

# AtomDB: A Transparent Off-line Mobile Database

Nikunj Mehta

Ashish Motivala

Colm Divilly

Garret Swart

Oracle  
500 Oracle Parkway  
Redwood Shores, CA 94065  
USA

{nikunj.mehta | ashish.motivala | colm.divilly | garret.swart}@oracle.com

## Abstract

AtomDB provides transparent read-write access to Web data even when the device is disconnected from the data server, thus enabling seamless on-line/off-line applications. AtomDB is based on a local Web data proxy and synchronization architecture for single master data sources. Using AtomDB and without making any significant changes, we show how an existing and a new Web application, each using Atom feeds for their data needs, can be made robust to unreliable connectivity.

## 1. Introduction

With an ever-increasing array of personal computing devices, users aspire to timely access to their data on each such device, personal as well as corporate, at all times. The multi-master model that served quite well in the past when data was intended to be consumed and produced mainly on mobile devices and not shared across multiple users is unsuitable in a highly collaborative environment. Most current applications require collaboration among users and access to data on both fixed and mobile devices. In this environment, a single master model is preferable as the source of truth for the data needs of such applications. Many of these computing devices are also highly resource-constrained and provide weak and unreliable connectivity to data networks [1]. As a result, there is an urgent need for adaptive and seamless applications that provide timely access to a user's data on multiple devices in the face of network unreliability.

AtomDB is a new technology aimed at fulfilling the previously described ubiquity aspiration by (i) providing transparent read-write access to locally stored Web data even when no network connectivity is available and (ii) synchronizing this local data using the same interface as that used for communication by an application client.

AtomDB is motivated by a set of adaptive application requirements. It should be: capable of disconnected

operation, responsive, extensible, secure, interoperable, and easy-to-deploy and evolve. Local storage and processing is the only way of dealing with disconnected operation in an online application. Treating the server storage as a backup gives the user a simple but limited model. Alternatively, the approach taken by AtomDB is to consider local storage as a write-back cache.

The goal of our work is to enable adaptive mobile applications that adjust to connectivity 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 provide a brief motivation for AtomDB. We will then describe AtomDB, which is an implementation of a data proxy and synchronization architecture for adaptive mobile applications. We demonstrate AtomDB through two adaptive applications, one synthetic and another unmodified from its original implementation.

## 2. Motivation

Research suggests that adaptive mobile applications should make local storage “*translucent to users through controlled exposure of cache management internals* [2].” Despite that, the state-of-the-practice employs separate applications for on-line and off-line usage. Witness a number of fat-client applications developed for mobile devices using mini relational databases that are synchronized to databases running in data centers.

### 2.1 State-of-the-art

The current state-of-the-art for network-based applications is much more in line with existing research and produces off-line behavior that is translucent to end-users by packaging off-line behavior inside regular online applications. An important design choice in such

applications 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 cache and the server store use distinct data models and access methods.
2. The local cache and server use the same data model but distinct access methods.
3. The local cache and the server store have the same data model and access methods.

State-of-the-art applications are designed using the first approach. Applications locally store a transformation of online data that is better suited to processing in a local 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 second approach, applications still must use a switch as in the second approach but do not transform data. They store results of online communications in a local database for off-line operation. Applications need extra processing when using the local data for off-line operation. However, it is easier to perform generalized data synchronization than in the second approach.

The last approach does not require additional processing for local data and, hence, does not require a data switch. As a benefit, 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.

## 2.2 Caching and synchronization for Web data

Caching is common in adaptive applications [3] although state-of-the-art involves custom synchronization for each data source. The two problems created by custom synchronization are (i) every application need to perform its own synchronization, and (ii) synchronization of the same data is repeated across applications. Moreover, state-of-the-art requires specialized interfaces for performing synchronization that is different from the interfaces used for client-server communication.

The emergence of 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] make it possible to perform automatic

data synchronization as well as provide transparent on-line/off-line data access.

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. 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 or not the user's device is connected to the network at that time.

## 3. AtomDB

AtomDB – *A Transparent Off-line Mobile Database* – is an implementation of a data access architecture for adaptive mobile applications that performs secure data synchronization between local caches and Web servers using existing application interfaces. Currently, AtomDB is implemented as a browser plug-in for Windows Internet Explorer 7 and Windows Internet Explorer Mobile 6.

### 3.1 Architecture

AtomDB builds upon HTTP and fully leverages its caching semantics. Moreover, in order for applications to function while they are off-line it is necessary to approximate server behaviour locally. For that, AtomDB uses Atom feeds and the Atom publishing protocol as the semantics for off-line fulfilment of standard HTTP operations such as GET, PUT, POST, and DELETE. Bi-directional synchronization depends on Atom publishing protocol. If the server providing Atom feeds does not support this protocol, AtomDB can subscribe to a read-only copy of the feed contents.

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

In this architecture, an application interacts with its server through a browser or another suitable HTTP user agent through either HTML or Ajax style requests.

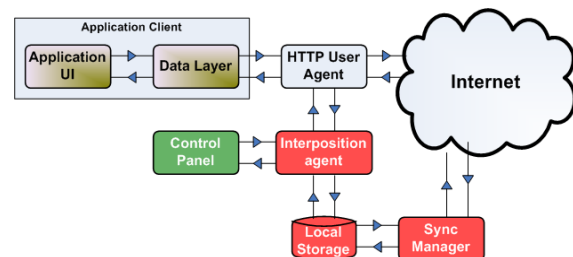


Figure 1: Architecture of AtomDB

AtomDB consists of three parts, an *Interposition Agent* that listens in on every interaction of the application with its servers. However, AtomDB only responds to requests for resources that it knows about. A resource is known to AtomDB if either it is a feed registered for synchronization or if it is linked in the feed, e.g., an image, and the link type is registered for subscription in some registered feed.

AtomDB can be configured directly by an application. This requires minor changes to the application so that it can identify feeds and link types as well as authorization headers to AtomDB. Alternatively, AtomDB also provides an experimentation feature where an application's feeds and other resources can be registered without making any changes to the application. To use this feature, a developer can provide configuration as a local script file directly to an AtomDB control panel and identify the application to which the configuration applies. In either case, AtomDB will manage the configured resources. The control panel also provides users information about the state of synchronization, including any errors.

The *Sync manager* is a background process that interacts with every registered data source to maintain the local cache in persistent storage and replays off-line update requests to the server when a connection is available.

AtomDB satisfies intercepted requests only 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 respond to subsequent requests for the same resource. This behaviour enhances application responsiveness by replacing server interaction 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 based on Atompub semantics. AtomDB continues to provide the tentative off-line response to future GET requests for that resource until the pending request is relayed to the server.

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 as well as any application listening for such errors.

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 choose to handle conflict resolution itself by listening to conflict errors.

### 3.2 Applications

We demonstrate AtomDB's utility using two applications – a synthetic new application and an existing Ajax application both using an Atompub data source.

#### Worklist

This is a new application written for a mobile device works against an Atompub server. As shown in Figure 2, this application presents a list of tasks to a user and provides a means of updating the details of a task as well as add and delete tasks. Every task is an entry in the feed of tasks and can be separately edited. Upon first use, the application registers its required feed. While AtomDB processes the registration and locally hoards that feed's data, the application transparently fetches data online. Within a few seconds, the data is available locally and application requests are answered locally. At this point the application can seamlessly work even if it gets disconnected.

All the functions including editing or deleting an existing task and creating a new task can be performed off-line as well as online. If changes are made off-line the latest changes always appear in the application. If a new task is created off-line, it always shows up at the top of the *Worklist* until 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. Once connection is re-established, AtomDB relays all update requests to the server.

#### Birthday Manager for Google Calendar

Birthday Manager [11], shown in Figure 3, is an existing Google Calendar API sample, that uses Ajax calls to access and update calendar feeds in Google's calendar entry format [12]. Google expects an authorization token to be present on every request for data from the Birthday Manager application. This required us to programmatically configure AtomDB with the token provided by Google to Birthday Manager.

We use this example to demonstrate how little an application changes in order to work off-line using

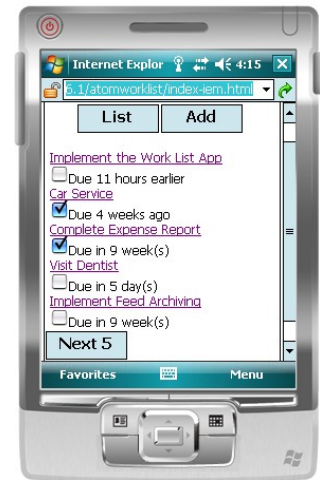


Figure 2: Worklist

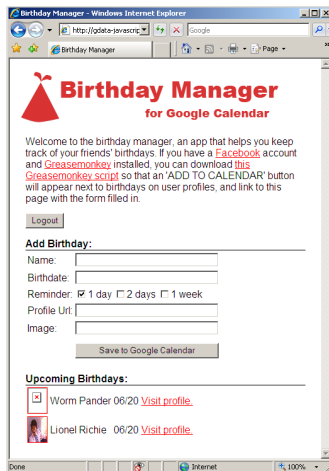


Figure 3: Birthday Manager

AtomDB even when security constraints are involved. To that end, we constructed a 941-byte JavaScript file that registered the feeds required by this application including the resources used by the application as well as provide the authorization token to AtomDB. This file is provided to the AtomDB control panel, which enables the application to even be started off-line. This subscribes the application to the Google calendar feed as well as the a feed of the resources that make up the application such as JavaScript, images and html. Once again, this application starts off using online requests and seamlessly switches to local data as it becomes available. Furthermore, off-line requests are handled by AtomDB without any negative impact on the application. AtomDB is also able to interoperate with the security requirements of the Google Atompub server and present locally cached data only to those applications authorized by Google.

#### 4. Summary and Future Work

AtomDB is an Atom-aware local caching proxy and synchronization manager implemented as a browser plugin. Configured by either a control panel or by application code, it maintains a transparent, secure, extensible, local write-through cache of the data needed by Web applications. It provides an off-line implementation of the Atompub protocol to enable applications to function and update their data even when disconnected. This architecture builds on the Web technologies including 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.

AtomDB is primarily relevant to single master data sources. 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. Also, since it performs stateless synchronization, AtomDB is fit for use with very large numbers of users and devices. This approach can also benefit from the existing Web infrastructure such as caches in speeding network interactions and scaling server processes. Moreover, it works with arbitrary Atom feed sources and, hence, is viable as an enabler of off-line applications today.

Since AtomDB provides built-in synchronization and conflict resolution primitives, there is no need to rewrite applications or create branches on code paths for off-line data access. Furthermore, simple conflict resolution is built-in and can be easily configured on a per-application basis. Our experience in mobile business applications has also indicated that complex conflict resolution algorithms are not required and simple techniques are sufficient to deal with minor conflicts that arise in practice and AtomDB provides a good test bed to validate this.

Our work shows that it is possible to accomplish adaptiveness in mobile applications without being dependent on the relational data model or being tied to complex disconnected transaction models. AtomDB is an initial step towards adaptive mobile Web applications. More work remains to be done to improve efficiency and timeliness of synchronization, and emulating server query processing on cached data.

#### References

- [1] Satyanarayanan, M. *Fundamental Challenges in Mobile Computing*, Principles of Distributed Computing, May 1996
- [2] Ebling, M., John, B., and Satyanarayanan, M. *The importance of translucence in mobile computing systems*. ACM Trans. On Computer Human Interactions. 9(1), March 2002.
- [3] Satyanarayanan, M. *The evolution of Coda*. ACM Trans. on Computer Systems. 20(2), May 2002.
- [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](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>