MashMaker: Mashups for the Masses

Rob Ennals
Intel Research Berkeley
robert.ennals@intel.com

Minos Garofalakis
Yahoo Research
minos@yahoo-inc.com

Categories and Subject Descriptors: H.4.3 [Information Systems Applications]: Information Browsers

General Terms: Management, Design, Human Factors, Languages

Keywords: Mashup, web, end-users.

1. INTRODUCTION

MashMaker is an interactive tool for editing, querying, manipulating, and visualizing *"live" semi-structured data*. MashMaker borrows ideas from word processors, web browsers, and spreadsheets. Like a word processor, MashMaker allows ad-hoc, unstructured editing of data. Like a web browser, MashMaker encourages users to find information by exploring, rather than by writing queries. Like a spreadsheet, MashMaker allows users to mix computed values with their data, including editing "live" (i.e., continuously-updated) data assembled through the web and/or user queries. MashMaker represents a novel paradigm for the ad-hoc exploration and management of diverse, heterogeneous collections of live data, that draws on the design principles of more "natural" software tools (like web browsers and spreadsheets) and simple scripting languages, rather than formal database models, schemas, and queries.

The goal of MashMaker is to allow non-expert users to easily create their own mashups based on data and queries produced by other users and by remote sites. Non-expert users often have lots of data that they would like to be able to query and otherwise manipulate. For instance: "Which of these houses has the largest number of good restaurants nearby?", "Do any of these news stories affect people I know?", "Show me a map with the addresses of everyone with my surname" or "How much would each of these recipes cost to make if I bought the ingredients in Safeway?" Right now, performing these kinds of queries and sharing them with others is possible, but not easy. The goal of MashMaker is to make such tasks easy and intuitive.

MashMaker is designed around the following key principles:

- **Untyped Tree Data Model:** Data is structured as a tree, where nodes have content, child nodes, and properties. There are no constraints on the form of the tree.
- **Mixed Data and Queries:** Users can extend/enhance their view of the data by adding *computed nodes* whose values are computed based on the surrounding data.
- **Sharing Queries as Widgets:** Useful queries can be bundled and exported as *"widgets"*. If a user computes something interesting over their data, they can then choose to turn the query into a widget and share it with their friends.

SIGMOD'07, June 11–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

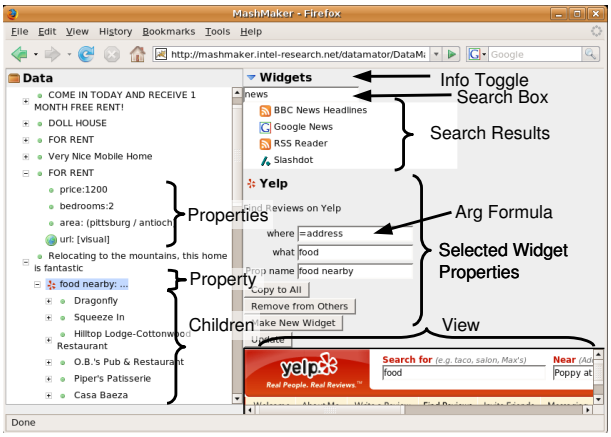

Figure 1: The MashMaker Interface.

- **Overlayed Editing of Live Data:** Users can apply local edits to their data views (including query results). These edits are treated as an *overlay* (or, an incremental *edit script*) on top of the data, allowing the data to be updated while preserving user edits.
- **Example-Driven Queries based on Interactive Data Exploration:** Users never have to think about textual queries in a formal query language. Instead, all queries and data manipulation tasks can be formulated through interactive data browsing and exploration. Query tasks can also be specified *by example*; for instance, to compute a property for a collection of data elements, a user can compute that property for one element, and then copy the property to other elements in the tree (e.g., the node's siblings).
- **Collaborative Exploration of Data:** Users can share data and widgets with friends and other members of their social network. During data exploration, MashMaker can automatically suggest widgets that have been applied to similar data inside the user's social network.

MashMaker differs from previous work on personal information management [1, 3, 2] through its focus on ad-hoc, interactive data exploration and manipulations, rather than structure extraction and support for just querying the data. MashMaker improves on previous tree-structured spreadsheets [4] through its support of live data and query results, its ability to package up queries as widgets, its support for user collaboration and sharing, and its exploration-based, formula-free query model.

Figure 1 shows a screenshot of the MashMaker interface running as an AJAX application within the FireFox web browser. The

left-hand portion of the window is taken up by the *data tree* which contains the user's data. On the right, we see the *widget pane* containing a widget search box, and configuration information for the selected node. The widget pane can be collapsed by clicking on the toggle button at the top (Figure 2).

2. ARCHITECTURE

In this section, we provide more detail on the MashMaker system architecture, focusing on the six key designed principles outlined in Section 1.

2.1 Untyped Tree Data Model

All data in MashMaker is structured in the form of a tree. Each node in the tree has *content, child subtrees,* and a set of *properties*, where each property is a pair of a name and a subtree. Following the design paradigm of spreadsheets or scripting languages, MashMaker has no formal data schema and the tree is permitted to take an arbitrary form. While it is certainly the case that not all queries can be applied to all data, data mismatch issues are explained at query time, rather than ahead of time. The content of a node can be either text or a visualization (Figure 2).

2.2 Mixed Data and Queries

Like a spreadsheet, MashMaker does not separate derived query results from data. To compute a value from some data, a user can insert a node directly into the MashMaker tree, and define the value of the node using a formula written in terms of the surrounding data. (While expert users can explicitly specify such formulae, non-experts typically express data-manipulation tasks through interactive data exploration and browsing, as discussed later in this section.)

A node's formula defines not only the content of the node, but also its child nodes and properties. For instance, in the MashMaker interface in Figure 1, the "food nearby" formula evaluates to a tree that contains nodes for nearby restaurants and their properties. MashMaker also distinguishes between *direct nodes* which have their own formula, and *derived nodes* which are defined through the formula of a parent.

Much like a spreadsheet formula, a MashMaker formula is either a *literal* (just some text), or an *expression* (starting with an "=" sign). An expression is either a path to another node relative to the current location in the tree, or a function application taking further formulae as arguments. Since a formula can contain relative paths to other nodes, its value depends on the location at which it is evaluated. Some simple example formulae include:

- Hello : The literal string "Hello".
- =boss.age : The value of the "age" property of the "boss" property.
- =Weather(=country) : The weather in the place referred to by the "country" property.

Once again, note that non-expert users can specify such formulae interactively through browsing over the MashMaker tree (with no knowledge of the formula syntax). Expert users can however enable a *formula box* that allows them to type in node formulae directly.

2.3 Overlayed Editing of Live Data

Following the spreadsheet model, all data is editable, including *computed nodes* generated through queries. A user can generate a subtree with a query, and then modify it by editing, adding, and deleting child nodes. Since MashMaker data and queries are live,

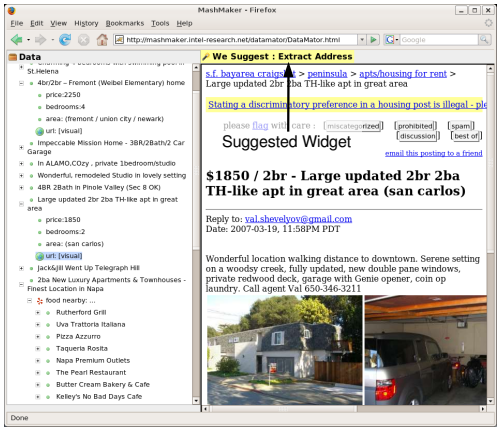

Figure 2: View with the widget pane collapsed

the results of the query can dynamically change over time. When the base data changes, MashMaker does not lose the user's edits. Instead the edits are treated as an *overlay* (essentially, an *edit script*) which is reapplied to the base data when changes arrive. It is important to note that any edits made by the user stay local to the user's view — no attempt is made to propagate changes to the original source data.

2.4 Query Widgets

A query widget is the combination of a *function with associated metadata*. The function takes formulae as arguments and produces a subtree as a result. The metadata includes a description of what the widget does, an icon to be used for nodes defined by the widget, and a optional custom interface for configuring it.

Users can search for widgets by typing keywords into the *widget search box* (Figure 1). MashMaker will also automatically suggest widgets that it believes may be applicable (Figure 2). This suggestion is based on what widgets other users have applied to similar data, where data similarity is determined using several different criteria (e.g., the data was created by the same widget, or has similar structure and/or properties). For example MashMaker might suggest "distance" if there are two addresses in scope, suggest "map locations" for a list of nodes with addresses, or suggest "find home page" for a person's name.

MashMaker query widgets generally fall into one of four categories:

- **Importers** import data into MashMaker from an external source; for example, listings from Craigslist or email addresses from Outlook.
- **Visualizers** define a node whose content is a visualization; for example, showing locations on a map, plotting a graph, or representing a list of nodes as an editable spreadsheet.
- **Evaluators** define a new subtree in terms of existing subtrees; for example, "Sum these numbers", "Filter this list", or "Find the distance between these two addresses".
- **Extractors** find structure in unstructured data; for example, "Find Address" or "Make this house description canonical".

A widget can be created as a *builtin widget* (i.e., system-provided for a certain data type), an *extension widget*, or a *custom widget*. Extension widgets are written as web services which MashMaker connects to. Custom widgets are defined by the user in terms of other widgets.

Users can create a custom widget by wrapping up a query that they have already performed. To create a widget, a user selects the node that holds the result of the query, and then clicks the "New Widget" action from the action panel. MashMaker now prompts the user to specify the name of the widget, the widget's description, an icon, hints on where to apply the widget, and what to do with each of the other nodes that the selected node depends on. There are two ways that MashMaker can treat a node that the result node depends on: (i) *Local:* The node is internal to the widget; or (ii) *Argument:* The value of the node must be specified as an argument to the widget. This is very similar to creating a function in a programming language by copying an expression and deciding which of the other expressions it depended on should be copied across as well. As with Apple's Dashboard Widgets, a user can create their own custom configuration interface for their widget if they like.

Once a user has defined a custom widget, they can publish it to the world, or share it more locally with their friends and other members of their social network.

2.5 Exploration- and Example-Driven Queries

MashMaker enables non-expert users to query their data through interactive data exploration and browsing, rather than by writing abstract queries. If the user selects a node that they are interested in, then MashMaker can suggest a set of query widgets that it thinks might produce interesting information from that node and the nodes around it (Section 2.4, Figure 2). This model allows the user to casually browse around their data, applying widgets to the data they have, and browse and query the data produced by the widgets, until they eventually run into something they find interesting and possibly bundle this up as a widget to share with others. Like the web, MashMaker allows users to absent-mindedly wander through their data in the hope of running into something interesting.

Note that, while users have the ability to specify queries by entering formulae directly into a formula box, the intention is that most users will not use this feature, and indeed that it will usually be turned off.

MashMaker also enables users to casually specify operations over collections of data *by example*. For instance, much like using a spreadsheet, a MashMaker user can query a collection of elements by first defining a query for an individual element and then copying this query to all elements in the collection. To do this, the user can browse into a node, define new computed nodes with the property that they are interested in for just that node, and then apply a "copy to all" action to copy the new property to all sibling nodes. As an example, if my data is a list of houses, I might browse into one house, use a "Crime Level" widget to compute a property node whose value is the crime level near that house, and then use the "copy to siblings" action to give all other houses a crime level property. I might then use the "Sort" widget to create a new copy of the house list that is sorted by crime level.

Note that, since MashMaker data is live, the "copy to all" action does not just copy the selected property to the current siblings, but to all future siblings as well. If the data is updated to include additional sibling nodes, then the property is copied into them as well. The underlying model behind this is that every node contains a (normally) hidden *#prototype* property that contains properties that should be copied to new siblings.

2.6 Collaborative Exploration

MashMaker allows users to collaboratively share and explore data and queries. Users can share data, widgets, and widget suggestions — all using a simple social network dynamically maintained by MashMaker. If a user has created some data that they think might be of interest to their friends, they can then publish this to their friends, either as writable or read-only data. This allows users to create their own ad-hoc social-networking applications, making use of data published by friends to define data they publish themselves. Similarly, users can share widgets they have created, and the set of MashMaker-suggested widgets dynamically adapts based on the widgets shared/used across a user's social network.

3. DEMONSTRATION SCENARIO

The user is planning to rent a house, and wants to know which of the houses available has the most good restaurants nearby. First, the user goes to a housing website and clicks the "Add to MashMaker" icon on their browser bookmark bar. MashMaker starts up and the list of houses on the website is displayed in the data tree, with one node for each house, and the properties of each house reflecting the information provided.

The user selects one of the house nodes, and MashMaker automatically suggests that the user apply the "Things Nearby" widget to search for things nearby. The user clicks on this suggestion and is presented with a query box to allow them to configure the query. MashMaker has automatically filled in the address of the house as the location to search from and the user enters "food" as what they want to search for, and "food nearby" as the property name.

Seeing this list, the user decides that they would like to know how many restaurants there are within 0.5 miles with a rating of three stars or higher. The user clicks on the "filter" widget (which MashMaker automatically suggested) and uses the filter widget's custom interface to set up these criteria. They then use the "count" widget, to count how many restaurants meet this criteria, calling it "restaurant count". They then click "copy to all" to give all the other houses a "restaurant count" property, allowing the user to compare the houses.

The user decides that this "restaurant count" property is useful, so they click the "make new widget" to save this property as a widget that they can again use in the future. MashMaker asks the user which of the values "restaurant count" depended on should be treated as arguments, and the user selects the house, and the kind of thing being searched for. The user types a description, uploads an icon, and publishes the new widget. Now, the next time one of the user's friends looks at a house, this new widget will be suggested.

The user goes on to create additional widgets that compute the number of the user's friends that live within a mile of each house, and the user's commute time to work for each house. The user then plots all the houses on a map, color coded according to how they rate on these scores.

4. REFERENCES

[1] CAI, Y., DONG, X., HALEVY, A. Y., LIU, J., AND MADHAVAN, J. Personal information management with semex. In *SIGMOD Demo Program* (2005).

[2] FAABORG, A., AND LIEBERMAN, H. A goal-oriented web browser. In *CHI Proceedings* (2006).

[3] FRANKLIN, M., HALEVY, A., AND MAIER, D. From databases to dataspaces: A new abstraction for information management. In *SIGMOD Record* (2005).

[4] TAKEICHI, M., HU, Z., KAKEHI, K., HAYASHI, Y., MU, S.-C., AND NAKANO, K. TreeCalc: towards programmable structured documents. In *Japan Society for Software Science and Technology* (2003).