Reliable Data Distribution and Consistent Data Replication
Using the Atom Syndication Technology

Firat Kart, L. E. Moser, P. M. Melliar-Smith
Department of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA, USA

Abstract. Atom is a lightweight syndication technology, based on XML, that allows data to be published on, and retrieved from, the Web. Atom does not currently provide reliable data distribution or consistent data replication. In this paper we describe a novel Reliable Data Distribution and Consistent Data Replication infrastructure for Atom. Reliable Data Distribution ensures that the intended consumers have obtained the data that the publisher published and that the publisher can garbage collect the data. Consistent Data Replication provides high availability and fast local access to the data at the consumers. We also describe a potential application of, and present performance results for, the Reliable Data Distribution and Consistent Data Replication infrastructure that we developed.

Keywords: Atom, consistent data replication, feed, reliable data distribution, RSS

1 Introduction

Lightweight architectures and protocols are becoming increasingly important in the Internet. Most enterprises do not have enough resources, time or knowledge to implement applications using heavyweight mechanisms. Atom and RSS [5, 8, 9, 13, 14, 20] are lightweight syndication technologies, based on XML, that allow data to be published on, and retrieved from, the Web. They can be used to replace or augment technologies such as Web Services [21, 22, 23, 24], which have generated concern [12] particularly for event-based applications.

Atom and RSS are typically used in best-effort mode with no guarantee of reliable delivery, i.e., they do not ensure that the intended consumers have received the data that the publisher published. For some applications, such a best-effort strategy, with its lower costs, is appropriate. However, there are other applications, such as supply chain visibility applications, that require access to the data with high reliability, high availability and good performance. Consequently, we have extended Atom with a novel infrastructure that provides:

- **Reliable Data Distribution** – Complements the information feed at the publisher with an acknowledgement feed at the consumer to achieve reliable delivery of information feeds
- **Consistent Data Replication** – Mirrors the publisher’s data at the consumer for high availability and fast local access.

Atom and RSS typically require several seconds to process and propagate information. Our Reliable Data Distribution and Consistent Data Replication infrastructure adds a bit more delay. Thus, it is not intended for interactive human access to remote data. Rather, it provides more convenient and rapid access to a mirrored local copy of the data. For many applications, a delay of several seconds before remote updates become available locally is acceptable, because local data access enables the applications to be constructed more easily.

Our Reliable Data Distribution and Consistent Data Replication infrastructure does not aim to provide the distributed atomicity that group communication systems [7] provide with their consequent high costs. It matches more closely publish/subscribe systems [17, 18], but with less difficulty in constructing application programs. Publish/subscribe systems, and also Atom and RSS feeds, can scale to large sizes.

2 Background

Publish/subscribe systems have a substantial history with successful commercial products from Tibco [17] and Vitria [18]. Considerable effort has been devoted to enabling such systems to scale to large sizes, including Gryphon [2] which exploits multiple intermediate brokers to facilitate scaling, and Elvin [15] which quenches the publication of notifications for which no one is listening.
Publish/subscribe systems are often based on push technology, where a publisher sends information updates to the consumers. Push technology was originally thought to be an effortless method for delivering information; however, it is not lightweight [4]. It requires the consumer software to be continuously available to the publisher through a firewall, and a socket connection to remain open to receive pushed information from the publisher. Moreover, push technology consumes considerable network bandwidth.

With pull technology, the consumer periodically checks for information at the publisher, rather than receiving it at unpredictable intervals. Media content (text and graphics) is not communicated directly to the consumer; rather, the consumer retrieves the media content off the Web. The consumer software does not need to be visible through a firewall, or to keep a socket connection open to receive pushed information from the publisher.

Syndication technology, in which information is pulled rather than pushed, has become popular for distributing information over the Internet. Two syndication technologies in use today are Really Simple Syndication (RSS) [9, 13, 20] and Atom [5, 8, 14]. Both are based on XML, which enables the sharing and communication of information between heterogeneous platforms by making it self-describing. Both allow a publisher to make information available to consumers, which retrieve that information at their convenience. The information is delivered as an XML file, called a feed.

RSS was originated by UserLand in 1997 and subsequently used by Netscape, and then evolved into publication by news Web sites (Reuters, CNN, BBC, etc). Today, there are two versions of RSS. RSS 1.* derived from the Netscape RSS version, and is maintained by the RSS-DEV Working Group [13]. RSS 2.* derived from UserLand, and is maintained by the Center for Internet & Society at the Harvard Law School [20].

Perceived deficiencies in both versions of RSS led to the development of Atom [5, 8, 14], which is more complete and less ambiguous than RSS. Supporters of Atom claim that Atom relies more heavily on standard XML, supports auto-discovery, and specifies a payload container that can handle different kinds of content unambiguously. Supporters of RSS claim that, by introducing another specification, Atom confuses the marketplace. A comparison of RSS and Atom can be found in [3].

Recently, extensions to RSS have been proposed to allow two-way data exchange and, more broadly, to bring database-like capabilities to the Web [9]. In the database-of-the-future vision of Bosworth (CTO of Google) [1], the core database primitives are mapped onto the core HTTP primitives, and there is a simple way to insert, replace, and delete entries within a feed based on HTTP.

Little university research on, or applications of, Atom and RSS exist. One notable exception is the work of Williamson and Stanger [19], who have proposed a lightweight data integration architecture based on Atom. Another is Corona [11], which can handle large numbers of publishers using multiple brokers that cooperate to scan the publishers for new feed entries. Because Atom is designed for interoperability, scalability and extensibility, our Reliable Data Distribution and Consistent Data Replication infrastructure is based on Atom.

3 Reliable Data Distribution

A reliable data distribution infrastructure must provide two-way communication, from the publisher to the consumers and from the consumers to the publisher, so that the publisher knows that the consumers have received the data intended for them — only then can the publisher garbage collect that data.

The Atom Publishing Protocol [5] itself does not guarantee reliable data distribution, i.e., it does not guarantee that the consumers reliably receive a feed. Moreover, the Atom Publishing Protocol does not guarantee that the publisher knows when the consumers have received the feed and, thus, that it can garage collect the feed.

Our Reliable Data Distribution infrastructure requires the consumers of the information to publish an acknowledgement feed for the publisher to check periodically. The consumer includes, within its acknowledgement feed entries, the standard Atom XML unique id and timestamp of the information feed entries it received. The consumer can acknowledge a later feed entry that it has read successfully, even though it has not acknowledged an earlier feed entry. The publisher pulls the acknowledgement feeds that the consumer has published. If it detects that the consumer did not successfully receive an earlier feed entry, the publisher regenerates and publishes that feed entry. Providing reliable delivery of information feeds requires the publisher to retain the information feed entries in an information feed database until it can discard those entries.

Our Reliable Data Distribution protocol differs from other protocols in that a timestamp suffices to provide reliable delivery of feeds. Consecutive sequence numbers are not required. The timestamps must be distinct, but do not need to be
monotonically increasing, although they probably are, because they are obtained from the publisher’s clock. Each acknowledgement feed includes the timestamp of the information feed that it is acknowledging. Our protocol does not require the consumer to read the feed entries in the order in which they were published or in timestamp order. Some applications do require source order, but others do not.

Figure 1 shows the modules of our Reliable Data Distribution infrastructure on the consumer side and on the publisher side. Both the consumer and the publisher middleware have two modules, namely, a Publish module and a Read module. These modules work closely together and have different responsibilities, depending on whether they are used at the publisher or the consumer.

3.1 Publish Module
The Publish module is responsible for making information or acknowledgements available as an Atom feed, which is published using a simple Web server. The Publish module uses a feed database from which it retrieves the outstanding entries to publish in the feed and converts the information to XML-formatted data for inclusion in the feed.

At the publisher, the Publish module makes information available for the consumer to read as a feed. It periodically checks the feed database, and updates the feed, as appropriate.

At the consumer, the Publish module makes acknowledgements available for the publisher to read as a feed. It creates the acknowledgement feed from the feed database, which contains a table of acknowledgements that the Read module manages.

The Atom Publishing Protocol requires a unique identifier for each feed entry. Our Reliable Data Distribution infrastructure uses this identifier to identify the feed entry that is being acknowledged.

3.2 Read Module
The Read module is responsible for checking the status of the feed file and for gathering the content when new information is available.

At the publisher, the Read module updates the information feed records in the feed database, and tags those that the consumer(s) received. When the publisher receives all expected acknowledgements, the publisher tags its feed database entry as received, and removes the feed entries that the consumer(s) acknowledged.

At the consumer, the Read module inserts acknowledgements into the acknowledgement table of the feed database. The Publish module acknowledges successful reception of an entry by publishing an acknowledgement.

The publisher does not publish acknowledgements of the acknowledgements that it has received. Such acknowledgements are implicit in the publisher’s removal of the acknowledged entry from its information feed. When the consumer does not see a feed entry in the publisher’s feed that it has already acknowledged, it knows that the publisher received its acknowledgement.

Our implementation of the Reliable Data Distribution infrastructure uses the Java programming language and the Rome tool [16]. Rome includes generators and parsers for different kinds of feeds, e.g., Atom and RSS. Additionally, it allows conversion from one syndication format to another, increasing interoperability between applications that use different formats. In other words, it is possible for the middleware to publish Atom feeds and to read RSS feeds and vice versa, and to convert between them.

3.3 Application
Our Reliable Data Distribution infrastructure is particularly suitable for applications that are delay tolerant, such as Inventory Visibility applications [6, 10]. In Inventory Visibility applications, the status of the inventory at a manufacturer is made available to the suppliers over the Web. The supply chain can have multiple levels, where a manufacturer at level \( n \) is a supplier to a manufacturer at level \( n - 1 \).

In using our Reliable Data Distribution infrastructure for Inventory Visibility applications, the manufacturer publishes its inventory information as an Atom feed. The suppliers read the information in the feed to learn the inventory status of the manufacturer. Thus, the manufacturer plays the role of the publisher, and the suppliers act as the consumers. It is also possible to have the suppliers...
generate information feeds for the manufacturer to read. In that case, the manufacturer is the consumer, and the suppliers are the publishers.

4 Consistent Data Replication

Our Consistent Data Replication infrastructure, which is based on our Reliable Data Distribution infrastructure, aims to provide data replication for the purposes of high availability and fast local access. The Consistent Data Replication infrastructure replicates information from the back-end source database at the publisher to the back-end replica database at the consumer.

Figure 2 shows the operations on the back-end source database and the back-end replica database, the information flow between the publisher and the consumer, and the acknowledgement flow between the consumer and the publisher. The Publisher middleware creates a feed database. Triggers on the publisher’s source database insert new records into the feed database if there is an Insert, Update or Delete operation on the source database. The Consumer middleware creates a feed database into which it inserts feed entries that it has received from the publisher. Triggers on the consumer’s feed database are used to modify the consumer’s back-end replica database according to the type of operation, Insert, Update or Delete.

4.1 Publisher

At the publisher, the information feed database contains the same tables as the source database and, in addition, several tables that are specific to the Consistent Data Replication infrastructure, as shown in Figure 3 (a) for the Inventory Visibility application. In particular, the ADR_Consumer table keeps track of the expected consumers of the published feeds. The ADR_ConsumerFeed table maintains the relationship between a consumer and a reference to a feed entry.

The fields of the tables in the information feed database are the same as those in the source database, but include a few extra fields that are specific to the Consistent Data Replication infrastructure. During initiation, the infrastructure adds the extra fields to the tables in the information feed database. Figure 3 (b) shows the fields for the Stock table of the Inventory Visibility application.

The ADR_Feed_ID field contains the unique feed identifier of the entry. The Consistent Data Replication infrastructure drops the primary key fields of the source table and makes the ADR_Feed_ID field of the feed table the primary key. The ADR_Feed_Type indicates the particular database operation (Insert, Update, Delete) to be performed on the replica database at the consumer. If there is an update of the primary key of the Stock table in the publisher’s source database, the reference ADR_Feed_ItemID_OLD to the old primary key(s) allows the mapping to be performed on the replica database. The ADR_Feed_Database field keeps the name of the database to which the feed entry refers. Similarly, the ADR_Feed_Table field keeps the name of the table to which the feed entry refers.
At the publisher, triggers on the source database cause the records to be inserted into the information feed database when there is an operation on the source database. A new record for the operation is then inserted into the appropriate information feed table. After insertion of the record, the creation time of the feed entry is recorded in \textit{ADR\_Created}. When the feed is published, the publish time of the feed is recorded in \textit{ADR\_Published}. When the publisher receives acknowledgements from all of the expected consumers, the publisher tags the feed entry as complete by setting \textit{ADR\_Complete} and then it garbage collects the feed entry.

4.2 Consumer

At the consumer, the feed database contains the same tables as the back-end replica database (and the back-end source database at the publisher) and, in addition, an acknowledgement table (\textit{ADR\_Ack}), which is specific to the Consistent Data Replication infrastructure, as shown in Figure 4(a) for the Inventory Visibility application. The \textit{ADR\_Ack} table contains acknowledgement records that are inserted into the table after execution of the database operation on the back-end replica database, when a trigger on the feed database occurs.

The fields of the records in the \textit{Stock} and other tables in the consumer’s feed database are shown in Figure 4(b), with the same interpretation as for the publisher’s feed database.

At the consumer, triggers on the feed database occur when an operation is performed on it, and cause the corresponding operation to be performed on the back-end replica database. The type of operation that is specified in the \textit{ADR\_Feed\_Type} field determines the operation that is executed on the replica database.

The order in which operations are performed on the consumer’s back-end replica database depends on whether the consumer reads the feeds in source order. If so, the order of operations on the replica database is the same as the order of operations on the publisher’s source database. No guarantees are provided for ordering between different feeds.

4.3 Handling New Consumers

To provide reliable data distribution, the publisher must know all of the consumers that read its information feed, so that it can collect acknowledgements from all of them before it garbage collects the feed.

In our current implementation, a new consumer is added to the system at the publisher by a system administrator. When a new consumer is added, the Publisher middleware is made aware of the new consumer. Before the consumer starts reading the published feed, the back-end replica database at the consumer must be initialized. Thus, the publisher scans its back-end source database and creates a feed for the new consumer to initialize its back-end replica database. In this initial feed, all of the entries are of the type Insert. Any subsequent operation on the publisher’s source database has a timestamp greater than the timestamps of the initial feed entries.

When the consumer has acknowledged all of the entries in the initial feed, it starts reading the regular feed. The consumer discards any entry in the regular feed with a timestamp less than or equal to the timestamps of the initial feed entries. The publisher garbage collects the initial feed once the consumers have acknowledged the entries in that feed.

Our current implementation also allows a system administrator at the publisher to remove an existing consumer from the system.

5 Evaluation

We evaluated the performance of our Reliable Data Distribution and Consistent Data Replication infrastructure. We performed the experiments on 3.2 GHz Pentium 4 PCs with 2 GBytes of memory be-
time is 8.2 seconds. Most of the delay results from times for 1000 feed entries. The mean completion consumer every 5 seconds, and checks the acknowledgement feed at the con-
sumer, but also increases communication and propagation of data from the publisher to the con-
sumer, i.e. the time, in seconds, for the publisher to check a consumer’s acknowledgement feed. We let $T_{Inf oFeedCheck}$ denote the time, in seconds, for a consumer to check the publisher’s information feed. These times are highly dependent on the nature of the application. A low value of $T_{Inf oFeedCheck}$ results in more rapid propagation of data from the publisher to the consumer, but also increases communication and processing costs.

In the first experiment, there is a single publisher and a single consumer. The consumer checks the information feed at the publisher every 5 seconds, i.e., $T_{Inf oFeedCheck} = 5$. The publisher checks its feed database every 5 seconds, i.e., $T_{dbCheck} = 5$ and checks the acknowledgement feed at the consumer every 5 seconds, i.e., $T_{AckFeedCheck} = 5$.

Figure 5 shows the frequencies of completion times for 1000 feed entries. The mean completion time is 8.2 seconds. Most of the delay results from $T_{dbCheck}$, $T_{AckFeedCheck}$ and $T_{Inf oFeedCheck}$. Most of the overhead is incurred in the Rome tool in generating and parsing the XML.

Figure 6 shows that having more consumers does not adversely affect the completion time of the feed entries. Rather than being determined by the number of consumers, the completion time is dominated by the consumer that checks the information feed least often and is the latest to acknowledge an information feed entry. The number of consumers that can be supported is limited by the ability of the publisher to handle consumers’ requests for new feed entries. Servers that can handle 1000 requests per second are readily available and, thus, can support several thousand consumers concurrently.

### 5.2 Consistent Data Replication

For the Consistent Data Replication infrastructure, we measured the delay (response time) to retrieve information from a local database, a remote database via a direct connection, and a remote database using a Web Service. For this experiment, we used a simple query that retrieves information from a database table. In real-life situations, this query might be more complex and the result set might include more records.

Figure 7 shows the delay (response time) for obtaining information from a local database, a remote database using a direct connection, and a remote database using a Web Service. The graph shows that the performance of the Web Service degrades as the number of information records to be returned increases. The remote database access over the direct connection performs better than the Web Service, because there is no XML processing for constructing and parsing SOAP messages. As ex-
pected, making the information locally available results in the best performance.

In addition to better performance (less delay) due to local access, our Consistent Data Replication infrastructure provides higher availability of the information to the consumer because it is locally available. If the consumer must access the information from the publisher at the remote server and the remote server goes down, the consumer cannot retrieve the information from the publisher until the remote server is brought back up again.

6 Conclusion

We have presented a novel Reliable Data Distribution and Consistent Data Replication infrastructure that is based on Atom. Our Reliable Data Distribution infrastructure is particularly suitable for applications where there is tolerance for delays in the delivery of information. Our Consistent Data Replication infrastructure replicates information at the consumers, where the information is used, which improves the performance of applications and the availability of information.

In our Reliable Data Distribution infrastructure, the publisher must know which consumer(s) are reading its feed and from which it must obtain acknowledgements before it can garbage collect the feed entries. Currently, we manually configure this set of consumers; we are investigating means of automating this process. We are also considering the use of categories of feed entries, querying a feed for entries of interest, filtering a feed for relevant entries, and aggregating feeds. In addition, we are investigating privacy and security issues for our Reliable Data Distribution and Consistent Data Replication infrastructure.

References