Generic Peer-to-Peer Support for a Personal Service Platform

Fredrik Espinoza Lucas Hinz

Swedish Institute of Computer Science
Box 1263, 164 29 Kista, Sweden
Telephone: +46 8 633 1500
{espinoza, lphinz}@sics.se

Abstract

Building on previous work of the OASIS group at SICS, this paper presents a generic peer-to-peer system for the sView personal service platform. The sView platform provides each user with a personal service briefcase from which services may be reached using a variety of devices and interfaces. With the peer-to-peer system, called Briefcase Connectivity, we leverage the community of all sView users, the ultimate purpose of which is to simplify the propagation of cooperating services, to enable users to become individual service providers, and to bring to the users a greater number of specialized services. We suggest that sView with Briefcase Connectivity and today's electronic services demonstrate a viable model to make a step toward tomorrow's ubiquitous computing.

1 Introduction

On a daily basis people use a wide range of *electronic services* to enhance their lives—to communicate with friends and colleagues, to make travel reservations, or to access information. Electronic services may be found in many different networks: on the World Wide Web, in telecom operators' networks (e.g. voice mail), in home area networks (e.g. alarm control and home entertainment systems), or in interactive digital cable and TV networks. Unfortunately, the design of electronic services seldom focuses on the user, but instead focuses on the service provider or the devices used to access the services [5]. Recognizing this oversight, the OASIS group at SICS[1] developed the concept of a Personal Service Environment, or PSE [3]. The PSE is intended to make the use of electronic services easier for the user, and for this reason focuses on increased user control, enhanced service interoperability, and support for continuous, ubiquitous access to the user's whole set of services on a wide variety of devices. Conceptually, a PSE is an individually collected and tailored set of services, available to the user at all times, and at least partially independent of Internet access. The reference implementation of the PSE, called sView [4], is a virtual *briefcase* that serves as a storage and execution environment for electronic services.

SView is equipped with a number of mechanisms that mitigate the design of new services and facilitate service interoperability. For example, the *ServiceDesigner* service [6, 8] enables the user to collect and combine Web Services[2] into a single, collaborative service unit, furnish it with a graphical user interface, and generate a fully qualified sView service for use in the user's briefcase. This gives the user the freedom to fill his briefcase with services catered to his needs.

1.1 Motivation

Given the creative freedom that the sView architecture provides, it is conceivable that the body of sView developers and users in a near future will generate a large number of different services. Based on the success of today's popular peer-to-peer file sharing applications, it is obvious that sView users would benefit from a network in which they could share their services with each other. In one of our scenarios entitled *Individual Service Provisioning* each user can be a service provider. Using a tool like the ServiceDesigner the user creates a service and then shares it with other users in the sView community. This is the primary motivation for the present work.

But peer-to-peer computing is advantageous in other areas besides service sharing, exemplified by the multitude of systems that utilize peer-to-peer[3]. Ranging from instant messaging applications to wide-area storage architectures, new systems are being developed that incorporate peer-to-peer in their design. These systems leverage peer-to-peer to

[1] Swedish Institute of Computer Science, http://www.sics.se.

[2] http://www-106.ibm.com/developerworks/webservices/.

[3] In this paper, the terms peer-to-peer, peer-to-peer computing, and P2P denote the same concept.

enhance system performance, increase service availability, and to guarantee user anonymity.

To cater to both of these situations, we wish to extend the functionality of the existing sView platform with a generic peer-to-peer based communications layer. This capability, in the form of a service called *Briefcase Connectivity*, will bring sView users together in an always-on network of sView briefcases.

With Briefcase Connectivity it will be possible to design services that leverage the community of sView briefcases including, but not limited to: information and resource sharing services; collaborative services such as bulletin boards, chat rooms, and auctions, and other services that include social mechanisms, such as trust chains, real-time ratings, etc.; and distributed services that tackle complex problems.

1.2 Background

Two developmental trends characterize what is happening to the Internet today. First, the Internet is becoming increasingly service-oriented. Nearly every public and private institution has a web site today, and most of these give the user access to some type of service—to manage bank accounts, to book hotel reservations, or to search a directory for an address. Another indication that the Internet is becoming service-oriented is the Web Services movement, led by Ariba, IBM, Microsoft, and others. Web Services are intended to make web sites more modular, allowing service providers to make their web sites programmatically accessible. That is, service providers will have access to service functionality via the Web, facilitating the design of new services.

With the increased focus on *service oriented computing*, we argue that an environment like sView will be beneficial to users, as a unifying environment for all services. Thus, we here introduce the main concepts of PSEs and sView for the purpose of giving a backdrop to the present work on Briefcase Connectivity. For more details regarding lessons learned, the novelty of sView relative other service frameworks, and a more precise description of the technical details of sView, please see [3, 4, 5].

In sView, users can store, execute, and even create electronic services in a personal, virtual briefcase. Services can be reached using a variety of devices and platforms (including graphical user interfaces on ordinary PCs, web-based interfaces, and interfaces in mobile phones), and the accompanying sView server architecture ensures that a user's briefcase is always continuously running and ready at hand. SView is attractive because it collects all of a user's services in a single, uniform environment, freeing the user from having to alternate between service environments to access services.

The unifying theme for the sView system is user control, defined by openness, continuous and ubiquitous access, personalization, and collaboration.

Users should have complete freedom regarding which services they use in their briefcase. For this reason, sView's service architecture is open, meaning that any service provider, including the user, can design services for sView.

Services should be available to the user at all times and should be accessible on a variety of devices. The sView briefcase is accessible on a wide range of devices including laptop computers, PDAs, and mobile phones. SView achieves continuous access by enabling services to save their state, making it possible for the service briefcase to migrate between usage sessions and devices without having to restart the services it contains.

Many electronic services require and collect personal information about the individual user. The sView framework makes it easy to monitor and control which services have access to what information.

One of the shortcomings of many (proprietary) services is their inability to interoperate. The sView architecture promotes service interoperation by making it possible for services to share APIs with each other. Another level of interoperation is achieved by the ServiceDesigner, which allows a user to connect Web Services to each other and to other services in sView.

To further illustrate sView, let us consider a usage scenario: our example user Michael is working in his office and keeps his sView briefcase open and running on his office PC. In the briefcase he has a number of services including email, instant messaging, booking service for the spa in his apartment building, pizza delivery, airline booking, and so on. At five o'clock, Michael decides to leave work, and logs out of the briefcase and walks to the subway. As he logs out, the services in the briefcase are stopped, compressed, and transferred to a central server for off-line access—arriving at the server, the services are once again started and made active. While on the train, Michael decides that he feels like a soak in the spa and he therefore takes out his cell phone and gains access to his briefcase remotely, using the phone's wireless Internet connection. He checks the bookings and after finding the spa completely available for the whole evening, books the next two hours. He also decides to order a pizza to go with his soak in the tub. When he gets home, he gets the idea to invite his girl friend for the evening. Using his tablet PC, he logs into sView and first fires off an instant message with the proposal; his girlfriend, also being an sView user, immediately responds with a yes. Then he checks on the pizza order; the pizza has not yet been dispatched so he changes the order to a larger size. Finally, he changes the spa booking, adding two more hours. When Michael logs into sView on the tablet PC, it connects via the wireless home network to the remote server and fetches the active briefcase. Once again, Michael can in-

teract with his services using more powerful graphical user interfaces, since the services have been brought to the local machine where they are now executing; thanks to the sView server architecture, the session has never been interrupted and the services have kept their state.

The second developmental trend of today's Internet is marked by peer-to-peer's emergence as a competitive computing model. P2P's utility has historically been overshadowed by application designers' preference for the client-server model. The recent success of large-scale file sharing applications has rekindled an interest in peer-to-peer, compelling commercial and private industry alike to develop software that incorporate peer-to-peer in their design.

1.3 Paper Overview

In the next section we describe Briefcase Connectivity. Following this, we examine a use case involving a service that utilizes Briefcase Connectivity—Sentinel. The next section describes related work, and finally, in Conclusions, we summarize our work and give some hints as to possible future work.

2 Briefcase Connectivity

Briefcase Connectivity is a JXTA[4]-based generic peer-to-peer communications system for sView (more on JXRA below). It is intended to solve two problems. The first problem involves building and maintaining a network of sView briefcases. The second problem is providing a general communication mechanism that sView services can use to communicate over the sView network. On the one hand, Briefcase Connectivity must provide a component that is accessible to other briefcases via the network. On the other hand, it must provide an interface accessible by sView services contained in the briefcase.

Before we examine the design and implementation of Briefcase Connectivity let us consider the following questions:

What is the novelty of Briefcase Connectivity? Briefcase Connectivity is essentially a generic peer-to-peer system for the ordinary user. In our design and construction of the system we have aimed at making it as simple as possible to use. At the moment it is easy for a developer to use Briefcase Connectivity, and in our "Individual Service Provisioning" scenario[5], where every user can be a service provider, some of the services produced can easily be peer-to-peer enabled thanks to Briefcase Connectivity. By creating a network of sView users, who all contribute with services and resources, we hope to generate a network effect with positive feedback [12] where more services contributed by members leads to higher incentive to participate in the community.

What can you do with Briefcase Connectivity in sView? With support for peer-to-peer, service providers can leverage the potential small world effects of communities [1] and produce services that add value to sView users. As we will see below, with Briefcase Connectivity it is relatively easy to create a peer-to-peer enabled service or to add auxiliary peer-to-peer functionality to an existing service. Consider a standard MP3 player such as Winamp[6]. Now imagine adding to Winamp a peer-to-peer system in order to present to the user a top list of the songs being played, in all instances of Winamp, right now—or in other words the "Real-Time Top 10"[7]. To do this you would need to build a complete peer-to-peer system including the protocol design, network interfaces, discovery, caching, addressing, etc.—a lot of work to add a little function. In sView, with Briefcase Connectivity, you simply call the send method of the Briefcase Connectivity API. This broadcasts information about which song you are currently playing to other instances of the MP3 player. Each player in each briefcase then builds its own top list by compiling the information being sent within the network of MP3 players.

How does Briefcase Connectivity differ from Jini/RMI/Sockets/...? Briefcase Connectivity is at a higher level of abstraction. This makes it easier for developers to create services that make use of the peer-to-peer network. Furthermore, Briefcase Connectivity directly gives the developer and end-user access to a peer-to-peer system and a community of users. With Jini, RMI, Sockets, etc., and even plain JXTA, a developer must expend greater effort to achieve the same level of functionality.

What does JXTA add? JXTA adds the plumbing of the peer-to-peer network. We believe JXTA has the potential to become a de facto standard for generic peer-to-peer. At the very least we expect to be able to use JXTA for a number of development iterations of Briefcase Connectivity. As a new and improved version of JXTA is released, we plug it into Briefcase Connectivity and start to benefit from the enhancements. However, the Briefcase Connectivity system is built in such a way that we can use any type of underlying peer-to-peer system with minor alterations of interfacing code [9].

[4] Project JXTA: http://www.jxta.org.

[5] Individual Service Provisioning is described briefly in Sect. 1.1. A more thorough discussion is forthcoming in Fredrik Espinoza's PhD thesis, due in the fall of 2002.

[6] http://www.winamp.com.

[7] There are many reasons to use a peer-to-peer solution rather than a server-client solution—decentralization, load-balancing, robustness, scalability, etc.

2.1 Design

At the most basic level, our design of Briefcase Connectivity enables instances of the same service residing in different briefcases to exchange messages.

In Fig. 1 we see four briefcases running Briefcase Connectivity. Briefcase *D*, for example, is running three services, *S2*, *S3*, and *S4*. These services use Briefcase Connectivity to communicate with *S2* in Briefcase *A*, *S3* in Briefcase *B* and *C*, and service *S4* found in all the briefcases, respectively.

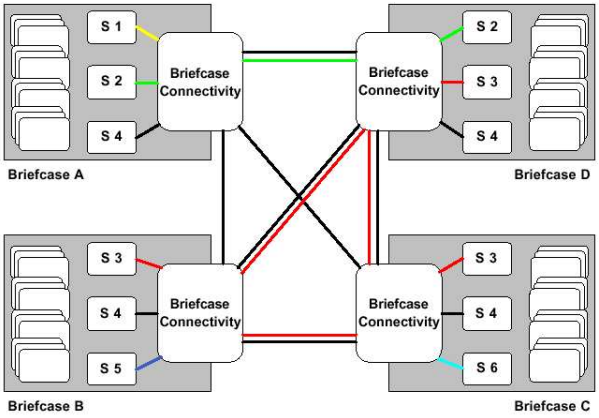

Figure 1. Different instances of the same service communicate via Briefcase Connectivity.

The following list presents the design criteria for Briefcase Connectivity: the sView network should not rely on a central arbitrator for any significant portion of its operation; the sView network should be dedicated to sView briefcases and their services; a briefcase must be able to discover the network; a briefcase must be able to discover other briefcases on the network; a briefcase should be able to discover a particular briefcase on the network; a briefcase must be able to join/leave the network intermittently; the joining/leaving of briefcases should not disrupt the network; a briefcase must be able to communicate with other briefcases; the communication protocol must carry service-specific protocols; the communication protocol must accommodate a heterogeneous set of service protocols; communication between briefcases should be secure; Briefcase Connectivity should maintain an accurate view of the network; Briefcase Connectivity must make a best effort to deliver messages

We divide the design into three parts: Sect. 2.1.1 describes the network component that enables a briefcase to participate in the network. Section 2.1.2 describes the network interface that is made available to sView services. Finally, Sect. 2.1.3 presents an architectural overview of the design of Briefcase Connectivity.

2.1.1 The Network Component

The sView network is comprised of briefcases upon which sView services that require this network can be deployed. Services that require a sView peer-to-peer network connection with their peers use Briefcase Connectivity as a provider of this function.

We have elected to use JXTA technology as a basis for our design and implementation. The peer-to-peer protocols [10] implemented by the JXTA reference implementation exceed the requirement criteria listed in the previous section.

Briefcase Peers. Our natural inclination is to represent the sView briefcase as a peer on the JXTA network. To achieve this, the Network Component must implement the JXTA protocols [10] and become a member of a peer group. By default, all peers are members of the "World Peer Group" and, by virtue of the protocols required for inclusion in this group, are capable of discovering resources within the group, including other peer groups, peers, pipes (i.e., connections between peers), etc. The uniform addressing scheme that JXTA provides solves the problem of addressing briefcases. However, we need a way for briefcases to interact with each other.

Peer Interaction. Peers in the JXTA network interact with each other by invoking services. For example, a peer might provide a client-server style service for downloading movies. When a peer wants to make a service available to other peers, it must create a pipe to which clients attach their own pipes. By connecting their pipes to the server pipe, clients can exchange data with the server and access the service.

In sView, we can envision how briefcase peers will interact. Each briefcase runs a messaging service that is equipped with two pipes: one to which other services connect, and the other for connecting to other services. When a briefcase joins the network, it publishes its messaging service advertisement to the peer group. Other briefcases collect these advertisements and extract the enclosed pipe advertisements when they wish to communicate. It is intended that the messages carry sView service specific protocol messages. To eliminate unnecessary overhead, the briefcase messages must be compact.

Briefcase Peer Group. The JXTA peer group abstraction enables us to build a briefcase peer group that is isolated from the rest of the JXTA network. This is advantageous

for a number of reasons. First, we can more easily implement trust mechanisms that are valid for the entire group. Second, we can leverage the peer group abstraction to create specialized sub-groups on top of the primary briefcase group. For example, a number of briefcases might deploy an auctioning service that creates a centrally coordinated network, similarly to the Sentinel, as we will see in Sect. 3. Briefcases interested in the service can then join the group and participate in a mediated auction. When the winner of an auction is announced, the buyer and seller may join a private one-on-one network where they conduct a secure transaction over encrypted channels. The peer group abstraction also gives us the flexibility to optimize the sView network's organization. If we discover that a fully decentralized network is sub-optimal, we can incorporate super briefcases (reminiscent of Milgram's highly connected peers) in the network to create a more hierarchical organization and enhance network performance.

2.1.2 The Service Interface

Now that we have a network component capable of carrying messages between briefcases, we need an interface that allows sView services to access the network to communicate. The idea is that any service that requires a network of sView briefcases should not have to design and implement a network architecture or bother with low-level network protocols. Instead, services should be able to access the sView network via a simple interface that Briefcase Connectivity provides. In this sense, Briefcase Connectivity can be seen as an abstraction that provides the API to the sView network protocol. The service interface defines operations for creating and releasing a network connection as well as operations for sending and receiving messages.

Several sView services in one briefcase may use Briefcase Connectivity simultaneously to participate in the network. For this reason, we need a way to deliver incoming messages efficiently and correctly to specific services. As messages arrive from the network, the service interface will deliver the messages to the appropriate service message queue of each service.

The Briefcase Connectivity protocol must be general in order to accommodate any type of service communication requirement, and simple in order to make designing sView network services easy. The operations mentioned above—create, release, send, and receive—suit our criteria of generality and simplicity. When a service creates a connection to the sView network, it receives a handle to a message queue identifiable by the service's name and ID. It receives messages directly from the queue, and sends messages by invoking the send operation defined by the service interface. When the service is finished with its network session, it releases its connection and loses its message queue.

2.1.3 Putting it Together

Combining the network component with the service interface yields Briefcase Connectivity. Our briefcase is represented by a fully qualified JXTA peer that participates in the JXTA network. Briefcases interact with each other by publishing and discovering advertisements for their message services. SView services that wish to communicate over the network must establish a connection to the network, which amounts to obtaining a handle to a private message queue from the service interface. The services can then send messages via the service interface that delivers them to the network component. The network component in turn sends the message on the JXTA network to another briefcase, where it is delivered to the specified recipient service.

2.2 Implementation

The Briefcase Connectivity architecture consists of a peer-to-peer module, a processing and demultiplexing module, service mailboxes, and a simple messaging protocol and message format. When the Briefcase Connectivity service is started in the sView briefcase, it first creates the peer-to-peer module, called JXTAMessenger. JXTAMessenger creates a briefcase peer on the JXTA network. Once the peer is established on the network, the processing and demultiplexing module, called CommManager, is created. This component prepares the internal mechanisms required for formatting and processing messages to and from sView services. When CommManager completes its initialization tasks, Briefcase Connectivity activates both mechanisms. At this point, other sView services may register themselves and receive mailboxes, called ServiceMailboxes, from which they receive messages. At the same time, they may send messages as well as receive network information from Briefcase Connectivity.

Messages arriving at the transport layer are passed up to JXTAMessenger where they are checked for correctness and delivered to CommManager. The messages are parsed, reformatted as SviewMessages, and then delivered to the appropriate ServiceMailbox. Services send messages by delivering them to CommManager where they are formatted and addressed. Once formatted, they are passed down to JXTAMessenger where they are put on the wire.

Please refer to [9] for more details about the implementation of Briefcase Connectivity.

3 Proof of Concept: The Sentinel

Let us consider electronic services in more general terms. Many electronic services demand a high level of user interaction and focus. Some of these services may present information in a graphical user interface, and may require

the user to physically interact with the GUI by making selections or typing information. Other services may prompt the user with visual or audio cues when they have completed a task or require input from the user. ICQ, for example, is an instant messaging application that is infamous for its highly audible "Uh-Oh!" that sounds when a new message arrives. In an informal setting, such as at home, these distractions are un-intrusive. However, in more formal settings, such as at the office or in a meeting, such distractions are intrusive and potentially disruptive. In a typical meeting, many of those attending have with them mobile phones, PDAs, and laptop computers. A potentially large number of electronic services may be running on these devices, heightening the risk of intrusive events disrupting the flow of the meeting.

The FEEL Project[8] is a cooperative endeavor between SICS, the FUSE group at the Royal Institute of Technology (KTH), and the IAM team at the University of Southampton, and is part of the larger, EU funded program The Disappearing Computer[9]. FEEL focuses on managing the intrusiveness of pervasive and ubiquitous technology. SICS role in the project has been to provide a software platform (sView) for implementing the project's concepts. FEEL envisions that each member of a planned or ad-hoc meeting has with him one or more computing devices running the sView briefcase. When the group convenes, their briefcases collaborate transparently, determining a level of intrusiveness appropriate for the meeting. Once the intrusiveness level is determined, all services executing in the briefcase are forced to behave in a manner appropriate to the decided upon level. For example, if the group decides upon a low level of intrusiveness, an instant messaging service would not give audio cues, and a mobile phone would redirect calls to voice mail. Briefcase Connectivity's ad-hoc, peer-to-peer capability tenders the type of core functionality that makes this possible. However, we need a service in the briefcase that carries out this task.

Sentinel (Fig. 2) is a distributed sView service that was developed for the FEEL project. The first implementation of Sentinel uses Briefcase Connectivity to perform a distributed voting task. Users running Sentinel are presented with a GUI that displays the number of active Sentinels, representing the users who are present at the meeting. The user adjusts a slider in the GUI to select his preferred intrusiveness level. Adjusting the slider activates a voting process that is carried out among all of the Sentinels. Each Sentinel presents its vote to the coordinating Sentinel and when the voting process is complete, the GUI displays the average of all the collected votes, the number of voters that participated in the election, and a color-coded panel that indicates whether or not the user's vote was counted in the most recent election. Based on the vote results, Sentinel regulates how other services interact with the user.

The Sentinel service is fully replicated in each briefcase. When the voting process begins one of the Sentinels involved will take on the role of coordinator for that particular vote. When the process is finished the result of the vote will be broadcast to all the Sentinels involved and for the next vote it is possible that a different Sentinel will take on the role of coordinator. The fact that any Sentinel can perform the coordination makes the voting system robust. If the current coordinator were to break down another would take its place.

One problem we noticed while working with Sentinel was the timeliness, or rather untimeliness, of some message deliveries. Running a group of Sentinels on a LAN produced no ill effects, but as soon as other Sentinels (which were scattered across the Internet) joined the group, Sentinel's accuracy declined considerably. The JXTA v1.0 Protocols Specification also states:

> "Due to [the] unpredictability of P2P networks, assumptions MUST NOT be made about the time required for a message to reach a destination peer. JXTA protocols SHALL NOT impose any timing requirements for message receipt."

Sentinel employs a timing mechanism that determines the length of the voting process; the fact that JXTA's protocols do not support real-time applications poses a problem for Sentinel. However, JXTA's flexibility allows us to

Figure 2. SView running Briefcase Connectivity and Sentinel.

[8] http://www.dsv.su.se/feel.

[9] http://www.disappearing-computer.net.

enhance any aspect of the technology, making it possible to design networks that provide real-time functionality. In addition, we can leverage JXTA's peer group facilities to create dedicated Sentinel groups. Within these small peer groups we are able to implement and guarantee a higher quality of service.

Apart from the timeliness of message deliveries, Sentinel is free from problems. The service was easy to implement, due primarily to the simplicity of programming to Briefcase Connectivity's API. At the time of writing, other Briefcase Connectivity services have been implemented, including an on-line help service and an instant messaging service.

4 Related Work

Today's peer-to-peer systems can be classified into three broad categories: centrally coordinated, hierarchical, and decentralized.

Centrally coordinated systems operate in a fashion similar to client-server systems. A central server coordinates the activity of the peers in the network and mediates information that the peers may later act on, for example, to contact each other. In hierarchical systems, the responsibilities of a central coordinator are delegated to a tree of coordinators around which peers form themselves into groups. Of the three categories, decentralized peer-to-peer has drawn the most interest, due to the fact that these systems are robust, scalable, and virtually self-sustaining—characteristics desirable in distributed applications. Four notable decentralized peer-to-peer systems are Gnutella[10], Freenet[11], OceanStore[12], and Project JXTA.

Gnutella. Gnutella is a protocol for distributed information searching and sharing, implemented primarily by file sharing applications. The Gnutella protocol defines the way in which servents[13] communicate over the network. Gnutella is currently the largest peer-to-peer network in use, boasting millions of users.

Freenet. Freenet is an anonymous information storage and retrieval system. This description is a bit of a misnomer, however, as Freenet does not guarantee permanent storage. The goals of the Freenet project are to provide a system that guarantees anonymity for both producers and consumers of information, thwarts third party attempts to deny access to information, and provides efficient storage and routing of information.

OceanStore. OceanStore is a wide-area peer-to-peer storage architecture [2]. Currently under development, OceanStore is intended to be a self-sufficient, globally spanning archival system that links together servers from around the world. In return for a nominal fee proportional to the amount of desired storage space, an OceanStore user has access to persistent data—personal files, configuration data for a personal computing device, etc.—from anywhere in the world.

Project JXTA. Project JXTA is an ongoing, monolithic open-source effort bent on improving modern distributed computing, especially in the area of peer-to-peer [7, 11]. JXTA's developers claim that the Internet's three fundamental assets—information, bandwidth, and computing power—are underutilized due to the prevalence of the client-server model. The project's design objectives are derived from what the JXTA designers consider to be shortcomings of existing peer-to-peer systems: the inability to inter-operate, platform dependence, and the lack of true ubiquitous support for heterogeneous devices. At the highest level of abstraction, JXTA is a set of protocol specifications that define the basic activities of any conceivable peer in a peer-to-peer network. It is intended that this will provide a flexible framework in which to design nearly any type of peer-to-peer system.

The common ground shared by all of these systems is threefold:

1. They embrace decentralization but do not necessarily exclude centralization. All of these systems attempt to minimize their reliance on central points that provide functionality essential to the workings of the system such as processing, address resolution, arbitration, etc.

2. They leverage resources—processing power, storage space, bandwidth, digital content, human presence, etc.—that would otherwise be underused or unused

3. They tolerate and even expect dynamic connectivity

Briefcase Connectivity is similar to Gnutella in that it is decentralized and nodes act as servers and clients. The main difference is that Briefcase Connectivity was designed to be a generic peer-to-peer system on top of which service specific protocols can be placed. This allows us to use the communication layer for more than file sharing. Freenet and OceanStore focus on providing reliable storage of data and files within the peer-to-peer community—Briefcase Connectivity is focused on communication between peers. When peers can communicate using any chosen protocol they can also establish service specific protocols and infrastructure in parallel. Thus Briefcase Connectivity is more general and could, in theory, implement

[10] http://gnutella.wego.com

[11] http://freenet.sourceforge.net

[12] http://oceanstore.cs.berkeley.edu/

[13] A Gnutella peer is called a servent because it acts as both a SERver and a cliENT.

the functionality of both Freenet and OceanStore, via separate services. JXTA is most similar to Briefcase Connectivity, and rightly so, since Briefcase Connectivity builds on JXTA. The difference is that Briefcase Connectivity is designed to work at a higher level of abstraction, providing peer-to-peer capabilities to other services in the sView briefcase.

5 Conclusions

We have presented the development of Briefcase Connectivity. To design a generic peer-to-peer service like Briefcase Connectivity, we needed to understand the field of peer-to-peer. Our analysis of this computing paradigm[14] taught us that it is powerful and flexible, but also that its benefits are not easily won.

Although it is relatively new, JXTA technology offers designers a powerful and flexible medium in which to build peer-to-peer systems. Our own design and implementation attests to this fact. Aided by JXTA, Briefcase Connectivity enables any service to access the sView network and communicate with other services. We hope that the success of Briefcase Connectivity and services like Sentinel generates an "sView network effect" and inspires service providers to build new and exciting services for the sView platform.

We intend to use Briefcase Connectivity as a testing ground for future research. This research includes, but is not limited to: analyzing JXTA's performance and optimizing Briefcase Connectivity; implementing an accountability mechanism to ensure the equal and proper usage of sView network resources; deploying a reputation system on the sView network to spin a "web of trust"; fine-tuning Sentinel for the FEEL Project; and realizing our ideas for sharing individually created services between users.

6 Acknowledgments

The work described in this paper was partially financed through the FEEL EU project and the sView SITI project[15]. The authors wish to thank the rest of the OASIS team at SICS, including Markus Bylund, Ola Hamfors, Anna Sandin, and Stina Nylander.

References

[1] *Nexus, Small Worlds and the Groundbreaking Science of Networks*. W. W. Norton & Company, 2002.

[2] David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Christopher Wells, Ben Zhao, and John Kubiatowicz. OceanStore: An Extremely Wide-Area Storage System. Technical Report UCB/CSD-00-1102, University of California at Berkeley, Computer Science Division, Berkeley, California 94720, May 1999.

[3] Markus Bylund. Personal Service Environments - Openness and User Control in User-Service Interaction. Licentiate of Philosophy Thesis, Uppsala University, 2001.

[4] Markus Bylund and Fredrik Espinoza. sView – Personalized Service Interaction. In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 215–218, Manchester, UK, April 2000. The Practical Application Company Ltd.

[5] Markus Bylund and Annika Wærn. Personal Service Environments - Openness and User Control in User-Service Interaction. Technical Report T2001:07, Swedish Institute of Computer Science, Kista, Sweden, May 2001.

[6] Fredrik Espinoza and Ola Hamfors. ServiceDesigner: A Tool to Help End-Users Become Individual Service Providers. In *Proceedings of HICSS-36, Hawaii International Conference on System Sciences*, January 2003. Forthcoming.

[7] Li Gong. Project JXTA: A Technology Overview. Web, 2001.

[8] Ola Hamfors. Service Designer - Lets the End-user Create her Own User-Interface to Web Services. Master's thesis, Royal Institute of Technology, Stockholm, Sweden, 2001.

[9] Lucas Hinz. Peer-to-Peer Support in a Personal Service Environment. Master's thesis, Uppsala University, Uppsala, Sweden, 2002.

[10] JXTA v1.0 Protocols Specification. Web, 2002.

[11] Project JXTA: An Open, Innovative Collaboration. Web, 2002.

[12] Carl Shapiro and Hal R. Varian. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston, Massachusetts, 1999.

[14] This analysis was more extensive than is apparent in this paper. Please see [9] for more details.

[15] http://sview.sics.se.