Rapid Development of Spreadsheet-based Web Mashups

Woralak Kongdenfha and Boualem Benatallah
University of New South Wales
Sydney, Australia
woralakk, boualem@cse.unsw.edu.au

Julien Vayssière[*]
SAP Research
Brisbane, Australia
julien.vayssiere@sap.com

Régis Saint-Paul[†]
CREATE-NET
Trento, Italy
regis.saint-paul@create-net.org

Fabio Casati
University of Trento
Trento, Italy
casati@dit.unitn.it

ABSTRACT

The rapid growth of social networking sites and web communities have motivated web sites to expose their APIs to external developers who create mashups by assembling existing functionalities. Current APIs, however, aim toward developers with programming expertise; they are not directly usable by wider class of users who do not have programming background, but would nevertheless like to build their own mashups. To address this need, we propose a spreadsheet-based Web mashups development framework, which enables users to develop mashups in the popular spreadsheet environment. First, we provide a mechanism that makes structured data first class values of spreadsheet cells. Second, we propose a new component model that can be used to develop fairly sophisticated mashups, involving joining data sources and keeping spreadsheet data up to date. Third, to simplify mashup development, we provide a collection of spreadsheet-based mashup patterns that captures common Web data access and spreadsheet presentation functionalities. Users can reuse and customize these patterns to build spreadsheet-based Web mashups instead of developing them from scratch. Fourth, we enable users to manipulate structured data presented on spreadsheet in a drag-and-drop fashion. Finally, we have developed and tested a proof-of-concept prototype to demonstrate the utility of the proposed framework.

Categories and Subject Descriptors

D2.2 [**Software**]: Design Tools and Techniques—*Modules and interfaces*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical user interfaces, Interaction styles, Prototyping*; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Design

Keywords

Web data mashups, spreadsheets, component model, spreadsheet-based mashup patterns

[*]Julien Vayssière is now with the Smart Services CRC, Sydney, Australia, and can be reached at Julien.Vayssiere@smartservicescrc.com.au.

[†]Work done while the author was at the University of New South Wales.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.
WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

1. INTRODUCTION

Service Oriented Architecture (SOA) and Web 2.0 foster a transition from desktop applications to web applications. This trend is enabled by Web accessible services such as Flickr, MySpace and Yahoo! Groups that allow users to store and manipulate their data, as well as build their own applications on the Web. While these services enable access to individual Web data sources and applications, users also demand for the creation of value-added applications by aggregating existing services [22]. For example, a user may want to collect Sydney's attractions and restaurants suggested by her friends in Yahoo! Groups, and collect photos related to the suggestions from Flickr. To support this need, a proliferation of online mashup development services have been developed, which allow users to create mashup applications by composing data, presentation and application functionalities from disparate Web services. Examples are Yahoo! Pipes [7], Microsoft Popfly [5] and Intel Mash Maker [3].

While the existing mashup tools improve the development of mashup applications, important challenges also emerge. Specifically, the advancement of techniques for creating mashups are driving companies to build their business models around mashups [16]. These developments seek to enable knowledge workers to effectively perform their routine tasks, which typically involve accessing, analyzing and integrating information from various sources. We believe that knowledge workers, who typically have no programming background, should be able to benefit from the power of the SOA and Web 2.0.

In this paper, we aim to address the above needs by providing a framework that allows users to develop Web data mashups within spreadsheets. Spreadsheets are an ubiquitous tool for the analysis and manipulation of data by desktop users [26]. They have compelling advantages that today we take for granted. They are simple, intuitive and work very well for performing data visualization and manipulation. The fact that they are used daily by a vast majority of users not only proves that they are very usable and useful, but also allows us to capitalize on the fact that users are accustomed to this paradigm. This motivates us to investigate the opportunity of using spreadsheets to access, analyze and manipulate Web data.

In order to provide such a framework, there are several challenges to be tackled.

Access to and representation of complex data from spreadsheets. One of the key benefits that spreadsheets bring to data management is the flexibility in terms of data formatting [23]. Spreadsheets do not impose many constraints regarding the data layout: data can be

organized based on criteria such as subjective importance, e.g., by placing important data on the top-left corner, or related pieces of data next to each other. Furthermore, the spreadsheet data model can be considered unstructured and supports only simple data types such as string, integer, etc. On the contrary, data accessible from Web data services are complex data such as JSON, RSS, etc. The challenge here is to bridge this data representation mismatch.

Synchronization between spreadsheet data and Web data. Spreadsheets provide an incremental approach for building fairly complex applications with immediate feedback (continuous evaluation) provided at each step of the application development process [17]. More precisely, spreadsheet cell formulas may contain references to other cells. When a referred cell is manipulated, the values of all referring cells are evaluated and shown to users immediately. Spreadsheet users are accustomed to this kind of behavior, and therefore the challenge here is to identify how to provide immediate feedbacks to spreadsheet users when Web data are manipulated, as well as to manage updates on the Web data when users manipulate them on the spreadsheets.

Reuse-driven of spreadsheet-based Web mashups. As mentioned earlier, although existing mashup tools have produced promising results that are certainly useful, they are primarily targeted at professional programmers. We argue that it is important to provide technique and automated support that would shift the efforts of developing mashups from scratch to that of reuse and customization. This will simplify mashup development tasks and increase user productivity.

Easy manipulation of complex data. Spreadsheets typically offer users with direct data manipulations such as edit/delete/copy/move cell contents. Manipulations of complex data instead are usually specified by queries in a SQL-like language, which may not be intuitive to spreadsheet users. We argue that it is important to provide support for organizing and manipulating complex data through concepts familiar to spreadsheet users, as well as maintaining the same simplicity and cleanliness of the paradigm, to the possible extent.

To address the above challenges, we have developed a framework for accessing, visualizing and manipulating Web data within spreadsheets, and implemented it specifically for MS Excel. We chose MS Excel because it is the most widely used spreadsheet product. However, we remark that the concepts presented in this paper are generic and can be applied to other spreadsheet applications. Our framework offers the following contributions.

- To support access to Web data, we interpose a data model, a variant of the Entity-Relationship (ER) model, between the spreadsheet and heterogeneous Web data services. This intermediate data model enables uniform data format and access interface to data services, hidden behind the intermediate layer. We then extend the spreadsheet data model such that entities become first class values of spreadsheet cells [25]. A formula language is also proposed to select and manipulate structured data defined by the ER-based model. This language is built on top of the standard spreadsheet formula language.

- To support the synchronization between spreadsheet data and Web data, we propose a new component model that enables the superimposition of spreadsheet data views over ER-based data views. The proposed component model consists of a data view, presentation, and interaction components. Data view components allow access to and construct data views over the ER-based model. Presentation components display the contents of data view components in the tabular grid of the spreadsheet and manage user interactions on the spreadsheet. Data view and presentation components expose a set of operations that allows other components to query and modify their internal information, as well as a set of events that notifies other components that some changes occur. Interaction components consist of a set of synchronization rules that translate events generated by a component onto operations of other components.

- To increase simplicity and productivity of mashup developments, we propose the notion of spreadsheet mashup patterns. Each pattern provides necessary functionality for developing Web mashups including accessing data from Web data services, presenting complex data on the spreadsheets, and handling the synchronization between spreadsheet data and complex data. We envision that although concrete data accesses and presentations are application-specific, in many cases it is possible to capture in a generic way the types of data access among data services, and presentations among spreadsheet applications. This allows users to reuse and customize spreadsheet mashup patterns instead of developing such mashups from scratch.

- To support simple manipulations on complex data, we envision an approach that allows spreadsheet users to perform drag-and-drop operations. These operations are then translated into queries over structured data. This is the key to enable exploration and understanding of the complex data, and will move the manipulations of complex data from writing SQL-like queries to concepts which are familiar by spreadsheet users.

To demonstrate the value of our approach, we have developed a prototype, called SpreadATOR [25], for a sales opportunity identification scenario, in which data are aggregated and combined from three different data sources: Nasdaq RSS service, Google RSS News service, and a CRM system. The prototype can be extended to other data sources. With our proposed framework and implementation, it is possible to access a variety of Web data sources, represent them on the spreadsheet, and manipulate data (including imposing changes on the source, if needed) from the "comfort" of the spreadsheet and with analogous flexibility and simplicity.

In the next section, we use a running example to describe some requirements in mashup developments. Then we discuss mechanisms for accessing complex data from spreadsheets (Section 3), followed by the proposed component model (Section 4). We present the proposed set of spreadsheet-based mashup patterns in Section 5. We describe our development tool in Section 6. Finally, we discuss related work and conclude in Section 7 and Section 8, respectively.

2. RUNNING EXAMPLE

To illustrate the approach, we use a scenario of a salesperson who wants to identify opportunities for selling software products. To do so, she monitors stock markets, looking for companies with the largest gains in their stock prices. A strong rise of a company's stock is often a sign that a significant event just happened in the company, and any such event may be an opportunity for selling the software. For instance, a sharp increase in the stock price may be a consequence of new plans to expand the business, or of the company becoming the target of an acquisition. Since expansions and mergers often result in IT projects which might rip and replace existing softwares, the salesperson could have opportunities to sell software if she reacts quickly to this event.

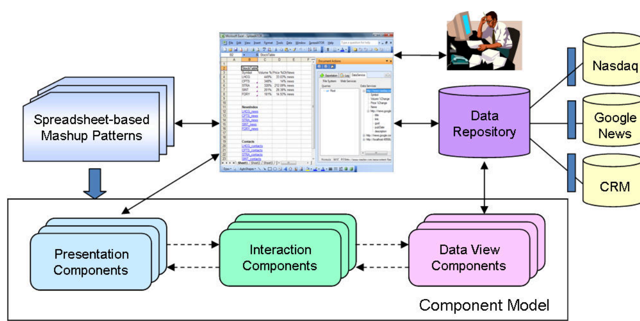


Figure 1: Reference scenario: development of Web data mashups for a sales opportunity identification

In this scenario, the salesperson accesses five stocks with biggest gains from Nasdaq.com. She then wants to get more information about each stock's company in the list. In particular, she would like to read news related to the company, as well as to know if these companies have been contacted by her own company before. She decides to use our tool to create a mashup that aggregates stock list from Nasdaq Stock information service, news related to each stock from Google news service, and contact details and purchase histories of each stock's company from her corporate's CRM system. Figure 1 depicts this scenario, in which the salesperson connects Nasdaq, Google News and CRM services into the tool, selects and customizes spreadsheet-based mashup patterns to build component models that capture necessary mashup functionalities. The desired outcome is the mashup shown in Figure 3. In the following, we describe what the salesperson expects our tool to simplify her mashup development tasks.

First, since the Nasdaq, Google News and CRM services use different data access methods (e.g., HTTP or SQL queries) as well as data representations (e.g., XML or JSON), ideally a tool would need to hide this heterogeneity from the user. Second, the data obtained from the CRM system may contain thousands of records, hence a tool would need to cater for both simple query specification (e.g., to filter unwanted data), and presentation (e.g., to display large set of data on the spreadsheet). Third, the user may want to present stock data and news data in different ways. For example, she may want to display stock data using a typical tabular presentation with each stock as a row and its attributes (e.g., price change, volume traded, etc.) as columns. News about the company should instead be presented as a list of hyperlinks which allows the user to quickly access to news data related to a given company that she is interested in. As an example, Figure 3 shows the application that the salesperson would like to have at her disposal. By clicking on an index in cell B12 of Sheet1, the salesperson is shown with a collection of news related to a symbol RATE in Sheet2. A tool therefore would need to provide different methods, which should be commonly used by spreadsheet users, for laying out data on the spreadsheet. Fourth, in some situations, the salesperson may want to manipulate contact details on the spreadsheet, hence a tool needs to support simple data manipulations, which should also preserve the spreadsheet metaphor. The tool should also be able to push the contact details back to the CRM system after manipulations. Finally, stock data and news are frequently updated, so a tool needs to provide users with ability to browse up-to-date information on the spreadsheet.

In the remainder of the paper we discuss the model and tool that allow users to build and interact with these kinds of spreadsheet-based Web data mashups.

3. ACCESSING COMPLEX DATA

This section discusses how, from a spreadsheet, we can access heterogeneous Web data sources and construct data views over these data that hide the heterogeneity from spreadsheets users, in a manner that is as simple and usable as possible. We then present how we bridge the link between these data views and the spreadsheet world by way of a formula language. Since the formula language was presented in an earlier work [25], we limit ourselves to a short description for the self-containment of this paper.

3.1 Constructing views over Web data services

Uniform data access. To deal with the heterogeneity of data models and data access methods of services, we leverage the data service technology [10, 11], a recent advent of SOA for exposing data as services. We particularly leverage the Web data service framework which is integrated as part of the ADO.Net 3.5 [11]. It offers a variant of the ER model to describe the structure of the underlying data sources. Specifically, when accessing data from non-data service sources, additional adapters are required to map data formats, access and manipulation operations between SpreadATOR data services and underlying data sources.

In mashup creations, users typically want to create complex data views, which may involve "joining" data from multiple data services. For example, in our sales opportunity scenario, the user may wish to display news for each company, which requires joining stock data with their corresponding news through a relationship (called AppearedIn). However this relationship does not exist, the user needs to provide it to our system. We refer to this kind of relationship as a user-defined relationship. This relationship is then used by our system to collect only a set of news entities that satisfies the AppearedIn relationship. Specifically, these news entities are obtained as a result of a semi-join between News and Stock entity types. Figure 2 shows a subset of the data schema for our reference scenario. It consists of two entity types: NasdaqStock, GoogleNews. Each entity type has a particular set of attributes. The reference attributes whose values are associations to entities of another entity type are denoted by "*". As an example, the NasdaqStock entity type has an association attribute GoogleNews, whose value is a reference to an instance of type GoogleNews. All other attributes are atomic.

NasdaqStock(Symbol, Volume, Price, News, *GoogleNews)
GoogleNews(title, link, guid, category, pubDate, description)

Figure 2: A subset of schema for the reference scenario

Constructing data views. The *Service Browser*, illustrated in Figure 3, greatly simplifies the task of constructing data views for spreadsheet users. This is achieved by presenting to users the entities that are accessible from data services using a tree representation. The construction of a data view begins when a user adds the URL of a Web data service in the tool. If the corresponding service is accessible, the service browser displays its ER-based schema using a series of trees in the following manner: each entity is represented as a distinct tree, the attributes of a given entity are represented as leaves in that tree and related entities are presented as children nodes. These children nodes may further be expanded to display related entity attributes and their own related entities in a recursive manner. The result is a set of trees where each represents a possible path through the ER-based schema starting from each of the entities of the schema. Figure 3 partially shows a tree representing the schema in Figure 2. The root node represents Nasdaq-

Stock entity, which has three leaves representing attributes Symbol, Volume and Price. The related entity GoogleNews is represented as an expandable child node which, when expanded, presents the attributes and related entities of GoogleNews. By navigating to the node representing GoogleNews, users can construct a data view that contains only news related to stock data. Specifically, this user interaction is translated to *join* operation over the ER-based schema.

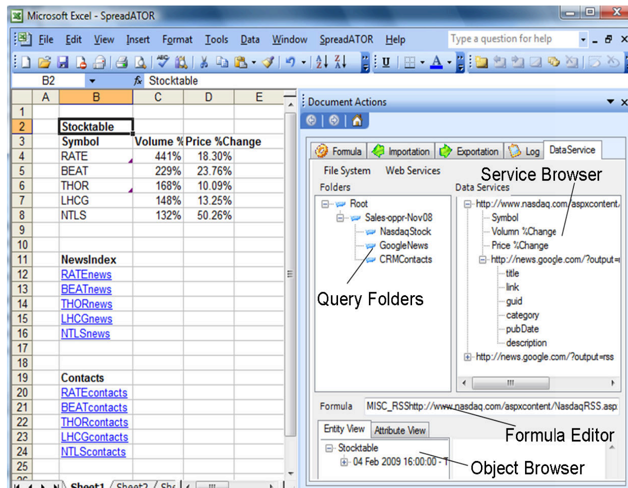


Figure 3: Spreadsheet-based mashup for the reference scenario

We also provide a series of graphical primitives to allow users to create more sophisticated queries in a form called *Preview*, as shown later in Figure 8. By dragging a URL node from the service browser onto a cell in the spreadsheet, this Preview form will be shown to the user. From the Preview, users can perform the following operations: (i) *Projection*: users can select attributes of interest by ticking corresponding check boxes, (ii) *Filter*: users can limit instances of an entity to be retrieved to the subset that matches a given filter predicate. The maximum number of instances displayed in a data view can also be specified, and (iii) *Sort*: users can order instances in a data view according to their attributes in ascending or descending order. The result of all these operation is immediately showed to the users (hence the name *preview*) who can refine a query until they are satisfied.

Once data views have been constructed, we allow users to store them in the *Query folders*. With the query folder, users can create new virtual folders, populate these folders with data views (containing sets of objects obtained from external Web services). This enables users to flexibly manage constructed views using their familiar concepts of file systems. Consider the salesperson in our reference scenario. She can create a folder named SalesOppr-Nov08 (as shown in Figure 3) to store all data views constructed for the reference scenario as files in a single folder. When select a file in the folder, the user can browse the data view contents represented in a tree structure in the *Object Browser*, as shown on the bottom right of Figure 3. The user can also simply drag a file in a folder to a cell in the spreadsheet. This action enables the user to bring complex data contained in a data view into spreadsheet cell. In the following subsection, we describe how our tool support complex data in spreadsheets.

3.2 Supporting complex data in spreadsheets

In the previous subsection, we described how users can graphically construct data views over a given ER-based schema. Once data views have been constructed, they need to be displayed on the tabular grid of spreadsheets. However, a major challenge here is the difference in the representations between data contained in the data views (i.e., complex data as described in Section 3.1) and that supported by the spreadsheet (i.e., simple data of types string, integer, etc.). To bridge this mismatch, we extend the spreadsheet data model so that cells can contain complex data. The details of our model and formula language are presented in [25]. In this paper, we only summarize the key points using examples.

Like in any spreadsheets, we refer to a cell by its column and row coordinates. For example, cell B2 refers to a cell located at column B and row 2. Each cell has a formula which is evaluated into an atomic typed value such as integer, float, string, datetime, and displayed to the user. A cell may contain a reference as a hyperlink to another cell in the same or different worksheet. We extend standard spreadsheet formula language such that a formula can be expressed by one of the following:

- B2 = http://www.nasdaq.com/...: defines contents of cell B2 as a URL of the data service from which complex data is retrieved from. We refer to cells containing this kind of formula as *container cells*. A container cell holds complex data, as a set of objects which are instances of a particular entity in the ER-based model.
- B4 = <<1.B2>>.[0]/_symbol: defines contents of cell B4 based on contents of cell B2. The formula in cell B4 contains a value selection expression, which particularly returns the value of attribute symbol in the first object in the set. Similarly the name of an attribute can be obtained by a formula like <<1.B2>>.[0]/#symbol. We refer to cells containing value selection expressions as *presentation cells* since they are used to present contents of complex data stored in a container cell.

We would like to note that formulas, specified in our formula language, are maintained in a separate context, called the *external mapping* definition, which leaves untouched the standard spreadsheet formula language and overall behavior of the hosted spreadsheet application. Specifically, the set of objects, held by a container cell, is handled by our system; for spreadsheet (MS Excel in particular) a cell simply contains a user-defined label as shown in Figure 3. The advantages of this formula language are threefold: (i) complex data now become first class values of cells, and their contents can be laid down on the tabular grid of spreadsheets, (ii) as our system maintain complex data in a separate context, we maintain the simplicity of spreadsheet paradigm, and (iii) the synchronization between spreadsheet data and complex data is possible since formulas maintain correspondences between them.

4. SUPERIMPOSITION OF SPREADSHEET VIEWS OVER DATA VIEWS

We propose a new component model that is designed to manage the synchronization between complex data contained in data views (described in Section 3.1) and spreadsheet data (described in Section 3.2). The design of this component model comes from our observation that there are always some elementary features required for implementing any data mashups: data have to be retrieved from data services, a representation suitable for spreadsheet display has to be built and interactions of the user with the spreadsheet environment may need to be translated to operations on the underlying data and vice versa. Our proposed component model therefore consists of three elements: data view, presentation, and interaction modules (also called tool components or simply components hereafter). It is

somewhat analogous to the Model-View-Controller design pattern (MVC), which has proved effective for building interactive applications. The *data view component* is responsible to retrieve data and cache a view of these data (it would correspond to the Model in traditional MVC). The *presentation component* is responsible for presenting data on the tabular form of spreadsheets (analogous to the View in MVC). The *interaction component* is responsible for synchronizing the data views and spreadsheet presentations (analogous to the Controller in MVC).

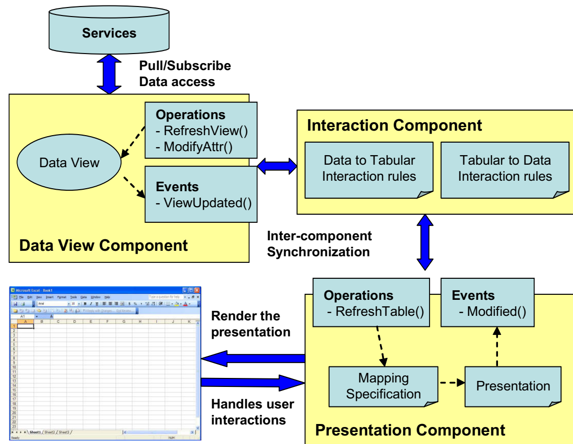


Figure 4: Component Model

We identify the following abstractions for each of the components. Data view and presentation components have the notion of states such that when changes occur, they can notify other components to update their states accordingly. The states of data view components corresponds to their data views used to cache data accessed from external services, while the states of presentation components corresponds to the presentations of data on the spreadsheet. Data view and presentation components expose events to notify their state changes to other components, as well as operations that act as state change requests. Interaction components consist of a set of rules that are used to handle the synchronization between data view and presentation components. We detail each of these components below.

4.1 Data view components

Data view components allow accessing data from external data sources accessible through Web data services (see Section 3.1). We provide two types of data view components: push component and pull component. They capture two data access patterns commonly found on the Web, i.e., request-response and publish-subscribe. We enable these two data access methods through the use of the Jabber framework [4]. Both push and pull components, once obtained data, will store the data in a data view. The contents of a data view is a set of objects corresponding to entities of the ER-based model.

Data view components expose a set of operations and events that allows them to interact with other components of the model.

Operations. Operations allow other components to query and modify the contents of a data view component. Table 1 shows a set of operations that are common to any data view components. This set of operations is classified into: *data access operations* that allow querying the contents of a data view component, *update operations* that allow modifications of objects in the data view, e.g., adding or removing their attributes.

Events. Events allow a data view component to notify other components of updates in its contents. Specifically, when the contents of a data view component is updated, it sends `D_ViewUpdated` event to notify other components. This event passes complex data contained in the data view as its parameter.

Data Access Operations
dv:getObjects() returns a set of objects in a data view
dv:getObject(o_j) returns a particular object
dv:getAttrName(o_j, a_k) returns the name of an attribute
dv:getAttrValue(o_j, a_k) return the value of an attribute
Update Operations
dv:modifyValue(a_k,old,new) changes the value of an attribute
dv:insertAttr(a_k, N) adds a new attribute a_k with value N
dv:deleteAttr(a_k) deletes an attribute from the data view
dv:insertObj(o_j, N) adds an object o_j with new value to the data view
dv:deleteObj(o_j) removes an object from the data view
dv:refreshView() replace the data view with a new set of objects
dv:dropView() drops the data view
dv:sortBy(a_k, order) sorts the current set of objects in the data view by attribute a_k according to the order condition, which can be ascending or descending
dv:filter(pred) conditionally selects a subset of the current set of objects in the data view according to condition pred

Table 1: The list of operations of data view components

4.2 Presentation components

Presentation components allow displaying data in the tabular grid of spreadsheets. Each presentation component embeds a *presentation specification*, which describes how the contents of a data view component is mapped to a tabular display. The presentation specification itself is built by composing lower level presentation abstractions that model the organization of data on the spreadsheet.

Presentation specification. The presentation of data on the spreadsheet needs to adhere to the tabular data model that has made spreadsheets so popular (described in Section 3.2). Structured data may be represented on the spreadsheet in a variety of ways. Some examples of presentations will be discussed in Section 5. To allow constructing these presentations, we introduce hereafter a compositional framework which allows to specify mappings from structured data to clusters of cells on the spreadsheet.

The spreadsheet presentation is modeled with the following constructs:

- ATTRIBUTE specifies a cell that contains an attribute name.
- VALUE specifies a cell containing an attribute value.
- RECORD specifies a range of cells that displays contents of an object.
- SET specifies a range of cells that presents a collection of objects.
- SHEET specifies a worksheet.

Figure 5 shows an example of the presentation specification where attribute names and values of entities (originating from a data view component) are displayed. It consists of a container cell (shown by a user-defined label) and an expandable number of rows. The first row consists of a collection of cells presenting attribute names (also called ATTRIBUTEs), while other rows present their corresponding values (also called VALUEs). The rows displaying attribute values are called RECORDs.

Internally, presentations are specified relatively to the coordinate of a container cell, i.e., the top-left cell. All other cells' contents are computed from this container cell by iterating through the data objects presented in the data view and through attributes of these

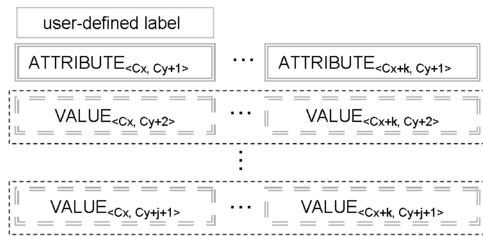


Figure 5: Presentation Specification

objects. Iterations are figured on the graphical representation of Figure 5 by ellipsis.

These constructs are used as building blocks that can be composed to build tabular presentations similar to the concept of report building systems such as ASP.NET [2], which supports generation of web pages by binding data with basic web-page components. Following a similar approach, we need to bind these presentation constructs to data views. This is achieved by the Data-to-Tabular Mappings.

Data-to-Tabular Mappings. The objective of Data-to-Tabular Mappings (DTM) is to bind contents of a data view component to the presentation specification. This is achieved using the formula language discussed in Section 3.2, as described in the following.

$$\langle C_x, C_y \rangle = \text{dv:getObjects()} \quad \text{----------} \; (1)$$

$$\langle C_x{+}k, C_y{+}1 \rangle = \text{dv:getAttrName}(o_1, a_k) \;;\; 1 \le k \le \text{dv:countAttrs}(o_1) \quad \text{----------} \; (2)$$

$$\langle C_x{+}k, C_y{+}1{+}j \rangle = \text{dv:getAttrValue}(o_j, a_k) \;;\; \left.\begin{array}{l} 1 \le j \le \text{dv:countObjs}(\langle C_x, C_y \rangle) \\ 1 \le k \le \text{dv:countAttrs}(o_j) \end{array}\right\} \text{-----} \; (3)$$

Figure 6: DTM for the Table presentation component

A DTM for the presentation in Figure 5 is shown in Figure 6. The mapping states that the content of cell $\langle C_x, C_y \rangle$ is derived from a set of objects in a data view component (obtained by operation `dv:getObjects`). Hence, cell $\langle C_x, C_y \rangle$ will act as a container cell because its content is associated with complex data (here, a list of complex objects). The mapping also specifies that the attribute names are displayed on the row located below the container cell. This is achieved by iteration over the attributes of the objects referenced in the container cell (and maintained by the data view component). All objects are assumed to have the same set of attributes, i.e., they are different instances of the same entity. When executed, each iteration (indexed by k) populates a cell in column $C_x + k$ with the corresponding attribute name. Finally, the mapping also iterates through the list of objects referenced in the container cell. When executed, each iteration step (indexed by j) produces a row in the presentation for displaying attribute values of the object corresponding to this step. Within a row, the cells are populated by iterating over the object attributes.

Note that the data-to-tabular mappings are generic, i.e., they are not tied to a specific application, and they are able to map the contents of any data view components onto the presentation. The application specific details are provided by the user as customization parameters and these are the only inputs required from users. These parameters include a *content* parameter and a *spatial* parameter. The content parameter specifies the data view whose contents needs to be displayed on the spreadsheet. The spatial parameter corresponds to a coordinate in which the presentation begins. These parameters are used to generate a set of mapping formulas specified in the formula language (discussed in Section 3.2), which we will explain in more details later in Section 5.2.

Operations. A presentation component exposes a set of operations (see Table 2) that allows interfacing the presentation component with the spreadsheet environment and allows its synchronization with other components. Based on the mappings presented above, spreadsheet presentation can be adjusted according to user manipulations.

The behaviors of operations on *presentation cells* depend on their cells' types, e.g., ATTRIBUTE, VALUE, RECORD. In particular, when the user performs a manipulation, the system checks type of the selected presentation cell and invokes a corresponding operation. For example, assume that cell B5, in Figure 3, is being deleted. The system checks its cell type, i.e., VALUE cell, thus invokes the operation `deleteVALUE` accordingly. Now assume that cell B3 is being deleted. In this case, since cell B3 is an ATTRIBUTE cell, displaying the name of an attribute symbol, the operation `deleteATTR` is invoked. This operation deletes the contents of cell B3 itself and all the VALUE cells displaying values of the attribute symbol (cells B4:B8), as well as removes all external mappings of cells B3:B8.

Users have access to operations on *container cells* by a context menus obtained by a right click on the container cell. This context menu allows them to modify the display of the complex data referenced in the container cell. For example, the `Refresh` operation triggers a new query on the data source and updates the spreadsheet presentation with the latest values; the `Selection` operation allows users to restrict the set of objects displayed by applying a filter; the `SortBy` operation lets the user order a set of objects according to some criteria.

Events. A presentation component exposes a set of events to which other components may subscribe to obtain notifications of changes in the presentation state. This is useful when other components need to react to user manipulations performed in the spreadsheet environment. For example, the presentation component in Figure 3 will fire a "`P_VALUEchanged`" event when the user edits contents of a VALUE cell B4. As shown in Figure 4, our component model is only concerned with component-defined events, not native events defined by the spreadsheet applications. Figure 10 illustrates the distinction between component-defined events and native spreadsheet events. Essentially, user actions on the spreadsheet (e.g., edit content of cell B4) trigger native spreadsheet events. Presentation components intercept the native spreadsheet events, check type of the cell being manipulated (i.e., VALUE), and process them internally (by calling operation `modifyVALUE`), and trigger component-defined events (`P_VALUEchanged`) to signal other components of their states change. A set of events defined for the Table presentation component is shown in Table 2.

4.3 Interaction Components

The role of interaction components is to synchronize data view and presentation components. They respond to events from presentation components (resp. data view components) by invoking operations on data view components (resp. presentation components) following specifications expressed in *interaction rules*. Essentially, an interaction rule establishes a publish/subscribe relationship between data view and presentation components in terms of event publisher, event type, event subscriber and an operation of the subscribing component. When event parameters and operation parameters are not compatible, interaction components may contain additional data transformation logic.

Depending on their directions, interaction rules can be classified into presentation-data and data-presentation interaction rules.

Presentation-data interaction rules define how to map events, generated by a presentation component, onto operations of data view components. An example of presentation-data interaction rule

Operations on presentation cells
ui:modifyVALUE(V_j,old,new) changes content of a VALUE cell from old to new
ui:deleteVALUE(V_j) deletes contents of a VALUE cell
ui:deleteATTR(A_j) removes a set of ATTRIBUTE and VALUE cells
ui:renameATTR(A_j,old,new) changes content of ATTRIBUTE cell from old to new
ui:insertATTR(A_j,N) adds a set of ATTRIBUTE and VALUE cells
ui:deleteREC(R_i) removes a range of cells referred by a RECORD
Operations on container cells
ui:SortBy(A_j,order) updates a presentation for a set of objects sorted by attribute A_j in ascending or descending order
ui:Selection(pred) updates a presentation with a set of objects satisfying pred
ui:Refresh() updates a presentation with a new set of objects from external service
ui:delete(C) deletes a container cell and all its depending presentation cells
Events
P_VALUEchanged(V_j,old,new) notifies that a VALUE cell is modified
P_VALUEdeleted(V_j) notifies that a VALUE cell is deleted
P_ATTRinserted(A_j,N) notifies that a set of ATTRIBUTE and VALUE cells is added
P_ATTRdeleted(A_j) notifies that a set of ATTRIBUTE and VALUE cells is deleted
P_ATTRrenamed(A_j,old,new) notifies that content of ATTRIBUTE cell is modified
P_RECdeleted(R_i) notifies that a RECORD is removed
P_sorted(order) notifies that a set of RECORDs is reordered
P_selected(pred) notifies that RECORDs are selected based on condition pred
P_refreshed() notifies that the presentation is replaced with a new set of objects
P_dropped() notifies that the presentation is dropped

Table 2: Interface of the Table presentation component

specifies interactions between presentation component StockTable and data view component StockDataView in our reference scenario is shown below (interfaces of these components are omitted here for the sake of clarity).

```
<interaction
  publisher="StockTable"     event="P_VALUEChanged"
  subscriber="StockDataView" operation="dv:modifyValue"/>
```

This rule maps the `P_VALUEChanged` event from StockTable component onto the operation `dv:modifyValue` of StockDataView component. Consider an example when the user modifies contents of cell B4. The presentation component StockTable captures this user manipulation and triggers an event `P_VALUEchanged`. Upon receiving this event, the interaction component dispatches the event parameters and passes them to the operation `modifyValue` of the StockDataView component. This may involve an additional data transformation (see Section 5.2).

Data-presentation interaction rules define how to map state change events, generated by a data view component, onto operations (i.e., state change requests) of presentation components. An example of data-presentation interaction rule that facilitates the interactions between StockDataView and StockTable components is shown below. It maps the `D_ViewUpdated` event from StockDataView component onto the `Refresh` operation of StockTable component.

```
<interaction
  publisher="StockDataView"  event="D_ViewUpdated"
  subscriber="StockTable"    operation="ui:Refresh"/>
```

5. PATTERNS FOR SPREADSHEET-BASED WEB MASHUPS DEVELOPMENT

Developing a complete mashup application using the component model we have presented is not trivial if users have to specify interaction rules and build presentation from scratch. However, this architecture can be brought to users in a convenient way through pre-defined and reusable patterns that capture common methods of data access and their possible tabular presentations as found in spreadsheet applications. We now present a set of common tabular presentations frequently found in spreadsheet applications, and then discuss how we capture these presentations in reusable patterns for developing spreadsheet-based Web data mashups.

5.1 Common tabular presentations

In [18] we have analyzed a collection of spreadsheet applications. We found that there are a set of presentations that are commonly used to display data on spreadsheets as shown below.
1. Content presentation. This presentation displays an *object* of a data view (e.g., a stock RATE) as a range of two columns on a spreadsheet. The first column presents attribute names of the object, while the second column displays its attribute values (shown in Figure 7(a)).
2. Repeater presentation. This presentation presents a *set* of objects (e.g., a set of stocks) by repeating the presentation of the content presentation (in Figure 7(a)) in either vertical or horizontal direction (shown in Figure 7(b)).
3. Table presentation. This is the most popular presentation that we have found, and presents a *set* of objects as a table which each row presents an object and each column presents an attribute's value. The top-most row presents attribute names (shown in Figure 7(c)).
4. Index presentation. This presentation displays a set of objects as *hyperlinks* that allow navigation to their presentations displayed in separate worksheets (Figure 7(d)). The presentation of each object is generated using the content presentation in Figure 7(a).
5. Relationship-Index presentation. This presentation presents a set of objects as *hyperlinks* that allow navigation to their related sets of objects in separate worksheets (shown in Figure 7(e)). The presentation of each related set of objects are displayed using the presentation of either a repeater presentation (Figure 7(b)) or a table presentation (Figure 7(c)).
6. Hierarchical presentation. This presentation displays sets of objects related by relationships in a hierarchical structure. For example, for each stock symbol, its related set of contacts is displayed using the Table presentation (Figure 7(c)) as shown in Figure 7(f).

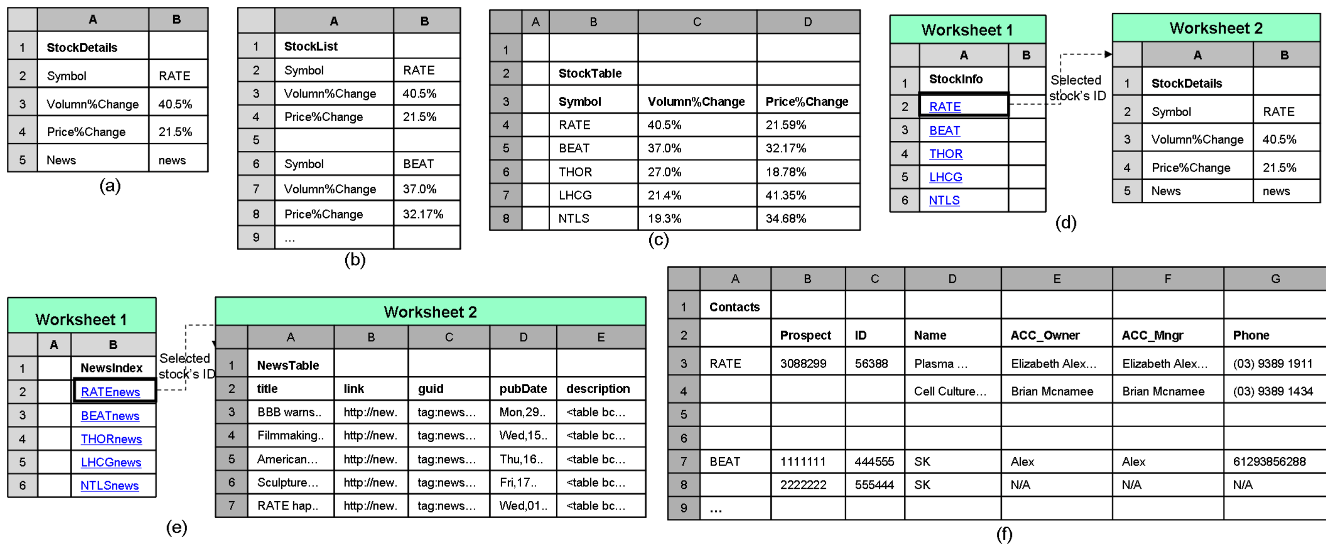
5.2 Spreadsheet-based Web mashup patterns

In this work, we capture the above set of common tabular presentations as presentation components that encapsulate reusable presentation logic for displaying data on tabular grid of spreadsheets. These presentation components, together with data view components and interaction components, are packaged as *Spreadsheet-based Mashup Patterns*, which allow users to create mashups where interfaces are tabular grids in spreadsheets.

Consider the *Table Mashup Pattern* that captures the way in which data are presented in tabular format as shown in Figure 7(c). This pattern is specified as a combination of the data view component in Section 4.1, the tabular presentation component in Section 4.2, and the interaction component in Section 4.3. It captures the default behavior (default presentation, data access method, and interaction feature) that needs to be customized to meet the specific needs of a user. The customization includes constructing a data view, and specifying a method for refreshing data (e.g., user-driven refresh, periodic pull, or push).

As an example, consider a user who selects this Table Mashup Pattern to import and present stock data. The user customizes the pattern to import top-five gain stocks from the Nasdaq service, and present values of attributes symbol, volume and price in the spreadsheet. With further customization, the user can enable three possible methods for refreshing data from Nasdaq. Depending on the user's option, three possible scenarios can be generated from this pattern. These scenarios share the same presentation component but differ in their data view components used to access data and interaction components used to synchronize spreadsheet data and structured data views.

Now we discuss how mashups can be built from spreadsheet-based mashup patterns. To conserve space, we only illustrate a



(a)

	A	B
1	StockDetails	
2	Symbol	RATE
3	Volumn%Change	40.5%
4	Price%Change	21.5%
5	News	news

(b)

	A	B
1	StockList	
2	Symbol	RATE
3	Volumn%Change	40.5%
4	Price%Change	21.5%
5		
6	Symbol	BEAT
7	Volumn%Change	37.0%
8	Price%Change	32.17%
9	...	

(c)

	A	B	C	D
1				
2		StockTable		
3		Symbol	Volumn%Change	Price%Change
4		RATE	40.5%	21.59%
5		BEAT	37.0%	32.17%
6		THOR	27.0%	18.78%
7		LHCG	21.4%	41.35%
8		NTLS	19.3%	34.68%

(d)

Worksheet 1

	A	B
1	StockInfo	
2	RATE	
3	BEAT	
4	THOR	
5	LHCG	
6	NTLS	

Selected stock's ID

Worksheet 2

	A	B
1	StockDetails	
2	Symbol	RATE
3	Volumn%Change	40.5%
4	Price%Change	21.5%
5	News	news

(e)

Worksheet 1

	A	B
1		NewsIndex
2		RATEnews
3		BEATnews
4		THORnews
5		LHCGnews
6		NTLSnews

Selected stock's ID

Worksheet 2

	A	B	C	D	E
1	NewsTable				
2	title	link	guid	pubDate	description
3	BBB warns..	http://new.	tag:news…	Mon,29..	<table bc…
4	Filmmaking..	http://new.	tag:news…	Wed,15..	<table bc…
5	American…	http://new.	tag:news…	Thu,16..	<table bc…
6	Sculpture…	http://new.	tag:news…	Fri,17..	<table bc…
7	RATE hap..	http://new.	tag:news…	Wed,01..	<table bc…

(f)

	A	B	C	D	E	F	G
1	Contacts						
2		Prospect	ID	Name	ACC_Owner	ACC_Mngr	Phone
3	RATE	3088299	56388	Plasma …	Elizabeth Alex…	Elizabeth Alex…	(03) 9389 1911
4				Cell Culture…	Brian Mcnamee	Brian Mcnamee	(03) 9389 1434
5							
6							
7	BEAT	1111111	444555	SK	Alex	Alex	61293856288
8		2222222	555444	SK	N/A	N/A	N/A
9	…						

Figure 7: Some examples of data presentations in spreadsheets

single task of creating a table presentation of stock data as shown in Figure 3. Table Mashup Pattern requires parameters to instantiate its DTM definition as discussed in Section 4.2. We provide a graphical tool, called *Preview*, to support this task (as shown in Figure 8). In the following, we describe how this *preview* can be used to instantiate the Table Mashup Pattern.

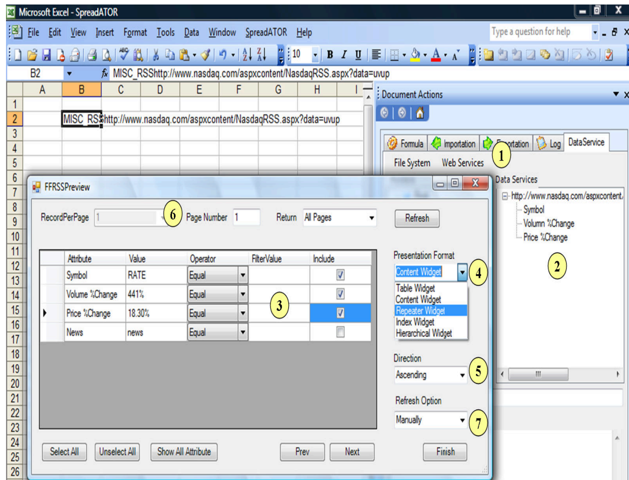


Figure 8: Constructing data views with selection and projection

1. *Configure a data view component.* The first task that the user needs to do when developing a mashup is to specify which data they wish to access. *Step 1* in Figure 8 shows the dialog used for configuring a data view component. The user simply needs to provide a URL corresponding to the data source to access. Once the URL has been provided, the user can browse the source schema and construct data views by using the *Service Browser* (*Step 2*) combined with the *preview* form. This form is displayed when the user selects the entity to import in the *service browser* and assign a container cell for that entity (simply using a drag-and-drop operation to move the URL above a cell). The *preview* form displays a tabular presentation of the entity used for specifying additional query parameters such as a selection or a projection (*Step 3*).

2. *Configure a presentation component.* The user can then choose among several tabular presentations for displaying the query result on the spreadsheet (as shown in *Step 4*). Tabular presentations are not only displaying values of the entity but also include functionalities for sorting and paging operations (as shown in *Step 5* and *Step 6*). The sorting operation allows the users to sort a set of objects in ascending or descending order, while the paging operation allows users to specify a maximum number of objects to be presented on the spreadsheet.

3. *Configure an interaction component.* Finally, the user can specify how the mashup should react to notifications from the data source (in *Step 7*). By default, the presentation is not updated automatically, the user needs to manually request for a refresh.

	A	B	C	D
1				
2		http://www.nasdaq.com..		
3		<<1.B2>>.[0]/#symbol	<<1.B2>>.[0]/#Volumn%Change	<<1.B2>>.[0]/#Price%Change
4		<<1.B2>>.[0]/_symbol	<<1.B2>>.[0]/_ Volumn%Change	<<1.B2>>.[0]/_ Price%Change
5		<<1.B2>>.[1]/_symbol	<<1.B2>>.[1]/_ Volumn%Change	<<1.B2>>.[1]/_ Price%Change
6		<<1.B2>>.[2]/_symbol	<<1.B2>>.[2]/_ Volumn%Change	<<1.B2>>.[2]/_ Price%Change
7		<<1.B2>>.[3]/_symbol	<<1.B2>>.[3]/_ Volumn%Change	<<1.B2>>.[3]/_ Price%Change
8		<<1.B2>>.[4]/_symbol	<<1.B2>>.[4]/_ Volumn%Change	<<1.B2>>.[4]/_ Price%Change

Figure 9: Generated mapping formulas

Once the user finished pattern configuration in the *preview* form, a data view component (StockDataView), a presentation component (StockTable) and an interaction component are constructed. To generate the presentation of stock information, the StockTable component uses the configuration parameters to generate a set of mapping formulas, as shown in Figure 9, from the DTM definition (in Figure 6). These mappings are used to bind contents of the StockDataView component to the presentation specification in Figure 5. As a result, a table presentation is displayed on the spreadsheet starting from the cell where the URL is dropped on (cell B2) as shown in Figure 7(c).

After the presentation has been generated, it can be modified by user manipulations on the spreadsheet or updates from the data source. Consider the example in Section 4.3 when the user changes the contents of cell B4 from "RATE" to "RATECorp". Figure 10

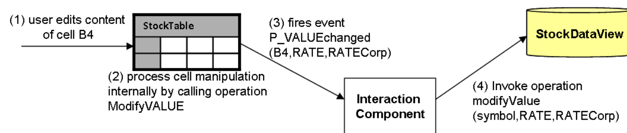


Figure 10: Interaction handling

provides a simple illustration of what happens at the runtime based on the *presentation-data* interaction rule specified in Section 4.3:

- *Capturing event from the presentation component* involves: (1) the spreadsheet application fires a native event upon the modification of cell B4, (2) the component StockTable captures this native spreadsheet event, calls its internal operation `ModifyValue` and triggers an event `P_VALUEchanged`, and (3) the interaction component captures this event.
- *Invoking operations of subscribing components* involves: (4) the interaction component locates components and operations matching this event based on the interaction rules. For each interaction, the interaction component transforms event parameters to corresponding operation parameters, and then invokes the specified operation on the subscribing component. In our example, the interaction component transforms the event parameter B4 into an attribute name "symbol", while the other parameters are the same. Then the interaction component invokes the operation `modifyValue` of the component StockDataView and passes to this operation a set of parameters "symbol", "RATE", and "RATECorp".

The above example illustrates how our framework shifts the efforts of developing mashups from scratch to that of customization. Users can create mashups simply by instantiating spreadsheet-based mashup patterns. Tasks such as building tabular presentations as well as coordinating data view and presentation components are delegated to the system, thereby simplifying mashup development.

5.3 Drag-and-drop re-organization

Spreadsheets are typically used as a tool to visualize, manipulate and analyze data. When introducing complex data into spreadsheet data model, we need to provide data manipulation operators for such complex data. We follow the spreadsheet paradigm and allow users to manipulate their data using drag-and-drop fashion. In Section 4.2, we have pointed out that users can invoke some operations on container cells by using context menus. We also allow users to do direct cell manipulations on presentation cells such as copy/move/delete/edit. Due to space limitations, we cannot discuss all of the operations in detail. Rather we summarize some of them, mainly by using examples.

Consider the salesperson who wants to create a new copy of stock data so that she can manipulate and perform different analysis on the data. In this case, she copies container cell B2 to cell G2. This operation works as if one were dragging a data view from a folder into a spreadsheet cell. That is, creating a new copy of stock data in cell G2. To create a presentation of this stock data, the user can use our formula language or simply selects and customizes one of our spreadsheet-based mashup patterns.

Now assume that there exist a data view PODataView containing hundreds of purchase orders, and a presentation POTable displaying these purchase orders in cells H1:K250. The salesperson may wish to copy only ten purchase orders into another location on the spreadsheet so that she can have a closer look at their details. In this case, assume that she copies a range of cells H20:K30 presenting purchase order data to cell M1. Copying presentation cells results in the creation of a new presentation component that also displays contents of the data view PODataView. The presentation of this new component consists of ten RECORDs, each RECORD consists of four ATTRIBUTEs and VALUEs displaying attribute names and values of purchase orders. This new presentation component needs to subscribe to the current interaction component managing synchronization between PODataView and POTable so that its presentation can be kept synchronized with the contents of PODataView component. In this case, when the contents of PODataView component is updated, e.g., by new notifications from the CRM service, the presentations of both POTable and the new presentation component are updated accordingly.

By these drag-and-drop manipulations, users do not need to go through all the process of constructing data views again when they want to apply some changes to data views. For example, when users want to remove attribute volume from the presentation of stock data, the user do so by simply deleting cell C3. Our tool offers various options for users to reorganize complex data displayed on the spreadsheet. Users can choose an option that is suitable for their jobs at hand.

6. IMPLEMENTATION

The implementation of our prototype consists of a backend server to execute mashups and a set of adapters to communicate with external data sources. The backend server is an extension of our previous work [25, 18]. It has been implemented as an add-in to MS Excel using C# and VSTO [21]. The information from the Excel sheet being manipulated by the user is captured by a VSTO program and sent to our system. The C# modules are compiled and run as backend server.

The backend server consists of a code generator. Given a mashup pattern, the code generator outputs a set of mappings necessary for binding data to Excel sheet, component definitions, and necessary code that models the component interactions. The backend server then executes the mappings to populate Excel sheet, instantiating the components, and executing the interaction code to coordinate the interactions among the components. The generated interaction code manages events subscriptions and operation invocations.

To communicate with non-data service sources, we have built a number of adapters. An adapter implements the mappings between the ADO.Net Web data service and the data sources. In our sales opportunity identification scenario, we have implemented adapters for two RSS services: Nasdaq, Google News; these adapters could be adopted easily to communicate with any RSS services. Since the data transformation code is already built, we need to only map their URLs into the data service standard.

7. RELATED WORK

Many commercial tools have been developed to help spreadsheet users to access external data sources. MS Excel, in particular, allows to import data from external sources using, e.g., XML Mapping tool [24], SQL importation [19], Web data importation [13], or Analysis Services [1]. However, these tools have several limitations. First, they do not maintain references to complex data, rather they convert complex data into Excel's supported types when presented on the spreadsheet. Second, they denormalize the nested structure that might exist in the imported data, e.g., converting hierarchical XML documents, into flat tables. Last but not least, the fact that a distinct environment is used to access each type of data sources hampers the integration of the various data sources, thus

it is not possible for authoring any form of composition between these different sources.

There exist some efforts for supporting homogeneous data access from different sources in spreadsheets through Web data services. For instance, Visual Studio Tool for Office (VSTO) allows developers to access data services from ADO.Net framework. Another example is to manipulate data services entities by using the macro language that accompanies spreadsheet environments. However, these technologies are intended for professional programmers with a solid background in object-oriented programming. Other approaches, offered simpler web services invocations, exist. For instance, StrikeIron [6] is a commercial add-in to MS Excel that allows users to invoke web services using drag-and-drop manipulations. This initiative does not consider complex objects obtained from web services as first class entities, rather they are translated as a collection of atomic values presented in a range of cells. It is not possible to refer to the complex objects in subsequent data visualizations or analysis, leave alone the relationship that may exist among the imported data. In addition, the correspondences among elements in the data service and cell contents of the spreadsheet are lost once imported, hence it is not possible to refresh the spreadsheet with new data or update the Web data when users manipulate them on spreadsheets.

Some research work have proposed models to capture data presentations in spreadsheets. In [15, 20], the authors have provided a powerful data model to represent a broad class of tables covered by both relations and spreadsheets. However, the specification of presentations needs to be done manually. [8, 14] proposes a specific table presentation as the purpose is to ensure the spreadsheet correctness. However this approach also provide support for the specification of presentations through the notion of template. Our approach is located in between these two approaches: it defines a small set of frequently used spreadsheet data presentations and provides support for their specifications.

The work presented in [9] automatically infers presentation templates (as proposed in [8, 14]) from existing spreadsheets. This work is especially useful when users want to export spreadsheet data to external sources. Our data presentation templates can be adopted by such approach.

In the area of Web application development, there are several work [12, 2] that provide data presentation patterns that are used to model web applications. These patterns can be adopted by our approach, however unlike these existing work, we follow spreadsheet paradigm and provide interactive approach for data importation and presentation with immediate results.

8. CONCLUSION AND FUTURE WORK

Skilled knowledge workers are moving towards creating mashups to fulfill their routine tasks. In this paper, we proposed a framework that allows users to easily build mashups within their familiar spreadsheet environment. As we have illustrated, while our tool makes complex data as first class spreadsheet cell values, we maintain the same simplicity of the paradigm. Users can therefore employ their familiar spreadsheet concepts to explore, manipulate, and analyze complex data.

In addition, with the proposed component model, our tool can be used to build fairly sophisticated mashups, involving joining data from multiple Web data services, keeping spreadsheet data up to date with Web data. We also simplify mashup development by proposing a set of spreadsheet-based mashup patterns. These patterns simplify mashup development tasks and increase users productivity by shifting the efforts of building mashups from scratch to that of reuse and customization.

We have used our tool to create prototypes of mashups that involve accessing data from different types of data sources, e.g., RSS feeds services, relational databases, and Flickr. Our experience demonstrated the superiority of our spreadsheet-based mashup tool compared to existing tools in terms of both simplicity and development productivity.

9. REFERENCES

[1] Designing Reports with the Microsoft Excel Add-in for SQL Server analysis services. Microsoft Corp., 2004.
[2] ASP.Net. http://asp.net/.
[3] Intel mash maker. http://mashmaker.intel.com.
[4] Jabber Framework. http://www.jabber.org.
[5] Microsoft popfly. http://www.popfly.ms.
[6] StrikeIron Web Services for Excel. http://www.strikeiron.com/tools/toolssoaexpress.aspx.
[7] Yahoo! pipe. http://pipes.yahoo.com/pipes.
[8] R. Abraham, I. Cooperstein, S. Kollmansberger, and M. Erwig. Automatic generation and maintenance of correct spreadsheets. *Proc. ICSE'05*, pages 136–145.
[9] R. Abraham and M. Erwig. Inferring templates from spreadsheets. In *Proc. ICSE '06*, pages 182–191.
[10] M. Carey. Data delivery in a service-oriented world: the bea Aqualogic data services platform. In *Proc. SIGMOD '06*, pages 695–705.
[11] P. Castro and A. Nori. Astoria: A programming model for data on the web. *Proc. ICDE'08*, pages 1556–1559.
[12] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. In *Proc. WWW'00*, pages 137–157.
[13] J. R. Durant. Web queries and dynamic chart data in Excel. Technical report, TR. Microsoft Corp., 2003.
[14] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein. Gencel: a program generator for correct spreadsheets. *J. Functional Programing*, 16(3):293–325.
[15] M. Gyssens, L. Lakshmanan, and I. Subramanian. Tables as a paradigm for querying and restructuring. In *Proc. PODS '96*.
[16] A. Jhingran. Enterprise information mashups: integrating information, simply. In *Proc. VLDB '06*.
[17] S. Jones and M.Burnett. A user-centred approach to functions in excel. *SIGPLAN J.*, 38(9):165–176, 2003.
[18] W. Kongdenfha, B. Benatallah, R. Saint-Paul, and F. Casati. Spreadmash: A spreadsheet-based interactive browsing and analysis tool for data services. In *Proc. CAiSE'08*.
[19] K. Laker. Exploiting the power of oracle using microsoft excel. Technical report, Oracle Corp., 2004.
[20] L. Lakshmanan, S. Subramanian, N. Goyal, and R. Krishnamurthy. On query spreadsheets. In *Proc. ICDE'98*.
[21] E. Lippert and E. Carter. *.Net programming for office: C# with Excel, Word, Outlook, Infopath*. Addison Wesley, 2005.
[22] D. Merrill. Mashups: The new breed of web app. Technical report, IBM Corp., 2006.
[23] J. Pemberton and A. Robson. Spreadsheets in business. *IMDS J.*, 100(8):379–388, 2000.
[24] F. Rice. Creating XML mappings in excel 2003. In *TR. Microsoft Corp.*, 2005.
[25] R. Saint-Paul, B. Benatallah, and J. Vayssiére. Data services in your spreadsheet! In *Proc. EDBT '08*.
[26] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users programmers. In *Proc. VLHCC '05*.