Everyware. Nevertheless, it is important to point out that Web services standards offer little support for data description and representation. Data integration involves semantic tagging of content, allowing machines to process it automatically. Semantic Web [2,5,11,25] initiatives are the most obvious approaches to overcome these limitations. Ontologies that represent data from a specific domain can be used to solve inconsistencies and ambiguities on service requests and responses.

Future web services implementations will be able to receive semantic information as input, process it using one or more ontologies and send back a response that is also semantically tagged.

Everyware intends to achieve this goal by extending coordination and service description languages in order to express the semantics of exchanged data among peer over the network.

A real-world peer network based on Everyware needs to contain some security mechanisms to avoid malicious behavior from untrustworthy peers. A digital signature infrastructure is being conceived to assure non-repudiation from any part. Message encryption also is expected to guarantee that third parties will not read messages.

Support for different devices is planned. A J2ME version of Everyware LW will enable low-capacity devices to provide services to other Everyware peers.

Finally, Everyware is expected to incorporate a series of standard personal services such as file sharing, e-mail, database sharing, presence and contacts that can help users manage their information and content from any device connected to the Internet.

6. ACKNOWLEDGMENTS

This work was partially supported by scholarship from CAPES, and grants from CNPq. Special thanks to Frederico Guimaraes for the help on the calendar service implementation and to Gustavo Carvalho for his contribution to the coordinator architecture.

7. REFERENCES

[1] Agha, G. Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge, MA, 1986.

[2] Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. Scientific American, May 2001.

[3] BPEL Business Process Execution Language for web Services Specification [online] http://www-106.ibm.com/developerworks/library/ws-bpel/

[4] Davenport. T., and Prusak, L. Working Knowledge. Harvard Business School Press, 2000.

[5] Dumbill, E., "The Semantic Web: A Primer" [online] http://www.xml.com/pub/a/2000/11/01/semanticweb/

[6] Enterprise JavaBeans Technology [online] http://java.sun.com/products/ejb/

[7] Fr0Lund, S. Coordinating Distributed Objects. An Actor-Based Approach to Synchronization. MIT Press, 1996.

[8] Fremantle, P., Weerawarana, S., Khalaf, R. Enterprise Services. Communications of the ACM Volume 45, Number 10, October 2002, 77-82.

[9] Gamma, E. et Al. Design Patterns. Element of Reusable Object-Oriented Software. Addison Wesley, 1995, 207-217.

[10] Groove™ Networks [online] http://www.groove.net

[11] Hendler, J., "Agents and the Semantic Web" [online] http://www.csumd.edu/users/hendler/AgentWeb.html

[12] Hewitt, C. Viewing Control Structures as Patterns of Passing Messages. Journal of Artificial Intelligence, 8(3):323-364, 1997.

[13] HTTP. Hypertext transfer protocol. Internet Engineering Task Force [online] ftp://ftp.isi.edu/in-notes/rfc2616.txt

[14] ICQ™ [online] http://www.icq.com

[15] JXTA v 1.0 Protocols Specification [online] http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html

[16] Kreger, H. Web Services Conceptual Architecture. IBM Software Group White Paper, May 2001.

[17] Microsoft Corporation, Microsoft® .NET My Services Specification. Microsoft Press, 2001, Chapter 2

[18] Napster™ [online] http://www.napster.com

[19] Orlean, D., Ferreira, F., Lucena, C. Everyware: Dealing with e-commerce Pervasiveness. Proceeding of the International Conference on Internet Computing. Las Vegas, 2001.

[20] Pritchard, J. COM and Corba Side by Side, Addison-Wesley, 1999.

[21] Project JXTA: An Open, Innovative Collaboration. [online] http://www.jxta.org/project/www/docs/OpenInnovative.pdf

[22] Sametinger, J. Software Engineering with Reusable Components. Springer-Verlag, 1997

[23] SMTP. Simple Mail Transfer Protocol. Internet Engineering Task Force. [online] http://www.ietf.org/rfc/rfc821.txt

[24] Stal, M. Web Services: Beyond Component-Based Computing. Communications of the ACM Volume 45, Number 10, October 2002, 71-76.

[25] Swartz, A., "The Semantic Web in Breadth" [online] http://logicerror.com/semanticWeb-long