Mobile AJAX Applications: Going Far Without The Bars

Nikunj Mehta Garret Swart Colm Divilly Ashish Motivala

Oracle
500 Oracle Parkway
Redwood Shores, CA 94065
USA
{nikunj.mehta | garret.swart | colm.divilly | ashish.motivala}@oracle.com

Abstract

Mobile application developers can't avoid human computer interface challenges, limited network connectivity, and harried users, but they shouldn't have to deal with new data storage and communication models and synchronization issues. In this paper we explore how the AJAX application model and the Atom data model, widely used for on-line applications, can be combined to create seamless on-line/off-line model for mobile applications. We show how existing applications can be made robust to intermittent connectivity and limited battery capacity without any application changes. This is achieved through a local Web data proxy and synchronization architecture called BITSY. We illustrate this approach with two robust mobile Web applications, one existing and another new, running on AtomDB, an implementation of BITSY built as a browser plug-in.

1.Introduction

Today's users aspire to use multiple devices, to use best-of-breed Web applications, for corporate as well as personal needs, and to be able to use their applications and data from any device at anytime notwithstanding any network or battery condition [1]. In recognition of these aspirations, both Microsoft [2] and Apple [3] have recently announced software and services to replicate user data across a range of devices based on a master copy managed at a central site. To fully satisfy these aspirations, we must produce applications that are responsive and available where the user needs them, even when the user's device is temporarily disconnected from the network.

From the early days of the Web, users downloaded portions of the Web onto their personal computers. On the content-driven Web this was easy; a crawling application would copy an indicated portion of the Web into a disk resident Web cache and allow the user to browse content from this cache when off-line. These personal crawlers were useful because they hid the slow speed and intermittent connectivity provided by the dial-up lines of the time. As server-based Web applications became the focus of Web activity, this approach became less interesting because server applications (written in PHP, Perl or Java) could not be downloaded, and even if they could be downloaded, the application's execution environment was not present on the PC.

With the emergence of Ajax, a new form of replication is becoming feasible. Instead of just content being cached, the applications themselves are cached and can be run even when the device is disconnected from the network. Of course, most interesting Ajax applications need information services to provide them with data to be rendered. Many Ajax applications define XML [4] or JSON [5] based data models and access protocols based on Atom feeds [6] and the Atom publishing protocol [7], e.g., GData [8] and OpenSocial [9]. These standards help regularize the web and save developers from defining and teaching new developers a new protocol for each application. In this paper, we show how to take advantage of feed-style HTTP application interfaces to perform cached data access and off-line data manipulation that is later synchronized with the server.

The goal of our work is to enable adaptive mobile applications that adjust to connectivity and/or battery levels by transparently switching to local storage without adding significant effort for either the use or development of such applications. We do this by giving developers one model for Web application data access that works whether the application is deployed over the network or installed locally, executed on a PC or on mobile devices, smart phones, and any other Internet-enabled device.

In the rest of this paper we'll describe two applications where today's mobile computing solutions are difficult to develop and manage and typically result in less than usable applications. We will then describe BITSY, a data proxy and synchronization architecture that enables adaptive mobile applications. Finally we describe an

implementation of BITSY as a browser plug-in, called AtomDB, and we show it in action supporting two applications, one written specifically for BITSY and another unmodified from its original implementation.

2. Motivation

Applications are moving away from client-based storage because of its propensity to fail and the difficulty in sharing it either between different users or between a single user's many devices. This move is a difficult one for mobile enterprise applications as the mobile computing environment is far from benign: (i) servers are often not configured for high availability, (ii) networks are not ubiquitous, fast and secure, (iii) power starved mobile devices must gulp energy to communicate on high speed networks, and (iv) users rely on getting their answers right now. We want to resolve these seeming contradictions and get the responsiveness of local storage with the retention and sharing semantics of global storage.

Furthermore, while we are in the midst of a revolution in the quality of the interaction in data-intensive Web applications, their mobilized counterparts have not kept pace. This is largely due to connectivity challenges and the high cost of developing and maintaining applications for the fragmented mobile market. User's standards have risen and mobile applications have to meet them.

In this section, we discuss application scenarios that illustrate the mobile computing problem, we identify the main requirements for a mobile application architecture and finally, we consider some of the architectural possibilities that lead us to BITSY.

2.1 Use cases

The first case involves a corporate road warrior on a tablet PC and the second case a user working on a smart phone.

Pharmaceutical sales representative mobile assistant

For a pharmaceutical sales representative, their tablet PC acts as a rather large digital assistant – it carries their communications, presentations, appointments, notes, and enables them to connect with others instantaneously. They need up-to-date product and inventory information, clinical evidence, comparative analysis, and tailored presentations at their fingertips. They also collect information from doctors during meetings aimed at increasing sales. However, they have no network connectivity when visiting doctors due to policies at clinics and hospitals. Like other users they are not interested in solutions that require extra effort on their part to prepare for network disconnection.

Integrated address book

A smart phone carries a built-in address book typically stored on the local flash storage. However, a large part of a user's contact network is corporate in nature – colleagues, customers, and partners. In addition, today's users often use social networking tools to interact with indirect contacts, even though they don't always have up to date information about such contacts. Users should be able to get all their contact information over-the-air directly from their sources, make modifications when necessary and be able to place a phone call, IM, or email to any of their contacts from their device. They should also be able to limit access to their private data and switch devices or services without losing access to their contacts. Needless to say, users need a responsive address book that is accessible at all times.

2.2 Requirements

These use cases highlight the requirements that a mobile application environment should support.

Disconnected operation: In order to bridge temporary connectivity gaps, applications should function close to normal even when disconnected.

Responsive: Applications must be responsive enough to match the human computer interaction capability in the devices.

Extensible: Data may be of any content type.

Secure: We need to protect one application from another while still allowing data sharing.

Interoperable: The server, the device software and the application may each be developed by different organizations and they must work together without change.

Deployable and Evolvable: Once an application is developed, it needs to be easily deployed in the field and gracefully accept updates.

2.3 Architectural Alternatives

Local storage and processing is the only way of dealing with disconnected operation in an online application. Local storage can be considered as a write-back cache or the server storage can be treated as a backup of the client store. Treating the server storage as a backup gives the user a simple but limited model.

The other issue is how the data model and data access method match up between the client and the server. We consider the following possibilities:

1. The local store is the application repository and it is backed up on the server.
2. The local cache and the server store use distinct data models and access methods.
3. The local cache and server use the same data model but distinct access methods.
4. The local cache and the server store have the same data model and access methods.

The first approach is the traditional one used in off-line applications. Some applications take this model to

the limit installing the entire server stack on the client device, including a database, an application server and the application client. The database and any other application resources must be synchronized over the network using application-specific techniques. This approach results in applications that are harder to evolve due to information integration issues between the local database and the server database. Moreover, it leads to security that is either too tight or too loose – applications tend to either not cooperate at all or have full access to other's data.

In the second approach, applications locally store a transformation of online data that is better suited to local processing in a local database, e.g. a local relational database. Applications must then employ a data switch between on-line and off-line operation, explicitly accessing the local database only when off-line. The problem with this approach is that the application devolves to two separate applications that are each accessible with the same UI. Worse, the application-specific data transformation can make it harder to perform application-independent synchronization.

In the third approach, applications use HTTP requests and responses when operating on-line and store the HTTP response in a local database for off-line operation. The application still must use a switch as in the second approach but does not transform data. In addition, applications need extra processing when using the local data for off-line operation that is not required for on-line operation. However, it is easier to perform generalized data synchronization than in the second approach.

The last approach is an improvement on all the preceding in that it does not require additional processing for local data and, hence, does not require a data switch. As a result, developers don't need a distinct off-line version of their application. This minimizes additional tools and techniques that an application developer needs to master. This approach essentially is a write-back cache of online data. It does require that applications use the same interface for communication as for synchronization with the server. An advantage of this approach is that it is possible to take full benefit of existing application-layer protocols, such as Atom, instead of reinventing those mechanisms in a synchronization-specific protocol.

For these reasons we have chosen to base BITSY on Atom as a standards-based data synchronization mechanism that fits inside a standard Web browser and allows developers to build regular Web applications using AJAX and HTML. As long as standardized synchronization protocols are used, no proprietary technologies are required and applications are able to obtain data from any sources that speaks the Atom protocol. Most importantly, applications can transparently switch between online and local data because support the same API, the only distinction is that one is proxied locally while the other is performed directly on the server. This is the approach we take in an architecture we have developed and BITSY, the architecture and its control API, is described next.

3. BITSY

BITSY — A Web Interface To Synchronization — is architecture for supporting mobile applications. By providing lightweight, secure data synchronization between local caches and Web servers and by supporting the existing communications APIs, BITSY results in easily mobilized applications that meet the requirements of the previous section.

AtomDB is an implementation of BITSY as a plug-in for Windows Internet Explorer 7 and Windows Internet Explorer Mobile 6.

3.1 Data access architecture for mobile applications

BITSY is an architecture designed in the native style of the Web [10], building on URLs, ReSTful data access, and hypermedia, that is, mixed content and hyperlinks. Data managed by BITSY is identified using URLs enabling its use across existing applications. Figure 1 below is a pictorial representation of BITSY.

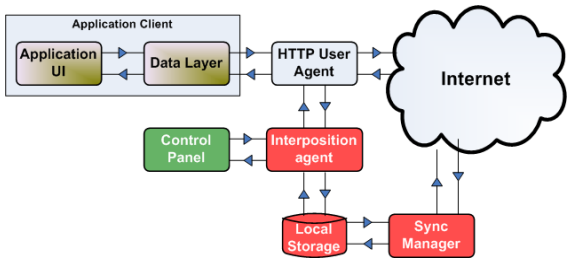

Figure 1: BITSY Architecture

In this architecture, an application is made up of a UI and a data layer. The application interacts with its server through a browser or another suitable HTTP user agent. BITSY consists of three parts, an *interposition agent* that listens in on every interaction of the application with its servers. The interposition agent may be configured using a *control panel* that is itself a BITSY application and can be provided either by the BITSY provider directly, by the platform vendor acting on behalf of the BITSY provider, or a third party. The control panel allows for the configuration of applications and data sources that are cached and synchronized as well as providing users information with the state of the device synchronization, including any errors. Finally the *Sync manager* is a background process that interacts with every registered data source to maintain the local cache and push updated data back to the server.

Applications continue to access feed data as they normally do either through HTML or AJAX. Only if

BITSY is configured to cache a data source will it intercept requests made to it. Otherwise the browser handles requests as usual. The cache may satisfy requests that are intercepted if the required headers specified through the control panel are present on a request. This allows applications to control the visibility of their data. If the agent intercepts a GET request, then it responds to the request from its cache immediately. If intercepting a known unsafe request, such as PUT or DELETE, the agent attempts to make the request directly to the server. If a network connection is available, then the server's response is recorded and forwarded to the application. This response is later used to subsequent requests for the same resource. This behaviour makes applications more responsive by replacing remote requests with local processing. Of course, this comes with the chances of slight staleness of responses to GET requests. If the server is unreachable for any reason, then the agent records the request in a queue and generates a tentative response to the application. Until this request is processed on the server, the plug-in continues to provide the tentative response to future GET requests for the same resource.

When network connectivity is restored, the synchronization manager forwards any queued requests to the server and stores the responses as in the on-line case. If the response indicates an error, then the synchronization manager alerts the control panel application. Applications may also be modified to directly handle such errors, if they choose to.

BITSY's synchronization leaves applications free to deal with required application data structures and logic. For the most part an application developer doesn't need to worry about synchronization as long as the underlying data model is based on feeds. The control panel allows for a selection of conflict resolution algorithms, or the application can chose to handle conflict resolution itself.

In addition to the generic control panel that allows for user control, a BITSY application can choose to configure its own data sources or an existing application may be wrapped inside of an installation application that handles the BITSY configuration. This latter case is a key adoption issue as it allows for both the users and the application to be unaware of BITSY. Only the application specific BITSY installer needs to know what is going on.

3.2 Feeds for mobile data access and synchronization

BITSY builds upon HTTP and fully leverages its caching semantics. However, in order for applications to function while they are off-line it is necessary to approximate server behaviour locally. The BITSY architecture uses Atom feeds [6] and the Atom publishing protocol [7] for communication and synchronization as the semantics for various HTTP operations such as PUT, POST, and DELETE. Bi-directional synchronization depends on Atom publishing protocol. If the server providing Atom feeds does not support this protocol, BITSY can still be used to subscribe a read-only copy of the feed contents.

In addition to avoiding language engineering, this choice helps synchronization because Atom formats are versatile and ready for archiving, and when used with the publishing protocol provide support for the complete data life cycle.

Atom documents are versatile data envelopes capable of communicating an individual record or feeds with thousands of multiple records. They have been used to communicate mapping information, personal information, financial information and so on. Atom feeds contain XML entries (records) each of which represents a single application entity such as a contact, appointment, or an order, represented in any text or binary format including, but not limited to, XML. This makes it possible to deal with any kind of content used in applications. An application simply needs to know the URLs of Web feeds it needs and BITSY does the necessary work to manage all the records in those feeds and their supporting data.

Furthermore, Atom feeds are designed for archiving. Atom feeds provide permanent identifiers to each piece of information making it possible to correlate copies of the same data. Atom feeds contain timestamps that allow applications to check whether their version of data is up-to-date with its source. Correlation and up-to-date checks can be made on lists of resources, i.e., feeds, without requiring a large number of HTTP requests.

Finally, Atompub provides complete support for data life cycle: applications can add to, remove from, and change feed contents using Atompub. Feeds from Atompub servers also provide a network location for every record, thus enabling direct modifications of individual records.

Moreover, both the protocol and format are extensible allowing significant room for new application needs. There are already many tools for creating, publishing and viewing documents. Finally, Atompub has already been chosen by various organizations [8, 9] as the application data interface of choice. The end result is that applications can access information (such as messages, contacts, appointments, and so on) from any service regardless of whether the user's device is connected to the network at that time or not. All is needed is that the feed be configured for local caching.

4. Evaluation

We evaluate BITSY in terms of its contribution to the robustness of mobile Web applications in this section. We also contrast BITSY with existing techniques for Web data synchronization and off-line Web applications. Finally, we discuss a few limitations of Atompub that need improvement to make Atompub a protocol of choice for Web data synchronization.

4.1 AtomDB in action

We worked on two applications with AtomDB. We chose to experiment with a new application first and then with an existing Ajax application that used an Atompub data source.

Worklist

The first application was written for a mobile device from scratch to work with a plain vanilla Atompub server. This application, residing on a single URL, presents a list of tasks to a user and provides a means of updating the status of the task as well as create and remove tasks. Every task is an entry in the feed of tasks and can be edited at its Atompub *edit* URL. A new task can even be created at the Atompub task feed URL. The application subscribes to the task feed using AtomDB to enable offline operation. If a task is deleted AtomDB removes the task from the feed and hence the task does not show up in the application. If a new task is created, it always shows up at the top of the *Worklist* as long as the new task is not synchronized with the server. Details of every task can be viewed and edited, except the task which is locally created cannot be edited.

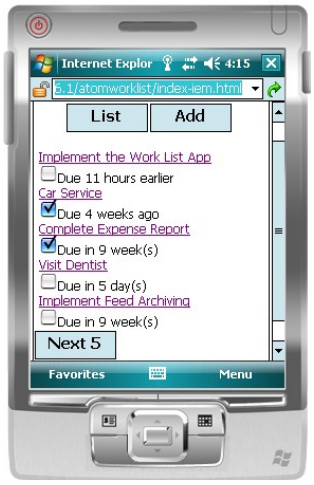

Figure 2: Worklist

Birthday Manager with Google Calendar

Birthday Manager [11], is an existing Google Calendar API sample, that uses Ajax calls to access and update its data. This application's data is provided as feeds of calendar items in Google's calendar entry format [12]. Google requires user authentication before the feed is provided to another application. In order to relay to AtomDB the authorization token provided by Google to Birthday Manager, we used a JavaScript injecting browser add-on. Using this add-on we inject JavaScript code in to the Birthday Manager Web page for subscribing it to the Google calendar feed as well as the resources that make up the application such as JavaScript, images and html. This acts as a control panel that subscribes the application to the Google calendar feed as well as the resources that make up the application such as JavaScript, images and html. Further, the JavaScript API used by this application makes a request to a Google Atompub proxy, which in turn forwards the requests to the Google Calendar Data server, translating both the request and response in the process. When the device is disconnected, this proxy behaviour is unavailable. Therefore, we emulated the Atompub proxy behaviour in AtomDB to allow the application to continue its operation when off-line. The same injected JavaScript unsubscribes from the calendar feed when the user logs out of the Birthday Manager.

4.2 Comparison with existing off-line approaches

SyncML [13] a proprietary synchronization protocol is used for synchronizing personal information such as contacts and appointments between PDAs and desktop software. Synchronization is performed either through tethered connections or over-the-air. It is capable of syncing any device to any other device. The HTTP binding for SyncML used for performing over-the-air synchronization uses the POST method for all its operations and sidesteps various HTTP constructs such as conditional processing and cache semantics. As a result, SyncML is not suitable for online applications. Furthermore, it requires a server to keep track of the synchronization state of every synchronized client creating challenges when dealing with large numbers of clients and during recovery from data corruption.

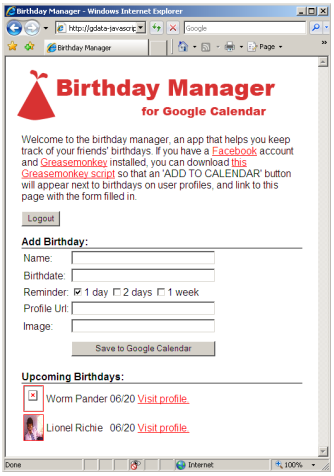

Figure 3: Birthday Manager

FeedSync [14] is a protocol to perform synchronization of data on XML feed formats such as RSS or Atom. FeedSync is suited to scenarios where there is no single "master" copy. Atompub and BITSY are designed for HTTP-based client-server systems where the server owns the master copy. FeedSync has additional restrictions on the Atom feeds and can only work with feeds specially prepared for synchronization with FeedSync. BITSY can work using plain old Atom feeds with no extensions. Furthermore, FeedSync places additional burden on feed sources to keep additional FeedSync metadata, while BITSY does not place any obligations on the server. FeedSync scales well as the number of independent copies of a given data item grows, whereas BITSY and Atompub scale best when large numbers of users access the same synchronization sources.

Gears [15] provides tools to extend Web applications for off-line operation. Gears provides a SQL data store that applications can use for local data access and a local HTTP server that can service GET requests. The current draft of HTML 5 [16] also provides an SQL data store API in Web browsers. However, neither provides built-in

synchronization primitives forcing applications to come up with their own. Moreover, applications are also required to use a SQL-based programming model to take advantage of the local storage capabilities requiring the application to be rewritten to acquire off-line capabilities.

4.3 Atompub for Web data synchronization

Despite being designed for publishing and editing Atom data, Atompub is claimed by a few to be a general-purpose data management protocol for the Web. Our use of Atom and Atompub for synchronization of Web data has shown many benefits and also some limitations.

Atompub was not designed for its sole use to be from within a Web browser. As a result parts of Atompub cannot be effectively used from online applications, and that makes such capabilities less useful in off-line applications. An important such capability is publishing and editing media resources and their metadata.

Another restriction is that Atompub does not provide a standard means for a server to offer a feed that supports "repeatable read": Every request may produce a different response. In the context of feed paging, which is used to limit the size of each Atom response, this makes it difficult to construct a consistent view of an entire Atompub collection.

Another challenge with Atompub-style resource creation is that the namespace is controlled entirely by the server and a client can only offer hints for minting names. As a result, when data is created off-line, it does not have a URL, which makes it hard to edit or link to this data until the server processes the request. This also potentially breaks applications that expect URLs on all data that they obtain.

Similarly, Atompub does not provide an efficient and standard means of discovering entries that have been deleted from a collection. As a result, a client needs to inspect every possible entry of a collection to determine the fate of existing entries.

5. Summary and Future Work

AtomDB is an Atom aware proxy server built as a browser plug-in. Configured by either the control panel or by application installers that make use of the BITSY architecture, it maintains a transparent local cache of the data needed by the application and provides a local implementation of the Atompub protocol to allow the application to function and update its data even when it is disconnected. This architecture builds on the Web 2.0 infrastructure (Atom feeds and Ajax) and provides a uniform data access model for all Web applications, whether the application is connected or disconnected, running on a PC or in an embedded device. This approach improves the robustness of Web applications used in mobile devices without creating a dependence on proprietary technology or requiring a new model for applications or data.

More work remains to be done to improve efficiency and timeliness of synchronization, processing cached data for response to queries, and supporting feed paging. Our current implementation offers the latest snapshot of data when responding to requests, even though all the necessary data may not be cached locally. As a result, applications may get an inconsistent view of data. We plan to improve this capability. Finally and ironically, although we are pushing the envelope of the mobile Web, the current crop of mobile Web browsers are unfriendly to work with and impose a variety of restrictions on both plug-in and content developers. Desktop browsers are much more friendly and we hope more mobile Web browsers open themselves up to extensions such as AtomDB.

References

[1] Satyanarayanan, *M. Fundamental Challenges in Mobile Computing*, Principles of Distributed Computing, May 1996

[2] *A first look at Live Mesh*, Microsoft, April 2008

[3] *MobileMe*, Apple, www.apple.com/mobileme

[4] Yergeau, F. et al. *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C, Feb 2004

[5] Crockford, D. (eds). *The application/json media type for JavaScript Object Notation (JSON)*, IETF RFC 4627, July 2006

[6] Nottingham, M. and Sayre, R. (eds). *The Atom syndication format*, IETF RFC 4287, Dec 2005

[7] Gregorio, J. and de hOra, B. (eds). *The Atom publishing protocol*, IETF RFC 5023, Oct 2007

[8] Google Data APIs, http://code.google.com/apis/gdata/overview.html

[9] *RESTful API Specification version 0.8*, OpenSocial, http://code.google.com/apis/opensocial/docs/0.8/restfulspec.html

[10] Jacobs, I. And Walsh, N. *Architecture of the World Wide Web, Volume One*, W3C, Dec 2004

[11] *Birthday Manager for Google Calendar*, http://gdata-javascript-client.googlecode.com/svn/trunk/samples/calendar/birthday_manager/birthday_manager.html

[12] Google Calendar Data APIs and Tools, http://code.google.com/apis/calendar/reference.html

[13] *SyncML Sync Protocol, version 1.1*, Open Mobile Alliance, Feb 2002

[14] Ozzie, J. et al. *FeedSync for Atom and RSS*, Microsoft, Dec 2007

[15] Gears, http://code.google.com/api/gears

[16] Hickson, I. And Hyatt, D. *HTML5: A vocabulary and associated APIs for HTML and XHTML*. W3C Editor's Draft, June 2008.