

DataMator: Mashups for the Masses

[Demo Proposal]

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

Minos Garofalakis
Intel Research Berkeley
minos.garofalakis@intel.com

1. INTRODUCTION

DataMator is an interactive tool for editing, querying, manipulating, and visualizing “live” *semi-structured data*. DataMator borrows ideas from word processors, web browsers, and spreadsheets. Like a word processor, DataMator allows ad-hoc, unstructured editing of data. Like a web browser, DataMator encourages users to find information by exploring, rather than by writing queries. Like a spreadsheet, DataMator 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. DataMator 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 DataMator 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 DataMator is to make such tasks easy and intuitive.

DataMator 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

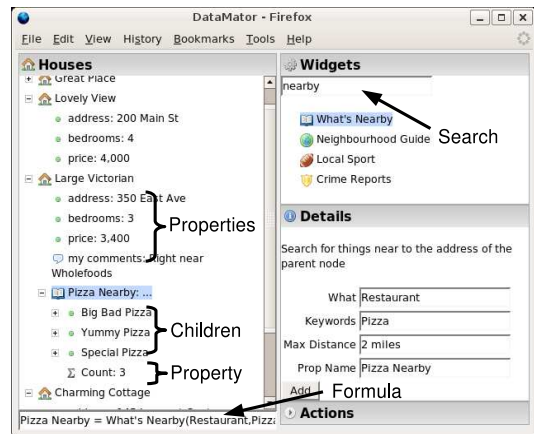


Figure 1: The DataMator Interface.

- **Overlaid 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, DataMator can automatically suggest widgets that have been applied to similar data inside the user’s social network.

DataMator 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. DataMator 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 DataMator 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. Below the data tree, we see the *formula box* which reveals the underlying formula for the selected node. On the right, we see the *widget selector*, the *details panel*, and the collapsed header for the *action panel*.

2. ARCHITECTURE

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

2.1 Untyped Tree Data Model

All data in DataMator 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, DataMator 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, DataMator does not separate derived query results from data. To compute a value from some data, a user can insert a node directly into the DataMator 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 DataMator interface in Figure 1, the “What’s Nearby” formula returns a “Pizza Nearby” subtree that comprises nodes for nearby pizza stores and their properties. DataMator 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 DataMator 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 of the parent node.
- =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 DataMator tree (with no knowledge of the formula syntax). The *formula box* (Figure 1) reveals the defining formula for the selected node, or “#derived” if the selected node is generated by the formula for a parent node.

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 DataMator data and queries are live, the

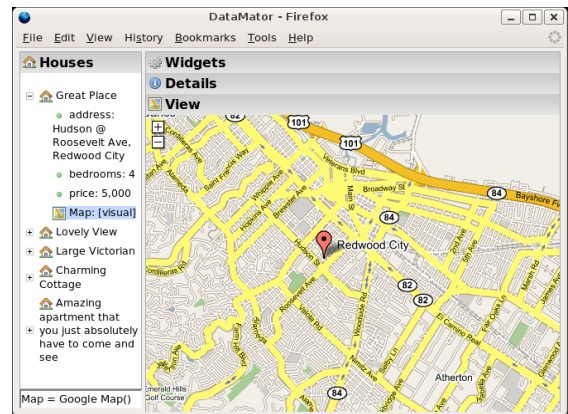


Figure 2: Example DataMator Visualization Node.

results of the query can dynamically change over time. When the base data changes, DataMator 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 an optional graphical skin for configuring it.

Users can search for widgets by typing keywords into the *widget search box* (Figure 1). DataMator can also automatically suggest widgets that it believes may be applicable. 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 DataMator 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.

DataMator query widgets generally fall into one of four categories:

- **Importers** import data into DataMator 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 (Figure 1), 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” 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 DataMator

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. DataMator 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 three ways that DataMator can treat a node that the result node depends on: (i) *Local*: The node is internal to the widget; (ii) *Argument*: The value of the node must be specified as an argument to the widget; or, (iii) *Dynamic Scope*: When the widget is instantiated, it looks for an argument node using the same path from the site at which it is instantiated. 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 visual theme for their widget if they like, affecting the look of both the details panel and the listing of that widget in searches.

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

DataMator 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 DataMator 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). 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, DataMator 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 the formula box, the intention is that most users will not use this feature, and indeed that it will usually be turned off.

DataMator also enables users to casually specify data-manipulation tasks by *example*. For instance, much like using a spreadsheet, a DataMator 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 siblings” 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 DataMator data is live, the “copy to siblings” 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

DataMator 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 DataMator. 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 DataMator-suggested widgets dynamically adapts based on the widgets shared/used across a user’s social network.

3. DEMONSTRATION SCENARIO

The user is concerned that that one of her friends might be killed in a disaster without her knowing. The user uses the “BBC News” widget to search for disaster news stories with “dead” as a keyword. When the user browses into one of these news stories, DataMator suggests the “Extract Address” widget, because other users have applied it to news stories before. The user applies this widget, adding an “address” field to the news story.

Next, the user wants to know if any of her friends live near to where this disaster occurred. She drags her address book into the news story, creating a new property called “friends” that is a live copy of her address book. When the user selects one of her friends, she follows DataMator’s suggestion that she should apply the “Distance” widget to calculate the distance from the friend’s home to the site of the news story. She then uses the “copy to siblings” action to give all her friends such a property, and uses the “Filter” widget to create a filtered list of friends, called “friends in peril”, including only those friends who live within 10 miles of the news story.

Having determined which of her friends live near one of the disaster areas, the user wishes to determine which of her friends live near other disasters. She uses the “copy to siblings” action to copy across the “friends in peril” property to all the other news stories. The “copy to siblings” action notices that “friends in peril” has a dependency on the computed “address” and “friends” properties, which are not present in the other news stories, so DataMator asks the user whether she would like to copy these properties as well. The user agrees, adding all three properties to all news stories. The user now creates a filtered property, called “important disasters” containing only those news stories with a non-empty “friends in peril” property.

The user is pleased with her work, and decides that she would like to publish the “important disasters” node as a public widget for others to use. She clicks on the “create widget” action, specifies that the address book be taken as an argument but that the news source be hard-coded, and types in a brief widget description.

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 dataspace: 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).