A Spreadsheet Tool for Creating Web Applications Using Online Data

Kerry Shih-Ping Chang
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
kerrychang@cs.cmu.edu

Brad A. Myers
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
bam@cs.cmu.edu

Abstract
We present a tool called Gneiss that extends the familiar spreadsheet model to support creating data-drive web applications that use data from the Internet. Gneiss allows the user to extract data from arbitrary REST web services to a spreadsheet and send spreadsheet data to web services dynamically without writing conventional code. It provides new spreadsheet functions and interface elements to support using and manipulating structured data and streaming data. Moreover, Gneiss allows users to create not only visualizations but also full-fledged interactive web applications that can present and modify the spreadsheet data.

Author Keywords
Spreadsheets; end-user programming

ACM Classification Keywords
H.4.1. Information Systems Applications – Spreadsheets

Introduction
This paper presents a tool called Gneiss that aims to extend the familiar spreadsheet model for the Internet era to reduce the barrier of using and publishing data online for end-users. Conventional spreadsheets take local, static data and manipulate it to produce local, static graphs and charts. However, many data sources

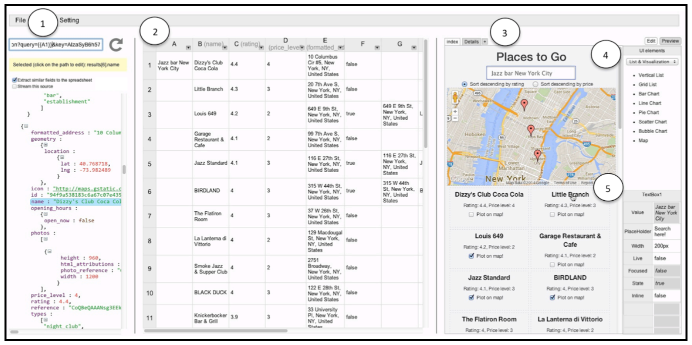
nowadays are no longer static files but dynamic, real-time data sources on the Internet such as web services and mobile sensors. Further, what many people want to create are also not static document files anymore but rather interactive websites or web applications [6]. While some conventional spreadsheets like Microsoft Excel are being enhanced with some ability to access web data and share content online, they provide little or no support to using structured data (such as JSON and XML which are common formats for web services) and streaming data (such as sensor data, financial data feeds), and can only let the user publish data as tables and visualizations. Prior research spreadsheet tools have explored extending the spreadsheet language to support programming GUI applications [1,2] and facilitating cleaning data through programing-by-demonstration [7]. Different from these systems, Gneiss focuses on enabling users to create data-driven applications that interact with a variety of data sources.

Gneiss

Gneiss makes contributions by extending the spreadsheet model to support interacting with arbitrary REST web services, using structured and streaming data, and programming data bindings and interactive behaviors in web applications. Here we briefly describe key features with respect to these contributions. Full features of Gneiss are described by publications at VL/HCC'14 [4], UIST'14 [3] and CHI'15 [5]. A screenshot of Gneiss is shown in Figure 1.

Two-way data flow with web services

Gneiss currently supports arbitrary REST web services that return JSON data. To load data from a web service, the user enters an API in the URL bar in the source pane (Figure 1 at 1). The returned JSON data

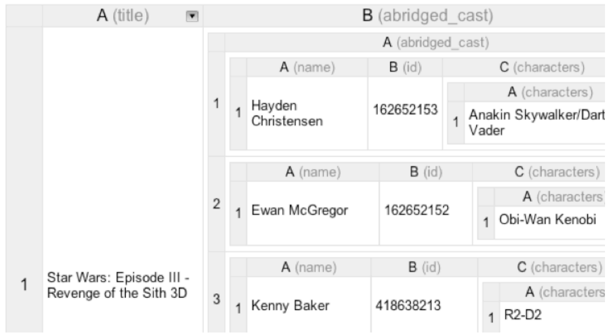


	A	B (name)	C (rating)	D (price_level)	E (formatted_…	F	G
1	Jazz bar New York City	Dizzy's Club Coca Cola	4.4	4	10 Columbus Cir #5, New York, NY, United States	false	
2		Little Branch	4.3	3	20 7th Ave S, New York, NY, United States	false	
3		Louis 649	4.2	2	649 E 9th St, New York, NY, United States	true	649 E 9th St, New York, NY, United States
4		Garage Restaurant & Cafe	4.1	2	99 7th Ave S, New York, NY, United States	false	
5		Jazz Standard	4.1	3	116 E 27th St, New York, NY, United States	true	116 E 27th St, New York, NY, United States
6		BIRDLAND	4	3	315 W 44th St, New York, NY, United States	true	315 W 44th St, New York, NY, United States
7		The Flatiron Room	4	3	37 W 26th St, New York, NY, United States	false	
8		La Lanterna di Vittorio	4	2	129 Macdougal St, New York, NY, United States	false	
9		Smoke Jazz & Supper Club	4	2	2751 Broadway, New York, NY, United States	false	
10		BLACK DUCK	4	3	122 E 28th St, New York, NY, United States	false	
11		Knickerbocker Bar & Grill	3.9	3	33 University Pl, New York, NY, United States	false	

Figure 1. The Gneiss interface. (1) is the source panel where the user can load data from a web service. (2) is the spreadsheet editor where the user stores and manipulate the data. (3) is the web interface builder where the user can create a web application by dragging-and-dropping GUI elements from the toolbar on the right (4) to the output page. The user can select a GUI element in the output page and view and edit its properties in (5).

are shown below the URL bar. The user can extract any field in the return data to the spreadsheet by dragging-and-dropping it to any cell in the spreadsheet. Gneiss uses a programming-by-example approach to facilitate extracting similar items in the returned documents, such as the first field of all array items. When the "Extract similar fields to the spreadsheet" checkbox in the source pane is checked, the system will let the user drag a field to the first cell of a spreadsheet column. It will then automatically fill in the rest of the column with similar fields based on the document structure.

The user can also send the value of a cell in the spreadsheet to a web service by replacing any part of the web API in the URL bar with the name of the cell, using the syntax `{{cellName}}`. When a cell is used in a web API, every time the cell changes, the system will send a new API request using the new cell value and



	A (title)		A (name)	B (id)	C (characters)
1	Star Wars: Episode III - Revenge of the Sith 3D	1	Hayden Christensen	162652153	Anakin Skywalker/Dart Vader
		2	Ewan McGregor	162652152	Obi-Wan Kenobi
		3	Kenny Baker	418638213	R2-D2

Figure 2. Gneiss's nested table

update any other spreadsheet cells extracted from this web service with the new return data. This makes the spreadsheet program created in Gneiss easily reusable, as the user can retrieve new data from a web service by editing spreadsheet cells.

Supporting structured data with nested tables
Structured fields extracted from web services (such as a JSON object or array) will be shown using nested tables (see Figure 2). Cells in a nested table can be used in formulas like regular cells, using the syntax `cell.childCell`. For example, in Figure 2, `B1.A2.A1` refers to the cell with value "Ewan McGregor". To assist the user in further manipulating the data in the nested tables, our tool provides a "flatten" function that flattens a nested table column and stores all values in a column. To use the flatten function the user can enter `=flatten(columnName)` in the top cell of a column, and then our tool will automatically fill the cells below with the appropriate data. The syntax for `columnName` is similar to referring to a cell in a nested table, which is `column.childColumn`. Using Figure 2 for example, the user can let column C be all actors in all Star Wars movies by entering `=flatten(B.A.A)` into cell C1 (`B.A.A` is the column that stores the actor names in each abridged cast of each movie). The flatten function populates column C with a flattened list of actor names. The user can then apply filtering to the column to remove duplicate names and get a clean list.

Cell metadata for manipulating streaming data
Gneiss allows the user to stream data from arbitrary REST web services to a spreadsheet by checking the "Stream this source" checkbox in the source pane and then dragging a desired field to a spreadsheet column. The system then starts to stack the column with the latest values of the field pulled from the web service every 3 seconds. By default the values are sorted descending by time, so the newest value appears at the top of the column. The streaming frequency and how the data are sorted can be changed in a dialog box.

Gneiss introduces cell metadata that describe other attributes of a cell's value and allow users to manipulate spreadsheet data using these attributes. Each cell has metadata of its value's provenance and fetched time. This enables users to manipulate data using not only their values but also temporal information of when the value is retrieved. For example, to view the 5 highest values of the day, a streamed column can be filtered to show only the data retrieved today, sort them descending by value and filter to show only the top 5 rows. We design new functions `FETCHTIME(cellName)` and `SELECTBYTIME (startTime, endTime, range)` to let users get the fetched time of a cell and select values in certain cells that are streamed between certain times. For example, suppose column B 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 articles fetched between 9-10am on September 21, 2014 containing the phrase "White House".

Programming interactive, data-driven applications
We extend the spreadsheet model to allow a GUI element property (such as "color" and "value" of a heading element) to use spreadsheet formulas as its value and also be used in other spreadsheet cells. The syntax for refereeing a GUI element property is `ElementID!PropertyName`. To further allow users to program interactive behaviors in web applications, every GUI web element in Gneiss has "interactive properties" whose value will change based on how the user interacts with the element. For example, the "state" property in every GUI element will change based on how the mouse cursor interact with it (such as "idle", "hovered" and "clicked"). Or the "value" property in a text box will change based on what the user types in the text box.

This design allows data bindings and interactive behaviors in a web application to be programmed using the spreadsheet's equation-based evaluation model without the need of conventional event-based model. For example, spreadsheet cell A1 can be set to whatever the user types in a text box in a web application by setting its value to `=TextBox1!Value`. Then A1 can be sent to a web service by replacing the query parameter in a web API with `{{A1}}`. Suppose the search result extracted from this web service is put in column B. Text in a list element in the web application can then be set to `=B:B` to show the data. This creates an interactive search web application. Or the sorting rule of a column can be set to `=IF(Checkbox1!Checked, "Descending", "Ascending")` to sort the column dynamically based on if the user checks the checkbox. The user can also use Gneiss's spreadsheet as a database for the created web application by setting a column to stream data from a web input element (such as a text box). The column will then store the user's input in the element to the spreadsheet as storing a data stream from a web service. This allows users to create data entry applications like a self-tracking application for custom behavior [6].

Future work

We are developing new functions to help users do data transformations and manipulate nested tables in a spreadsheet. We will also enhance Gneiss to support creating mobile web applications and getting data from web pages and mobile sensors to the spreadsheet.

We can bring a demo of Gneiss to the workshop.

References

[1] Burnett et al., Forms/3: A First-order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *J. Funct. Program. 11*, 2 (2001), 155–206.

[2] Burnett et al., FAR: an end-user language to support cottage e-services. *Proc. HCC,* IEEE (2001), 195–202.

[3] Chang et al., Creating Interactive Web Data Applications with Spreadsheets. *Proc. UIST*, ACM (2014), 87–96.

[4] Chang et al., A spreadsheet model for using web service data. *Proc. VL/HCC*, IEEE (2014), 169–176.

[5] Chang et al., A Spreadsheet Model for Handling Streaming Data. *Proc. CHI*, ACM (2015), To appear

[6] Choe et al., Understanding Quantified-selfers' Practices in Collecting and Exploring Personal Data. *Proc. CHI*, ACM (2014), 1143–1152.

[7] Kandel et al., Wrangler: Interactive Visual Specification of Data Transformation Scripts. *Proc. CHI*, ACM (2011), 3363–3372.