The Tablecast data publishing protocol

Ka-Ping Yee
Google
kpy@google.com

Dieterich Lawson
Medic Mobile
dieterich@medicmobile.org

Dominic König
Sahana Foundation
dominic@nursix.org

Dale Zak
Medic Mobile
dalezak@gmail.com

ABSTRACT

We describe an interoperability challenge that arose in Haiti, identify the parameters of a general problem in crisis data management, and present a protocol called Tablecast that is designed to address the problem. Tablecast enables crisis organizations to publish, share, and update tables of data in real time. It allows rows and columns of data to be merged from multiple sources, and its incremental update mechanism is designed to support offline editing and data collection. Tablecast uses a publish/subscribe model; the format is based on Atom and employs PubSubHubbub to distribute updates to subscribers.

Keywords

Interoperability, publish/subscribe, streaming, synchronization, relational table, format, protocol

INTRODUCTION

After the January 2010 earthquake in Haiti, there was an immediate need for information on available health facilities. Which hospitals had been destroyed, and which were still operating? Where were the newly established field clinics, and how many patients could they accept? Which facilities had surgeons, or dialysis machines, or obstetricians? Aid workers had to make fast decisions about where to send the sick and injured—decisions that depended on up-to-date answers to all these questions. But the answers were not readily at hand.

The U. S. Joint Task Force began a broad survey to assess the situation in terms of basic needs, including the state of health facilities. The UN Office for the Coordination of Humanitarian Affairs (OCHA) was tasked with monitoring and coordinating the actions of the many aid organizations that arrived to help. The Haitian national ministry of health (MSPP) had its own list of health facilities, with one version from 2005 and another version from 2009. The Pan-American Health Organization (PAHO) created a Haiti Health Facilities Master List by combining the data from MSPP with more recent data from MEASURE Evaluation, USAID, US Health and Human Services (HHS), WHO, Ushahidi, the UN Stabilization Mission in Haiti (MINUSTAH), the Sahana Foundation, PEPFAR, and Shoreland Inc. (PAHO, 2010). This list was posted as a Microsoft Excel spreadsheet on a public webpage hosted by Google Sites (PAHO, 2010). It had one row for each facility and had columns for the facility name, geolocation, contact information, damage, operational status, and other information.

It took significant time and effort to gather data from all these sources and compile a comprehensive list. But the Master List had only old MSPP data for most facilities, with no indication of their status after the quake. As soon as the list was published, it began falling out of date: bed capacities changed from day to day; staff were replaced and their contact information changed; new facilities opened to meet new needs. New versions of the Master List were produced, each time requiring considerable manual effort. There were duplicate and obsolete records. Some facilities had multiple geolocations from different times (2005 and 2009) or different sources (MSPP, HHS, and WHO). Facility names were a mix of English and French, inconsistently spelled. The facilities were originally identified by a 5-digit PCode assigned by MSPP, but because the PCodes kept changing, PAHO assigned a 7-digit number called a HealthC_ID, to be used instead as a stable primary key.

The organizations that actually set up, staffed, and ran the health facilities lacked strong incentives to use the Master List. Each such organization naturally had its own list of its own facilities, which would always be more up to date and thus more useful to consult than the Master List. Further, investing the extra effort required to update the Master List would not yield results until the next version of the list was published.

Reviewing Statement: This short paper has been fully double-blind peer reviewed for clarity, relevance, and significance.

Both Shoreland and Google tried to facilitate updates to the list in order to address this problem. Shoreland deployed its Travax software to run a website where the list could be viewed and searched, and users with authorized accounts could make edits. Shoreland's call center staff also made edits as requested by telephone. A Google team built the Google Resource Finder site, which allowed the Haiti health facilities to be viewed, filtered, and edited on a map. Google Resource Finder accepted edits from anyone online or by e-mail, and tracked the author and version history of each field. Both these projects made the list more accessible and made updating easier in different ways—but they resulted in two different data sets. Each attempt to reconcile them required lengthy negotiation and manual work to export, clean, and import the data. Despite such efforts by both Shoreland and Google, the data sets continually diverged as different edits were made on each site.

The cholera outbreak that began in October 2010 created two kinds of pressure on the Master List. First, many new specialized facilities were being set up to treat cholera. PAHO assembled a list of the cholera treatment facilities, but this list contained many duplicates and required manual validation before it could be added to the Master List; and again the list was quickly out of date as facilities opened and closed and capacity was shifted to follow the geographical progress of the epidemic. Second, additional information related to cholera became important to track—the type of cholera treatment available at each facility, the number of cholera beds, the number of cholera cases observed—which required new columns in the Master List spreadsheet. These would require corresponding schema changes in Shoreland's and Google's systems.

As of January 2011, there remain multiple, differing, incomplete, partially outdated lists of facilities. Some are synchronized through continuous manual effort, and some have no regular synchronization process.

PROBLEM

The story of the Haiti Health Facilities Master List illustrates several difficulties of real-life data management:

- The data came from multiple sources with widely varying latency and reliability.
- Different sources provided data on different items (i.e. rows) or in different fields (i.e. columns).
- Different sources provided different values or versions of the same field for a particular item.
- Users of the data had varying needs for reliability and had different levels of trust in different sources.
- The data changed constantly, too fast for a central organization to reconcile changes by hand.
- The data was being maintained in multiple places and stored in very different storage systems.
- New fields (columns) of data were added to the live data set in response to new information needs.

Our hypothesis is that these issues are common to many kinds of crisis information, and that they are significant obstacles to the coordination of relief actors in general. Although comprehensive situational awareness would be highly valuable to most actors, the capacity and motivation to provide the necessary data is too low and further diminished by the large overhead of assembling and reconciling the data.

Perhaps if all actors edited a single shared data set on one central website, many of the above difficulties would vanish. But total centralization is neither practicable nor politically feasible, particularly in a crisis context:

- Most organizations already use their own software and practices for collecting and managing data.
- A single site is unlikely to have a feature set that fully suits every actor's needs.
- Central management of user identities and permissions across all actors is prohibitively difficult.
- Not all actors want to see the same view of the data: some actors may be willing to tolerate less reliable data in exchange for speed or coverage, but the level of tolerance will vary from context to context.
- Not all actors have reliable Internet access, and will need to collect and edit data while offline. Those that do have Internet access often lack the bandwidth to support live online editing.
- Users of the data may need to keep that data updated in other systems; for example, OpenStreetMap also contained copies of the Haiti health facility data in the form of map features.

What is needed is a decentralized way to share, aggregate, and reconcile changes to crisis data—a common protocol rather than a common tool. The above challenges suggest these design requirements for a protocol:

- It should facilitate the integration of partial data from multiple sources.
- It should enable data consumers to select which sources to accept data from.
- It should allow for multiple versions of a piece of data, at different times or from different sources.
- It should accommodate data that is collected or edited offline or delayed in transit.

- It should accommodate schema changes.

DESIGN

Tablecast is a way of broadcasting the changes in a data set. For simplicity, Tablecast assumes that data sets have a two-dimensional tabular structure, following the widespread practice of managing data in relational databases and in spreadsheets that mimic relational tables. The Tablecast protocol refers to each row of a table as a *record* and identifies each column by a *field name*. In order to enable different parties to refer to the same record, Tablecast requires each record to have a unique *record identifier*.

The Tablecast specification (see http://tablecast.org/) has two parts: it specifies an Atom-based data format (Tablecast feeds) and an HTTP-based API (Tablecast services). It is possible for an application to import or export the data format (for example, in an offline context) without necessarily implementing the API.

Tablecast feeds

A Tablecast feed is an Atom feed representing a stream of changes to a table, called *edits*. Each edit is contained in one Atom entry in the feed. A Tablecast edit updates the values of some of the fields in a particular record. Each edit has an *author* (the party responsible for the new values) and an *effective time* (the time that the new values took effect). Each field can have an optional *comment* to explain the change. Using a timestamp (the effective time) and an idempotent operation (replacing with a new value) enables applications to apply a set of edits in any order and get the same result. This is an example of a Tablecast edit:

```
<tc:edit tc:record="tag:example.org,2010:1234567"
         tc:author="mailto:user@mailprovider.org"
         tc:effective="2010-06-29T15:27:39Z"
         tc:type="{http://schemas.google.com/tablecast/2010}row">
  <tc:row>
    <tc:field tc:name="facility_name">"New name"</tc:field>
    <tc:field tc:name="available_beds"
              tc:comment="estimated by doctors on site">55</tc:field>
  </tc:row>
</tc:edit>
```

This says that the record with identifier `tag:example.org,2010:1234567` was updated by someone with the e-mail address `user@mailprovider.org` at 15:27:39 on June 29, 2010, UTC; the `facility_name` field was set to the string value "New name" and the `available_beds` field was set to the numeric value 55.

This example reveals a few syntactic details about the format. The edit is a single `<tc:edit>` element, to be wrapped inside an Atom entry contained in an Atom feed. The record identifier is a tag: URI, the author identifier is also a URI, and the effective time is expressed as an RFC 3339 timestamp in UTC. The field values are expressed in JSON. Like a spreadsheet, the Tablecast data model allows an arbitrary set of columns with dynamically typed values. The table schema is not fixed; new columns may be added at any time.

Publish/subscribe model

PubSubHubbub (Fitzpatrick, Slatkin, and Atkins, 2010) is a protocol by which:

- A subscriber indicates to a hub that it is interested in a particular Atom feed.
- A publisher of a feed notifies the hub when the feed is updated.
- The hub picks up the updates and promptly pushes them out to all the interested subscribers.

Tablecast is designed around this publish/subscribe model. For any given data set, a data provider can publish a Tablecast feed of the updates to the data. It can choose to make the feed available at a public URL, or to use an unguessable URL that it shares only with specific other parties. Each data consumer can choose to subscribe to any feeds it likes, and accept or ignore any of the edits according to its own policies. Since subscribers can also publish their own feeds, they can produce curated streams of edits for others to subscribe to, or do any other kind of processing they wish. We believe there are interesting possibilities for stream processing agents that subscribe to Tablecast feeds, process them, and publish the results as Tablecast feeds.

In this model, publishers compete to provide reputable feeds, and subscribers choose which feeds they trust. Subscribers derive confidence in data by inspecting the attribution in the `tc:author` field and by choosing publishers who maintain that field in a trustworthy manner.

Tablecast services

Although we have described a Tablecast feed only as a stream of changes so far, the stream alone is insufficient: a new subscriber also needs a way to acquire the data that already exists in the data set. Thus, we define two different views of the data set:

- A stream view, which conveys an ongoing stream of changes to the data
- A snapshot view, which conveys all the records in the data set at the current moment

A Tablecast service is a URL that supports specific sets of query parameters such that clients can retrieve both of these views. When a client first subscribes to a service, it can request a snapshot view to obtain the state of the entire data set at that moment. From then on, it can use the stream view to receive incremental updates.

The stream view allows the client to request only the edits that are newer than a particular time, and iterate forward to the present. Note that "newer" refers to the time that the publisher *began publishing* an edit, not the effective time of the edit. This allows a client to retrieve just the edits it has not previously retrieved, and thereby perform an efficient incremental update.

The snapshot view allows the client to iterate over the data set in order by record identifier. The entire data set need not be retrieved in one atomic operation. Because edits are idempotent, it is safe for the client to retrieve a snapshot in multiple requests, observing the values of different records at different times, as long as it retrieves edits from the stream view starting from the time of the first snapshot request.

IMPLEMENTATIONS

TextForms

In disaster situations, communication infrastructure often fails, leaving many without reliable telephone and Internet access. Text messaging is often one of the first communication methods to become available again. To take advantage of this quick rebound time, Medic Mobile has developed an application called TextForms that allows organizations to collect structured data using only simple text messages. With TextForms, a clinic disconnected from the Internet can provide decision makers with up-to-date information by sending text messages to a TextForms server, which in turn forwards the information to a Tablecast service.

To set up TextForms for data collection, one defines fields and easily textable abbreviations for these fields. For example, one can define a field named "Hospital Status" with the abbreviation `hospstat`. A user would then submit a "Hospital Status" report by sending the text message "`hospstat damaged`" to a TextForms server. The general form of a TextForms message is a field abbreviation followed by the field value. Multi-step text message interactions are used for data validation and complex data types such as checklists.

To provide data in the Tablecast format, TextForms allows administrators to link each field to its corresponding Tablecast field name. Using the field names, TextForms can generate Tablecast XML and post it to a configurable Tablecast service URL. Because of the limitations of the text messaging interactions, TextForms does not implement the comment attribute of the field tag or the record attribute of the edit tag.

By bridging the gap between SMS and the Internet, TextForms has the potential to improve the quality of data gathered in crisis situations and bring internet-disconnected facilities inside the information loop.

Google Resource Finder

Google Resource Finder maintains a data set containing records for the Haiti health facilities. Its purpose is to bridge several methods for updating and retrieving health facility information: the data can be viewed on the website or retrieved by e-mail, and it can be updated on the website, by e-mail, or (via TextForms) by SMS.

Google Resource Finder is both a Tablecast publisher and a Tablecast subscriber. When a user on the website edits a health facility record, a Tablecast edit reflecting the changes is added to the outgoing feed. Google Resource Finder can also be configured to subscribe to any number of other Tablecast feeds, via PubSubHubbub. When an edit arrives from one of these feeds, the change is reflected on the website, and the edit is also republished on the outgoing feed where it can be picked up by any other subscribers.

Integration with TextForms is achieved using Tablecast. TextForms does not itself play the role of a Tablecast server, because it is designed to run on an in-country PC, which may not have a constant Internet connection and a stable IP address. Instead, TextForms posts its Tablecast edits to an intermediary server (at a reliable Internet hosting facility), which publishes them in a Tablecast feed. Google Resource Finder subscribes to this latter feed. PubSubHubbub notification ensures that Google Resource Finder receives new edits within seconds.

RELATED WORK

FeedSync

FeedSync (Microsoft, 2010) is a protocol for "bidirectional, asynchronous synchronization of new and changed items" in Atom and RSS feeds. Although both FeedSync and Tablecast can be used for maintaining identical copies of a data set, they have different design goals and capabilities:

- FeedSync is aimed at ensuring global consistency of two or more copies of a collection, and specifies a merging algorithm that participants must perform to achieve this. Tablecast neither guarantees nor requires this; a subscriber can maintain a subset of the data or a result computed from the data set.
- FeedSync is designed to handle any RSS items or Atom entries (which can contain arbitrary XML), whereas Tablecast is primarily focused on tabular data.
- Tablecast takes advantage of PubSubHubbub to transmit immediate updates to subscribers.

S3XML

S3XML is the native XML import/export format used by Sahana Eden, an open source platform for disaster management. Sahana Eden exposes its data as table-like collections through a REST interface, in both S3XML and S3JSON (an isomorphic JSON format). This REST interface also supports other XML formats by applying XSLT stylesheets to transform incoming or outgoing feeds into or from S3XML.

S3XML is similar to Tablecast in that it includes a unique record identifier, a timestamp, and source and record owner attributes, and can specify the values of any subset of fields in the record. However, it does not carry timestamps on individual fields and does not exploit PubSubHubbub for broadcasting updates.

Database and XML delta formats

Many free and commercial tools exist to compare database tables (SQL Compare, SQL Delta, tablediff, and so on); most of them emit differences in the form of a SQL script that can be applied to one table to make it match the other. There are also existing data formats for representing changes in an XML document (such as Delta Update Language and RFC 5261); conceptually they express sequences of editing operations on the elements in the document. The Tablecast format contrasts with these SQL and XML formats primarily in that Tablecast is less expressive, simpler, and designed to tolerate edits arriving out of order.

CONCLUSION

We have presented Tablecast, a protocol for publishing updates of data tables to interested subscribers, inspired by a real-world crisis data management scenario. The first few independent implementations of Tablecast are being developed and tested. We are hopeful that Tablecast will substantially reduce the barriers to data sharing and coordination among crisis tools and relief organizations.

ACKNOWLEDGEMENTS

Thanks to Alyson Rose-wood at U. S. HHS and Bill Lang at Shoreland for their input on this paper.

REFERENCES

1. Fitzpatrick, B., Slatkin, B., and Atkins, M. (2010) PubSubHubbub Core 0.3 Working Draft, available at: http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html
2. Microsoft Corporation. (2010) FeedSync for Atom and RSS, available at: http://feedsyncsamples.codeplex.com/wikipage?title=FeedSync%20for%20Atom%20and%20RSS%20(v1.0)%20specification
3. Pan-American Health Organization. (2010) Field definitions and metadata for Haiti Health Facilities, available at: http://new.paho.org/disasters/index.php?option=com_docman&task=doc_download&gid=812
4. Pan-American Health Organization. (2010) Haiti Health Facilities Files & Documents, available at: https://sites.google.com/a/netspective.org/haiti-health-facilities/files