A Spreadsheet Model for Handling Streaming Data

Kerry Shih-Ping Chang and Brad A. Myers
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{kerrychang, bam}@cs.cmu.edu

ABSTRACT

We present a spreadsheet model for working with streaming data. Our prototype tool presents techniques to let the user stream data from web services and web input elements to a spreadsheet without preprogramming those sources into the tool. Spreadsheet cells record metadata about where and when the data came from, allowing the user to view and manipulate streaming data using temporal information. Starting and pausing a data stream in the spreadsheet can be controlled programmatically using values computed by spreadsheet cells, making the spreadsheet program highly dynamic and interactive. We demonstrate the range of our design with a series of examples highlighting its ability to create different kinds of applications that process real-time data from the web using simple spreadsheet formulas.

Author Keywords

Spreadsheets; streaming data; web services; end-user programming; live programming

ACM Classification Keywords

H.4.1. Information Systems Applications – Spreadsheets

INTRODUCTION

Much data analysis work nowadays uses real-time data such as market prices, geo-locations of people and vehicles, or social network feeds. Tools that allow easy, ad-hoc use of streaming data sources could be very useful for people such as data analysts, emergency first responders, or even casual users who want to explore real-time web data. For example, one might want to monitor weather alerts from National Weather Service along with real-time searches for Twitter feeds in the alert areas for emergency events, or to monitor item auction prices on eBay to see the trends and guess the right time and price to bid. Currently, using real-time web data requires programmers to writing complex code that streams the data from the providers and dynamically manipulates the collected data. Commercial data analysis or visualization software can let users play with built-in data sources but often lacks customizability in terms of adding new data sources and using them in an ad-hoc manner.

This work extends our system called Gneiss[1] [3] (see the next section for an overview) that lets users create web applications by linking web UI elements to spreadsheet cells. In this paper, we describe a spreadsheet model that lets users work with streaming data from web data sources. Our spreadsheet model has several innovations: first, it presents techniques that allow users to stream any fields from arbitrary REST JSON web services without needing a developer to preprogram those sources into the tool (which is often a requirement in prior end-user mashup or spreadsheet tools such as [5,6]). Second, it introduces a design for spreadsheet cell "metadata" which describe other attributes of a cell's value and can be used to manipulate spreadsheet data. In this work, each cell automatically records metadata of its value's provenance and fetched time, allowing users to view or manipulate streaming data in the spreadsheet using temporal information, such as getting the daily maximum and minimum values. Lastly, Gneiss allows streaming to be paused and restarted using conditions computed live from spreadsheet data using formulas. These features make the created spreadsheet program very dynamic and interactive.

To show the generalizability of our model, we demonstrate how the same mechanism that handles streaming data from web services can be used for collecting user data in web input elements such as textboxes on web pages. A spreadsheet column can be set to pull data from a web UI element either when the input element changes or when triggered by live conditions. The data will be saved as a data stream in the spreadsheet, making the spreadsheet work as a backend database for the web application.

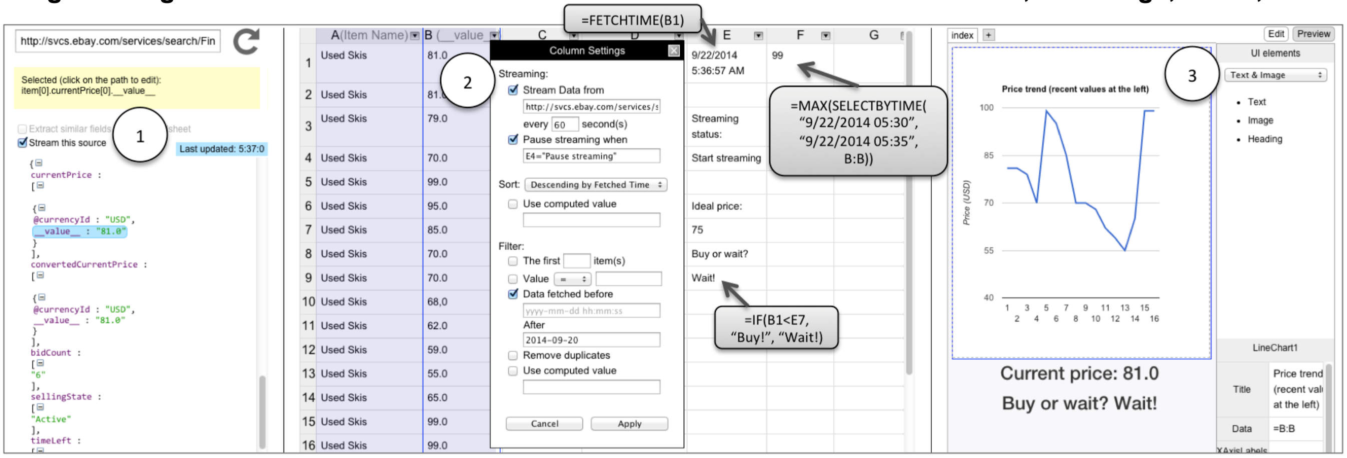
Combining all these features, this paper contributes a novel spreadsheet model for using streaming data, where custom real-time applications require only a few spreadsheet formulas that otherwise would require writing complex code.

BACKGROUND

Our prior system called Gneiss [3] introduced a spreadsheet environment for using web service data and creating inter-

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3145-6/15/04...$15.00.
http://dx.doi.org/10.1145/2702123.2702587

[1] Gneiss (pronounced the same as "nice") is a kind of rock. Here it stands for Gathering Novel End-user Internet Services using Spreadsheets.



http://svcs.ebay.com/services/search/Fin

Selected (click on the path to edit): item[0].currentPrice[0].__value__

□ Extract similar fields sheet
☑ Stream this source
Last updated: 5:37:0

```
{⊟
currentPrice :
[⊟

{⊟
@currencyId : "USD",
__value__ : "81.0"
}
],
convertedCurrentPrice :
[⊟

{⊟
@currencyId : "USD",
__value__ : "81.0"
}
],
bidCount :
[⊟
"6"
],
sellingState :
[⊟
"Active"
],
timeLeft :
```

	A(Item Name)	B (__value	C	D	E	F	G
1	Used Skis	81.0			9/22/2014 5:36:57 AM	99	
2	Used Skis	81.0					
3	Used Skis	79.0			Streaming status:		
4	Used Skis	70.0			Start streaming		
5	Used Skis	99.0					
6	Used Skis	95.0			Ideal price:		
7	Used Skis	85.0			75		
8	Used Skis	70.0			Buy or wait?		
9	Used Skis	70.0			Wait!		
10	Used Skis	68,0					
11	Used Skis	62.0					
12	Used Skis	59.0					
13	Used Skis	55.0					
14	Used Skis	65.0					
15	Used Skis	99.0					
16	Used Skis	99.0					

Column Settings

Streaming:
☑ Stream Data from
http://svcs.ebay.com/services/s
every 60 second(s)
☑ Pause streaming when
E4="Pause streaming"

Sort: Descending by Fetched Time
□ Use computed value

Filter:
□ The first item(s)
□ Value =
☑ Data fetched before
yyyy-mm-dd hh:mm:ss
After
2014-09-20
□ Remove duplicates
□ Use computed value

=FETCHTIME(B1)

=MAX(SELECTBYTIME(
“9/22/2014 05:30”,
“9/22/2014 05:35”,
B:B))

=IF(B1<E7,
“Buy!”, “Wait!)

Current price: 81.0

Buy or wait? Wait!

Title	Price trend (recent val at the left)
Data	=B:B
XAxisLabels	

Figure 1: A screenshot of our tool. Here the user creates a real-time application that streams the name and price of a product on eBay to the spreadsheet (column A and B) every 60 seconds. Each streamed cell records when its data is retrieved (E1) and this temporal information can be used in sorting and filtering, or in spreadsheet formulas (F1) to select cells.

active web applications. Gneiss contains three panes (see Figure 1): at the left (1) is a “source pane” that let users load data from arbitrary REST web services returning JSON data; at the center (2) is a spreadsheet that holds the data to be shown in a web application, and supports arbitrary computations using that data and sorting and filtering rules; at the right (3) is a web interface builder where web pages can be created interactively. Data flow is two-way between the spreadsheet and a web service as a spreadsheet cell value can be referenced in a web API (entered in the URL textbox at the top of the source pane) using the syntax `{{cellName}}` which will invoke the web service whenever the cell’s value changes. In the web interface builder on the right (3), the user can create UI elements and visualizations through drag-and-drop, and assign properties of UI elements to be computed from spreadsheet cells. UI element properties can be referenced in the spreadsheet using the syntax `UIID!PropertyName`. This enables two-way data flow between the spreadsheet and the web interface builder, thus allowing the creation of web applications that can interactively retrieve and present backend data.

Our work here extends Gneiss by providing a spreadsheet model for working with *streaming* data. The previous Gneiss system did not handle data streams nor did it support cell metadata that allow spreadsheet data to be manipulated based on the time fetched.

RELATED WORK

Some conventional spreadsheet tools have pre-programmed formulas or widgets that pull real-time data from external sources, such as the GoogleFinance formula in Google Spreadsheets or Microsoft Excel’s web query. These tools by default do not support creating data streams – they update individual cells with latest values from the sources but do not preserve past sequences of data. Woo et al. [6] extend Excel to collect and process sensor data. Sensor values are stored on a server and streamed to a spreadsheet to use in formulas or visualizations. ActiveSheets [5] is another Excel extension for streaming data. It provides more controls of how and what data should be streamed, such as letting users create new streams using formulas or pause a stream using a button. However, neither system supports manipulating streaming data using temporal information of when the data are retrieved. They also require data sources to be preprogrammed into the server. Forms/3 [2] is a spreadsheet programming tool for creating graphical interfaces. It has a time model for storing and retrieving GUI I/O events, but it does not deal with data sources. There are other languages that support using streaming data, such as Microsoft’s StreamInsight [1] for creating streaming applications in C#. In contrast, our work extends the spreadsheet metaphor, because it is already familiar to many end-users.

KEY FEATURES

Here we describe the key novel features in our tool for handling streaming data.

Create a Data Stream in the Spreadsheet

Figure 1 shows a screenshot of our tool’s interface. Our tool supports REST web services returning JSON data. The user can enter a web API at the top URL bar in the source pane and view the return data below. In our tool, arbitrary web services can be turn into a streaming source if the checkbox “Stream this source” is checked. The system then starts to send the web service request every 3 seconds (configurable) and updates the source pane with the latest return data. We use a roll-up text animation when replacing old values in the return data to show that they have been refreshed. Unchecking the checkbox stops the streaming. The last updated time of the data is shown in a blue label next to the streaming checkbox.

To create a data stream in the spreadsheet, the user selects a desired field in the return data and drags it to the top cell of a spreadsheet column. The system then starts to stack the column with the latest values of the field pulled from the web service. By default the values are sorted descending by time, so the newest value appears at the top of the column. Using this mechanism, the user can easily stream multiple fields from one or multiple web services in a spreadsheet.

Manipulate Streaming Data using Temporal Information

Streaming data are inherently time-series data, and so the ability to view or manipulate streaming data in the spreadsheet by time is essential. To enable this, we designed each spreadsheet cell to have metadata that describe attributes of its value. The metadata are by default not visible but can be exposed through formulas and be used to manipulate, sort and filter spreadsheet cells (note that cell metadata in our tool are different from user comments in conventional spreadsheets as they are set and maintained automatically by the system). In our spreadsheet model, each streamed cell stores not only its display value but also metadata about its provenance and fetched time, allowing data to be viewed and manipulated using its value, source and temporal information. For example, a streamed column can be filtered to show only the data retrieved today, sort them descending by value and filtered to show only the top 5 rows, to view only the 5 highest values of the day. Our tool applies sorting and filtering rules *dynamically* in the spreadsheet – they are re-evaluated every time when new data are retrieved. So in the above example, the column will update continually to display the 5 highest values. Sorting and filtering rules of a column are controlled by a dialog box (Figure 1 at 2) that is brought up by clicking on the arrow button at the top of the column. By default, columns extracted from the same streaming source are sorted and filtered together and are highlighted in the same color when the dialog box is open.

As mentioned earlier, cell metadata can also be accessed through spreadsheet formulas. We provide a new formula `FETCHTIME(cellName)` that returns the retrieval time of a streamed cell. The return value is in standard ISO 8601 format and can be used with conventional spreadsheet time functions. We provide another formula `SELECTBYTIME( startTime, endTime, range)` that returns an array of values in `range` that are streamed between `startTime` and `endTime`. The `SELECTBYTIME` formula can be easily used together with many conventional spreadsheet formulas that process a set of values. For example, suppose column B in the spreadsheet holds latest news streamed from a news data source. The formula:

```
 =COUNTIF(SELECTBYTIME(“2014-09-21 9:00”,
“2014-09-21 10:00”, B:B),“*White House*”)
```

returns the number of news articles fetched between 9-10am on September 21st, 2014 that contain the phrase “White House”.

Control Streaming Timing

Our model also allows the user to set when it should pull data from a streaming source. To do so, the user opens the column setting dialog box of a streaming column. There she can set the tool to pull data periodically every *x* seconds (for example, “60 seconds” in Figure 1 at 2). She can also choose to pause a stream when a given condition is true (if the condition is not specified, the stream pauses immediately). For example, in Figure 1, the user selects the “Pause streaming when” checkbox and enters “E1=’Pausing streaming’” as the condition, then our tool will pause streaming from the source if cell E1 becomes “Pause streaming”, and restarts streaming when E1 becomes other values. This feature also allows the user to create applications that dynamically stream data from different sources.

Stream Data from Web Input Elements

Our spreadsheet model can also be applied to stream data from web input elements such as textboxes or forms. The user can set a column to pull data from a web input element by checking on the “Stream data from” checkbox in the column setting dialog box, and setting the input element to be the streaming source. For example, entering `TextBox1!Value` as the streaming source sets the column to stream the value of `TextBox1`. By default, the column pulls data from an input element whenever its value changes. The user can further use the “pause” mechanism described earlier to start streaming only when certain condition is true. For example, using `Button!State!=“clicked”` as the pause condition makes the column stream data from `TextBox1` only when `Button` is clicked. Like spreadsheet cells storing data streamed from web services, cells storing data streamed from web input elements also have the same metadata and can be manipulated by retrieval time. This feature also enables our spreadsheet to be used as a form of database for a web application, as it stores input values as streams in the spreadsheet.

Save and Close a Spreadsheet

The user can save her spreadsheet on the server and decide whether to save the streaming data after she closes the spreadsheet. By default, the streamed data are stored on the server’s database, and all streaming stops when the spreadsheet is closed. All streaming restarts when a spreadsheet is opened. The user can also choose to remove all streaming data on the server when closing the spreadsheet, or to keep streaming even when the spreadsheet is closed (although our server has a limit for how much data can be stored in the database[2]). As described in [6], the web pages created in the web interface builder can also be exported as a web application running on the server.

IMPLEMENTATION

Our tool is implemented as a web application that contains a client-side spreadsheet tool and a backend server. All streaming data are stored in the backend server for better performance. Based on the data selection rules that the user sets in the spreadsheet (such as filtering), when a new data value arrives, the server selects the appropriate data and returns them to the frontend. We use node.js and MongoDB in the backend, and ConstraintJS [4] in the frontend.

[2] We use MongoDB as the database. See its limitations at http://docs.mongodb.org/manual/reference/limits/

DEMONSTRATIVE EXAMPLES

Here we use two examples to demonstrate our tool's ability to create real-time applications that use live streaming data requiring only a few simple lines of spreadsheet code.

Auction Price Tracker

Online auction marketplaces such as eBay let customers bid on an item. The price of an item could change as bids are added or if the seller adjusts the base price. Using our tool, the user can create a real-time price tracker. Suppose the user wants to monitor the price of a product, see its highest and lowest value daily, and get notified when the price is below a threshold (Figure 1 shows a screenshot of a similar example).

The user first uses eBay's FindAPI to get information about the desired product. In the source pane she sets the tool to stream data from eBay and extracts the name and price field from the return data to spreadsheet columns A and B using drag-and-drop. The tool starts to stack data streamed from eBay to columns A and B, with the latest data in the top row. To get the highest price value of today, she enters

```
=MAX(SELECTBYTIME("2014-09-21 00:00:00",
        "2014-09-21 23:59:59", B:B))
```

in cell D1 (as September 21st is the date of today). She then enters the same formula with dates replaced with September 22nd in D2, selects both D1 and D2, and drags the selection down to D7 to fill in D1 to D7 with the highest value each day in the week of September 21st, using the familiar "auto-fill" mechanism in spreadsheets. She similarly fills in E1 to E7 with the daily lowest price. Finally, she enters a price threshold in cell F1 and enters `=IF(B1<F1, "Bid!", "Wait")` in cell F2 to remind herself when to make a bid. In the web interface builder, she creates a web page that has two graphs: one visualizing column B (the price trend, shown in Figure 1 at 3), and another graph showing columns D and E (the daily high and low values) in real-time, along with a text label showing the current price (B1) and another text label showing if she should make a bid (F2).

Real-time Weather Alerts based on Locations

Suppose a user works in a company that has an internal web service tracking current locations of company trucks and wants to create an application that monitors the weather condition at a truck's current location to alert the driver of issues such as if the atmosphere visibility becomes too low. The user first sets column A to stream GPS coordinates of a truck from the company web service and sort the data descending by time, thus the most recent coordinates are in cell A1. To retrieve weather data of the truck's current location, the user uses Yahoo's Weather API, replaces the query value to refer to cell A1 and streams the visibility field from the return data to column B. Cell B1 thus becomes the atmosphere visibility reading of the truck's current location. The user can then use an `IF` formula to check B1's value and see if it is below a certain threshold.

As mentioned earlier, our tool also allows the user to pause a data stream programmatically. Suppose the user stores the location of the truck's destination in cell C1. She can then set the pause condition to be `A1=C1`, so the streaming stops when the truck arrives at its destination.

CONCLUSIONS AND FUTURE WORK

This work contributes a model for using streaming data in spreadsheets. It includes techniques to let users stream data from web sources to a spreadsheet without writing conventional code, a design for spreadsheet cell metadata to let users manipulate spreadsheet data using temporal information, and ways to dynamically control when to pull new data using the spreadsheet language and interaction techniques. Based on this model, our prototype tool provides a live environment for dealing with live streaming data. For future work, we would like to support more types of streaming data sources, such as streaming web APIs (like Twitter's) and mobile sensors. We are interested in exploring more usage of cell metadata, such as to assist people in integrating data from multiple sources or editing a collaborative spreadsheet. We will also run a formal user study evaluating the full Gneiss system.

ACKNOWLEDGEMENT

This research was funded by the NSF under grant IIS-1314356. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

REFERENCES

1. Ali, M., Chandramouli, B., Goldstein, J., and Schindlauer, R. The Extensibility Framework in Microsoft StreamInsight. *Proc. ICDE*, IEEE (2011), 1242–1253.
2. Burnett, M., Atwood, J., Walpole Djang, R., Reichwein, J., Gottfried, H., and Yang, S. Forms/3: A First-order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *J. Funct. Program. 11*, 2, Cambridge University Press (2001), 155–206.
3. Chang, K.S.-P. and Myers, B.A. Creating Interactive Web Data Applications with Spreadsheets. *Proc. UIST*, ACM (2014), 87–96.
4. Oney, S., Myers, B., and Brandt, J. ConstraintJS: Programming Interactive Behaviors for the Web by Integrating Constraints and States. *Proc. UIST*, ACM (2012), 229–238.
5. Vaziri, M., Tardieu, O., Rabbah, R., Suter, P., and Hirzel, M. Stream Processing with a Spreadsheet. In R. Jones, ed., *ECOOP 2014 – Object-Oriented Programming SE - 15*. Springer Berlin Heidelberg (2014), 360–384.
6. Woo, A., Seth, S., Olson, T., Liu, J., and Zhao, F. A spreadsheet approach to programming and managing sensor networks. *Proc. ISPN*, ACM (2006), 424–431.