Intel® Mash Maker

Rob Ennals
Intel Research
2150 Shattuck Ave
Penthouse Suite
Berkeley, CA 94704, USA
robert.ennals@intel.com

ABSTRACT

Intel® Mash Maker is a mashup creation tool that was initially developed at Intel® Research and is now being developed by Intel's Software Solutions Group.

Mash Maker allows a user to customise and improve web pages by applying mashups that add additional content. Such "overlay mashups" add content that is visually distinguished from the host web page, but is integrated into the normal page layout.

Mash Maker encourages users to take a "suck it and see" approach to find mashups that they like. As a user browses the web, Mash Maker suggests mashups that it believes the user will find useful. The user can then turn a mashup on, see if the content it added looks useful, and turn it off again if they do not like it.

Mash Maker uses a three level structure to create mashups. Information is extracted from web sites using wrappers that are written collaboratively by users in a wiki-like model. Widgets written in javascript query this information and add new information and visualisations to the page. A user can then arrange several widgets on a page to create a mashup which they publish and share with other users.

INTRODUCTION

Intel® Mash Maker [14, 15] is a browser extension for Firefox or Internet Explorer. Mash Maker allows users to create, share, and find mashups that improve existing web sites by adding additional content to existing page layouts. In this chapter, we discuss several of the key concepts behind Mash Maker:

Overlay Mashups: Mash Maker allows users to apply mashups that add content to existing web sites.

Mashup Suggestions: Mash Maker tries to guess what mashups a user will find useful, based on their past behaviour.

CHI 2009, April 3 - 9, 2009, Boston, MA, USA.

Collaborative Creation of Web Wrappers: Mash Maker uses a Wiki-model to allow users to teach it how to extract meaning from web sites.

The Shared Data Tree: Mash Maker mashups are composed from widgets which communicate by modifying a shared data tree.

Untrusted Widgets: Mash Maker avoids the need for users to trust mashups by isolating untrusted widgets from untrusted code

Copy and Paste: Mash Maker allows a user to combine web sites using a simple "copy and paste" metaphor.

Intel® Mash Maker was originally created at Intel® Research and is now being developed and maintained by the Intel Software Solutions Group. You can download Intel® Mash Maker from the following URL:

`http://mashmaker.intel.com`

OVERLAY MASHUPS

Mash Maker allows a user to customise the web sites that they browse by applying mashups that modify the pages to make them more useful. For example one might add a map to a news site to show where the stories are taking place (Figure 1), add price comparison information information to a shopping site to see what the prices were like on other sites, add legroom information to flights on a travel site (Figure 2), or add a button to every phone number that calls the number if you click it.

There has been a lot of prior work on writing modules that modify existing web sites. OreO [7] and WBI [3, 33] use a proxy to modify web pages without requiring support from the web browser. Other tools such as Greasemonkey [1], Chickenfoot [5] and Koala/CoScripter [32] run as extensions to the web browser that modify web pages on the client. Mash Maker was initially implemented as a web proxy [15] and was then reimplemented as a browser extension [14].

Unlike previous work, Mash Maker restricts mashups by requiring that they only be able to add new content to an existing page, and cannot modify a page in any other way. We refer to this restricted class of mashups as *overlay mashups*.

[1] http://greasespot.net

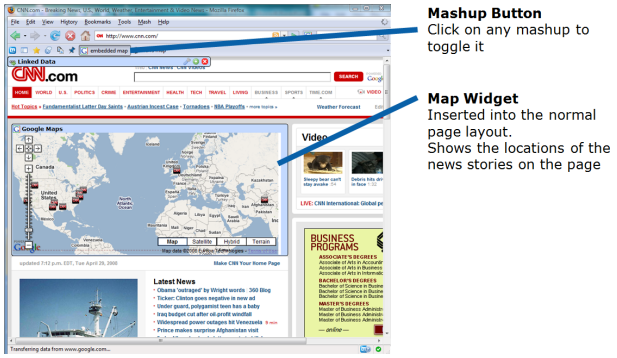

Figure 1. Showing CNN News Stories on a map

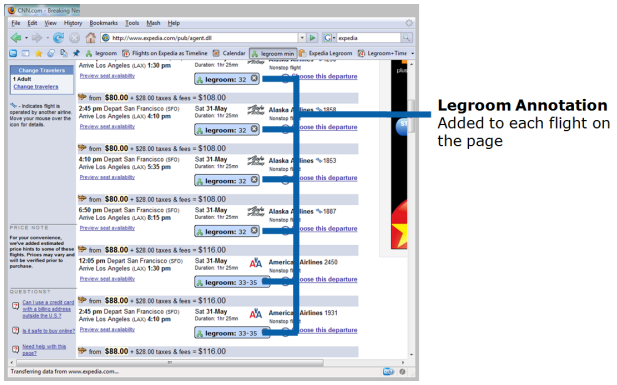

Figure 2. Adding legroom to Expedia

By adding this restriction Mash Maker excludes some mashups that can be created in tools such as Chickenfoot, for example, a Chickenfoot mashup can change the text on a web site, make text boxes resizable, add keyboard shortcuts to a web site, or provide automatic login for web sites.

There are three reasons why Mash Maker restricts mashups to be overlay mashups:

To make it clear what a mashup does: Unlike previous work, Mash Maker assumes that a user will apply a mashup to a web site without knowing in advance what it is that the mashup does, or whether they should trust the mashup. Mash Maker distinguishes content added by a mashup by surrounding it with a blue border. A user can thus quickly see which content on the page is from the original web page, and which content has been added by a mashup. By making it visually clear what content has been added by the mashup, Mash Maker makes it clear to the user what it is that the mashup has done to the web site, making it easier for them to evaluate whether the mashup is useful.

To prevent mashups misbehaving: Since Mash Maker assumes that a user will be applying an untrusted mashup to a web site without knowing what it does, it is important that Mash Maker restrict the extent to which mashups can do things that are unexpected or harmful. Restricting ourselves to overlay mashups makes this easier.

To reduce legal concerns: If a mashup can modify a page

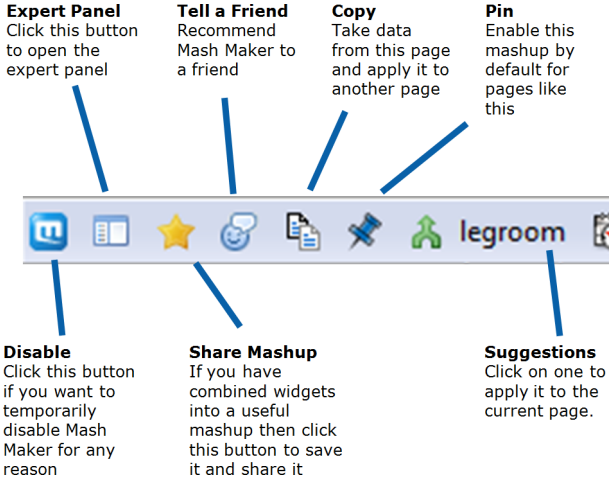

Figure 3. The Mash Maker Toolbar

in an arbitrary way, what happens if a mashup modifies a page in a way that the content owner does not approve of? One of the reasons why Mash Maker restricts itself to overlay mashups is to reduce the likelihood that a content owner sees a mashup as an unlawful derived work. Since the all additional content is visually distinguished from the page, and does not directly affect the page content, one can argue that Mash Maker is just a browser that shows additional content above a page, rather than a tool that modifies someone else's content.

More generally, if mashups are to become widespread then it is important that they be structured in a way that is beneficial or at least acceptable to content owners. Legal concerns are also the primary reason why Mash Maker cannot, at present, remove content from a web page. If content could be removed then uses could upset content owners by removing adverts.

MASHUP SUGGESTIONS

As the user browses the web, Mash Maker suggests mashups that it believes the user might like. Suggested mashups appear as toggle-buttons on Mash Maker's toolbar (Figures 1 and 3). A user can turn a particular mashup on and off by clicking on the button for that mashup. Mash Maker assumes that a user does not typically know what existing mashups will be useful for them, but that they they will recognise useful mashups when they see them applied to a page.

Mash Maker builds on a lot of prior work that recommends content based on user browsing behaviour [44, 42, 40, 25, 18]. The key factor that distinguishes Mash Maker from prior systems is that it recommends mashups rather than web pages. Mash Maker bases its suggestions on the website currently being viewed, the user's recent browsing history, the mashups the user seemed to like previously, and the behaviour of other users.

When Mash Maker suggests a mashup to the user, the only information the user has is the name of the mashup and possibly an icon. While a user can get a more detailed descrip-

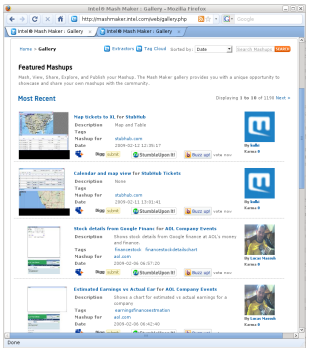

Figure 4. The gallery provides another way to find available mashups

tion of a mashup by hovering over the mashup button or browsing the mashup gallery (Figure 4), in most cases the user will just click on the button and see what happens. If the mashup looks useful then the user can keep it enabled. If it doesn't look useful then they can turn it off again. Since a user is not expected to know in advance what the mashups they enable do, Mash Maker ensures that any content added by a mashup is visually distinguished from the page so that the user can easily see what the mashup has added.

If a user decides that they really like a mashup then they can *pin* it by clicking on the pin button on the toolbar (Figure 3). Once a mashup is pinned, it will be enabled by default whenever the user visits that web site.

It is possible for a user to apply multiple mashups to the same page simultaneously. When a user has several mashups turned on, all mashups will add their content to the page. In some cases, one mashup can use information added by another mashup. For example if a user turns on a mashup that annotates apartment listings with nearby restaurants, and then turns on a mashup that displays all addresses on a map, then the restaurants will also be shown on the map. If the new combination of mashups is itself an interesting mashup, then the user can save the new mashup by clicking on the share button on the toolbar (Figure 3).

Mash Maker's suggestion system is largely orthogonal to the rest of the system. One could conceivably use Mash Maker's suggestion toolbar to automatically suggest mashups created with a tool such as Chickenfoot [5]. The advantage of combining Mash Maker's suggestion system with Mash Maker's restricted *overlay mashups* is that a user can easily see what a mashup has done, it is easy to remove a mashup without reloading the page if the user decides they do not like it, and it is harder for a mashup to do something undesirable.

If the suggestion toolbar does not suggest an interesting mashup, a user can also use Mash Maker's more conventional web-based mashup gallery (Figure 4) to find mashups. The gallery allows one to search for mashups by keyword and can show either mashups for a particular page type, or mashups for an arbitrary page types. Clicking on a mashup in the web gallery takes the user to an example web page with that mashup enabled. If the user likes the mashup, then they can pin it and navigate to the page they were actually interested in.

There is a trade-off between privacy and suggestion accuracy. In order to provide useful suggestions Mash Maker needs to download information related to the web pages that you browse. To reduce privacy issues, the Mash Maker client always requests information about an entire domain (e.g. google.com) rather than an individual URL. All requests are sent anonymously, and headers are set to allow requests to be cached by intermediate proxies.

COLLABORATIVE CREATION OF WEB WRAPPERS

To apply a mashup to a web page, Mash Maker needs to extract machine-readable data that it can use as inputs to the mashup. For example, if a mashup wants to add a map to an apartment listing site showing the location of each apartment, then Mash Maker needs to extract the apartment addresses from the web site. While standards such as microformats [2] and RDFa [38] exist to allow a web site to expose machine readable data directly in its HTML, at the time of writing most web sites do not do this. Mash Maker thus extracts machine-readable information from raw HTML using user-created *wrappers*.

A wrapper [30] is a set of rules that can be used to extract machine-readable data from a web site. While some authors have had some success extracting data from a website without any human assistance [11, 10, 43, 8, 1], most systems, including Mash Maker, require some form of user guidance to teach the wrapper-generator what a web page means [29, 2, 35, 48, 24, 4].

Mash Maker's web wrapper system has two unusual features. Firstly, Mash Maker organises its wrappers in a *wiki-style* model in which there is a single canonical web wrapper for any given kind of web page, and any user can edit any wrapper. Secondly, wrappers created by Mash Maker include *drop zones* that indicate where additional content should be inserted into the page layout.

Mash Maker's wiki-style model differs from the more conventional model in which multiple users create their own competing wrappers for different web sites and writers of mashups pick the wrappers that work well for them. One advantage of the wiki model is that it removes the need for a user to look through a list of potentially broken wrappers in order to find the one that works best with their web page. For example, if a user wants to create a new mashup that visualises the data on a particular web page using a calendar, then they simply need to drag a calendar widget onto the page, and do not need to first find the best wrapper. Choosing a wrapper can be a confusing process: "I just want to apply the calendar to this page. Why do I have to choose something from this list?"

Another advantage of the wiki model is that it makes it easier to recover when a change to the structure of a web page breaks a wrapper. In a study conducted by Dontcheva et al,

[2] http://microformats.org

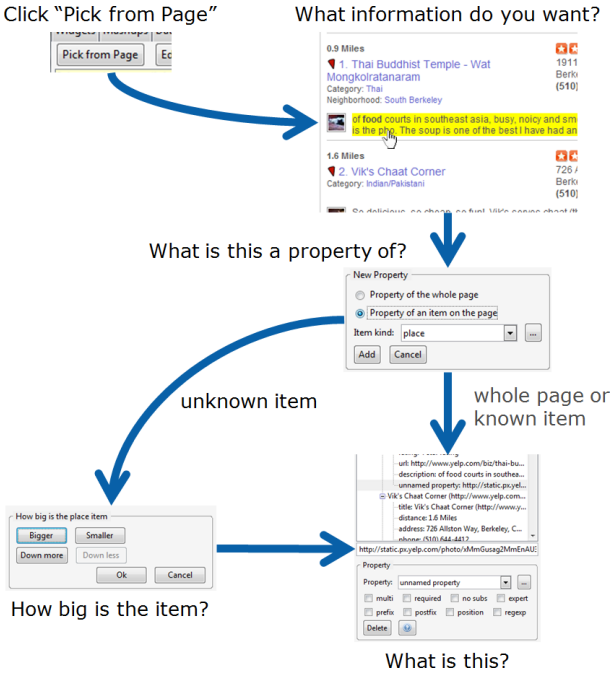

Figure 5. Teaching Mash Maker what something means

74% of tested web sites underwent a significant structural change of the kind that would be likely to break a wrapper within the 5 month test period. If a broken wrapper was owned by a particular person then it would be necessary for users to either wait for the wrapper owner to fix their wrapper, or to move all affected mashups to a new wrapper, adding the old wrapper to the list of dead wrappers that mashup authors need to avoid. With the wiki approach, the first user to notice that the wrapper has broken can open Mash Maker's wrapper editor and fix the problem. While techniques exist to repair a broken wrapper automatically [37, 34, 9] these techniques are not yet bulletproof.

The big disadvantage of the wiki model is that it opens Mash Maker up to potential vandalism [12]. If anyone can edit a wrapper, then anyone can break a wrapper. A common problem during the early days of Mash Maker deployment was that a novice user would decide to experiment with the wrapper editing tools and break something high profile such as google search without realising that their changes were affecting all users. Mash Maker's solutions to this problem are the same as those taken by text wikis such as Wikipedia [39] - a history is recorded so that bad edits can be rolled back, and sensitive wrappers can be locked. In the future, one could potentially also use wrapper verification [28] to prevent users saving wrappers that had obviously been broken.

Mash Maker's wiki-model for web wrappers is largely orthogonal to it's other features. Mash Maker could have used the more conventional "roll your own" model, in which case a user would be required to chose a suitable wrapper when creating a mashup. Similarly, Mash Maker's wrapper database can be useful for other other tools; indeed Yahoo Search

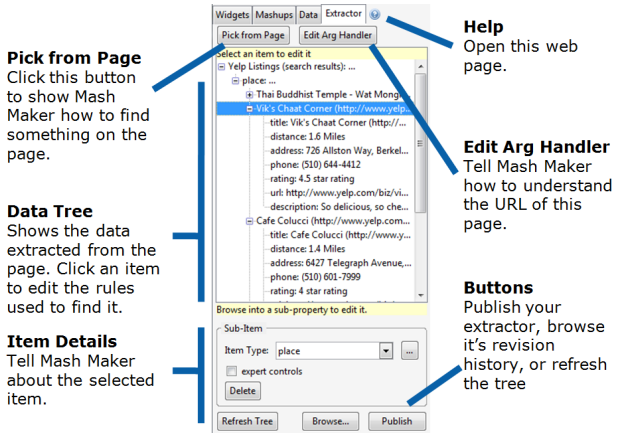

Figure 6. The Wrapper Editor

Monkey[3] uses Mash Maker's wrapper database to help it extract data from web pages [20]. Mash Maker's wiki-model is also largely independent of Mash Maker's particular choice of wrapper editor. One could potentially adapt other wrapper editors to support Mash Maker's wiki model by adding support for revision tracking and other collaborative features.

A user can edit a wrapper by opening Mash Maker's wrapper editor. This is part of Mash Maker's *expert panel*, which can be opened by clicking on the *expert* button on the Mash Maker toolbar (Figure 3). The wrapper editor is distributed as part of the main Mash Maker plugin and is available to all users.

Mash Maker's wrapper editor works in a similar way to other user-guided wrapper editors such as Pictor [48], Lixto [4], WIEN [29], STALKER [35], Dapper [4], and Irmak/Suel [24]. A user gives examples of things on the page that they think are interesting and Mash Maker attempts to infer a wrapper from these examples. For example, if a user wants to teach Mash Maker how to extract a product price, they click on the price, and then tell Mash Maker that this is a price (Figure 5). Given enough examples, Mash Maker will try to infer rules that it can use to recognise other prices. A web page will often contain several objects each of which have their own properties. E.g. a page may contain a list of products, each of which has a price. In this case, a user can tell Mash Maker that the price they clicked on is part of a product, and then use a set of "bigger/smaller/up/down" buttons to show Mash Maker the physical boundaries of the product that the price is part of (Figure 5).

Mash Maker uses a similar example-driven approach to decide which wrapper to apply to a page. When a user navigates to a page, they can explicitly select the existing wrapper that should apply to this page. Mash Maker uses these examples to infer a URL regular expression for each wrapper that covers these pages. While more sophisticated techniques exist for inferring what wrapper should be applied to

[3] http://developer.yahoo.com/searchmonkey/
[4] http://dapper.net

a web page [49] and a URL regexp is not always a good way of identifying web pages, URL regexps have the advantage of being easy to understand and debug when things go wrong. Mash Maker saves the list of example URLs as part of each wrapper and encourages the to check that their wrapper works on the example URLs before saving it.

It addition to specifying where information can be found on a page, Mash Maker wrappers also specify the *drop zones* on a web page where extra content should be added. A drop zone is a place on a web page where it is good to insert additional content without disturbing the layout of the web page. Since the best drop zones on a web page are largely independent of the particular content being inserted, it makes sense that drop zones be factored into the wrapper, rather than requiring each mashup to specify its own layout from scratch. A mashup is free to ignore the drop zones in the wrapper and place content in other locations if the author thinks that is appropriate.

Mash Maker uses wrappers to extract data from a web page even if a web site provides programmatic APIs that can be used to query its data. This is for several reasons. Firstly, if an API was used, then it would still be necessary to use a wrapper to determine where the data was on the page (as done by d.Mix [21]). Secondly, since the web page is already loaded, it is more efficient to get the information from the HTML rather than accessing an external API. A mashup may however use APIs to bring in additional information from other sources that should be added to the current page.

It is useful if the wrappers for different pages agree on a common vocabulary to describe their data. For example, if one wrapper says "price" while another says "cost" then it becomes harder to write widgets that can work across multiple web sites. Mash Maker addresses this problem by encouraging wrapper authors to choose type and property names that conform to a common ontology. This ontology is editable by all users using a collaborative ontology editor. Mash Maker's collaborative ontology editor is significantly more primitive than systems like Protege [46] or Freebase [6]. It allows users to specify type names, associated property names, and simple subtype relationships, but lacks higher level features. Unlike the ontology editor of FreeBase [6] Mash Maker anyone to edit the properties that can be associated with a type, not just its owner. The motivation here is to encourage people to reuse existing types rather than creating new ones.

Mash Maker does not currently use any kind of Data Detector (e.g. Miro [17]) to detect objects on a page. Mash Maker does however take advantage of microformats when they are available.

THE SHARED DATA TREE

Mash Maker uses a three-level architecture for creating mashups (Figure 7). Wrappers extract data from web pages, widgets visualise and manipulate the data extracted from the web page, and mashups connect multiple widgets together and arrange their content into drop zones in the page layout. These three layers are largely independent: A wrapper may be used by many widgets, a widget may accept data from many wrappers, a mashup may use many widgets, and a widget may be used by many mashups.

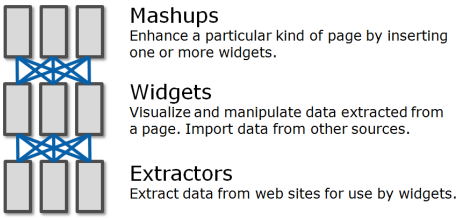

Figure 7. Wrappers, Widgets, and Mashups

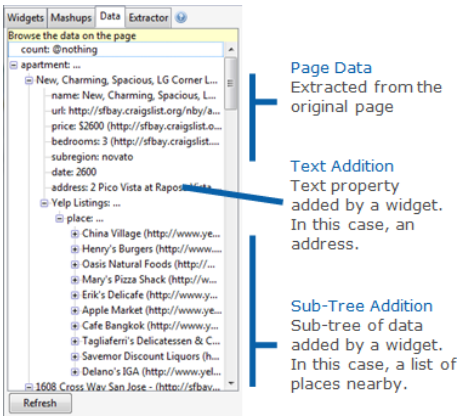

Figure 8. A data tree showing additions from widgets

Several other mashup tools allow one to create a mashup by composing one or more components. Good examples include Yahoo! Pipes [5], Microsoft Popfly [6], and Marmite [47] all. Pipes and Popfly adopt a visual dataflow programming model [23, 27, 36] in which wires are drawn between widgets to allow data to flow between them, and Marmite behaves like Apple's Automator by allowing one to create a mashup as a sequence of stages, each of which acts on the output of the previous stage. Mash Maker instead uses a Tuple Space [19] inspired publish/subscribe model [16] in which widgets communicate by reading and writing a shared data tree.

Mash Maker maintains a *data tree* for every web page currently open in the browser, showing a structured view of the data on the web page. Initially the data tree contains the information extracted from the page by the wrapper. Any widgets on the page can query the data on the page, add additional information to the data tree, and modify or remove information. An expert user can view the data tree for the current page using its tab in the expert side panel (Figure 8).

The data tree is the only means by which widgets can communicate with each other. The Mash Maker API allows a widget to ask to be notified when the result of a query

[5]http://pipes.yahoo.com
[6]http://popfly.com

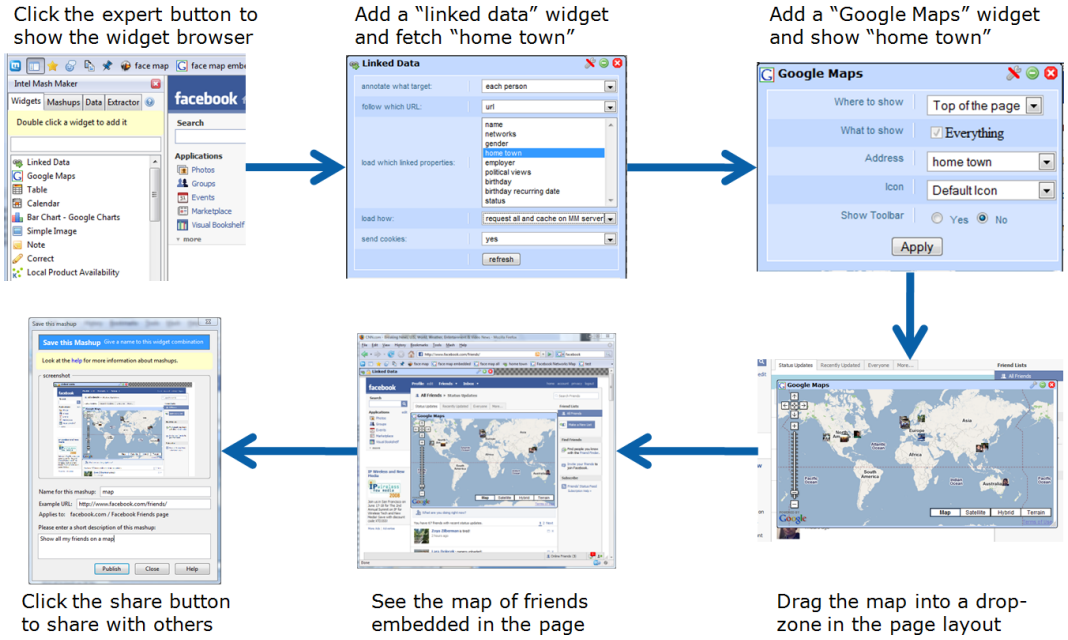

Figure 9. Steps to creating a mashup that shows facebook friends

changes due to actions by another widget. For example, the map widget asks to be notified when the set of objects with addresses changes. This allows the map widget to dynamically update its map when other widgets add or remove objects with addresses.

The mental model is that adding a widget to a page creates an *improved page*, which can itself be enhanced further by adding more widgets. A widget is not expected to distingish between information that was originally on the page and information that has been added to the page by other widgets. For example a price comparison widget does not care if the dollar-price of an item was calculated by a separate currency conversion widget.

Mash Maker could have used the same visual dataflow approach that is used by Yahoo! Pipes and Microsoft Popfly. Under this model, the wrapper would be treated as being just another box in the network whose extracted data could be fed to other boxes. Work started on Mash Maker before Pipes or Popfly were publically known, so no deliberate decision was taken to use a different model. There are however advantages of the "shared tree" model for the domain in which Mash Maker works:

One motivation for the "shared tree" model is that it allows a user to create a mashup by adding the features that they think that they want, without having to think about how they should fit into a logical structure. The widgets find each other by looking for information that they want that other widgets are providing. This works well for simple cases (e.g. find home country and visualise on a map), but for more complex mashups one may have to use a widget's settings panel to tell it which other widget it should be talking to.

Another motivation for the "shared tree" model is that it makes it easier for the physical location of a widget on the page to correspond to where it's primary visualisation will be inserted. If a visual dataflow model is used then some layouts of boxes will make the wires hard to read. Since Mash Maker's boxes are not connected by wires, this problem does not arise.

In addition to manipulating the data tree, a widget can also publish content that can be inserted into the layout of the page. Content can currently be either text, a clickable action icon, or an iframe that can contain arbitrary web content. Figure 1 shows a map visualisation running in an iframe that has been published by the Map widget. Figure 2 shows text annotations that have been associated with objects on a page.

Every piece of content published by a widget is associated with a particular node on the data tree. When the content is inserted into the page layout, it will be placed relative to the node that it is associated with. For example, the map in Figure 1 is associated with the whole page (the root of the tree) while each text annotation in Figure 2 is associated with a particular flight, and inserted into the layout for that flight.

A widget has no control of how its content will be integrated into the layout of the page, since the only view it has of the page is the data tree. It is entirely up to the user creating the mashup to insert content into the page layout appropriately. They can do this by either dropping content into drop-zones described by the wrapper, or placing content at a physical location relative to the node that it is associated with. This separation allows widgets to focus on the high level data processing task they are concerned with, without having to worry about how they might integrate into any par-

ticular layout.

Each widget can have a settings panel that a user can use to configure their behaviour. For some settings the default choice will usually be correct (e.g. a map widget should map everything on the page with an address), but for other settings a user is likely to want to set things manually in order to get good results (e.g. which property of an object should be used to decide its icon on the map).

Once a user has created a mashup that they think is useful, they can publish it by clicking on the "share" button. A user will then be prompted to enter a short description of their mashup, and will be shown the preview screenshot that Mash Maker will save with the mashup (Figure 9). Once a mashup has been published, Mash Maker can suggest it to other users.

Figure 9 shows the process of creating a mashup that adds a map to the Facebook friends list. The map shows the location of each of the user's friends based on the "home town" information they provided in their profile. In this example the user opens the expert sidebar, double clicks on two widgets to add them to the page, adjusts their settings appropriately, and then drops the map into an appropriate drop zone on the page. Once the user has created a mashup that they like, they can click the "share" button to make it available to other users looking at similar pages.

UNTRUSTED WIDGETS

Mash Maker has to be particularly careful about security since Mashups have access to private data and users are encouraged to run untrusted mashups without investigating them beforehand.

Since Mash Maker runs inside the browser, it has access to all the information that the browser shows to the user. This includes web pages that require logins, web pages on intranets, and content that is generated dynamically inside the browser. The advantage of this approach is that it allows Mash Maker to mashup useful content that would not be easily accessible to a mashup that ran on a separate server (e.g. Pipes or Popfly). The disadvantage is that some of this information may be private information that should not be leaked to third parties.

If one is not careful then one can easily open doors for hackers to steal confidential data. For example a mashup could scan email messages for passwords and use an external API to send them to a site run by an attacker. In the absence of security controls, any data that is visible on a web site could be scraped by a wrapper and sent to a malicious web site by a widget.

Other browser-extension mashup tools like Greasemonkey and Chickenfoot [5] suffer from this problem to a lesser extent. Unlike Mash Maker, Greasemonkey and chickenfoot do not suggest mashups to users automatically. Instead, their model is more similar to installing desktop software; a user browses a list of recommended mashups hosted on a trusted web site, picks one that seems useful and trustworthy, and installs it. By contrast, Mash Maker encourages users to turn on unvetted mashups written by unknown third parties with little information available about them other than their name.

Mash Maker addresses this problem by distinguishing between *trusted* and *untrusted* widgets. A trusted widget is once that has been checked by the Mash Maker administrators to make sure it cannot leak data to an untrusted server. The choice of which widgets are trusted is subjective. The Google Maps widget is considered to be trusted, even though it sends addresses to Google. If the addresses were highly confidential and Google was considered to be untrusted then one might not want this widget to be applied to a page.

If a widget is not trusted then it is not permitted to see any data that is fetched with cookies or HTTP authentication enabled. This restriction also prevents an untrusted widget being applied to information that another widget fetched with cookies. For example, if a calendar widget inserted information from your personal calendar then this could not be viewed by an untrusted widget. The intention is to restrict an untrusted widget to only be able to see content that it could see if it was running on another machine and prevent it from seeing content that was personalised for the current user. No mashups can be applied to a URL served as HTTPS.

One loophole in the "no cookies" security model is that an untrusted widget will still be able to see content that is private to a local intranet. The correct solution would be to consider any page fetched from a corporate intranet to be a "secure" page that cannot be seen by untrusted widgets, however it is difficult to determine what pages are on the intranet rather than being on the outside web. In particular, simply checking whether a page can be fetched from a remote server is not sufficient as some intranets provide private information on pages that have the same URL as a non-private page that is externally accessible. Mash Maker does not currently have a solution for this problem and so Mash Maker is not recommended for use on corporate intranet web pages.

A Mash Maker widget is implemented rather like a Google Gadget[7]. A widget is a piece of javascript code that runs inside its own iframe, embedded on the page. The browser's same-origin policy prevents a widget being able to directly manipulate the page that is being mashed up. This is in contrast to Greasemonkey which injects mashup scripts directly onto the page.

A Mash Maker widget needs to be able to access data that has been scraped from a web page and needs to be able to share data with other widgets via the shared data tree. This would normally be disallowed by the same-origin policy, so the Mash Maker browser extension extends the browser security model by providing additional API functions that a Mash Maker widget can call to query the data tree or publish additional visualisations. This approach is similar to MashupOS [22] and SMash [26].

[7] http://code.google.com/apis/gadgets

COPY AND PASTE

As a special case, Mash Maker allows users to create mashups by using a copy-and-paste metaphor to combine web sites. To create a mashup that inserts content from site A into site B, the user browses to site A, clicks the "copy" button on the toolbar (Figure 3), and then browses to site B and clicks "paste". For example, to add legroom information to a flight listing, one can browse to a web site that gives legroom information for different airlines, click copy, and then browse to a list of flights and click paste.

Mash Maker will try to guess how to combine the two sites together and create an appropriate mashup. In the current version of Mash Maker, the support for Copy and Paste is fairly simple. Mash Maker will look at the data structures for the two web sites and try to find a matching property that can be used for a simple join. The resulting mashup is implemented by adding an instance of the "paste" widget to the page. If the copy and paste result was not as desired then the user can tweak in using it's settings panel. Subsequent to the release of Mash Maker, this concept has been improved on by Karma [45] which more intelligent techniques to guess how web sites should be combined.

The core idea of using copy and paste to create web sites was inspired by previous work on web clipping tools [31, 41]. While Mash Maker uses web wrappers to extract data from a copied web site, D.Mix [21] takes a more elegant approach by determining an API that could be used to obtain information from the source site. While Mash Maker's copy and paste system combines a pair of pages, Dontcheva et al [13] take a more general approach in which the pages that should be joined together are found using a web search.

REFERENCES

1. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, pages 337–348, 2003.

2. N. Ashish and C. A. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, pages 8–15, 1997.

3. R. Barrett, P. P. Maglio, and D. C. Kellem. How to personalize the web. *CHI*, pages 75–82, 1997.

4. R. Baumgartner, S. Flesca, and G. Gottlob. Supervised wrapper generation with lixto. *VLDB*, 2001.

5. M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *ACM Conference on User Interface Software and Technology (UIST)*, 2005.

6. K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, New York, NY, USA, 2008. ACM.

7. C. Brooks, M. S. Mazer, S. Meeks, and J. Miller. Application-specific proxy servers as http stream transducers. In *WWW*, 1995.

8. C.-h. Chang and S.-c. Lui. Iepad: Information extraction based on pattern discovery. *WWW*, pages 681–688, 2001.

9. B. Chidlovskii. Automatic repairing of web wrappers. In *WIDM*, 2001.

10. V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *Journal of the ACM*, 51:731–779, 2004.

11. V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.

12. P. Denning, J. Horning, D. Parnas, and L. Weinstein. Wikipedia risks. *Communications of the ACM*, 2005.

13. M. Dontcheva, S. M. Drucker, D. Salesin, and M. F. Cohen. Relations, cards, and search templates: User-guided web data integration and layout. In *UIST*, 2007.

14. R. Ennals, E. Brewer, M. Garofalakis, M. Shadle, and P. Gandhi. Intel Mash Maker: Join the Web. *SIGMOD Record*, 36(4):27 – 33, 2007.

15. R. Ennals and D. Gay. User-friendly functional programming for web mashups. In *ICFP '07: Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming*, pages 223–234, New York, NY, USA, 2007. ACM Press.

16. P. T. H. Eugster, P. A. Felber, R. Guerraoui, and A.-m. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35:114–131, 2003.

17. A. Faaborg and H. Lieberman. A goal-oriented web browser. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 751–760, New York, NY, USA, 2006. ACM Press.

18. F. Gasparetti and A. Micarelli. Exploiting web browsing histories to identify user needs. *IUI*, pages 325–328, 2007.

19. D. Gelernter. Linda in context. *Communications of the ACM*, 32, 1989.

20. E. Goer. Swinging through the jungle with Mash Maker and SearchMonkey. *Yahoo! Developer Network Blog*, Oct. 2008.

21. B. Hartmann, L. Wu, K. Collins, and S. R. Klemmer. Programming by a sample: Rapidly creating web applications with d.mix. *UIST*, 2007.

22. J. Howell, C. Jackson, H. J. Wang, and X. Fan. MashupOS: Operating system abstractions for client mashups. In *Proceedings of the 11th USENIX workshop on Hot Topics in Operating Systems*, 2007.

23. D. Ingaiis. Fabrik a visual programming environment. In *OOPSLA*, 1988.

24. U. Irmak and T. Suel. Interactive wrapper generation with minimal user effort. In *WWW*, pages 553–563.

25. D. Kelly and J. Teevan. Implicit feedback for inferring user preference: A bibliography. *SIGIR Forum*, 2003.

26. F. D. Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama. Smash: Secure component model for cross-domain mashups on unmodi ed browsers. In *WWW*, pages 535–544, 2008.

27. D. Koelma, R. Van Balen, and A. Smeulders. Scil-vp: A multi-purpose visual programming environment. In *Applied Computing*, pages 1188–1198, 1992.

28. N. Kushmerick. Wrapper verification. *Word Wide Web Journal*, 2000.

29. N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. *IJCAI*, 1997.

30. A. H. F. Laender, B. Ribeiro-neto, A. S. Da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *Sigmod Record*, 31:84–93, 2002.

31. S. Lingam and S. Elbaum. Supporting end-users in the creation of dependable web clips. In *WWW*, pages 953–962, 2007.

32. G. Little, T. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: Capture, share, automate, personalize business processes on the web. In *CHI*, pages 943–946, 2007.

33. P. Maglio and R. Barrett. Intermediaries personalize information streams. *Communications of the ACM*, 43, 2000.

34. X. Meng, D. Hu, and C. Li. Schema-guided wrapper maintenance for web-data extraction. In *WIDM*, pages 1–8, 2001.

35. I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Autonomous Agents*, pages 190–197, 1999.

36. G. Raeder. A survey of current graphical programming techniques. *IEEE Xplore*, 1985.

37. J. Raposo, A. Pan, M. Alvarez, and A. Vina. Automatic wrapper maintenance for semi-structured web sources using results from previous queries 1. In *SAC*, pages 654–659, 2005.

38. RDFa. `http://www.w3.org/TR/xhtml-rdfa-primer/`.

39. D. Riehle. How and why wikipedia works: An interview with angela beesley, elisabeth bauer, and kizu naoko. *Computers and Society*, pages 3–8, 2006.

40. J. Rucker and M. Polanco. Siteseer: Personalized navigation for the web. *Communications of the ACM*, 40:73–75, 1997.

41. M. C. Schraefel, Y. Zhu, D. Modjeska, D. Wigdor, and S. Zhao. Hunter gatherer: Interaction support for the creation and management of within-web-page collections. In *WWW*, pages 172–181, New York, NY, USA, 2002.

42. Y.-W. Seo and B.-T. Zhang. A reinforcement learning agent for personalized information filtering. In *IUI*, pages 248–251, 2000.

43. K. Simon and G. Lausen. Viper: Augmenting automatic information extraction with visual perceptions. *CIKM*, pages 381–388, 2005.

44. J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR*, pages 449–456, 2005.

45. R. Tuchinda, P. Szekely, and C. A. Knoblock. Building mashups by example. *IUI*, pages 139–148, 2008.

46. T. Tudorache, N. F. Noy, S. Tu, and M. A. Musen. Supporting collaborative ontology development in protege. In *ISWC*, pages 17–32, 2008.

47. J. Wong and J. Hong. Marmite: end-user programming for the web. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1541–1546, New York, NY, USA, 2006. ACM Press.

48. S. Zheng, M. R. Scott, R. Song, and J.-R. WEN. Pictor: An interactive system for importing data from a website. *KDD*, pages 1097–1100, 2008.

49. S. Zheng, D. Wu, R. Song, and J.-R. WEN. Joint optimization of wrapper generation and template detection. *KDD*, pages 894–902, 2007.