RC22939 (W0310-125) October 21, 2003
Computer Science

IBM Research Report

Disconnected Portlets: An Overview

Marion Blount, Veronique Perret, Danny Yeh, Apratim Purakayastha
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Michael Moser, Yann Duponchel, Daniela Bourges-Waldegg, Marcel Graf
IBM Research Division
Zurich Research Laboratory
8803 Rueschlikon, Switzerland

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Disconnected Portlets: an Overview

Marion Blount, Veronique Perret, Danny Yeh, Apratim Purakayastha

IBM T. J. Watson Research Center
Yorktown Heights, New York
{mlblount, perret, dlyeh, apu}@us.ibm.com

Michael Moser, Yann Duponchel
Daniela Bourges-Waldegg, Marcel Graf

IBM Zurich Research Laboratory
Zurich, Switzerland
{mmo, ydu, dbw, grm}@zurich.ibm.com

Abstract. In the typical web application, a client renders markup and the application or service is implemented as a set of tiered functions in the network. However, clients can contain resources useful for an application and network connections cannot always be assumed. In this paper, we consider extending the reach of a web application to include 1) access to and use of local client resources, and 2) operation while disconnected from the network. We, however, try to preserve desirable programming model and management characteristics of web applications. We propose a system architecture and discuss an initial implementation using a portal as an example web application.

1 Introduction

Over the last three decades, the mainframe era, the client/server era, and the web era are recognized as technology sweet spots for software and application development. It is relatively easy to recognize a technology sweet spot when we are in the midst of it, the transitions are much harder to discern. There is an inherent tension of functional separation between the client/server and the web models. The web model enjoys the benefits of ease of deployment but lacks adequate client exploitation, whereas the client/server model exploits client capabilities but suffers from complex deployment. The tension is causing a transition where users/administrators want the best features of both models. One can either extend a client-centric programming model or extend a web-centric programming model to relieve the tension. In this paper, we discuss how we extend a web-centric programming model to exploit client capabilities.

Figure 1 outlines the characteristics of the major technical eras. In the mainframe era, (Figure 1a) the client was a simple character-oriented display that communicated with a mainframe using proprietary protocols. All the computing was centralized in the mainframe. The programs were also authored using a proprietary model. There was not really a concept of application-software vendors. The hardware, system software, and application provider was all one entity. One of the major advantages of this model was near-zero client deployment cost. As client-side hardware became cheaper and clients became more powerful, a new client/server model of computing emerged (Figure 1b). The client programs offered rich GUI, local application logic, local storage, and autonomous, possibly disconnected operations. For network resources, they used some forms of remote procedure invocation to contact servers. While this model offered tremendous advantages, diverse programming models on the client blossomed and in turn the total cost of ownership of client/server systems skyrocketed. In contrast with the client/server model, the current web model (Figure 1c) uses basic assumptions about client capabilities and centers around displaying markup. It uses standard protocols like HTTP and standard markup representation like HTML. The use of more standard representation of data using XML is also emerging. On the server side, a large community of open, standards-based programming using Java servlets/portlets and JSP pages has emerged. This model greatly reduces the total cost of ownership compared to client/server systems while offering a richer client experience than the mainframe era.

One of the shortcomings of the web model is its lack of exploitation of client capabilities. The web model treats today's highly capable clients as a pretty display. Users seem to want the best of both worlds. They like the simpler management and deployment model of the web, but like the richer local capabilities of the client/server model. This can be achieved in two ways; extending the client-server model for simpler management and deployment, or extending the web model to exploit client resources. In this paper, we choose to explore the latter. The reason is that various proprietary programming and data models are entrenched in the client/server world and it will be very hard to open it all up and standardize for interoperability and ease of management and deployment. The web model has open server-side

programming model options and open protocols already. We feel that it is easier to project that programming model down to clients.

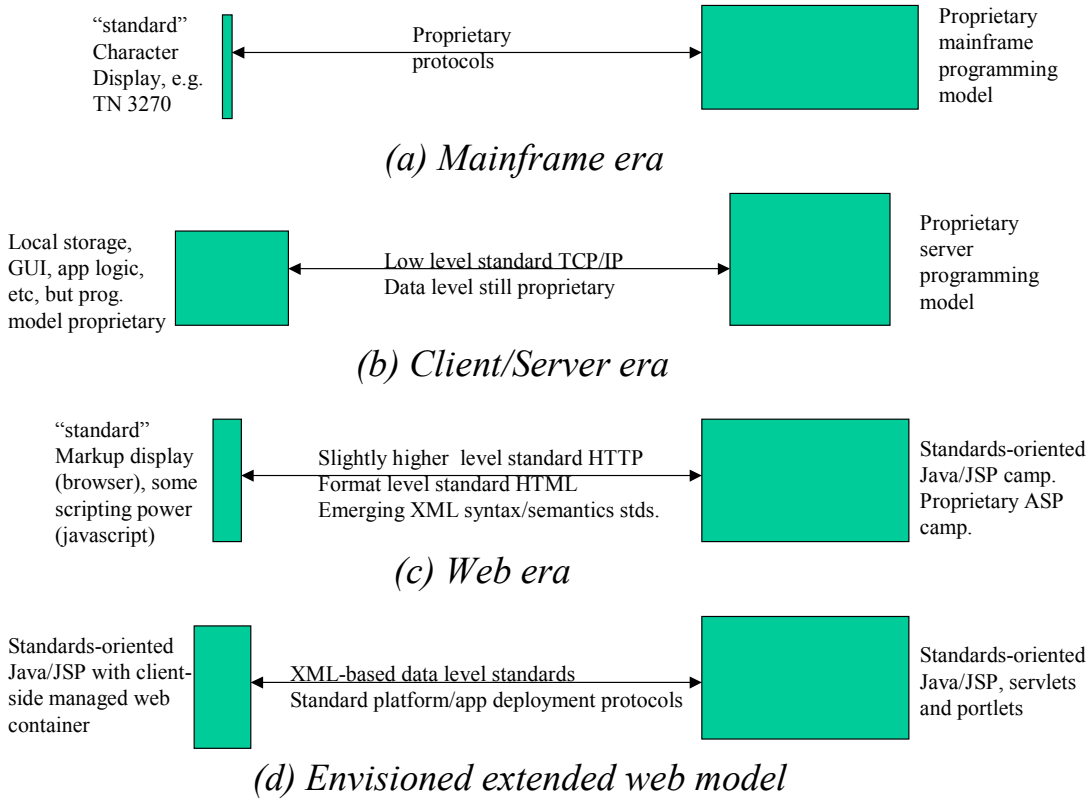

Figure 1. Summary of major eras of computing.

In this paper, we discuss design and implementation of a portal appliance, an instance of an extended web model. In this extended web model, the client side has a java-based web container in which web programs can execute. The web programs are written using the same programming model for those that execute on the server, with a few modifications and enhancements to exploit local resources, and to control data replication and transactions. Standards-based management technologies are used to deploy web programs on the client. In this system, a web program (like a portlet) can actually be executing on a client machine, thereby allowing disconnected operations, deferred transactions, accessing local display and other peripherals.

The rest of the paper is structured as follows. Section 2 provides a broad overview of the system architecture and design. Section 3 discusses implementation and early experience with the system. Section 4 discusses related work, and Section 5 outlines some ongoing and future directions.

2. Systems Architecture for the Managed Portal Appliance

The web application of focus for us, the portlet, is the key building block of web portals. A portal is a web-based single point of access to applications, data, and services. Each portlet is an application or service, and the presentation of several portlets can be aggregated on a single web page. A portal may offer hundreds of portlets and the user may access a selected set of them using a group of user-defined portal web pages. Each portlet relies on a set of services provided through a portlet API. Each major portal vendor has their own portlet API. This has given rise to an effort to standardize around a single portlet API. The Java Specification Request 168 – Portlet Specification should close in 2003 and has the support of the major portal vendors. [16] This will further accelerate the adoption of portlets as a major web application.

We introduce the notion of a portal appliance which allows portlets and data to be downloaded and operated locally on the client. The portlet can execute on the network portal in the "network" mode and on the appliance portal in the "local" mode. In the local mode, we may need to support disconnection from the network portal.

2.1 Design Principles for the Managed Portal Appliance

In the remainder of the paper, we focus on a portlet as the web application. Our goal is to provide the user a way to interact with a local web application (portlet) while possibly disconnected from the network. Here we highlight some design principles which guide our decisions on architecture and implementation.

User Experience Affinity. From a user's perspective, the network mode operation of a portlet and the local, possibly disconnected mode operation of the same portlet should not be seen as two applications. The look of the application and the interaction with it should be similar, if not the same, in the two modes. Disconnected operation will not be totally transparent to the user, but, the overall system should strive to make disconnection as transparent as it can be

Programming model affinity. The goal is that the programming model for network portlets and local portlets be the same except for those changes necessitated by disconnection or access to local resources. Development of both modes of the application should not be a two-application experience for the developer. The tools used to develop network portlets should be extended to aid in the development of local portlets.

Ease of management/deployment. There are no client side deployment issues for network portlets. Client side user administration is near zero. For local portlets, the goal is to minimize user deployment actions, to maintain the low level of user administration, and to remotely manage the client. We have to be able to deploy all of the client device software; that includes the operating system, drivers, the portal, and portlet applications. Multiple mechanisms may be needed to accomplish this.

Accommodate data source heterogeneity. Network access, remote access protocols, and web services make a lot of data sources available to a network portlet. A set of web applications, like portlets on a portal page, access a multiplicity of data sources. We must offer similar data source heterogeneity for local portlets, including supporting disconnection from the heterogeneous data sources.

2.2 Overall Architecture

Our proposed end-to-end high level architecture for supporting local, disconnected web applications is shown in Figure 2. To support local portlets, we establish a portlet execution environment on the client device. We call this the Managed Portal Appliance (MPA). The Network Portal Server (NPS) is the download source of the portlets, meta-data, and services for an MPA. Data is directly accessed on synchronization with back-end data sources. The NPS is kept out of the data path for reasons of scalability of the overall system. The browser on the client device is used to interact with local and network portlets. The Client synchronization Agent and the Server Synchronization Intermediary interact to provide initial data hoarding and synchronization of locally modified data to the back end data sources and to the server side user management and configuration data. Local portlets access data from the local cache through a mediator API. The Application Management System is responsible for the deployment of portlets to the MPA. Other deployment and management issues are handled by the Base Platform Management System. While it is not shown in the diagram, the MPA can also support transactions. A persistent messaging system, like IBM's MQSeries Everyplace [18] can be used to store and forward transactions.

We will discuss the architectural elements in more detail. In particular, we will discuss 1) the client portlet environment, 2) user experiences, 3) programming model, 4) data synchronization, and 5) security.

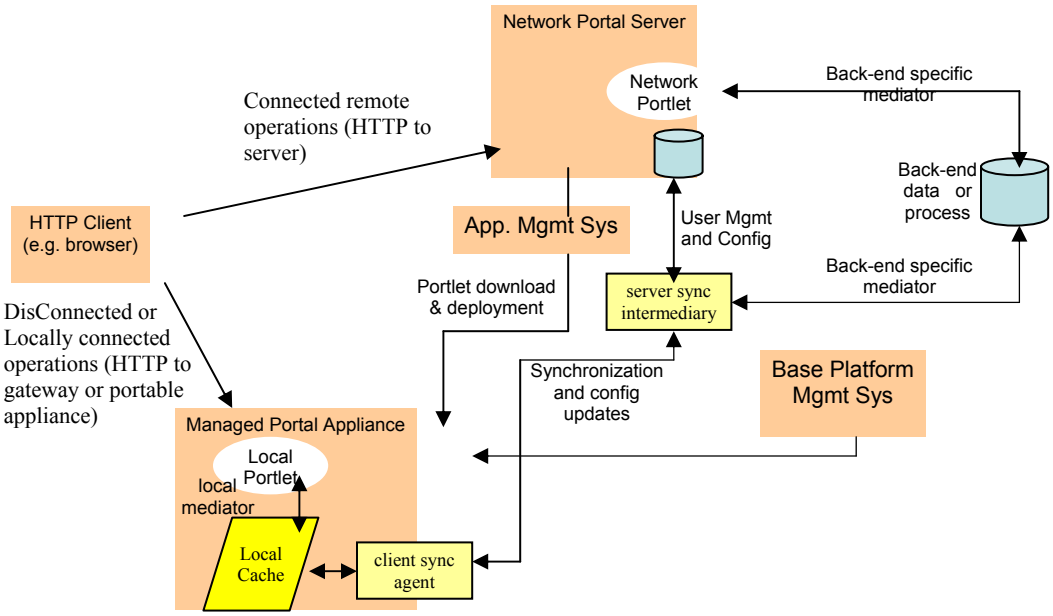

Figure 2. End-to-end systems architecture for supporting local disconnected portlet web applications.

2.2.1 Client Portlet Environment, Deployment, and Management

We want the user's experience of using local portlets to be the same as the user's experience of using network portlets. To do so, the MPA must 1) be able to run on a variety of client devices, 2) have extremely low user administration requirements and 3) have low deployment overhead for the MPA and local portlet applications.

We are targeting a set of client devices that range from PDA's to notebook computers. We have decided to implement MPA in Java. This solves the problem presented by multiple implementation targets. There are a number of Java Virtual Machines which have been designed for use on PDA class of client devices. The Java 2 Micro Edition (J2ME) standards [20] define a set of Java class libraries which are being supported by device makers and application developers.

The dynamic and static memory footprint of the MPA becomes a major issue when trying to run on client devices with low memory resources. The program memory space of some devices is` 16-32 MB. The performance of a Java application can be degraded if the memory utilization of the JVM is very high. We have adopted a strategy of implementing the MPA as a small memory footprint core that can be augmented with additional functions. The core will have small footprint portal services and will run in the program memory space of a PDA. On clients with more resources, we can substitute higher function portal services. For example, on notebooks, we may use a more extensive cache than we use on a PDA. A speech recognition function may be added for use by local portlets on client devices with the memory space and processing power.

In implementing MPA, we will use technologies which facilitate easy deployment and life cycle management of the MPA and local portlets. The Open Services Gateway Initiative (OSGi) Alliance [17] is a consortium of companies formed to establish standards and API's that facilitates the delivery of networked services to homes, vehicles, and portable devices. The OSGi standard specifies how services are packaged as a set of software components called bundles. Bundles are java packages with associated meta-data which specifies the services implemented in the bundle and the services the bundle is dependent upon. The OSGi framework has API's used in the download, install, update, and removal of bundles. Bundles can be installed and updated without necessarily restarting the device. The MPA and local portlets will be implemented as OSGi bundles. An OSGi framework, installed on the client device, will be basis for downloading and installing MPA and local portlets. While we do not reach zero administration for the user

and administrator, we have a solution which minimizes the tasks the user and administrator have to perform for supporting local portlets.

2.2.2 User Experience

Through the use of MPA, a user has the very positive user experience of accessing services and applications while disconnected from the NPS. The local version and the network version of the portlet will look and behave as similarly as possible. Disconnection will not be transparent to the user; that is, there are things the user will have to do in disconnected mode and in preparation for disconnected mode that are not necessary in connected mode. To have high affinity between connected mode and disconnected mode, the systems (MPA and NPS) should work together to minimize the impact on the user.

Portlet download, data synchronization, reconnection (disconnected mode to connected mode), disconnection (connected mode to disconnected mode) are new operations the user has to perform. The MPA's portal page will have icons for disconnecting, reconnecting, and synchronizing data. By monitoring a small number of items, like connection status and which portlets have modified data, the MPA will be able to automatically perform synchronization or prompt the user with reminders to perform synchronization upon reconnection to the Network Portal. The Network Portal has enough information to automatically package a hoard data set along with a local portlet in response to a portlet download request.

In a normal setup on a Network Portal, a user selects the portlets of interest and groups them in portal web pages. Our system can maintain this page grouping in disconnected mode, but, we will only display those portlets that can operate in disconnected mode.

There are a number of synchronization settings which apply to local portlets. Some parameter settings help define the hoard data set for the portlet and other parameters serve as input to the application. E-mail ID and email password may be hoard parameters for the email portlet and "Chinese" may be an input parameter to the Restaurant Find portlet. These settings can be modified on the NPS (using the network portlet) or on the MPA (using the local portlet). No matter where the changes are made our synchronization system will propagate the changes to the other system, thereby keeping the network portlet and local portlet settings consistent. Disconnection and data synchronization can be initiated from either the NPS or the MPA. In summary, while disconnection will not be transparent to the user, a small amount of additional function can significantly mitigate any negative impact on the user.

2.2.3 Programming model

What is the programming model that we want to preserve? The programming model for network portlets is based on the Model View Controller (MVC) design pattern. The View element deals with presentation and user interactions; this is done primarily through browser rendering using JSP. The Controller element uses the portlet API and other java classes to implement application logic. The Model element interacts with data sources to fetch data used by the portlet and to update the data source with data modified by the portlet.

Figure 3 shows the relationship between network and local versions of a portlet. For the majority of portlets, we hypothesize that the controller (C) and view (V) of local and network versions can be the same or almost the same. In some cases it will not make any sense to expose some things to the user in the local portlet. For example, a button that can be used to fetch local traffic conditions should not be active in a local portlet because there is no connection to the traffic service. Some portlets may want to handle data gathering and data synchronization in the application logic. So, we can say that, in most cases, C1=C2 and V1=V2. In the case of the network portlet, the model (M1) uses a (network or local) mediator to get data from and put data to the data source. M1 uses a different set of data get/put APIs for each data source. If the data source is a relational database, the mediator is normally a JDBC-based and M1 contains a set of SQL calls. If the data source is a web service, the mediator contains a web service client stack and M1 contains a set of SOAP calls. If the data source is another application, M1 contains a set of remote API calls for that particular application.

In general, the model (M2) for the local portlet makes calls to a local data store using a local mediator. If M2 is equal to M1, the Model element of the local portlet (M2) should use the same API calls as M1 and there must be a local data store to process the API calls. The variety of the data sources for network portlets makes it unlikely that M1=M2. It is not practical to recreate all of the data source API's on the client device just to reuse the model (M2) of the network portlet.

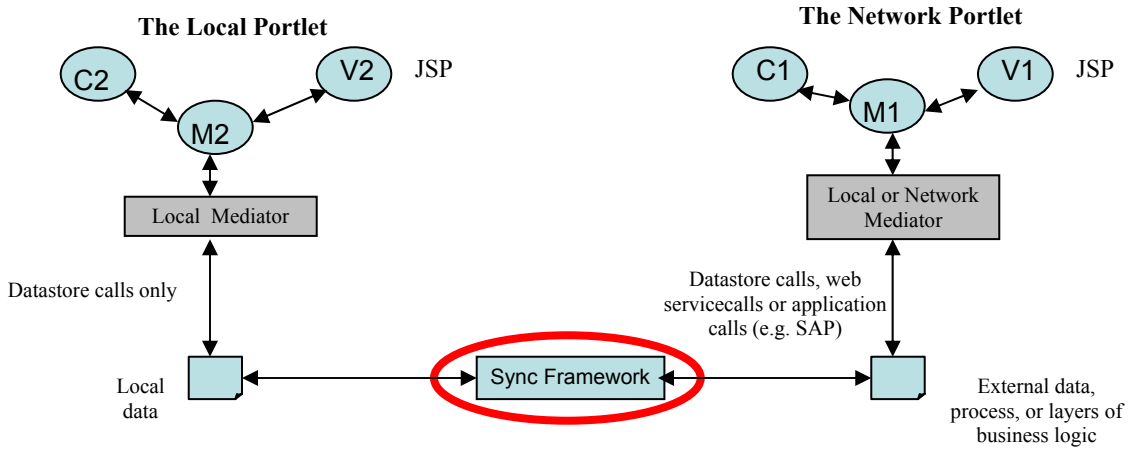

Figure 3. Relation between connected mode and disconnected mode of a portlet

Relational databases are an example of a type of back end data store that has an embedded, or small footprint, low/no administration implementation. Most database vendors offer an embedded implementation that supports the main API calls. There is also a synchronization component that works with that vendor's embedded and back end databases. The trend toward using web services and applications as back end data sources mean we must look for more general solutions.

The variety of data sources also complicates the synchronization framework. There is an adaptation needed for each type of data source; this means what starts as a single synchronization framework turns into N synchronization frameworks, where N is the number of types of data sources. We suggest an approach to solving this problem in the next section. In summary, network portlets and local portlets can use a single programming model, thereby enabling the sharing of development tools and developer expertise. They can also share significant common software between them. The challenges are in the areas of sharing Model elements.

2.2.4 Data Synchronization

Data synchronization refers to data transfers between MPA and back-end data sources related to data hoarding for local portlets and for synchronization of replicas as a result of changes at either node. In the prior section, we highlighted some problems caused by the varied types of back end data sources the system has to handle. The synchronization problem for the MPA is complicated due to the fact that even a small number of local portlets are likely to use data from a variety of back end data sources. The main advantage of a portal is that it is a convenient way to package applications and services in a concise package for users. Relational databases, file system, directories, web services, applications are just some of the back end data sources for portlets. The calendar local portlet may use a relational database, the contact list local portlet may use an LDAP directory, and a benefits enrollment local portlet may use a web service.

To support disconnected operation with data from a data source, one must first hoard data from the data source, replicate the data to the client, and synchronize modified data between the client and the data source. The more data sources there are, the more complicated a general solution, or just any solution becomes. If every data source had an embedded element that work in concert with the data source, then, the solution would be to deploy multiple embedded data sources and the accompanying synchronization middleware (for each data source). The problem is that embeddable data sources do not exist for every data sources. New types of data sources like web services and applications (i.e., SAP, Seibel) are less likely to provide an embeddable container for data. Over the near term, we may be limited to the embeddable data containers that are available and the ones that are implemented to support vital data sources. Over the long term, disconnected web applications require us to move to a system approach as shown in Figure 4.

Server side mediators have a connector component and a converter component. The connector component handles the particulars of accessing a specific type of data source. The converter component of the server side mediator 1) takes data from a data source and converts it to a structured data object (sdo), and 2) takes an sdo and converts it to a form suitable for the data source. An sdo is a high level canonical data object. A DOM is an example of an sdo. No matter the data source, the incoming data is converted to

the sdo model. Applications access sdo's through mediators. Sdo's are replicated to client side caches. Client side mediators understand the cache implementation. There is no need for a converter function in the client side mediators. Synchronization works with sdo's, so a single extensive synchronization framework suffices. Server side mediators handle conflict resolution.

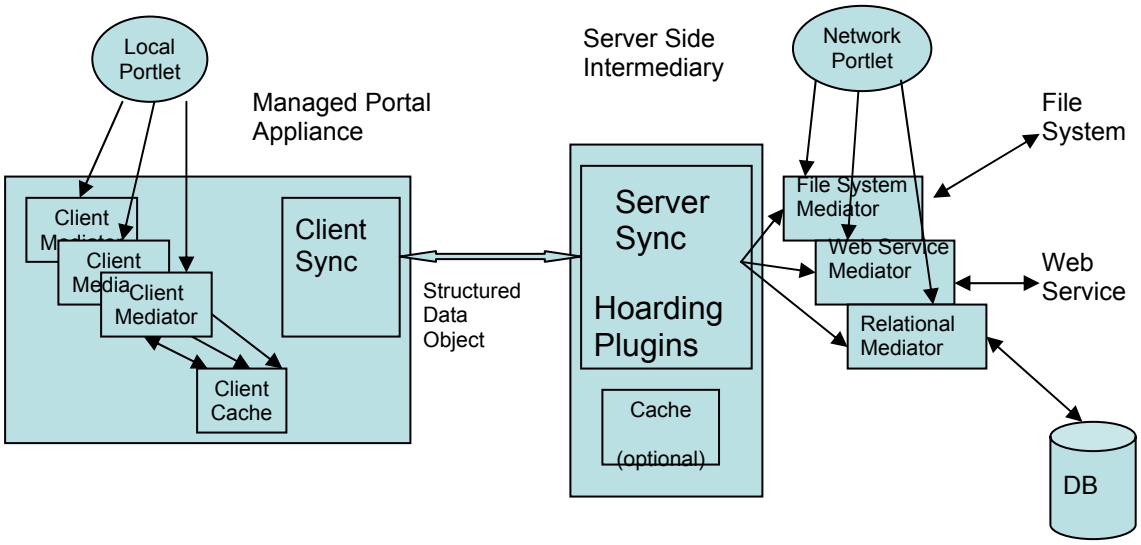

Figure 4. Synchronization architecture for disconnected portlet web applications.

Developers will develop hoarding plug-ins for local portlets. The hoarding plug-in specifies the data the local portlet needs for disconnected mode operation. The server side intermediary will provide the means of processing a hoarding plug-in. Given a hoarding plug-in, actions are taken to access the specified data source and obtain the data specified in the plug-in. There is a hoarding plug-in passed to the server side intermediary for each deployed disconnected portlet

There are major advantages of this type of synchronization for disconnected web applications. The connected and disconnected versions of the application use the same model (M), thereby, greatly increasing programming model affinity. This significantly eases the development complexity. Since only one data model is involved, programming tools will evolve to assist in synchronization, hoarding, and caching. A server side mediator is the primary element needed to plug new data sources into the system. This makes it possible to accommodate the varied data sources available in the web world.

2.2.5 Security

An important feature of portals is the single sign-on feature which allows a user to authenticate to the NPS and gain access to a number of applications and services. The NPS maintains a per-user set of credentials which it uses to access data sources and other services on behalf of the user. If we maintain the single sign-on model, a user will authenticate to the MPA and gain access to the local portlets and their data. If user authentication on the MPA is inferior to the user authentication on the NPS, then we have compromised the security of the data. This problem is not unique to the MPA, but is common to any solution that involves downloading data for disconnected operation. We could have chosen a design in which all of the credentials for back end data sources are maintained on the MPA, thereby, permitting the MPA to directly access back end data sources. There are a number of problems with this approach. The main problem is the security of the credentials on the MPA, particularly relatively low end clients like PDA's. Another problem is maintaining consistency with changes to the credentials on the back end system. Enterprises normally use periodic changes to passwords as a defense mechanism and provide tools to assist users in making these changes. It is unlikely tools will be extended to propagate changes to client devices. We believe that maintaining back end data source credentials on the NPS is the best approach.

Another set of issues is the lost or stolen MPA. Here the concern is unwanted access to data on the device. A common solution to this problem is to encrypt the data on the client device. Since the MPA can use local resources, we can use biometric technologies to implement enhanced authentication

mechanisms. In summary, we need an authentication interface in the MPA which accommodates a variety of authentication mechanisms. We will then be able to map an authentication scheme to the device and the level of security desired.

.

3 Implementation

The first prototype of the end to end MPA NPS system has been completed. The following section describes the prototype and the systems architecture design issues addressed in the prototype. A system overview of the implementation is shown in Figure 5.

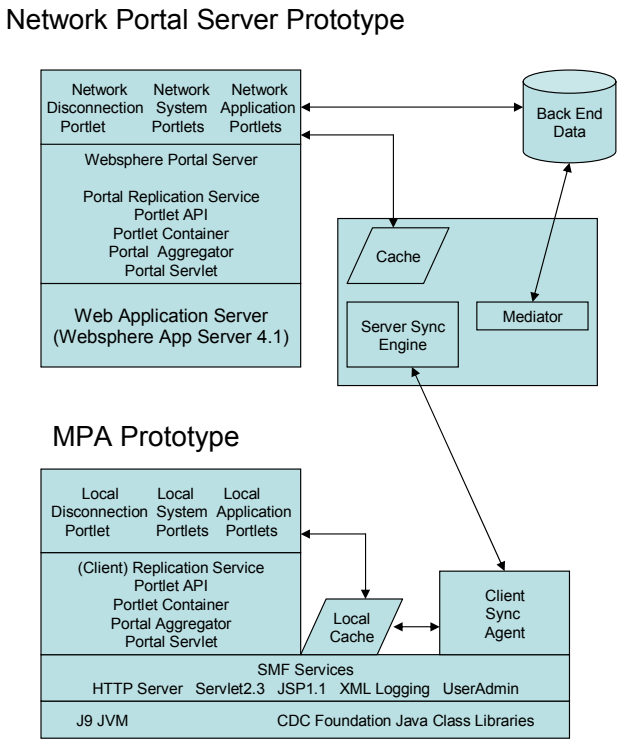

Figure 5. Network Portal Server and Managed Portal Appliance prototype.

3.1 Managed Portal Appliance Prototype

Both the Network Portal prototype and the MPA prototype are based on IBM's Websphere Portal Server (WPS) which runs on Websphere Application Server. We used a beta copy of WPS Version 4.1 as the NPS base. In support of local portlets, we added 1) a server intermediary, 2) a portal replication service, and 3) a Disconnection portlet to the base NPS system. The server intermediary handles the server synchronization function and the portlet deployment function. WPS has a number of portlets, called System Portlets, which are used by administrators to manage the portal and are used by users to customize their portal web pages. The Disconnection Portlet is a similar portlet used by users to download local portlets and to initiate synchronization from the server side. We had to implement a new portal service to give the server intermediary access to portlet configuration and parameter data. We have found that it is not necessary to make invasive changes to the Network Portal in order to support local portlets.

The MPA prototype was implemented on a notebook computer, but, we set a goal of being able to run in a 32-64 MB program space. This forced us to concentrate on the static and dynamic memory footprint. We chose the J9 JVM from OTI, Inc. [19] and the J2ME CDC/Foundation class library [20]. J9 has been ported to a broad set of client devices.

To implement the MPA, we first extracted from WPS the Java classes that implemented the WPS portlet API and portlet container. WPS's portlet API and portlet container were implemented inside a servlet, so the extraction was straightforward. The portlet container relies on a number of other services. Rather than use the WPS services, we looked for functionally equivalent, small footprint implementations. We chose IBM's Service Management Framework (SMF), an OSGi-certified framework and toolkit. In addition to providing the tools to build OSGi components, SMF contains a set of services based on OSGi service specifications. Some of the services we use include: 1)HTTP and Servlets (http server, servlet 2.3, JSP 1.1), 2) Log Service (general purpose message logger for OSGi environment), 3) User Admin Service (lightweight authentication function), and 4) XML function (JAXP XML). The MPA's local cache is a Cloudscape data base system. The MPA core was 8.5 MB static footprint. Overall, MPA has a static footprint of 12 MB with the data store and replication service included. The largest measured dynamic footprint of a (small set of) running two portlet was 20 MB.

We also used the SMF OSGi toolkit to implement local portlets as OSGi bundles. The Disconnection Portlet allows users to initiate download of a local portlet to the MPA and have it installed. This illustrates the value of OSGi technology in networked delivery and automatic installation of client software. This potential extends to parts of the MPA, but we did not experiment with it in this prototype.

Our synchronization system was an experiment in creating a data object view of data for our applications. The back end data source for our sample applications was a relational database. The server side mediator converts between relational database data and DOM data objects. The application portlets only use DOM data objects. The client sync agent and the server sync engine interact at synchronization time to synchronize the DOM data objects. We used a simple server-win conflict resolution scheme.

One of the main reason we implement prototypes is to discover and solve problems that are not anticipated at the design stage. Our system gives the user two places (MPA and Network Portal) to make changes to user information, portlet settings and portlet parameters. In order to preserve the user experience, we have to take the responsibility of maintaining consistency between the MPA and Network Portal copies of these meta-data. We used the synchronization system to maintain the consistency and implemented special case mediators in the server sync and client sync elements. This is another example of how we have to judiciously add function to the system in order to enhance the user's experience (remove burdensome tasks from the user).

3.2 Demonstration Application

We implemented a few applications on the MPA prototype. Each application was a single portlet that had a local mode of operation and a network mode of operation. In the mobile field worker application, the user is a field worker who makes service calls in a region defined by a zip code.

Figure 6 is a screen shot of the application's network portlet which show the job list at the beginning of the day. Note that the "See Traffic Alerts" button is active in the network portlet. Before leaving the office in the morning or while connected to the Portal Server from home, the worker replicates the job list data onto the MPA. During the course of the day, the worker updates the job status, makes notes about the jobs, all while disconnected Figure 7 is a screen shot from the local portlet on the MPA that reflects completion of one of the jobs. Note that the "See Traffic Alerts" button is not active, since this function is not available while the MPA is disconnected from the network.

The worker may connect to the Portal Server and perform synchronization in order to update the job status in the enterprise database and to see if new jobs in his area have been added. Figure 8 is a screen shot from the network portlet after synchronization, assuming no new jobs have been added to the list. Note that the "See Traffic Alerts" box is active. Since the list only shows incomplete jobs, Job#2 is not shown in the list.

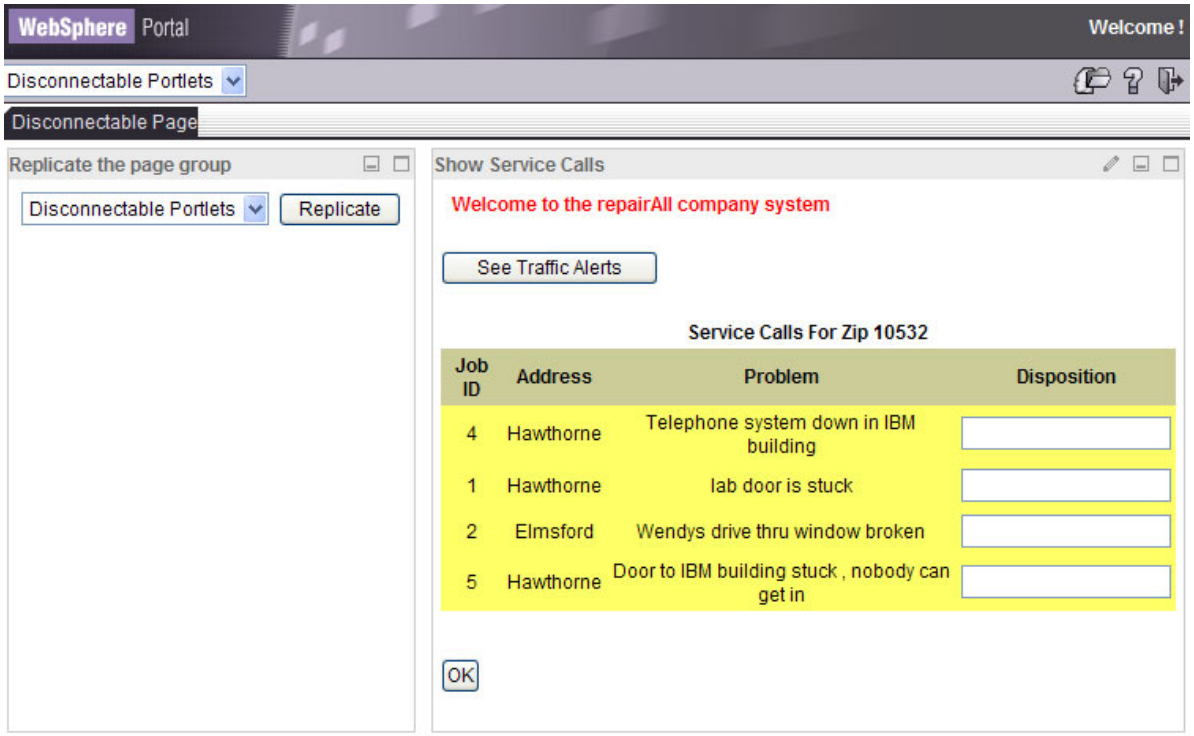

Figure 6. Screen shot from network portlet (running on the Network Portal)

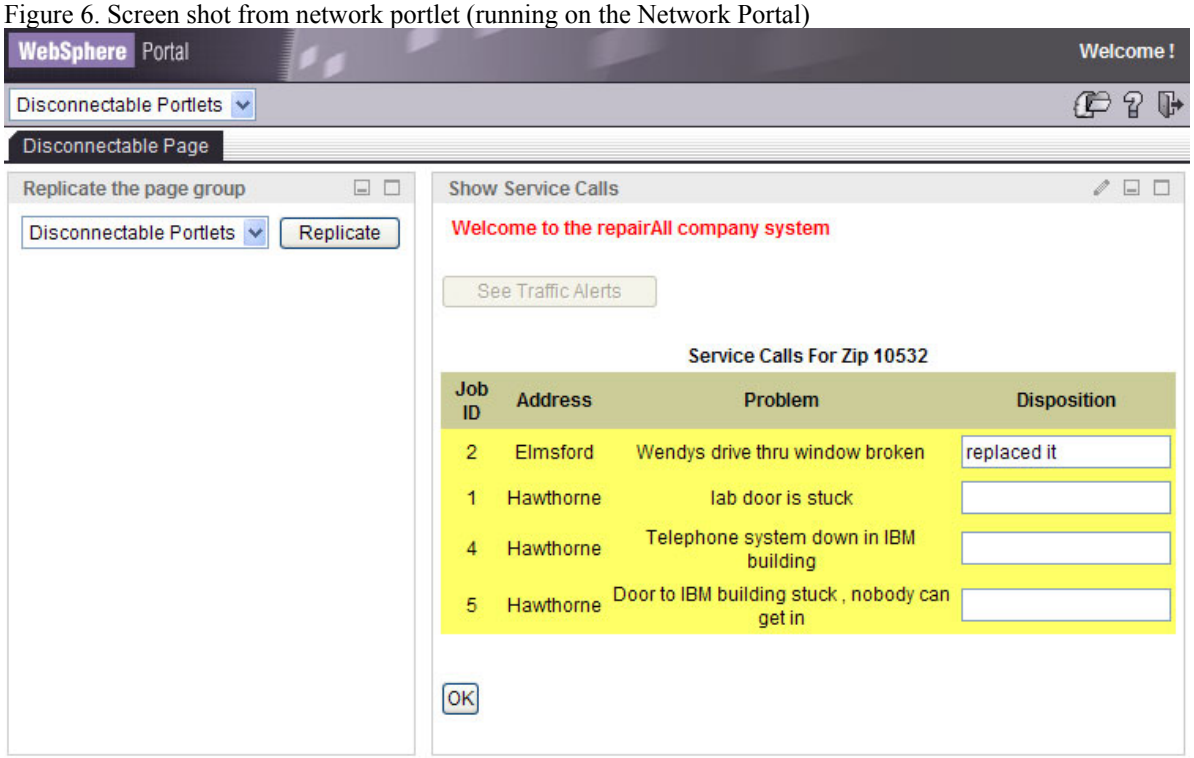

Figure 7. Screen shot from local portlet (running on MPA).

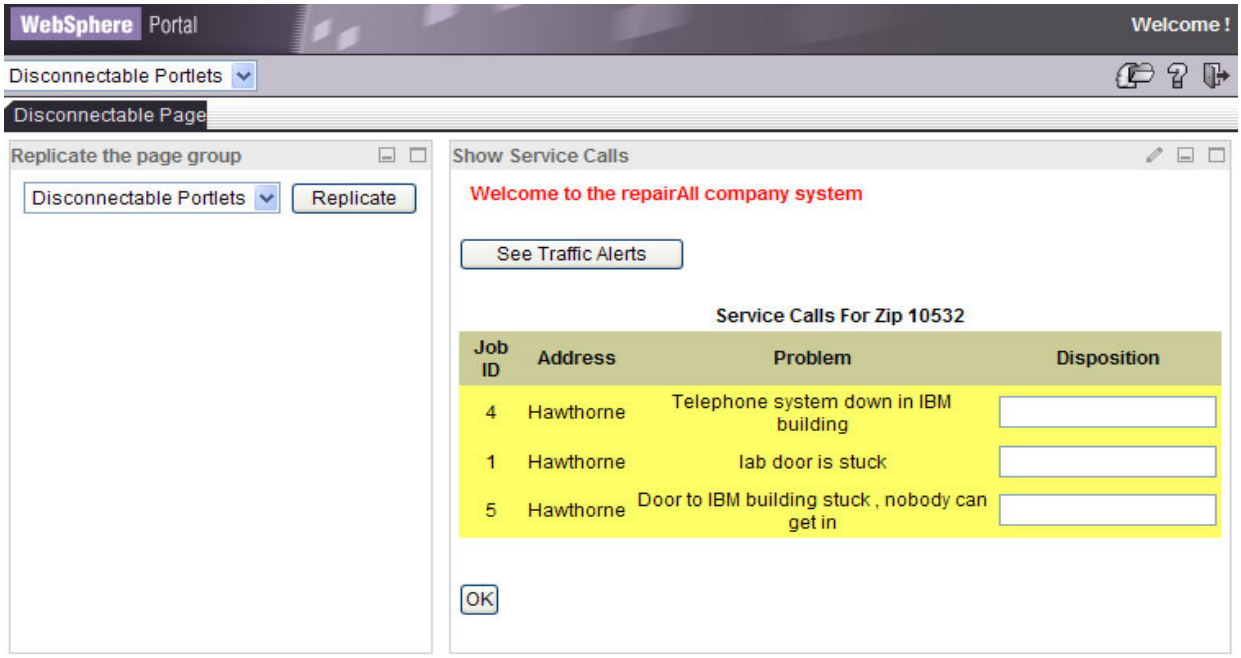

Figure 8.Screen shot from network portlet after synchronization.

4 RelatedWork

The scope of the end-to-end system that we are designing has relations with a number of previous efforts. Below we cover certain key aspects of related efforts.

Distributed MVC programming model: The fundamental programming model we espouse in the paper is a distributed MVC model. Various flavors of distributed MVC programming models have been developed before, primarily in the context of collaborative groupware. Marsic [1], and Krebs et al [2] focus on coordinating the MVC application state on heterogeneous machines. Graham et al [3] focus on state coordination across weak connection or disconnection. All of these efforts focus on coordinating in-memory application state and the MVC instances are tightly coupled to each other. In contrast, we have a simpler approach where state management across MVC instances is not required. The different MVC instances in our approach may not be executing at the same time. The instances only affect each other through the external data that they share.

The Windows DNA distributed MVC architecture [4] uses persistent message queues to communicate between client and server side system models. Again, in our system the client and server side system models do not necessarily communicate directly. We believe that our loosely coupled model makes application design and system operation simpler. There are also commercial systems such Group Kit [5] that let models drive different views. This is complementary to our work and can be applied to our system as well.

Disconnected Operations: One of the motivations for our work is to support disconnected behavior for web applications. There has been a large body of research and commercial work along disconnected operations. The Coda [6], and Ficus [7] systems explored replication of files between clients and servers. The Bayou [8] system explored peer-to-peer replication of objects. Commercial systems like Puma Intellisync [9], and Starfish TrueSync [10] offer systems to replicate personal information like calendar entries, and email. IBM's DB2Sync server [11] facilitates replication of relational databases to mobile clients. The SyncML [12] protocol is an interoperable message format and handshake protocol that a generic replication system can use.

The requirement for disconnected operations in our project is unique and broader than all of the systems above. First, web applications like portlets can access a variety of back end data systems including files, calendars, relational databases, and web services. Our replication subsystem needs to be able to

handle all the different kinds of data. It also needs to work across administrative boundaries and cannot assume a tight coupling between the back end data system and the replication system as in [10,11]. It also has to have a programmable hoarding component that is driven by the web application infrastructure, not a part of any of the above systems. SyncML protocol however, can be incorporated as the lower layer format and handshake protocol in our system. We take an approach where an application is able to specify the data it wants by implementing a particular class. The system invokes the class to obtain the data. During synchronization, data is represented in an XML/DOM format to capture various heterogeneous data types.

Offline web access: There are many commercial systems that offer offline web access. The Internet Explorer browser [13] has page caching and offline viewing support. Other systems like Backweb [14] offer further flexibility such as refreshing the cached client page as the server page changes. The AvantGo [15] system goes beyond offline HTML caching and actually caches data on the client. Using AvantGo applications the data can then be viewed on the client. Our system is distinct from the above systems as we focus on supporting the same Java-based open programming model on the server and the client and same/similar application and business logic execute on the server and the client.

5 Conclusion and Future Work

As outlined in the previous sections we have designed and implemented a web application model that is able to support distribution and disconnection of web applications. We have work ongoing or planned in the near future in a few areas.

One area of focus is implementation of the portal appliance on a pervasive device. We are targeting PDA class devices such as Compaq iPaq and Sharp Zaurus. A key challenge in the pervasive implementation is to reduce dynamic memory usage of the client container. Another key challenge is to revisit the aggregation model for a portal. Currently, the portal aggregator aggregates in a manner such that portlets share the real estate of the screen. In the pervasive form factor, this aggregation model is cumbersome. We are considering other simpler aggregation models (e.g., based on icons). The simpler aggregation model may also have a desirable side effect of reducing the static footprint and dynamic complexity of the aggregator.

Another key area of focus is further exploitation of the client. Once the web application is actually executing on the client, the application can begin to use local resources such as display, and other peripherals like a targeted native application. This will enable web application authors to be competitive in features and function with native applications.

Finally, we are also looking into improving the capabilities of the client web container by supporting other programming model elements like Enterprise Java Beans, lightweight messaging, and transactions.

Acknowledgements

We would like to thank James C. Colson for sharing with us some initial ideas which led to many ideas expressed in this paper. Martin Nally and Ravi Konuru made contributions to structured data objects and data synchronization. We would like to thank John Ponzo and Oliver Gruber for programming model and runtime management discussions.

References

[1] I. Marsic, An Architecture for Heterogeneous Groupware Applications, in *Proc. 23rd IEEE/ACM International Conf. on Software Engineering (ICSE 2001),* Toronto, Canada, May 2001.
[2] A. Krebs, M. Ionescu, B. Dorohonceanu, I. Marsic, The DISCIPLE System for Collaboration over the Heterogeneous Web, in *Proceedings of the Hawaii Int. Conf. on System Sciences*, Hawaii, January 2003.
[3] T. Graham, T. Urnes, R. Nejabi, Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware, *Proc ACM User Interface Software and Technology (UIST 96),* Seattle, Washington, November 1996, pp. 1-10.
[4] Distributed MVC: An Architecture for Windows DNA Applications, *Technical White Paper, Roguewave Corporation,* http://www.roguewave.com/products/whitepapers/mvcwp.pdf

[5] GroupKit, http://www.groupkit.org/
[6] J. Kistler, M. Satyanarayanan, Disconnected Operation in the Coda File System, ACM Transactions on Computer Systems, 10(1):3-25, February 1992.
[7] R. Guy, J. Heidemann, W. Mak, T. Page, G. Popek, D. Rotheier, Implementation of the Ficus Replicated File System, USENIX Conference Proceedings, UCLA, Los Angeles, California, June 1990, pp. 63-71.
[8] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, C. Hauser, Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System, In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, pp. 172-183.
[9] PumaTech Intellisync, http://www.pumatech.com/
[10] Starfish Software – Data Synchronization, http://www.starfish.com/solutions/data/data.html
[11] DB2 Sync Server, http://www.ibm.com/software/data/db2/everyplace/syncserver.html
[12] SyncML Data Synchronization and Device Management, http://www.openmobilealliance.org/syncml/
[13] Web Caching – Related Papers and Articles, http://www.web-cache.com/Writings/papers.html
[14] Backweb–The Offline Infrastructure Company, http://www.backweb.com
[15] AvantGo, Inc., http://www.avantgo.com
[16] JSR 168 Portlet Specification, http://www.jcp.org/en/jsr/detail?id=168
[17] OSGi (Open Services Gateway Initiative), http://www.osgi.org
[18] Websphere MQ Everyplace, http://www-3.ibm.com/software/integration/wmqe/
[19] Object Technology International, http://www.oti.com
[20] J2ME Technologies, http://java.sun.com/j2me