On demand Web-client technologies

by J. Ponzo
L. D. Hasson
J. George
G. Thomas
O. Gruber
R. Konuru
A. Purakayastha
R. D. Johnson
J. Colson
R. A. Pollak

This paper describes a comprehensive set of technologies that enables rich interaction paradigms for Web applications. These technologies improve the richness of user interfaces and the responsiveness of user interactions. Furthermore, they support disconnected or weakly connected modes of operation. The technologies can be used in conjunction with many Web browsers and client platforms, interacting with a J2EE™ server. The approach is based on projecting the server-side model-view-controller paradigm onto the client. This approach is firmly rooted in the Web paradigm and proposes a series of incremental extensions. Most of the described technologies have been adopted by IBM server (WebSphere®) and client products.

The World Wide Web has created an incredible growth environment by making business applications easy to deploy, manage, and access.[1] On the basis of open standards and a ubiquitous browser, enterprises have been able to interface their back-end databases and applications to the Web and create an environment where employees and customers can access a variety of applications from anywhere at any time. The Web successfully promoted hyperlinked documents, browsing, multimedia, and designer-quality look and feel. At the same time, Web adoption represented a step back from advanced desktop user interaction paradigms. It remained limited to a request-response paradigm with page-oriented documents and simple input forms, hindering the development of advanced e-business applications. It also remained limited to a fully connected mode of operation.

Today, customers for WebSphere* want the best of both the desktop and the Web paradigms. They want to retain the advantages of the Web, especially its ubiquity, low total cost of ownership, and strong back-end data integration. But they also want to regain the pre-Web advantages of desktop applications, such as rich and responsive user interfaces, standard widget technologies, and local processing of data.

From its early days, the Web has been evolving to address this concern. One can see an evolving trend that utilizes software from the server with a client-side model-view-controller (MVC) pattern. Originally, HyperText Markup Language (HTML)[2] was simply a view. Rapidly, Dynamic HTML (DHTML)[3] added a controller by adding scripting to the HTML view. More recently, with a move toward eXtensible Markup Language (XML),[4] the markup language may contain not only the view and the controller but also the model.

XML can be used to enhance the desktop paradigm. It can endow native applications with the attributes of Web applications such as back-end data integration. This approach is adopted by Microsoft Corporation, with XML underlying both its Internet Explorer and Office products. Internet Explorer fully supports XML and scripting, but it actually represents an extension to a native foundation. The foundation is a component architecture called Component Ob-

ject Model (COM)[5] combined with a visual embedding framework called ActiveX**.[6] The Web browser becomes simply a user-interface skin on an embedded browser control. The browser control is capable of rendering Web pages in any traditional user interface (UI). Conversely, any Web page may also contain a traditional UI through the ActiveX framework.

XML can also be used to enhance the Web paradigm. For example, it can endow Web pages with a page-local MVC, which translates into richer and more responsive UIs. By extending the classic Web programming model, such architectures can directly reuse current infrastructures and tools to develop more advanced Web applications. Web developers can begin to create Web pages that start to look and feel like native applications.

Enhancing the desktop paradigm enables the exploitation of the inherent native power of the client for richer UIs and local operations, while preserving back-end data integration. Conversely, enhancing the Web paradigm incorporates such natural advantages of the Web as low cost of management and good back-end data integration, while improving user interaction and allowing disconnected operations.

This paper presents a set of technologies that extends the Web paradigm to enable rich interaction. These technologies maintain affinity to the open Web/J2EE** (Java 2 Platform, Enterprise Edition) programming model and leverage the application and tool investments made in that model. One technology, which we call the Rich Browser Framework, extends the J2EE programming model and allows projecting a page-local MVC into current Web browsers, without requiring plug-ins or applets. Our extensions delivered from the server to the browser are rooted in JavaServer Faces,[7] truly making Web applications enjoy qualities of richness and responsiveness close to those of native desktop applications. The Rich Browser Framework technology is now available through IBM WebSphere Portal Server and associated tooling support in IBM WebSphere Studio.[8]

For more demanding applications that need local application logic or support for disconnection, our Rich Browser Framework may still fall short because it relies on markup and scripting technologies. One possible solution would be to adopt Java** applets as a client programming model; however, applets have not been widely accepted in the community. Another possible solution would be to promote a stand-alone client-server Java programming model with a new set of application models and tools. In fact, IBM Workplace Client Technology[9] was recently announced. It is based upon the open-source Eclipse framework and addresses customer requirements that necessitate a client-side application platform. The IBM Workplace Client offers a manageable client-side infrastructure and exports well-known client programming models (based upon J2EE and Eclipse) and associated tooling. The manageability of this client at its core is also based upon an open standard called Open Services Gateway Initiative (OSGi).[10]

IBM delivers products, referred to as Extension Services for WebSphere Everyplace, that leverage these open standards in conjunction with the J2EE programming model, enabling disconnectable business logic. The corresponding programming model, rooted in client-side servlet execution, is outlined in Reference 11. Although it represents a step forward, this approach does not yield the richness associated with a portal-type programming model. Hence, we propose and describe a technology—which we refer to as the *Disconnected Portal*—that extends the approach. It preserves the server-side programming model for the view and controller and allows interaction with parts of the server-side business logic and data projected onto a client-side runtime container. This approach not only results in improved responsiveness, but it also preserves the overall MVC programming model.

In promoting different technologies to develop advanced Web applications, we made sure to retain a seamless experience for end users of those technologies. Whether Web developers use the Rich Browser Framework or the Disconnected Portal as their technology of choice for one application or another, end users still access all their Web applications through their Web browser, as they always did. They remain unaware of crossing between different technologies, except that they now enjoy rich, responsive, and scalable applications.

In this paper, the next section provides an overview of the MVC projection approach and introduces our Rich Browser Framework and the Disconnected Portal. The third section outlines the design of our Rich Browser Framework as an extension to the J2EE programming model and illustrates its power with an example. The fourth section outlines the design and architecture of the Disconnected Portal. The fifth

section discusses related work, followed by the conclusion.

Overview

In recent years, the needs of Web users have been evolving. Simple browsing of documents from desktop machines is no longer sufficient. Web users are mobile and intermittently connected. Web applications are becoming more demanding in regard to the end-user experience and the scalability or flexibility of processing data sets. In this section, we introduce a set of Web-centric technologies that can be added to Web application servers, effectively moving the Web experience forward.

Although all customers want a better Web experience, the price that they are willing to pay for that experience varies widely. Customers have varied deployment, security, resource, and application requirements with different priorities. Some absolutely do not want any additional client footprint because of perceived increased management costs. Some may tolerate a managed-client footprint but want to preserve the Web application investment already made. The technologies we discuss here offer a spectrum of approaches, flexible and adaptable to different environments.

This approach allows developers to adapt to thin or thick clients, ranging from small pervasive devices up to high-end desktops or laptops. We embrace and extend the programming models of both the IBM WebSphere Application Server[12] and IBM WebSphere Portal Server,[8] thereby fully leveraging existing Web skills, tools, and infrastructures. Our approach combines server-side extensions with client-side runtime support, allowing developers to choose the appropriate degree of application partition between client and server sides. This degree of partition has two main components: the Rich Browser Framework and the Disconnected Portal.

The Rich Browser Framework targets scenarios where enhancing the existing Web browser experience is desirable, without requiring the installation of a managed client-side runtime. The impact of the Rich Browser Framework on traditional Web applications is depicted in Figure 1. The top part of the figure shows the current design of a Web application in a portal server. When a user accesses a Web page without the Rich Browser Framework, markup from each portlet[13] configured for that Web page is aggregated by the portal server and returned to the browser. Unless advanced DHTML programming is used by individual portlet developers, the user interactions with the page results in round trips to the server, especially if the page contains tables and forms. Interactions such as sorting, input validation, or incremental display of choices typically result in round trips to the server.

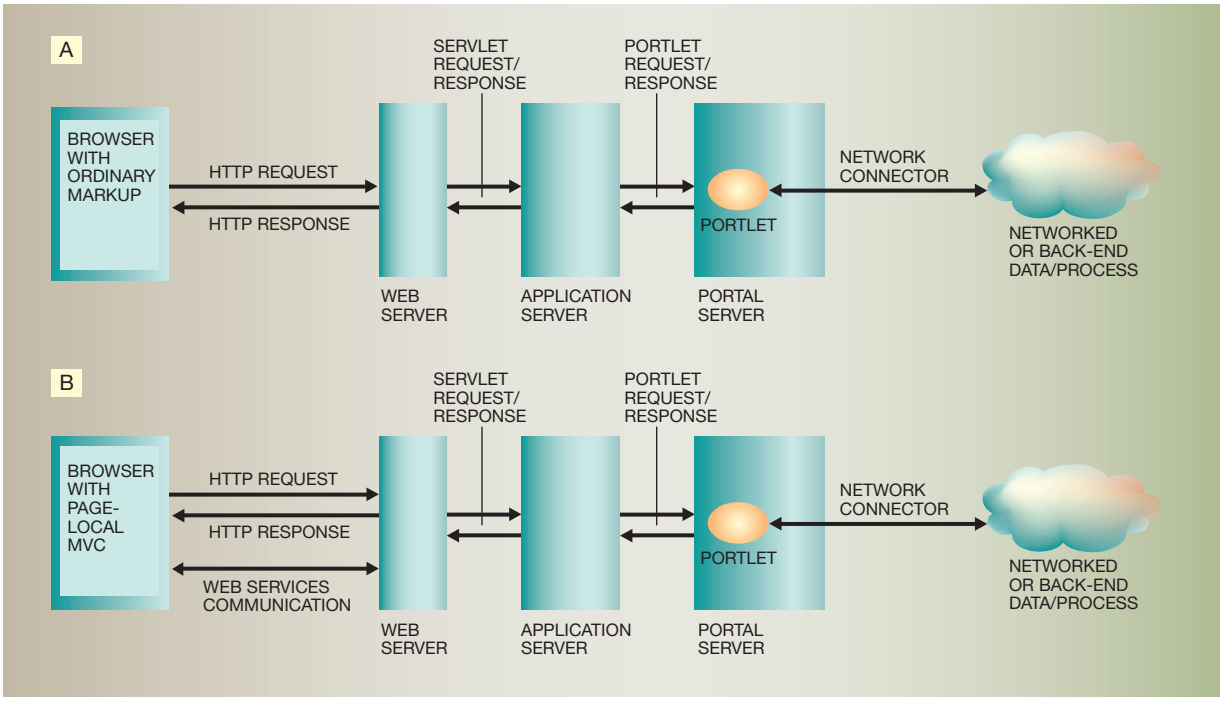
As shown in the lower part of Figure 1, the Rich Browser Framework allows a client-side page-local MVC to be projected from the server, based on XML, HTML, and JavaScript. This is a step forward from straight DHTML, as DHTML was a step forward from HTML. Although DHTML certainly allows a richer and more responsive user experience, it remains a complex technology and suffers from lack of a data model. In the Rich Browser Framework, page markup consists of a full browser-local MVC, containing a data model for multiple page-local views and controllers to render. Updates made to one view are instantaneously visible in other views. Furthermore, views have APIs (application programming interfaces) and may interoperate directly, offering, for instance, coordinated navigation based on the current selection.

In order to make this approach an easy-to-use end-to-end solution, the Rich Browser Framework enhances the server-side technology called JavaServer Faces (JSF).[7] JSF can be thought of as a way to formalize known patterns that have existed for a while in the JSP** (JavaServer Pages**) world and that introduce a strong focus on the creation of solid and maintainable UI front ends. In particular, JSF defines a component architecture and a true life cycle of these components. Components export markup to the client and import client data back to the server. However, JSF does not define what type of markup should be emitted, how the data coming back should be packaged, or even what the data should contain.

The Rich Browser Framework extends JSF by defining a client-side page-local MVC. Technically, this extension only requires DHTML from the Web browser and fits the overall JSP/JSF programming model. The generated Web page is XML-based, leveraging XHTML[2] rendering, XML data models, and advanced JavaScript controllers. The solution is ubiquitous across modern Web browsers and hosting platforms—no browser plug-ins or Java applets are needed.

But this approach can only go so far. Some customers may need enhanced function requiring client-side

Figure 1 Impact of the Rich Browser Framework. In A, almost all user actions result in server round trips. In B, many actions are locally serviced with incremental data refresh.



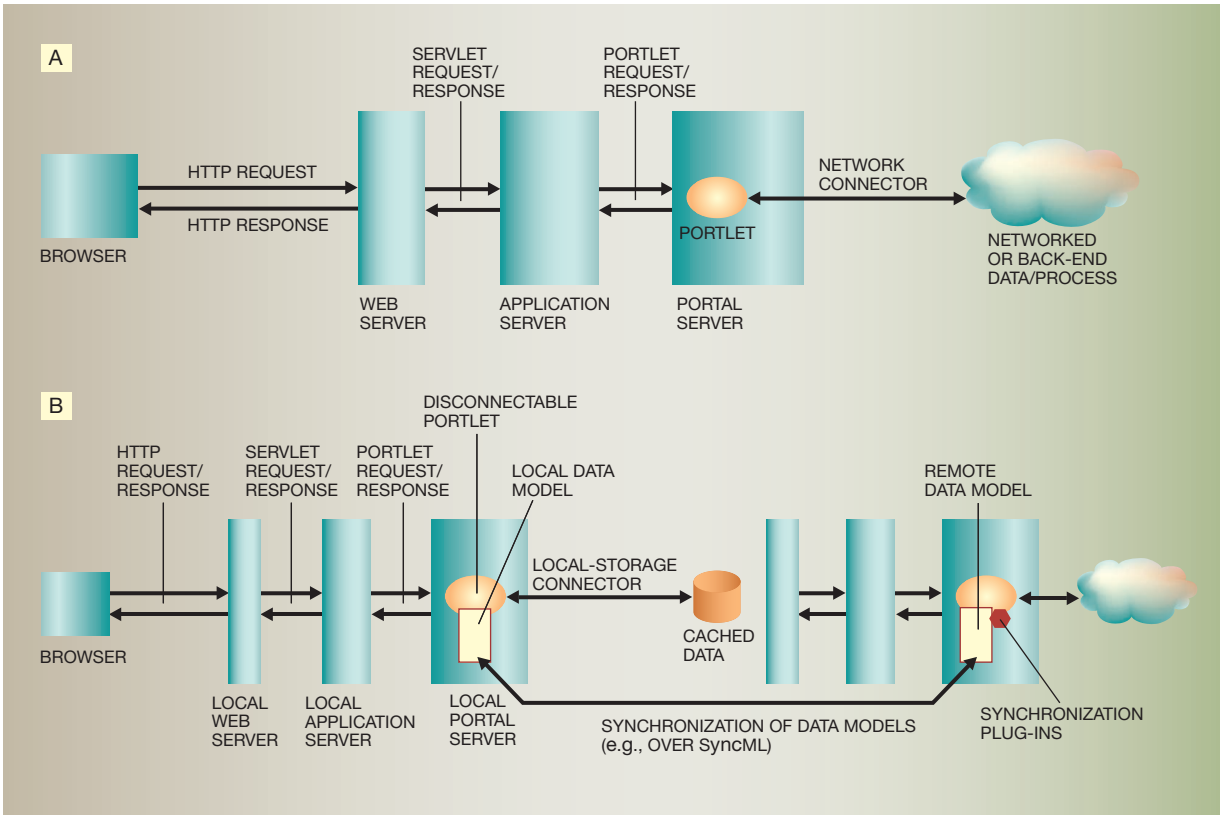
business logic and data. These customers are willing to accept the need to download a managed client platform. Furthermore, these customers want to use the advanced programming model associated with Web page aggregation while running this managed client platform disconnected. The point is that Web developers can continue to leverage their experience and Web skills and continue using their familiar tools. The Disconnected Portal technology addresses this set of users. Disconnected Portal consists of a client-side portal runtime container built as a component on a client platform that supports a portlet programming model with facilities for data caching and synchronization. It further extends the server-side programming model with client-side aggregation of Web applications. The Disconnected Portal maintains a low total cost of ownership by leveraging application life-cycle management and synchronization techniques defined by the OSGi alliance[10] and OMA DataSync[14] (formerly SyncML) respectively.

The impact of the Disconnected Portal is depicted in Figure 2. End users still use their local browser to access their aggregated Web applications. However, all or parts of those aggregated Web applications now run on their client machines, collocated with their browsers. We have shortened the connection length and thereby improved response time. Combined with the technology of the Rich Browser Framework, truly rich and responsive applications can be built. Additionally, because local portlets can be deployed along with business data on the client platform, the approach can support a richer disconnected mode of operation than that afforded by local view or controller. The local business data are a projection of the server-side business data. Advanced replication techniques are used.

Rich Browser Framework

The Rich Browser Framework extends the server-side programming model. It leverages the emerging JSF technology that is being widely adopted, extending it to allow the authoring of advanced Web pages that provide a rich and responsive user experience. In this section, we first discuss JSF technology and then introduce our extensions to it, illustrated with an example.

Figure 2 Impact of the Disconnected Portal. In A, today's portlets are shown. In B, tomorrow's disconnected portlets are shown.



Extending JavaServer Faces. Most Web applications are form-based applications and are comprised of one or more HTML pages. Typically, Web applications are structured around an MVC paradigm: JSPs[15] host view logic, and servlets provide controller logic, whereas JavaBeans** and Enterprise JavaBeans** (EJB**) contain the application data model. JSPs provide application developers with a programming interface for in-lining and dynamically generating HTML markup via custom Java logic or via a JSP custom tag library.

Although JSPs provide adequate support for dynamically generating HTML markup, there is no server-side infrastructure to help manage the states of UI components. Indeed, JSPs dynamically generate and emit markup without preserving the association between the markup and the code that produced it. This generation enables a relatively simple and centralized programming model but exhibits non-optimal responsiveness when compared with form-based Web pages, where the state is actually posted back to the server for validation and processing.

JSF addresses this problem and simplifies the building of user interfaces. JSF allows Web developers to assemble reusable UI components in a page, connect these components to an application data source, and connect client-generated events to server-side event handlers. JSF remains a server-side technology, helping to handle the complexity of the UI by introducing a server-side framework. JSF is most easily deployed using a custom JSP tag library.

When a Web page with JSP tags is processed to answer a page request, the JSF custom tags do two things. First, like any custom tags, they generate HTML markup within the outgoing Web page. Second, they generate a tree of UI components for that page. The tree is built and stored server-side. The

tree retains the states of the UI components. For form-based pages, the browser will post back the form values (new or modified), and the JSF framework will be able to remap those values to their corresponding UI components. Each component can then validate its new value and accordingly update the server-side application data model.

Although JSF facilitates the server-side handling of complex UIs, it does not inherently improve the richness or the responsiveness of Web applications. Under these circumstances the Rich Browser Framework steps in. We introduce the simple idea of projecting a page-local MVC to the Web browser, allowing most of the interactions to remain local, and thus avoiding round trips. First, this idea requires the ability to define server-side a page-local data model at the server that will be sent to the client side along with the Web page. Second, this idea also requires late-binding UI components that can bind on the client side with the page-local model as opposed to the server-side data model.

JSF is a perfect foundation for projecting a page-local MVC. The page-local data model is introduced through a custom UI component that is supported by a custom tag in JSP. The tag refers to a model expressed at design time and a data source to which it applies. In WebSphere Studio, the WebSphere tooling platform, the Web developer creates a data model using Service Data Objects (SDOs).[16] SDOs use a data access model based on the publicly available eCore framework in Eclipse[17] and follow an entity-relationship paradigm. Through SDOs, Web developers can access different data sources such as relational databases, XML stores, or even JavaBeans.

Thus, when a Web page with JSF model custom tags is processed, the model definition that appears in the custom tag is applied to the referenced data source, thereby generating the page-local data model. The data model is inserted in the outgoing page markup stream. The data model is emitted by producing JavaScript code, which, when executed in the browser, creates the data model as JavaScript objects. We have a JavaScript implementation of the SDO API, called JavaScript Service Data Object (JSDO), that allows JavaScript code in the page to access and manipulate the created data model. The JavaScript library for JSDO is very small, just over 100 KB. It is downloaded once and then cached by the Web browser.

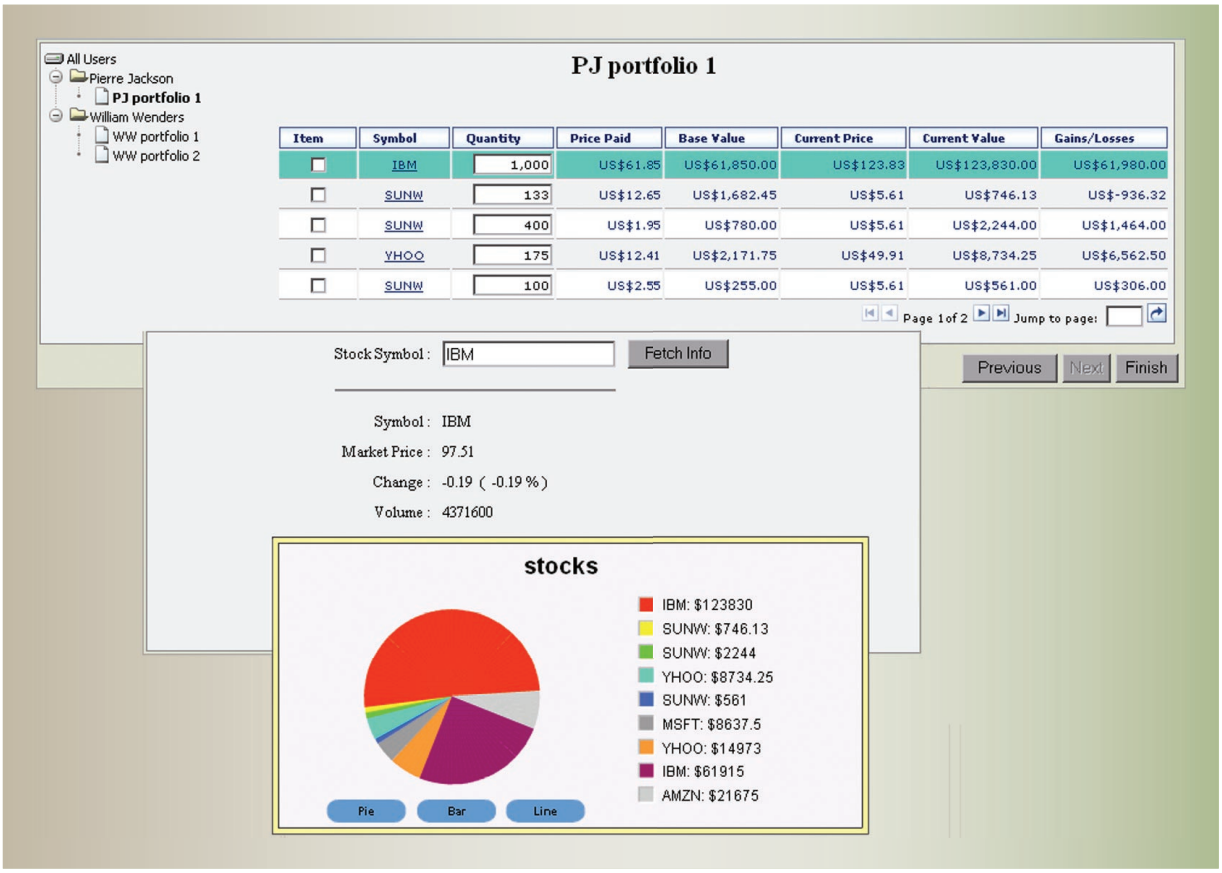
JSDO is the API used by late-binding UI components to bind the client side to the data model. Normal JSF UI components bind to the server-side data model when the JSP is evaluated. Therefore, the emitted HTML tags contain the value they render. For instance, an HTML input tag would contain the actual fragments of text (e.g. "45.00") that the tag is expected to consume. Late-binding UI components in contrast emit HTML tags containing only a reference in the page-level data model. The emitted HTML tags are associated with JavaScript logic called a binder. That binder, hooking the Web browser on-data-change event, will establish a "pub-sub" (i.e., publish-subscribe) relationship between its HTML tag and the data model through the JSDO API.

Updates to the data model are detected and recorded through the JSDO API. Only the modified values are sent back to the server. Through the normal JSF handling of Web browser forms, the modified values make their way back to the server-side UI component that created the data model. The modified values are then processed on the server side, and the application data model is updated accordingly. The validation process follows the normal JSF steps, allowing user-defined validations.

With this approach, programmers no longer need to look at a model through individual UI components and how they individually map to the server-side data model. The data model is defined once, as a whole for each page, at design time. The model is consistent, and its consistency is enforced through the JSDO API that enables the view logic to be applied, including type-checking values, range-checking values, and referential integrity for relationships between entities.

Currently, all standard JSF UI components and regular HTML tags are supported. We also provide an additional set of powerful UI components, such as DataGrid, TreeView, WebService, Flash-Based GraphDraw, and TabPanel. The power of late binding in a page-local data model can be illustrated through the DataGrid component. Our data grid is able to perform common operations, such as paging or sorting, without round trips to the server to refresh the page. Furthermore, because the data model is page-local and therefore shared across UI components, updates made through one UI component are immediately visible in other UI components. In other words, the end user experiences a seamless MVC experience across UI components in the page, without round trips to refresh the page as a whole.

Figure 3 A portfolio example



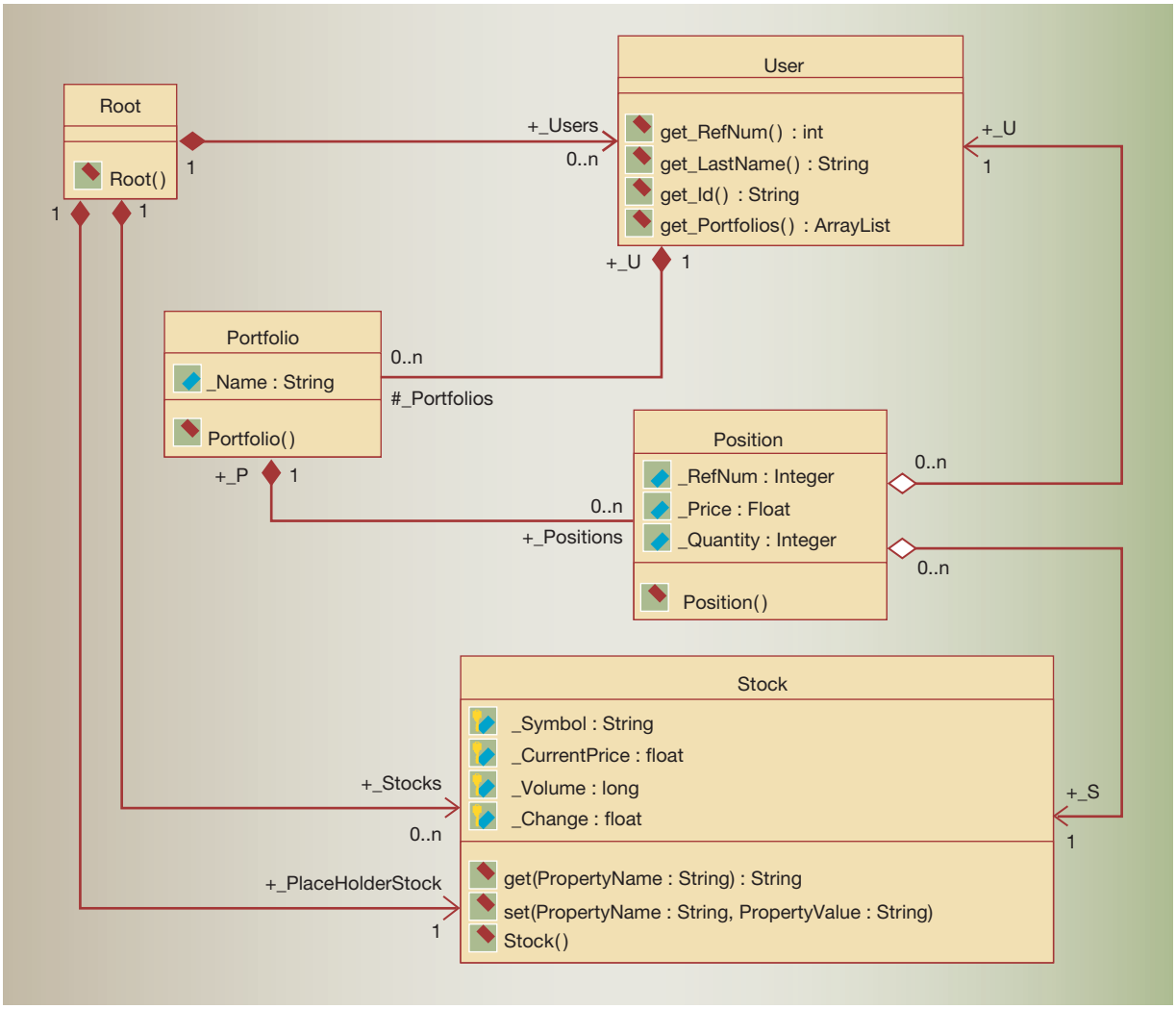
Additionally, by defining a programmatic JavaScript interface to these UI components, Web developers can integrate different UI components through page-local events, again without any round trips.

The programmatic interfaces to these UI components also allow integrating components developed in various other technologies. In fact, any Web browser technology that can call in JavaScript can easily be made to participate in two ways: interoperating with other UI components in the page or manipulating the shared page-local data model. For instance, one can write a Java applet, a Flash control, an SVG (scalable vector graphics) control, an ActiveX control, or a browser plug-in and make it play in the page as a first-class citizen.

Examples. Consider a stock portfolio application (Figure 3) where the user can select one portfolio from among several for display, select a stock, and retrieve its current price by using a Web service. The UI for this application is also shown in Figure 3. (Note that the names in the figures are fictitious. Any resemblance to names used in other contexts is coincidental and not intended by the authors.)

When the user selects a portfolio using the TreeView, the stock positions for that portfolio are displayed in the DataGrid. When the user selects a position in the DataGrid, the stock symbol is copied to the details form. When the user clicks on the "Fetch Info" button, a Web service is called, and the results are populated as output text elements below the input field. All of this happens without a page refresh and without going back to the server (except for the Web service call). The complete data set, all the controls, and their full set of interactions have been projected on the client side in the Web browser, and

Figure 4 Data model for the portfolio example application

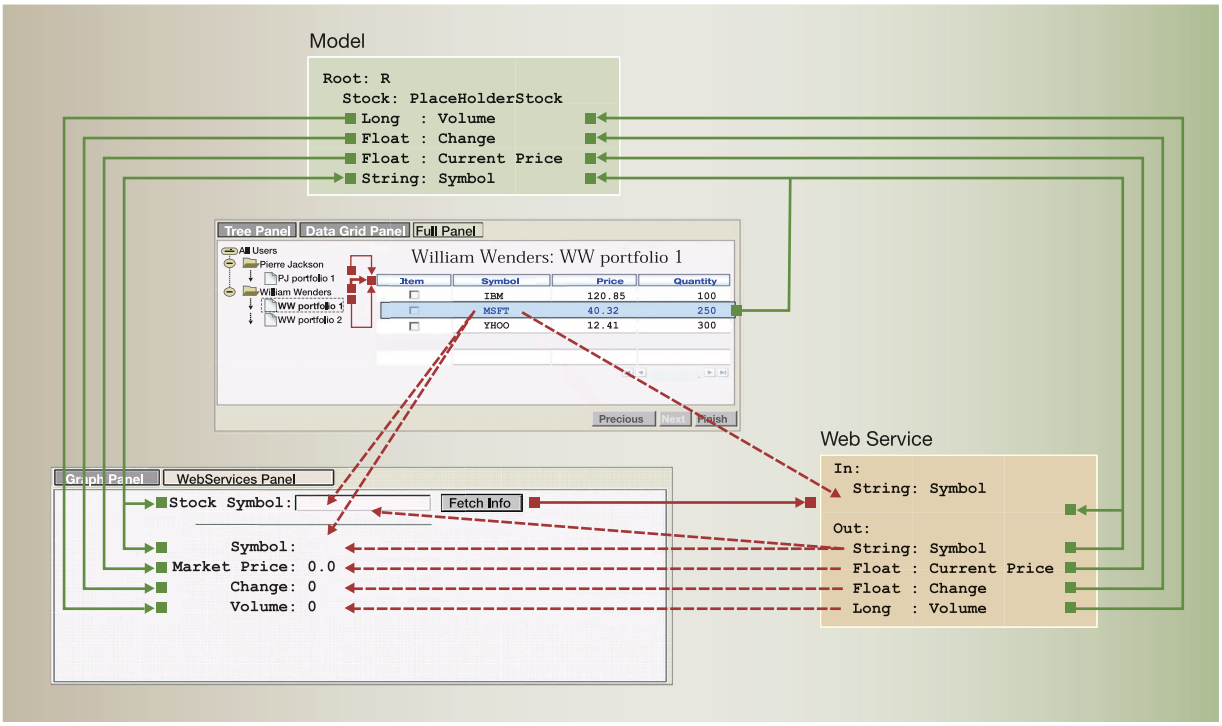


are all managed autonomously by the client-side MVC framework. The application is backed with the data model shown in Figure 4. The root contains a list of users and stocks and a placeholder stock object. Users own portfolios, portfolios own positions, and positions point to stocks.

All the UI components on the page (the TreeView, the DataGrid, the individual input and output fields, and the invisible Web service) are mapped in various ways to this model. The model is created on the server side when the page is generated and populated within the browser as the page is displayed. For this example, we focus on how the various data elements are bound between UI components and how events flow from one to the other.

Figure 5 shows the data model bindings for various UI components for the portfolio example. The arrows represent data model bindings. We show a set of controls mapped to the data model, namely, the inputs and outputs of the Web service, the input-text and output-text fields of the form, and the highlighted row in the DataGrid control. Each control is independently mapped to the model. This mapping provides a simple developer view of control-to-

Figure 5 Control-to-model relationships for the portfolio application



model relationships but affords a complex interaction between controls. The model acts as a hub to receive and dispatch data change events to subscribers. If a control makes a change to some data element, immediately the model will forward this data-change event to the other controls subscribing to the same data element. Those control-to-data relationships create virtual control-to-control relationships. When a row is highlighted in the DataGrid, it appears as though a value was copied to the destination input text, output text, and Web service input field. When the Web service is invoked, its output values are copied into the output-text fields of the form. Effectively, a virtual, programmatic drag and drop occurs. The dashed red arrows indicate those virtual relationships that exist implicitly based on simple control-to-model bindings.

Projecting a server-side MVC-based application onto the browser and retaining the full formality of that model once on the browser enables a new breed of Web pages that contain more interactions and better performance than traditional Web pages.

Disconnected Portal

The Disconnected Portal technology complements the Rich Browser Framework. Whereas the Rich Browser Framework promotes the concept of a page-level MVC for Web applications, the Disconnected Portal allows aggregation of local portlets that work in conjunction with projections of the full server-side MVC onto client machines. This aggregation is achieved through deploying portlets and replicating business data. We first discuss how to make the transition from a pure server-side architecture to a client-server one. We then detail the programming model and data synchronization framework. We finish with an implementation summary and status.

Architecture and design. Most Web applications, including portlets, use server-side business logic and data. Upon a page request, the IBM WebSphere Portal Server invokes individual portlets to generate markup that it will aggregate into one HTML page. To generate its markup, each portlet requests data from various back ends using logical mediators. We use the term *mediator* here to refer to a component

that communicates with a back end and exports a potentially transformed data format or model that may be different from that of the back-end data format. When a client is weakly connected or disconnected, this approach may be inadequate because the server-side Web applications, depending on their needs, can no longer be accessed from a Web browser. Disconnected Portal allows the aggregation of views in the context of autonomous and disconnected operations on client machines. It does so by introducing a client-side Web (portal) container that can aggregate local portlets. These portlets can interact with local instances of the server-side MVC application. Disconnected Portal can also utilize mediation mechanisms to synchronize the cached data with back-end data sources. One key element of the Disconnected Portal approach that contrasts with client-centric approaches is its J2EE programming model affinity. Ideally, local portlets in conjunction with local business logic run in the client-side container with little or no changes.

For instance, consider a simple e-mail portlet as a candidate for disconnection. On the server side, the portlet uses a JDBC** (Java Database Connectivity) mediator to access its e-mail database. The code can run as-is on Disconnected Portal, using a local database into which the end-user e-mails have been replicated. This assumes that the client machine has a local database and a JDBC mediator.

The preceding example illustrates the best-case scenario, in which the server-side business logic and corresponding portlets run as-is on the client. For many reasons, the business logic and corresponding portlets may have to be somewhat modified to run locally. The local business logic and corresponding portlet need to allow for flexible specification of the local dataset as a configurable parameter. In addition, the portlet may have to interact with the user in response to synchronization events such as failures or conflicts. Furthermore, sometimes the client-side dataset will have a different (reduced) schema or a slightly different mediator interface. Nevertheless, even if modifications are necessary, substantial affinity is preserved. The same programming language is used (Java), the same runtime container for business logic (e.g., EJB) and view/controller pairs (portlet containers) are used, substantial reuse of application code is likely, similar mediators are likely, and the overall data schema will bear many similarities.

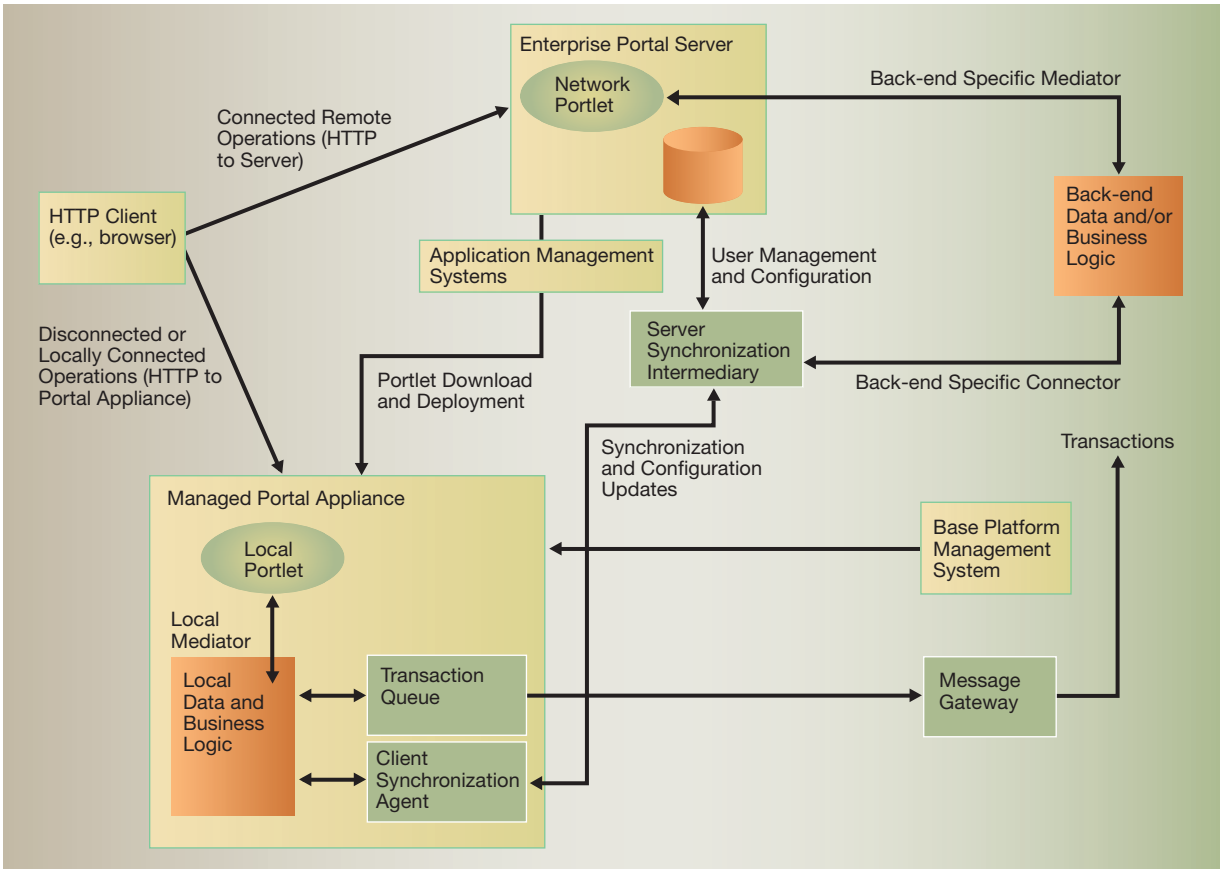
The above affinity principle led us to the architecture depicted in Figure 6. The overall system works in two modes for each Web application (portlet), *connected* and *disconnected*. In the connected mode, a Web application uses solely its server-side logic and data, expressed as portlets leveraging mediators. Portlet markups are aggregated on the server side. In disconnected mode, a Web application solely uses its client-side logic and data in conjunction with corresponding local business logic and data. The Web browser is simply redirected to the local client portal, where the client-side portlets execute the act of accessing local data through local mediators. Portlet markups are now aggregated on the client side.

The purpose of the rest of the infrastructure is to effectively enable the notion of local portlets in conjunction with local business logic and data and the management thereof. The *application management system* facilitates business logic, data, and portlet code download and deployment. The *base platform management system* facilitates updates to the underlying platform itself, which includes business logic containers as well as the Disconnected Portal container. The client and the server synchronization agents coordinate to materialize and keep data synchronized on each node. The synchronization agents may also be used to manage replication of portal user configuration data such as preferences and credentials. Off-line operations such as transactions are supported by a local persistent message queue that is coordinated with a message gateway interfacing to various transactional back ends.

This client-server infrastructure has two major goals: ease of management and deployment, and accommodation of data source heterogeneity. By extending the portlet programming model to clients and by requiring a client-side footprint, however small it may be, the relative zero-management attribute of the Web paradigm is compromised. It is therefore critically important to base the client-side infrastructure on platforms amenable to overall ease of management. The scope of management includes life-cycle management of both platform and application code, as well as management of configuration data.

Accommodating data source heterogeneity is also crucial because business logic and the corresponding portlets could potentially obtain their data from a multiplicity of data sources. These data sources could include databases, Web services, files, and many others. The data from these sources must be

Figure 6 The Disconnected Portal system architecture

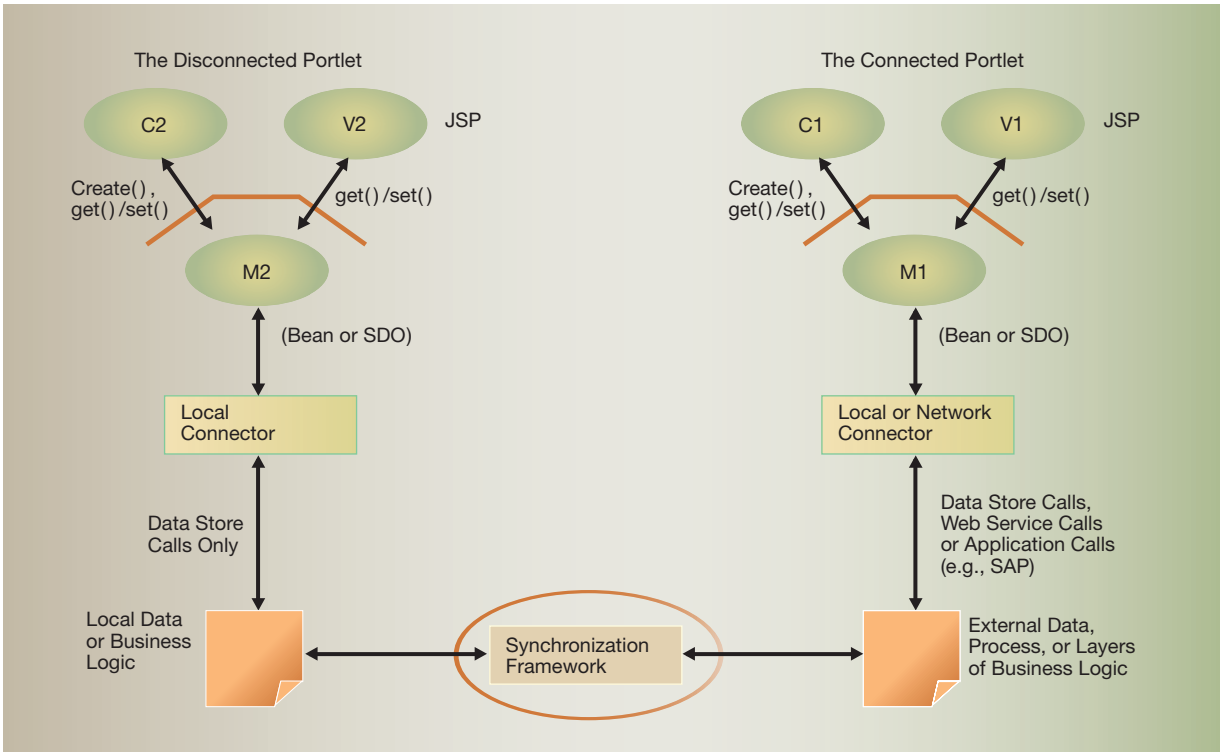


hoarded, that is, prefetched in anticipation of their use during disconnected operation. The hoarding infrastructure has to make it easy to create the data that need to be replicated across heterogeneous back-end data sources. One part of the challenge is to target client portal containers with a limited set of mediators. The following subsection discusses certain key aspects of the Disconnected Portal and corresponding data architecture, including the programming model and data synchronization, and also discusses user experience issues by means of an example.

Programming model. The Disconnected Portal espouses a distributed MVC model of programming (Figure 7) in which the different MVC instances are loosely coupled and only interact through changes they introduce to external data. Consider the MVC of the connected portlet. The view is often materialized as JSPs; the controller is Java code (servlets) that coordinates application flow. The model is often realized as JavaBeans or SDOs, providing a source-independent abstract view of data. The actual back-end business data for a portlet can be data sources such as databases or Web services that are accessed by mediators or connectors. The back-end data can actually be further abstracted by business logic encapsulated in EJBs.

The disconnected MVC is the same as the connected MVC, except that the client-side portlet uses a local connector or mediator to access locally cached data and corresponding business logic. If the API for the local mediator is identical to the connected mediator, then there is often no change to the portlet code. In some cases, programming the local portlet is not

Figure 7 Loosely coupled MVC model disconnected portlets



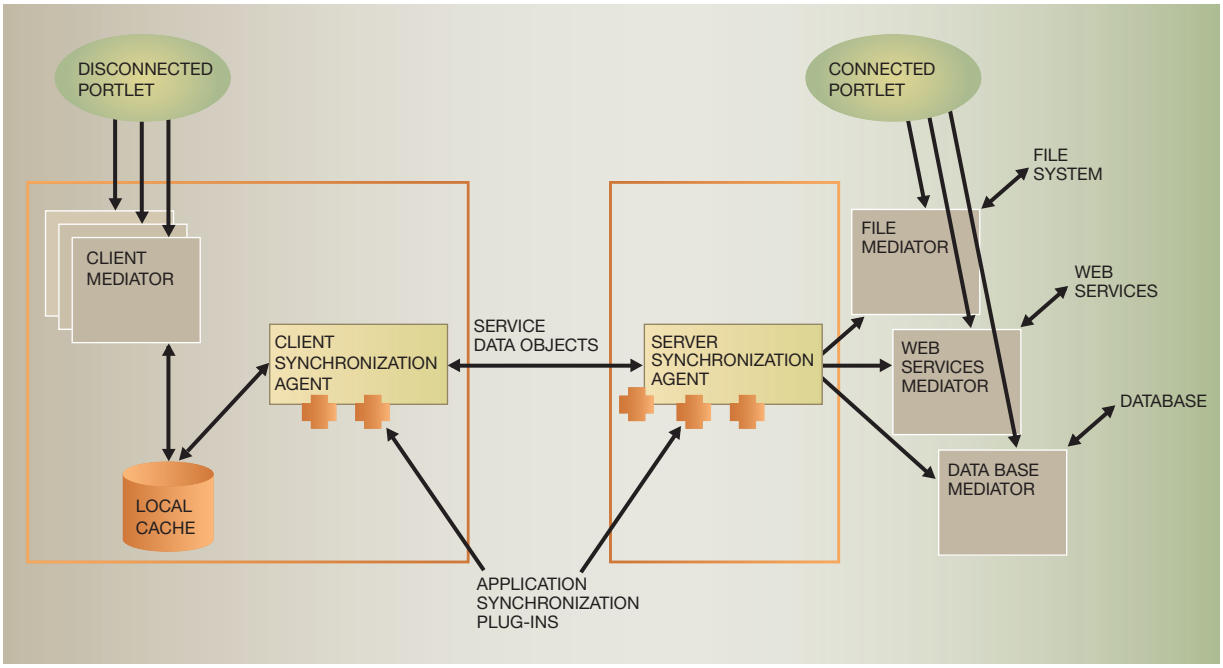
straightforward because the local mediator API may have a different form and semantics than the connected mediator (e.g., one for a Web service). When the data in the connected mode assume existence of a business logic layer such as an EJB, the data in the local disconnected mode require a local EJB, which in turn accesses local data such as a client-side database. In addition, the portlet author may need to implement a few additional functions, such as a specification of the actual data needed by the view layer and a potential mechanism for resolution of conflicts if that data (or the corresponding business data) can be modified simultaneously somewhere else. For many portlets though, this may be drastically simplified by tooling or system policies.

The disconnected and connected instances of the portlet are loosely coupled. They are not required to be coordinated at the time of program execution. The connected and disconnected versions may only affect each other via the corresponding back-end business data that are affected by their manipulation of view data. Any changes that collide at the business data layer may be reconciled through the synchronization framework or by some other means, such as a dominance policy whereby one node is considered a master and the other a slave. Other policies can easily be imagined as well.

Data synchronization. One form of delivering data for Disconnected Portal is data synchronization. A challenging problem in synchronizing data is that of data heterogeneity. Some synchronization solutions exist for specific back-end data types such as relational, file, and calendar data. Employing individual solutions for individual data types does not scale as diversity in data types increases. Moreover, a number of existing solutions (e.g., DB2* Everyplace Sync Server[18] and Lotus Domino*[19]) are difficult to simply "plug in" because of various assumptions made, such as schema-level data conformance, lack of programmatic control from applications, and relatively high management overhead.

The synchronization architecture (Figure 8) is centered on the notion of structured data objects, which abstract back-end data and are populated and maintained by means of disconnectable mediators. In turn,

Figure 8 Synchronization architecture for Disconnected Portal

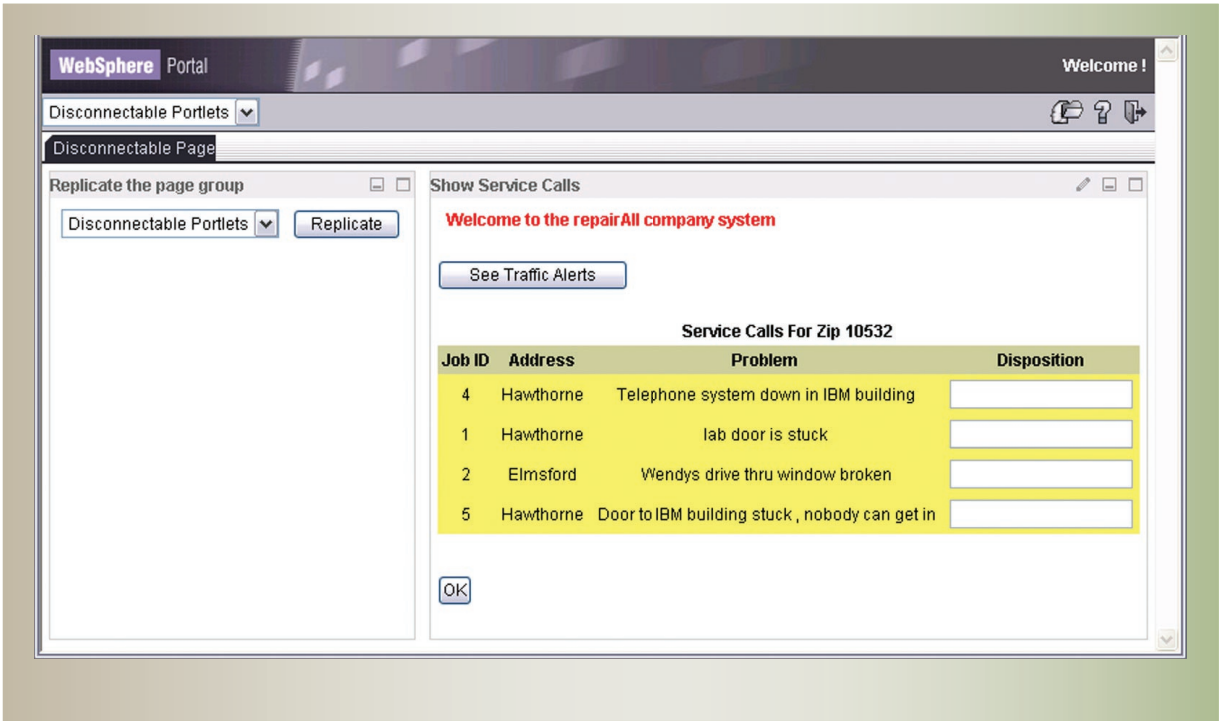


portlets create local data representations of these structured data objects using local mediators. One example of a structured data object is the SDO that was referenced earlier. Portlets are able to register synchronization plug-ins with a server synchronization agent that executes the plug-ins to create a set of structured data objects on the client. Disconnected Portal uses either the same or a similar mediator API to operate on locally cached SDOs. During replication, the modifications to SDOs are captured and reconciled with changes made to the back end. The application-supplied synchronization plug-in may again be invoked for managing conflict resolution. Standard synchronization protocols such as OMA Data Sync (formerly SyncML) can be used to encapsulate and manage the exchange of SDOs between the server and client agents.

Example. By facilitating portlet aggregation on the client and by providing a first-class portlet container, we were able to maintain the same look and feel of the connected portlet view and the same function of the connected portlet controller. In the general case however, it is not possible to completely abstract the distinction between connected and disconnected modes. For instance, because the local data store may have limitations, end users may perceive a lack of data availability during disconnection.

The limitations of disconnected mode impose additional demands on the users' knowledge of disconnected operation and their ability to handle disconnection-related tasks; for example, some portlet function may be disabled in the disconnected mode. Furthermore, for some portlets we expect the user to specify what data should be collected for use locally in preparation for disconnection because it is impossible in the general case for the system to automate that task. In some cases, the user also has to take explicit action to initiate disconnection of portlets and data before actually being disconnected. This action has effects on how local data delivery, local business logic, and local portlets operate because the View/Controller ongoing replication can be user-directed or transparent upon detection of network connection. Another demand placed on the user is caused by dealing with potential data synchronization conflicts. Nevertheless, our approach, illustrated by the following example, enables and promotes a strong linkage between connected and disconnected modes of operations.

Figure 9 Example of field worker portlet running on the server-side portal

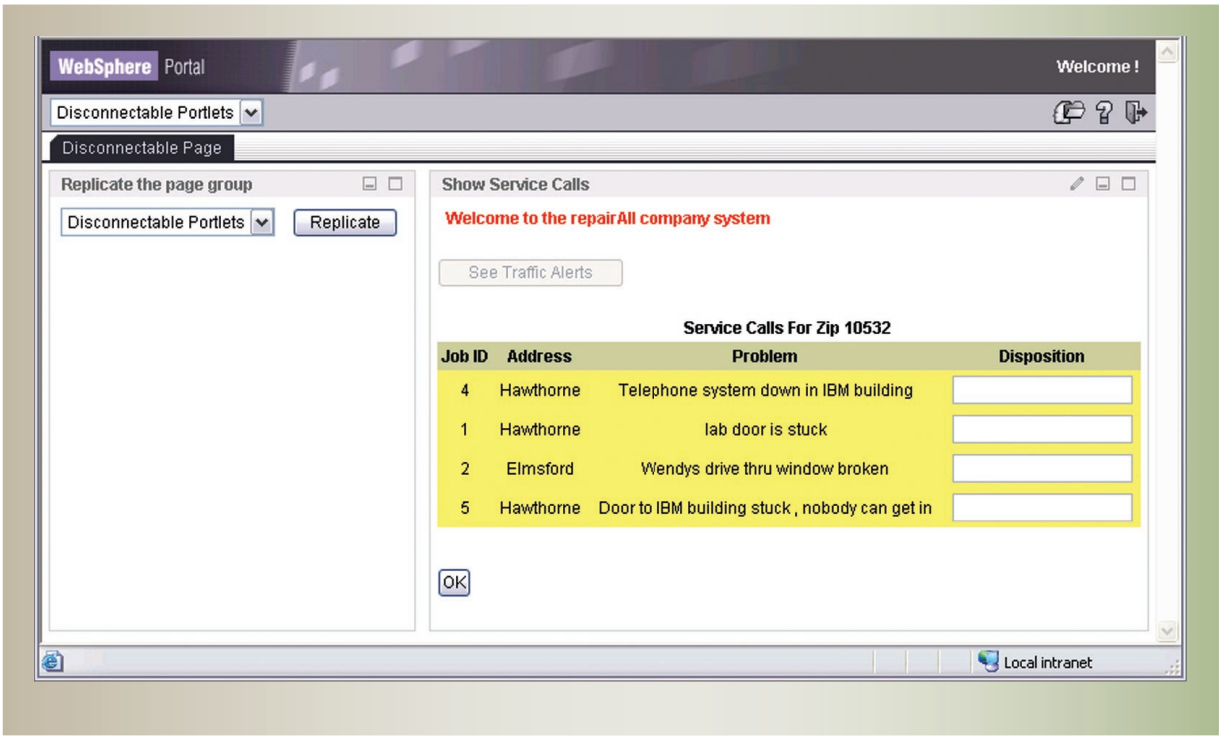


The example entails an application, in which the user is a mobile field worker who makes service calls in a region defined by a zip code. Figure 9 is a screen shot of the "disconnectable portlets" page group of the connected portal. This page group has one page with the application portlet and one system portlet. When connected, the system portlet allows the user to replicate portlets and portlet data pertaining to a page group, which is an additional action and demand placed on the user. The connected portlet shows the job list at the beginning of the day. Note that the "See Traffic Alerts" button is active in the connected 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 Disconnected Portal. This replication occurs by executing a mediator for the server-side business logic that extracts the job list data and delivers it to the client. If the client has no copy, then a local data copy is created. If there is already a local copy, the mediator executes the conflict resolution policies to ensure that any updates made to the previous copy are handled according to policy. During the course of the day, the worker updates the job status and makes notes about the jobs, all done while disconnected. Figure 10 is a screen shot from the disconnected portlet on the client that reflects completion of one of the jobs. Note that the "See Traffic Alerts" button is not active, because this function is not available while disconnected from the network (an example of disconnection affecting function and user experience).

The worker later reconnects to the network, at which point the mediator delivers the local data back to the server while simultaneously downloading any updates made at the server. Just as before, if there are conflicts, the conflict resolution mechanisms are invoked using the previously defined conflict resolution policy. The result is that new jobs are downloaded, job status is updated, and job status completion that is recorded at both places is "reconciled." The screen shot in Figure 11 is from the connected portlet after synchronization with the assumption that no new jobs have been added to the list. Note that the "See Traffic Alerts" button is active. Because the list only shows incomplete jobs, Job 4 has been removed.

Figure 10 Example of disconnected portlet running on the client device



Implementation summary and status. The Disconnected Portal is an ongoing project at this time, but certain milestones have been achieved. We chose to implement the client-side portal container on top of the IBM Extension Services for WebSphere Everyplace (ESWE) platform[20] using the Java language. This implementation enables the client-side portal container to run on various client form factors ranging from laptop-class to PDA (personal digital assistant)-class devices. For laptop-class devices, the implementation is a derivative of the server-side WebSphere Portal Server. Currently the static footprint is less than 12 MB. Server-side WebSphere Portal Server code has been carefully subset, and certain functions reimplemented to achieve an appropriate footprint for laptops. For smaller devices such as a PDA, the client portal container is based on the emerging portlet API standard,[13] and currently has a static footprint of less than 4 MB.

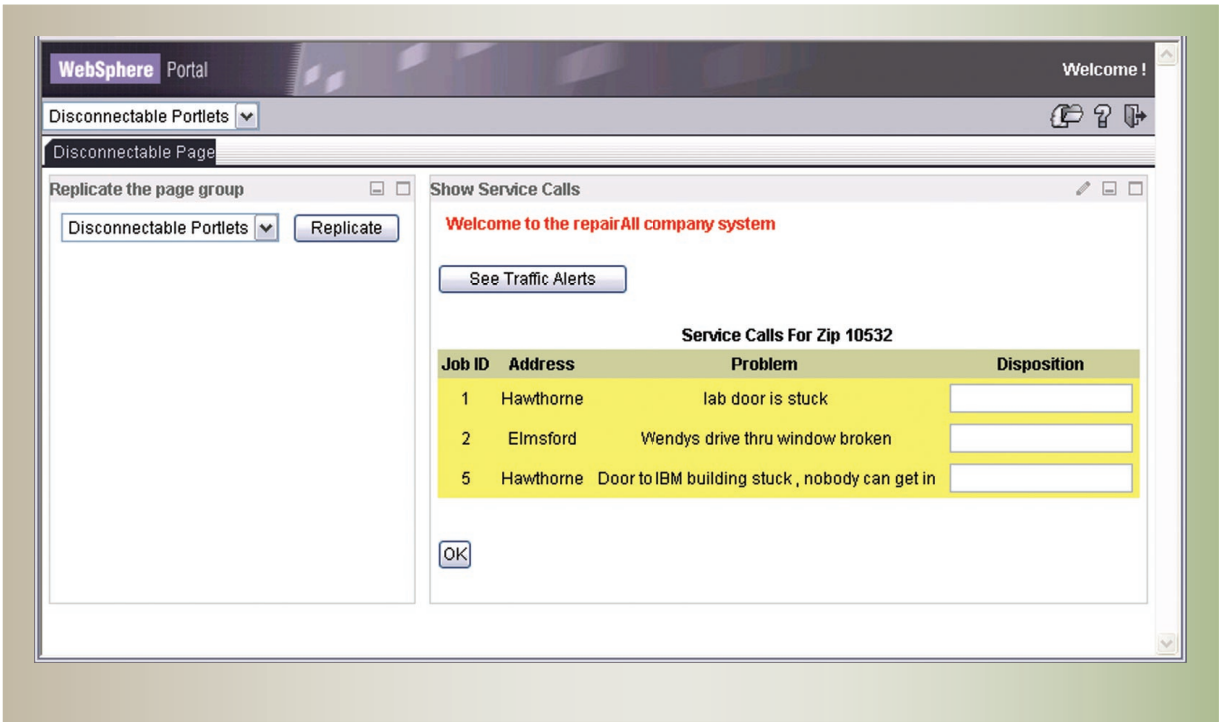
Both client portal containers (laptop and PDA) utilize OSGi life-cycle management for ease of deployment. The notion of bundles in OSGi and its overall service-oriented architecture afford a modular design of the portal and ease of incremental updates and maintainability. By virtue of using the ESWE platform, the client portal container supports the management and deployment of portlets (along with their configuration), as well as replication for certain data types.

The current focus areas of ongoing research include realization of a more general synchronization framework as has been described and more intelligent means of data caching to reduce user involvement in the overall process.

Related work

The overall on-demand client work presented here relates to a number of efforts. The basic ideas of MVC projections and distributed MVC have been studied before. In the context of collaborative groupware, Marsic,[21] and Krebs et al.[22] focus on coordinating the MVC application state on heterogeneous machines. Graham et al.[23] focus on state coordination across weak connection or disconnection. All of these efforts are aimed at coordinating the in-memory application state, and the MVC instances are tightly coupled to each other. In contrast, we have a simpler

Figure 11 Example of network portlet after synchronization



approach whereby state management across MVC instances is not required. The Rich Browser Framework simply projects a page-local MVC hosted by the browser.

XForms[24] is a W3C specification that supports building presentation-level applications with explicit MVC and the modeling of constraints. One of the motivations behind this work is to offload server-side processing so that the user has more responsive experiences. Some of our ideas overlap with that of XForms, except that we extend these ideas in the context of Java and J2EE programming model and tool availability.

The WinForms[25] technology offered by Microsoft Corporation is conceptually similar to a Rich Browser Framework. Microsoft provides a .NET-based[26] programming model using the Microsoft Common Language Runtime (CLR) and XML and Web service[27] technologies. The WinForms technology attempts to provide off-line form data access, application update and deployment, and a simple data model called ADO.NET[28] that is comparable to SDOs.[16]

Disconnected Portal employs a loosely coupled MVC model in which the different MVC instances may only affect each other through the external data that they share. This approach inherently makes our system simple because distributed state management and consistency is a difficult problem and need not be addressed for our domains. The Microsoft Windows** DNA distributed MVC architecture[29] 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 other commercial systems, such as Group Kit,[30] that let models drive different views. This method is complementary to our work and can be applied to our system as well.

One of the motivations for our work is to support disconnected behavior for Web applications. A large body of research and commercial work has been done on disconnected operations. The Coda[31] and

Ficus[32] systems explored replication of files between clients and servers. The Bayou[33] system explored peer-to-peer replication of objects. Commercial systems such as Puma Intellisync[34] and Starfish TrueSync[35] allow systems to replicate personal information such as calendar entries and e-mail. The IBM DB2 Everyplace Sync Server[18] facilitates replication of relational databases to mobile clients. The OMA Datasync[14] protocol is an interoperable message format and handshake protocol that can be used by a generic replication system.

The requirement for disconnected operations in the Disconnected Portal is unique and broader than all of the systems mentioned above. First, Web applications such as portlets can access a variety of back-end data systems, including files, calendars, relational databases, and Web services. A robust 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 Reference 18. It also has to have a programmable hoarding component that is driven by the Web application infrastructure and is not a part of any of the above systems. The OMA Datasync 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 are represented in an XML/DOM (Document Object Model) format to capture various heterogeneous data types.

Many commercial systems offer off-line Web access. The Internet Explorer browser has page caching and off-line viewing support. Other systems such as Backweb[36] offer further flexibility, such as refreshing the cached client page as the server page changes. The AvantGo[37] system goes beyond off-line HTML caching and actually caches data on the client. With use of the AvantGo applications, the data can then be viewed on the client. Disconnected Portal extends the basic ESWE system in a manner distinct from the above systems as it focuses on extending a portal-based programming model in conjunction with the same Java-based open programming model on the server and the client with same or similar application and business logic executing on the server and the client.

Conclusion

In this paper, we have presented two technologies involving projections of the server-side MVC programming model and applications onto the client. The Rich Browser Framework introduces the notion of a page-local MVC associated with each Web page. It allows client-side operations via scripting and allows manipulation of a structured dataset associated with a page. It improves Web application responsiveness without introducing any additional client footprint by simply exploiting features of modern browsers such as rich scripting support.

The Disconnected Portal project facilitates projection of the aggregation of local portlets to work in conjunction with local instances of the full server-side MVC application onto the client. We further extend the notion of disconnected operations for a collection of Web applications by extending the base platform to include local application data and corresponding replication along with store-and-forward operations, including conflict detection and resolution. Potentially, fully functional server-side Web applications can execute in the client, thereby substantially extending the reach of Web applications. Furthermore, the Rich Browser Framework can be juxtaposed with the Disconnected Portal on the client to further enhance Web application responsiveness and function. Together, they take a step forward in making the Web paradigm functionally comparable with the client/server paradigm while preserving the desirable characteristics of the Web paradigm.

Acknowledgments

The authors wish to thank Stephen Brodsky, Martin Nally, Jim Russell, John Schumacher, Brian White Eagle and B. J. Hargrave for technical discussions and their participation in helping shape the technical approach for this work. We also thank Joe Kubik, Michael Moser, Veronique Perret, Bill Gengler, Michael Burkhart, Dave Klein, Thomas Watson, Daniela Bourges-Waldegg, Marcel Graf, Yann Duponchel, Marion Blount, and Danny Yeh for their participation in the design and development of the Disconnected Portal client and data synchronization.

Cited references

1. D.F. Ferguson and R. Kerth, "WebSphere as an E-business Server," *IBM Systems Journal* **40,** No 1, 25–45 (2001).
2. "*XHTML™ 1.0, The Extensible HyperText Markup Language,*" 2nd Edition, W3C (August 2002), http://www.w3.org/TR/xhtml1/.
3. J. C. Teague, *DHTML and CSS for the World Wide Web*, 2nd Edition, Visual QuickStart Guide, Peachpit Press, Berkeley, CA (2001).
4. E. T. Ray, *Learning XML*, O'Reilly, Sebastopol, CA (2001).
5. J. Löwy, *COM and .NET Component Services*, O'Reilly, Sebastopol, CA (2001).
6. D. Chappell, *Understanding ActiveX and OLE*, Microsoft Press, Redmond, WA (1996).
7. JavaServer Faces Specification, JSR127, Java Community Process, at http://www.jcp.org/en/jsr/detail?id=127.
8. WebSphere Portal for Multiplatforms, IBM Corporation, http://www.ibm.com/software/genservers/portal.
9. IBM Workplace Client Technology, IBM Corporation, http://www.ibm.com/software/swnews/swnews.nsf/n/jmae5vgqjl.
10. OSGi (Open Services Gateway Initiative) Alliance, San Ramon, CA, http://www.osgi.org.
11. J. Colson, *Diversity Dictates in Pervasive Computing*, http://www.eetimes.com/story/OEG20010122S0037.
12. *WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide*, Redbook, IBM Corporation, http://www.redbooks.ibm.com/abstracts/sg246504.html.
13. Portlet Specification, JSR 168, Java Community Process, http://www.jcp.org/en/jsr/detail?id=168.
14. U. Hansmann, R. Mettala, A. Purakayastha, and P. Thompson, *SyncML: Synchronizing and Managing Your Mobile Data*, Prentical Hall, Inc., Upper Saddle River, NJ (2002).
15. *Servlet and JSP Programming with IBM WebSphere Studio and Visual Age for Java*, WebSphere Domain, Redbook, IBM Corporation, http://www.redbooks.ibm.com/abstracts/sg245755.html.
16. *Specifications: Service Data Objects, WorkManager, and Timers*, IBM Corporation (November 2003), http://www.ibm.com/developerworks/library/j-commonj-sdowmt/.
17. *Eclipse Platform Technical Overview*, Object Technology International (February 2003), http://www.eclipse.org/whitepapers/eclipse-overview.pdf.
18. DB2 Everyplace Sync Server, IBM Corporation, http://www.ibm.com/software/data/db2/everyplace/syncserver.html.
19. B. Hitchcock, *Domino Off-Line Services: An Administrator's and Developer's Guide*, IBM Corporation (2001).
20. Extension Services for WebSphere Everyplace, IBM Corporation, http://www.ibm.com/software/pervasive/everyplace.
21. I. Marsic, "An Architecture for Heterogeneous Groupware Applications," *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, Toronto, Canada (May 2001).
22. A. M. Krebs, M. Ionescu, B. Dorohonceanu, and I. Marsic, "The DISCIPLE System for Collaboration over the Heterogeneous Web," *Proceedings of the Hawaii International Conference on System Sciences*, Hawaii (January 2003).
23. T. C. N. Graham, T. Umes, and R. Nejabi, "Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware," *Proceedings ACM User Interface Software and Technology (UIST'96)*, Seattle, WA (November 1996), pp. 1–10.
24. "*XForms 1.0,*" W3C Recommendation 14, W3C (October 2003), http://www.w3.org/TR/xforms/.
25. C. Payne, *Teach Yourself .NET Windows® Forms in 21 Days*, SAMS Publishing, Indianapolis, IN (2002).
26. A. Fedorov, *A Programmer's Guide to .NET*, Addison Wesley Publishing Co., Boston (2002).
27. *Web Services Description Language (WSDL) 1.1*, W3C Note, W3C (March 2001), http://www.w3.org/TR/wsdl.
28. D. Sceppa, *Microsoft ADO.NET (Core Reference)*, Microsoft Press, Redmond, WA (2002).
29. *Distributed MVC: An Architecture for Windows DNA Applications*, Technical White Paper, Rogue Wave Software, Inc., Stingray Division (1999), http://www.roguewave.com/products/whitepapers/mvcwp.pdf.
30. GroupKit, http://www.groupkit.org/.
31. J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Transactions on Computer Systems* **10,** No. 1, 3–25 (February 1992).
32. R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page, Jr., G. J. Popek, and D. Rothheimer, "Implementation of the Ficus Replicated File System," *USENIX Conference Proceedings*, UCLA, Los Angeles (June 1990), pp. 63–71.
33. D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. I. Spreitzer, and C. Hauser, "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System," *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 172–183.
34. *Intellisync,* PumaTech, http://www.pumatech.com.
35. *Intellisync SyncML Server*, Intellisync Corp., http://www.truesync.com.
36. Backweb, The Offline Web Company, San Jose, CA, http://www.backweb.com.
37. AvantGo, a service of iAnywhere Solutions, Dublin, CA, http://www.avantgo.com.

Accepted for publication January 16, 2004.

John Ponzo *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (jponzo@us.ibm.com).* Mr. Ponzo is a Senior Technical Staff Member at the Watson Research Center. His major focus areas are Web application runtimes, Web development tools, and desktop client runtimes. He led several efforts at Research, including the IBM XForms Processor, Eclipse Web technology integration, and Sash projects. He made key technology contributions to WebSphere Studio, Eclipse, WebSphere Application Server, and Lotus Workplace. He received an M.S. degree in computer science from Polytechnic University, and a B.S. degree in computer science from Manhattan College.

Laurent D. Hasson *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (ldhasson@us.ibm.com).* Mr. Hasson is currently the lead architect for the Rich Client Browser Framework for the WebSphere line of products, aiming at delivering more interactive and responsive Web applications to standard unaugmented browsers. He joined IBM in 1996 in Toronto, where he spent two and a half years as the lead developer and architect for Net.Commerce. He moved to IBM Research in Hawthorne, New York in 1998, where he continued his role in the WebSphere Commerce line of products as a key contributor. In 2000 and 2001, he acted as the lead architect and development manager for WebSphere Commerce Suite, Marketplace Edition, an advanced business-to-business platform centered around hosted multiparty dynamic trading. Mr. Hasson received an M.S. degree in computer science from Brown University in 1994 and a B.A. in computer science and mathematics from New York University in 1992.

Jobi George *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (jobi@us.ibm.com).* Mr. George received a B.S. degree in computer science from the National Institute of Technology (NIT) at Surat, India in 1992 and an M.B.A. degree from New York University in 2002. He joined IBM Research in 1997 and has been working in the area of client technologies, Web application runtimes, and developer tooling. He was part of the team that designed and shipped the Sash Web Applications Framework, Xforms processor, WebSphere Studio script debugger, and Eclipse Web Integration technology. He has made technological contributions to IBM products such as WebSphere Studio, Eclipse, and Lotus Workplace.

Gegi Thomas *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (gegi@us.ibm.com).* Mr. Thomas is a research staff engineer in the E-Business Frameworks Department at the Watson Research Center. He received a B.A. Sc. in systems design engineering and a B.A. in economics from the University of Waterloo in 2002 and 2003, respectively. He subsequently joined IBM Research. He has also worked in the Emerging Technologies Department in the IBM Software Group (Raleigh), where he was a contributing member of the IBM Web Services Toolkit.

Olivier Gruber *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (ogruber@us.ibm.com).* Dr. Gruber received his Ph.D. in the field of object systems from the University Pierre et Marie Curie in France in 1992. Until 1995, he led a European project on large-scale persistent object systems at the French National Research Institute for Computer Science (INRIA). He joined the IBM Almaden Research Center in 1995 and worked on data management systems. In 1997, he transferred to the Watson Research Center, where he led the kernel team of the first research experiments around web application servers that changed into the present WebSphere Application Server. Since 1999, he has been involved with merging Java and Web client technologies, leading the research aspects of a client-side Java platform with server-side affinity. He started and led to success the synergy between Eclipse and OSGi, available in Eclipse 3.0, thereby moving Eclipse toward a Rich Client Platform (RCP). He is also very involved with the Lotus Workplace effort.

Ravi B. Konuru *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (rkonuru@us.ibm.com).* Dr. Konuru is a research staff member in the Systems and Software Department. His research interests are in the areas of operating systems and distributed computing. He joined IBM in 1995 after completing his Ph.D. at the Oregon Graduate Institute of Science and Technology. He has worked on and played key roles in many different areas, including a dynamic job scheduling and reconfiguration system for IBM SP2®, Atlanta Olympics Infrastructure, WOM, a Web application server framework which preceded WebSphere, Java Virtual Machine and Java platform performance, a pervasive Java environment experiment that eventually impacted the new OSGi-based Eclipse, next generation portal and client technologies and WSRP-related (Web Services for Remote Portlets-related) infrastructure and standardization.

Apratim Purakayastha *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (apu@us.ibm.com).* Dr. Purakayastha is a research staff member and manager in the Pervasive Computing Infrastructure Department. He joined IBM in 1996 after completing his Ph.D. from Duke University in computer science. His major focus area has been mobile data management and synchronization. He was a founding contributor of the SyncML data synchronization standard and contributed to the design and development of the IBM DB2 Sync Server and Enterprise Sync Server. He also played key roles in the development of IBM Notification and Context middleware. He is an IBM master inventor and a member of the ACM and the IEEE.

Robert D. Johnson *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (robertdj@us.ibm.com).* Dr. Johnson is a research staff member and senior manager in the Systems and Software Department at the Watson Research Center. He received his B.S. degree in physical chemistry from the University of Michigan in 1976, an M.S. degree in physical chemistry from Oregon State University in 1980, and a Ph.D. in physical chemistry in protein dynamics from the University of California at Davis in 1983. He subsequently joined IBM, where he has worked on advanced photolithography, disk drive technology, production of nanotube and fullerene materials, the Sash Web Applications Framework, the pre-J2EE WOM, a Web application server framework which preceded WebSphere, and tools for e-business application performance optimization. Dr. Johnson has received IBM Outstanding Technical Achievement Awards for his work in the areas of Web application server optimization, high-performance head/disk interface characterization, and in carbon-60 and metallofullerene nanotechnology. He is a member of the IEEE and the ACM.

Jim Colson *IBM Pervasive Computing Division, 11501 Burnet Road, Austin, Texas 78758 (jccolson@us.ibm.com).* Mr. Colson received his B.S.M.E. (mechanical engineering) from the University of Michigan, Ann Arbor in 1980, and M.S.M.E. and M.S. in computer science degrees from the University of Texas at Austin in 1985 and 1991, respectively. He is an IBM Distinguished Engineer and is currently the chief architect for pervasive computing and device software. Previously, he developed real-time, multiprocessor control systems for robotics and computer vision systems and solutions. This work led to his Masters Thesis on robot performance, which was subsequently adopted as an open standard (ANSI/RIA R15.05-1-1990 - Point-to-Point and Static Performance Characteristics). Mr. Colson is a member of the IBM Academy of Technology and an IBM master inventor.

Roger A. Pollak *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (pollak@us.ibm.com).* Dr. Pollak is a research staff member and manager in the Systems and Software Department at the Watson Research Center. He received his B.S. degree in physical chemistry from Polytechnic University in 1967, and M.S. and Ph.D. degrees in physical chemistry from the University of California at Berkeley in 1969 and 1973, respectively. He subsequently joined IBM at the Watson Research Center, where he has worked on the physics and chemistry of surfaces, electronic computer packaging, and more recently on software technology. In 1994 he received an IBM Outstanding Contribution Award for his work on the design and implementation of the IBM RS/6000® Scalable POWERparallel Systems® supercomputer. Dr. Pollak is a senior member of the Institute of Electrical and Electronics Engineers.