Mash-o-matic

Sudarshan Murthy David Maier Lois Delcambre
Department of CS, Portland State University
PO Box 751
Portland, OR 97207-0751, USA
+1 503 725 4061
{smurthy, maier, lmd}@cs.pdx.edu

ABSTRACT
Web applications called *mash-ups* combine information of varying granularity from different, possibly disparate, sources. We describe *Mash-o-matic*, a utility that can extract, clean, and combine disparate information fragments, and automatically generate data for mash-ups and the mash-ups themselves. As an illustration, we generate a mash-up that displays a map of a university campus, and outline the potential benefits of using Mash-o-matic. Mash-o-matic exploits *superimposed information* (SI), which is new information and structure created in reference to fragments of existing information. Mash-o-matic is implemented using middleware called the Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE), and a query processor for SI and referenced information, both parts of our infrastructure to support SI management. We present a high-level description of the mash-up production process and discuss in detail how Mash-o-matic accelerates that process.

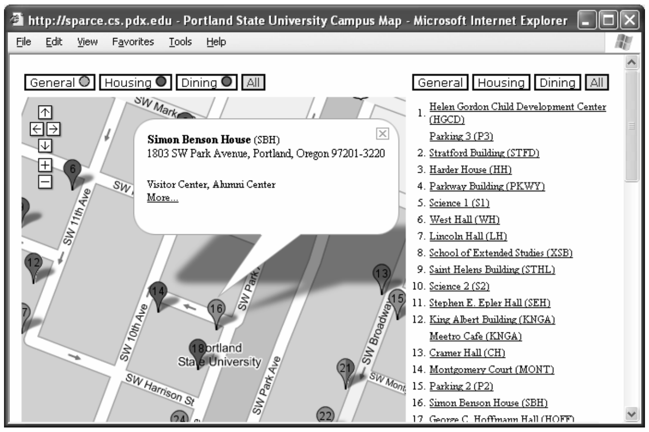
Categories and Subject Descriptors
I.7.2 [Document Preparation]: Multi/Mixed media,
I.7.5 [Document Capture]: Document analysis,
H.2.5 [Database Management]: Heterogeneous Databases,
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Information filtering, Retrieval models

General Terms
Management, Documentation, Design

Keywords
Mash-up, superimposed information, SPARCE, Sidepad, bi-level information, document transformation

1. INTRODUCTION
Mash-ups are web applications that combine information from multiple sources [8]. They often use third-party application programming interfaces (APIs) and display publicly available information in a novel manner. For example, Will James has created a map of the New York City subway system [29] using the Google Maps API [5]. Google Mapki [7] lists other examples.

Figure 1 shows a mash-up displaying a map of the Portland State University (PSU) campus [35]. The data for this mash-up comes from over 50 web resources. On the left is a map displayed using the Google Maps API. On the right is a list of landmarks. A numbered marker on the map represents one or more landmarks. For example, Marker 16 represents the landmark Simon Benson House, and Marker 12 represents two landmarks: King Albert Building and Meetro Cafe. Clicking on a marker in the map or on a landmark in the list pops up an information window above the corresponding marker. For each landmark related to the marker, this information window shows the name of the landmark, the address, an introductory text (such as departments housed), and a link to more information. Figure 1 shows the information window for Marker 16.


http://sparce.cs.pdx.edu - Portland State University Campus Map - Microsoft Internet Explorer
File Edit View Favorites Tools Help
General Housing Dining All
General Housing Dining All
SW Mark
SW 11th Ave
SW 10th Ave
SW Park Ave
SW Broadway
SW Harrison St
SW Mont
Portland State University

Simon Benson House (SBH)
1803 SW Park Avenue, Portland, Oregon 97201-3220

Visitor Center, Alumni Center
More...

1. Helen Gordon Child Development Center (HGCD)
   Parking 3 (P3)
2. Stratford Building (STFD)
3. Harder House (HH)
4. Parkway Building (PKWY)
5. Science 1 (S1)
6. West Hall (WH)
7. Lincoln Hall (LH)
8. School of Extended Studies (XSB)
9. Saint Helens Building (STHL)
10. Science 2 (S2)
11. Stephen E. Epler Hall (SEH)
12. King Albert Building (KNGA)
    Meetro Cafe (KNGA)
13. Cramer Hall (CH)
14. Montgomery Court (MONT)
15. Parking 2 (P2)
16. Simon Benson House (SBH)
17. George C. Hoffmann Hall (HOFF)

Figure 1: PSU campus-map mash-up

The tabs displayed at the top control the kind of information displayed. Tab names reflect types of landmarks. Selecting a tab above the map displays markers that correspond to landmarks of that type. Selecting a tab above the list lists only landmarks of that type. The tab named `All` (selected in Figure 1) displays all markers (and lists all landmarks). In this configuration, clicking on the tab named `Housing` above the list of landmarks will show only housing landmarks in the list, but the map will continue to display all markers.

The sources of a mash-up's data can vary in terms of format (HTML, PDF), structure (different address formats), and location (local file system, the web). Further, data might need to be transformed and elaborated before use. For example, a map-based mash-up needs map coordinates, but sources such as campus directories provide only addresses. Finally, mash-ups are unlikely to use the complete contents of a single source; they tend to use only fragments of a source's contents. Consequently, a mash-up developer might spend considerable effort in information selection and extraction activities.

Mash-o-matic can help a mash-up developer clean and combine extracted data and transform the combined data to the format a mash-up needs. When used in conjunction with a class of applications called *superimposed applications* [31], it can also help him extract and elaborate data from different sources. A *superimposed application* provides a means to reference heterogeneous data of varying granularity, and to superimpose new information (such as annotations) and structures (such as lists) on the referenced information.

In this paper, we provide a brief overview of the mash-up production process and discuss in detail how Mash-o-matic assists mash-up developers in that process. We use the PSU campus-map mash-up shown in Figure 1 to demonstrate the utility of Mash-o-matic.

With this paper, our contributions are: a description of the mash-up production process and a novel means to implement that process. From an implementation perspective, our contribution is a fully functional system to assist in producing map-based mash-ups. In the process, we also demonstrate a sophisticated application of our infrastructure to manage SI and referenced information.

In addition to producing the PSU campus map, we have used Mash-o-matic to produce a detailed map of grocery stores, called the *Portland Metro Food Markets* [34] for the Oregon Department of Agriculture (ODA). Outside our group, Mash-o-matic is being used to map embassies of different countries [Gopalakrishna, personal communication]. (The Mash-o-matic web site, http://sparce.cs.pdx.edu/mash-o-matic, will be updated as new mash-ups become available.)

The outline for the rest of this paper is as follows: Section 2 gives an overview of the mash-up development process and the various components of Mash-o-matic. Section 3 details an implementation of Mash-o-matic to produce map-based mash-ups. Section 4 describes an implementation of Mash-o-matic to produce map-based mash-ups. Section 5 discusses the potential benefits of using Mash-o-matic, customization of generated mash-ups, and the role of SI in generating mash-ups. Section 6 gives an overview of related work, and Section 7 presents some concluding comments.

2. OVERVIEW

We provide here an overview of the mash-up production process and show how the different components of Mash-o-matic fit into this process. We also give an overview of a superimposed application and the parts of our infrastructure for SI management that Mash-o-matic uses.

2.1 The Mash-up Production Process

Figure 2 shows the primary steps involved in producing a mash-up. Dotted arrows in this figure indicate data flow, solid arrows indicate control flow, and the boxes indicate process steps (or mash-up developer activities).

In the Select and Aggregate step, a mash-up developer collects selected information fragments from different sources. For example, he might copy landmark names and addresses from web pages and paste into a text file. In the Organize and Classify step, the developer structures (or restructures) the collected data. He might also elaborate the collected data. For example, he might group landmarks based on postal code. In the Clean and Transform step, the developer refines the organized data and re-purposes it for output in a particular format. For example, he might clean address information and transform the collected information and elaborations into XML format suitable for display on a map. The transformation step might involve use of external services. For example, the developer might use a web service to transform addresses to map coordinates. In the Display step, the developer implements the code needed to display the re-purposed data, possibly leveraging an existing display API. For example, he might use the Google Maps API to display information on a map. Alternatively, he might just invoke previously implemented code.

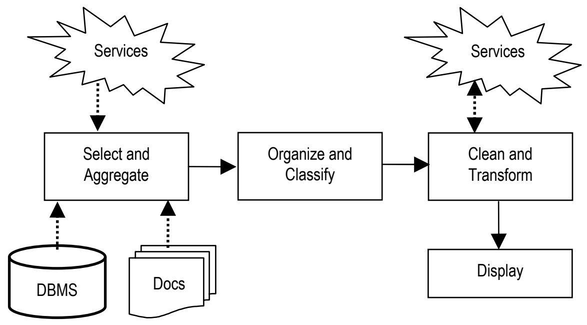


Figure 2: The mash-up production process.

The examples we have used thus far to describe the mash-up production process involve maps, but the process is not specific to map-based mash-ups. For example, one might produce a mash-up that displays information on historical events, collected from online encyclopedias, on a timeline.

A mash-up (specifically, the Display step) need not output information in a graphical manner; it could output text or HTML. For example, Yahoo! News [17] displays news aggregated from different news services as structured text in a web page. Finally, the steps in the mash-up production process might be performed manually, semi-automatically, or automatically, and the first three steps might be performed iteratively before displaying results.

In this paper, we use Mash-o-matic to produce map-based mash-ups. We use a superimposed application called *Sidepad* [33] to collect and organize data, express transformations as XSLT style sheets [19], and display results on a map using the Google Maps API. Section 2.3 mentions examples of mash-ups not based on maps. Section 5.2 discusses customization of Mash-o-matic for mash-ups not based on maps.

The following is the mapping of steps in the mash-up production process to various components of Mash-o-matic (Section 3 provides details):

Select and Aggregate: A superimposed application to identify information fragments and elaborate the fragments.

Organize and Classify: A superimposed application to structure the information fragments referenced and their elaborations.

Clean and Transform: Queries over the information fragments selected and their elaborations.

Display: An appropriate display function or program to present the results of transformation.

When using Mash-o-matic, a mash-up developer may use any toolset to select and organize information, but using a superimposed application enables the developer to more easily select, organize, extract and elaborate information fragments from heterogeneous, distributed sources. It also allows the developer to express transformations declaratively.

2.2 Superimposed Information and Sidepad

Superimposed information [31] (SI) refers to new information such as comments, and new structures such as lists, placed over existing *base information* such as HTML documents. In this setting, a user creates and manages SI in *superimposed applications* (SAs), which in turn use existing *base applications* such as web browsers to *activate* (or show in context) sub-documents, and to retrieve information such as text excerpts for sub-documents without activation.

Our approach to SI uses an abstraction called *mark* [25] to reference sub-documents of any format. We have implemented the mark abstraction for several base information types such as PDF, HTML, Microsoft® Word, and several audio and video formats. Support for new base types can be easily added without affecting existing applications. The mark abstraction is defined and implemented in the *Superimposed Pluggable Architecture for Contexts and Excerpts* [37] (SPARCE), our middleware for SI management.

Sidepad [33] is an SA we have developed using SPARCE and the aforementioned mark implementations. Figure 3 shows a Sidepad document. The boxes labeled `Type`, `Name`, and `Info` are *items*. The boxes labeled `Simon Benson` and `Albert` are *groups*, which are collections of items and other groups. An item has a name and a comment text. For example, in Figure 3, the item labeled `Type` in the group labeled `Simon Benson` has the comment text 'General'; the item labeled `Name` in the same group does *not* have a comment text. An item can also maintain a reference to a sub-document in the form of a mark. For example, the aforementioned item labeled `Name` is associated with a mark to a text fragment inside the HTML page shown in Figure 4. (The use of marks is not visible in Figure 3.)

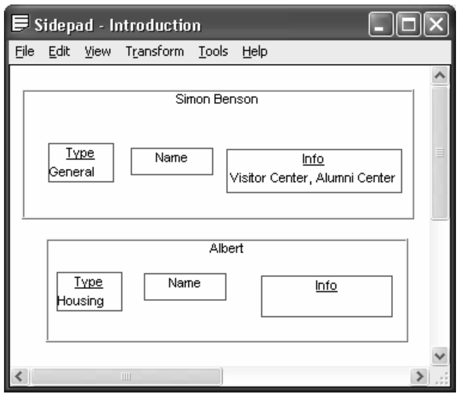
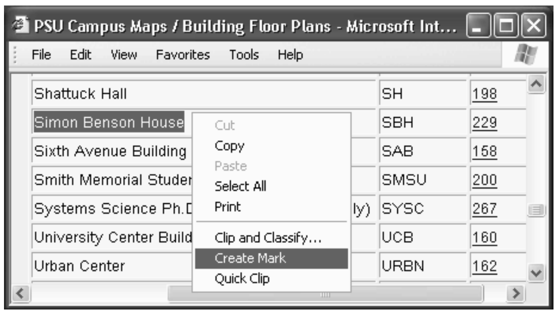


Figure 3: A Sidepad document.

Marks can be created manually using the add-ins we have developed for some base applications. Figure 4 shows a mark to a text selection in an HTML page being manually created. In this case, the user first selects the text to reference and clicks on the Create Mark menu item. He then performs a "Paste" operation in Sidepad, which creates an item associated with the text selection in the HTML page. Marks can also be created programmatically using the SPARCE API. For example, one can develop a script to create a mark into each row or into each cell in the HTML table shown in Figure 4.

Double-clicking an item in Sidepad *activates* the mark associated with that item. That is, it opens the base document in an appropriate base application and "highlights" the referenced sub-document. For example, activating the mark associated with the item `Name` in the `Simon Benson` group results in a screen similar to that shown in Figure 4, except the menu shown is not exposed.

With a mark associated with a Sidepad item, it is also possible to see *context information* such as text excerpt and containing paragraph for the mark from within Sidepad (that is, without activating the mark). Developers can also programmatically access context information using the SPARCE API.

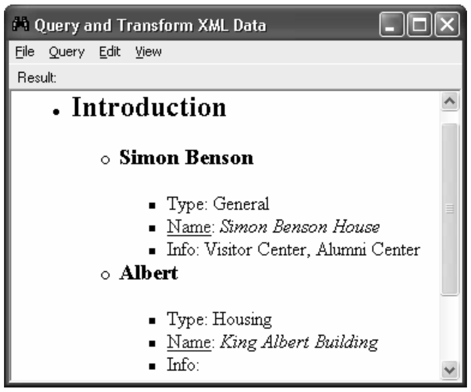

PSU Campus Maps / Building Floor Plans - Microsoft Int...
File Edit View Favorites Tools Help
Cut
Copy
Paste
Select All
Print
Clip and Classify...
Create Mark
Quick Clip

Shattuck Hall	SH	198
Simon Benson House	SBH	229
Sixth Avenue Building	SAB	158
Smith Memorial Studer	SMSU	200
Systems Science Ph.D ... ly)	SYSC	267
University Center Build	UCB	160
Urban Center	URBN	162

Figure 4: Creating an HTML mark.

2.3 Bi-level Queries

We have also developed a facility to query and transform the combination of superimposed and context information. That is, SI such as names and comment texts of Sidepad items can be combined with context information retrieved from marks, and the combined information can be queried and transformed. The combined information is called *bi-level information*, and queries over bi-level information are called *bi-level queries* [36].



- **Introduction**
  - **Simon Benson**
    - Type: General
    - Name: *Simon Benson House*
    - Info: Visitor Center, Alumni Center
  - **Albert**
    - Type: Housing
    - Name: *King Albert Building*
    - Info:

Figure 5: A Sidepad document transformed to HTML.

Figure 5 shows a transformation of the Sidepad document of Figure 3 to a hierarchical view in HTML, obtained using a bi-level query. The Sidepad document is at the root of the hierarchy and each Sidepad group in Figure 3 starts a sub-tree in the hierarchy. Each sub-tree includes the items contained in the corresponding group. For each item, the transformation shows the name of the item followed by the comment text, or the text excerpt retrieved from the associated mark if no comment text exists. (Text excerpts are italicized.) For example, the transformation shows the name and comment text for the item labeled `Type` of the group `Simon Benson`, but it shows the name and the text excerpt for the item labeled `Name` of the same group because that item's comment text is empty and the item is associated with a mark. The URL attached to the text containing an item's name (denoted by an underline) can be used to activate the mark associated with that item.

The transformation in Figure 5 is generated using an XSLT style sheet over an XML representation of bi-level information. Figure 6 shows partial XML bi-level information for the example Sidepad document. It contains a Group element for each Sidepad group and an Item element for each Sidepad item. It uses a Mark sub-element to encode the mark associated with an item, and each Mark element contains an XML representation of the context information for that mark, encoded in the Context sub-element. With this representation, the text excerpt of all referenced information can be retrieved using the XPath [18] expression '`//Mark/Context/Content/Text`'.

We have developed several useful and interesting transformations over bi-level information generated using Sidepad. For example, we have transformed strand maps [20] to the Scalable Vector Graphics format [21], concept maps [38] to the format suitable for the GetSmart concept-map tool [32], and course syllabus information to timelines. These examples and the example in Figure 5 are instances of the mash-up production process outlined in Section 2.1 (Figure 2). None of these mash-ups are based on maps.

The transformations are performed using a *bi-level query processor* we have developed as a part of our infrastructure for SI management. The query processor supports queries over XML bi-level information generated from any SA (not just Sidepad) and the queries can be expressed in XPath, XQuery [23], and XSLT.

```
<Document  name="PSU Campus Map">
   <Group  name="Simon Benson" ...>
     <Item  name="Type"> ... </Item>
     <Item  name="Name" ...>
       <Mark ID="HTMLMark..." ...>
        <Context>
         <Content>
          <Text>Simon Benson House</Text >    ...
         </Content >    ...
        </Context>
       </Mark>
     </Item>
     <Item  name="Info"> ... </Item>
   </Group>
</Document>
```

Figure 6: XML bi-level information generated from a Sidepad document.

3. THE MASH-O-MATIC MODEL

In this section and the next, we describe the use of Mash-o-matic in conjunction with Sidepad. In Section 5 we discuss the use of Mash-o-matic with other SAs and without any SA. Also, in the sequel we use the term "user" to mean a person that uses a mash-up. We use the term "developer" to mean a person that uses Mash-o-matic to create a mash-up.

Figure 7 shows a reference model for Mash-o-matic used to generate map-based mash-ups. Solid arrows in this figure show control dependencies, and dotted arrows show data flow. The shaded area shows components of Mash-o-matic.

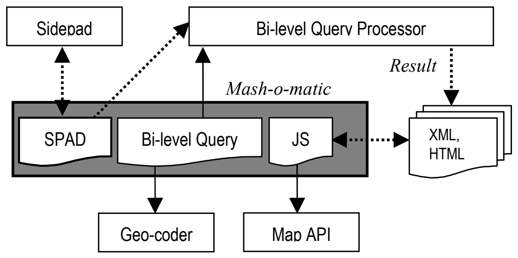


Figure 7: Mash-o-matic reference model for map-based mash-ups.

Mash-o-matic uses a Sidepad document (such as that shown in Figure 9) to collect and organize information (labeled SPAD in Figure 7). It uses a bi-level query to transform the collected information to a format the mash-up requires, and to generate the mash-up itself. The query uses third-part web services to translate addresses to map coordinates (latitude and longitude). Running the query results in four documents: `markers.xml` that contains information about markers (used to populate the left side of Figure 1), `landmarks.xml` that contains lists of landmarks (used to populate the right side of Figure 1), `dataSources.xml` that contains a list of all sources consulted to prepare the mash-up data, and `index.html`, the document a user browses.

The code for the mash-up is in the static JavaScript file `mash-o-matic.js` (labeled JS in Figure 7; the generated HTML document references this script). The script uses the Google Maps API to display maps and markers, and Ajax [27] to read data and to manipulate the user interface. The script also uses a XSLT style sheet to display information windows for markers.

Figure 8 shows the information model of map-based mash-up. As shown, a mash-up has a title, a center (specified as map coordinates), and a default zoom level for the map. It is associated with a set of *markers*, which are locations on a map. (The term *marker* is not to be confused with the term *mark* we introduced in Section 2.2.) A marker has a type (such as `Dining` and `Housing`), map coordinates, and the location of its icon (a file or directory specification). A marker is associated with *landmarks*, which are physical or logical entities with addresses. A landmark has a type, a name, an address, and a few other attributes (described later).

A marker's coordinates and the addresses of associated landmarks are expected to be related. The coordinates for a marker related to only one landmark could be the coordinates for the landmark's address. The coordinates for a marker with more than one landmark could be the coordinates for any one of its landmarks, or it could be the coordinates of a representative (common) area.

Of the six attributes Figure 8 shows for a landmark, only the `Type` and `Name` attributes are required. The `Address` attribute is optional, but we expect it will generally be used. The attributes `Acronym`, `Block`, and `Info` are additional attributes specific to the PSU campus map. We do not require them in order to render the map, display the markers, or list landmarks. Developers can change this set of attributes.

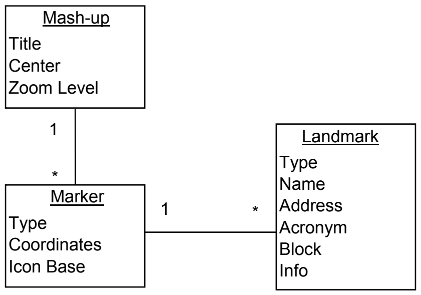


Figure 8: Information model of a map-based mash-up.

4. GENERATING MASH-UPS

In this section, we present the details of generating a mash-up using Mash-o-matic. We use the PSU campus-map mash-up for illustration.

4.1 Collecting Information using Sidepad

The PSU campus-map mash-up uses information from PSU's Architectural, Engineering and Construction Services (AECS) web site [11], the PSU Dining web site [12], Portland Citysearch [10], and web sites of various PSU departments.

The PSU AECS web site provides a listing of landmarks on campus. Figure 4 shows a partial list. For each landmark, the list provides a name, an acronym (which is an AECS-assigned identifier), a block (which is a city-assigned identifier), and a link to a page dedicated to the landmark. The dedicated page provides the address of the landmark, a picture, and a brief description.

Figure 9 shows the partial Sidepad document we used to collect information for the PSU campus map. The full Sidepad document contains a group for each marker on the campus map. Each marker group contains up to three items: one each for type, coordinates, and location of icon. A marker's group also contains a sub-group for each associated landmark. A landmark's group contains six items: one each for type, name, acronym, block, address, and introductory information for the landmark. This superimposed information structure corresponds to the information model in Figure 8. Out of the box, Mash-o-matic is designed to work with Sidepad documents that use this structure. Section 5 discusses the use of other structures.

Figure 9 shows groups for two markers. The highlighted group corresponds to Marker 16 in Figure 1. (The later transformation process automatically numbers markers from north to south.) It contains a sub-group for the landmark 'Simon Benson House'. Only two items in the sub-group for this landmark contain comment text; the other items have only names. The text for such items is obtained from the context of the associated marks (using a bi-level query).

Figure 9 also shows a group named `Config`. This group provides configuration information for the mash-up. The item labeled `Title` provides a title, the item labeled `Center` provides coordinates for the map's center. The item labeled `Zoom` defines the default zoom level. The item labeled `Key` supplies the Google Maps *application key* [5].

All Sidepad groups, except the configuration group, may be named anything, but the items must be named exactly as shown. The names of the groups and items are not visible to users, but the bi-level query that generates the mash-up data expects certain item names.

The complete Sidepad document in Figure 9 contains information for over 50 landmarks, collected from over 50 web pages. The process of collecting this information was relatively easy: We first created a *template group* for a marker, just like the highlighted marker group except that the items in the template are mark-less. We then cloned the template group (that is, copied and pasted) for each marker. For groups with more than one landmark, we cloned just the sub-group for the landmark. We manually created appropriate marks into web pages (see Figure 4) and associated the marks with items.

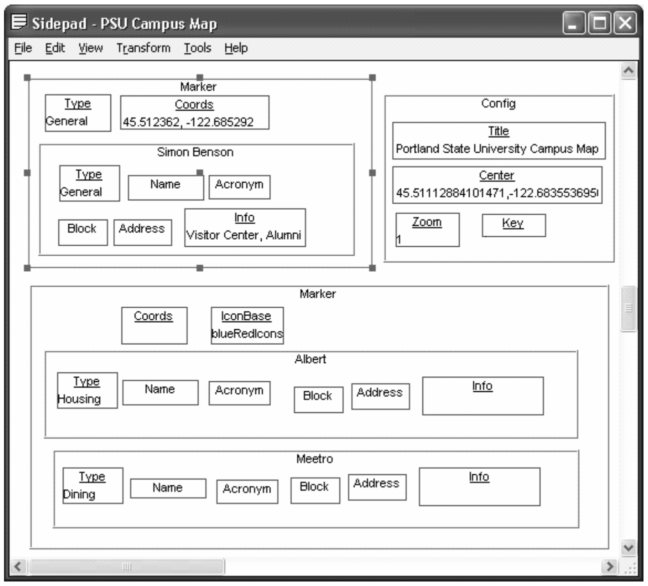


Figure 9: Sidepad document for the PSU campus map.

4.2 Transforming Information

Mash-o-matic uses a bi-level query to transform the Sidepad document of Figure 9 to the XML format a mash-up expects. The query performs four key tasks: it retrieves text excerpts from the marks associated with Sidepad items, computes map coordinates for markers if necessary, transforms the Sidepad data and the text excerpts retrieved to XML according to the schema a mash-up expects, and generates an HTML document containing the UI elements. The bi-level query is a composition of four XSLT style sheets and is processed using the bi-level query processor integrated into Sidepad.

Figure 10 shows a fragment of the XML document `markers.xml` generated by the transformation. The first `Marker` element corresponds to the highlighted Sidepad group in Figure 9. This element shows the details for the landmark 'Simon Benson House' corresponding to the information window shown in Figure 1. The second `Marker` element corresponds to Marker 12 in Figure 1. It shows two `Landmark` elements, one each for King Albert Building and Meetro Cafe. The details of these landmarks are not shown.

The element Config contains the configuration data generated for the mash-up (from the group labeled `Config` of Figure 9). The sub-elements named Marker are generated automatically: The mash-up uses them to create tabs for marker types and to determine icon locations.

The list of landmarks shown in Figure 1 is a transformation over the marker information shown in Figure 10. Because the list of landmarks changes only when the mash-up information changes, this transformation is applied only when the mash-up information is generated, and the results are stored in the document `landmarks.xml`. This document contains one table element for each landmark type. At run-time the mash-up loads the contents of an appropriate table element from this document into the list area (based on which tab above the list of landmarks the user selects). Maintaining the list of landmarks separate from the marker information enables a mash-up to "asynchronously" load the two kinds of information.

The content of an information window for a marker is generated at mash-up run-time using a transformation over the corresponding XML Marker element. For example, the information window shown in Figure 1 is generated by transforming the first `Marker` element in Figure 10. This transformation is performed using an XSLT style sheet named `markerInfo.xslt`. Developers may customize this style sheet.

```
<Mashup>
 <Markers>
  <Marker type="General" lat="45.512362" lng="-122.685292">
    <Landmark type="General" name="Simon Benson House"
        acronym="SBH" block="229">
     <Address>1803 SW Park Avenue, Portland, Oregon 97201-
        3220</Address>
      <Info>Visitor Center, Alumni Center</Info>
      <MoreInfo>www.fap.pdx.edu/…/index.html</MoreInfo>
     </Landmark>
  </Marker>
  <Marker lat="45.512742" lng=" -122.687426" type="Housing"
        iconBase="blueRedIcons">
   <Landmark type="Housing" name="King Albert Building>…
   <Landmark type="Dining" name="Meetro Cafe>…
  </Marker>
 </Markers>
 <Config>
  <Title>Portland State University Campus Map</Title>
  <Center lat="45.51112884101…" lng="-122.68355369567…" />
  <Zoom level="1" />
  <Marker type="General" iconBase="greenIcons" /> …
 </Config>
</Mashup>
```

Figure 10: Fragment of XML mash-up information.

4.3 Geo-coding

Geo-coding, which is the process of obtaining map coordinates, is a key activity when displaying landmarks on a map. Figure 10 shows some coordinates encoded in the attributes lat and lng (for latitude and longitude respectively). We maintain coordinates per marker, not per landmark. We choose this approach because when landmarks are situated too close to each other, plotting each landmark as a marker can make it hard to distinguish them on a map.

Merging landmarks into a single marker is easy with Mash-o-matic. The developer simply needs to move the Sidepad group representing a landmark into an appropriate marker group. For example, the second marker group in Figure 9 contains sub-groups for two landmarks. Transforming the changed Sidepad document to XML automatically completes the process.

The bi-level transformation Mash-o-matic uses *automatically* geo-codes markers, if the Sidepad document does not already specify coordinates. (It uses the address of the first landmark associated with the marker for geo-coding.) It can use either geocoder.us [3] or the Yahoo! geo-coder [16] (the choice is determined by a parameter; other geo-coders can easily be added). Mash-o-matic uses scripts embedded in an XSLT style sheet to integrate with these geo-coders. It uses a helper COM (Component Object Model) class [24] to work-around the script engine's sandbox because neither coder offers a JavaScript interface. (Both services offer interfaces based on Representational State Transfer [26] or REST).

To get an idea of the potential savings using an automatic geo-coder, we geo-coded some landmarks by hand when producing the PSU campus map[1]. To support hand coding, we altered the mash-up to display coordinates of a selected location. The process of hand-coding was simple, but time-consuming: We first located each landmark using Google Local (http://local.google.com) and located it again using the altered mash-up. We updated the Sidepad document with the coordinates the mash-up returned. Coding 33 landmarks took about 3 hours. In comparison, automatic coding takes almost no time: The Yahoo! geo-coder can code about four addresses each second with no user intervention [22]. However, the results do need to be manually verified.

Hand-coding might be necessary in some cases, even when automatic coding succeeds. For example, addresses of landmarks can result in markers too close to each other, but the landmarks might be too significant to be merged into one marker. In this case, it might be acceptable to use coordinates that are slightly off for one of the markers. Again, such changes are easily made in Mash-o-matic. The developer simply enters the hand-coded coordinates for the appropriate marker in the item `Coords`. The transformation process then uses the entered coordinates instead of computing them.

Because the accuracy of marker locations reflect on the quality of the mash-up, we recommend developers check the results of geo-coding on the map the mash-up displays. We recommend they merge landmarks and choose to hand-code based on their needs, and transform the Sidepad document again. We expect the fine-tuning to be complete in two or three iterations.

4.4 Validating Information

The Mash-o-matic transformation process validates the Sidepad document before generating the XML data for the mash-up: It does not generate the mash-up data if there are errors in the Sidepad document (data or structure). Figure 11 shows some example errors. An Error element describes an error. Its type attribute helps developers understand the location of error, and the

[1] We began hand-coding because we had evaluated only geocoder.us, and were not pleased with it. Later we evaluated the Yahoo! geo-coder, but continued hand-coding for calibration purposes.

Description sub-element contains the text of error description. For example, the first error in Figure 11 indicates an error in the Config group of the Sidepad document. An Error element may also include additional helpful information. For example, the last but one error in Figure 11 shows which landmark does not have a type assigned.

```
<Mashup>
  <Errors>
    <Error type="Config">
      <Description>Map center not defined or invalid
      center</Description>
    </Error>
    <Error type="Marker">
      <Description>One or more types of markers do not have
      default icon location</Description>
      <Marker type="General" iconBase="" />
    </Error>
    <Error type="Marker">
      <Description>One or more markers do not have a
      type</Description>
      <Marker type="">
        <Landmark name="Cramer Hall">1721 SW Broadway,
        Portland, Oregon 97201</Landmark>
      </Marker>
    </Error>
    <Error type="Landmark">
      <Description>One or more landmarks do not have a
      type</Description>
      <Landmark name="Art Building">2000 SW Fifth Avenue,
      Portland, Oregon 97201-4907</Landmark>
    </Error>
    <Error type="Geo-coding">
      <Description>Error 6: Possibly invalid address (The exact
      location could not be found, here is the closest match:
      1809 Sw 11th Ave, Portland, OR 97201)</Description>
      <Landmark name="King Albert Building">1809 SW
      Eleventh Avenue, Portland, Oregon 97201</Landmark>
    </Error>
  </Errors>
</Mashup>
```

Figure 11: A sample of errors detected during validation.

The last error in Figure 11 demonstrates an interesting situation in automatic geo-coding. A human could easily see that the address specified and the "closest" match reported are the same. The Yahoo! geo-coder successfully codes this address but generates a warning. Developers can direct Mash-o-matic to ignore such warnings a geo-coder might generate. To reduce the possibility of errors and warnings, Mash-o-matic cleans addresses before geo-coding. For example, it removes new line characters from addresses before coding.

To give developers an idea of the accuracy of the geo-coders we have tried: With warnings turned off, the Yahoo! geo-coder returned acceptable coordinates for 19 of 20 landmarks. (The address of the landmark in question is somewhat oddly assigned.) However, geocoder.us returned acceptable coordinates for only two landmarks: It placed landmarks in the middle of an intersection, or placed them on the wrong side of a street.

5. DISCUSSION

This section discusses the potential benefits of using Mash-o-matic, aspects of customization for mash-ups generated using Mash-o-matic, and the role of SI in Mash-o-matic.

5.1 Benefits of using Mash-o-matic

As described in Section 3, there are three parts to Mash-o-matic: a Sidepad document to collect information, a query to transform bi-level information, and a script to display markers and landmarks. A developer may use any combination of these parts. The benefits obtained vary accordingly.

Table 1 provides estimates on time spent in preparing the PSU campus-map mash-up using Mash-o-matic, and the time spent to develop Mash-o-matic itself (excluding the development of Sidepad and our infrastructure for SI management). The table shows three sets of activities (indicated by dashed lines): producing the PSU campus map, designing the Sidepad document structure and developing the bi-level query, and developing the mash-up code and the UI (the script and the HTML). Effort indicated in the first set of activities would be required to generate any mash-up using Mash-o-matic. The effort shown in the other two sets is potential savings due to using Mash-o-matic.

The first set of activities (Activities 1 and 2 in Table 1) shows the effort to produce the PSU campus map (5.5 hours total). The map consists of 45 markers and 53 landmarks. The information for the map comes from 54 base documents via 188 marks. We collected this information in two hours. We spent 3.5 hours geo-coding, of which 3 hours were spent hand-coding 33 landmarks. Automation saved us the effort to code the other 20 landmarks, but we did spend about 0.5 hours fine-tuning the codes generated automatically. We believe automated coding of *all* landmarks could have helped us develop the campus map in under four hours. Also, we created marks into web pages manually. Programmatically creating marks can potentially save time, but it assumes regions to be marked can be expressed in a script, and there is effort involved in scripting.

Table 1: Estimated time to develop Mash-o-matic and to produce the PSU campus map

Activity	Time (hours)	Remarks
1. Collect PSU information in Sidepad document	2.0	For 53 landmarks
2. Geo-code PSU landmarks by hand	3.5	For 33 landmarks and tuning
3. Queries to convert Sidepad document to XML	2.0	For XML schemas 1 to 4
4. Revise Sidepad document structure	0.5	For XML schema 5
5. Revise queries to convert Sidepad document to XML	1.5	For XML schema 5
6. Add validation	1.0	For XML transformation
7. Automate geo-coding	5.5	Research, code
8. Mash-up development	25.0	Learn, research, code
Total	**41.0**	

The second set of activities (Activities 3-6 in Table 1) help demonstrate a benefit of using Sidepad and bi-level queries to collect, organize, and prepare information for the mash-up. Over the course of its development, the mash-up's schema changed *five* times. The XML data initially contained only elements named Marker (one element per landmark), and all the information related to a marker were encoded as XML attributes. At the end of the development, the XML data contains one Marker element per group of landmarks. A Marker element contains elements named

Landmark, one per landmark at that location. The elements Marker and Landmark use a combination of XML attributes and sub-elements to encode information. The XML fragment in Figure 10 corresponds to the final schema.

With each revision of the mash-up schema, we revised only the bi-level query that generated the XML data, but we used the same Sidepad document (of Figure 9). We did restructure the Sidepad document when going from Schema 4 to Schema 5 (because the last schema needed new information elements). This revision to the Sidepad document took only about 30 minutes (we mostly used Copy and Paste operations).

The third set of activities (Activities 7 and 8 in Table 1) are related to developing the HTML and the JavaScript source, and automating geo-coding. The time to complete these activities is potential savings with each application of Mash-o-matic.

5.2 Customization

We have presented details of generating a mash-up that uses the Google Maps API, but developers might wish to change some aspects of the generated mash-ups.

Geo-coding: Mash-o-matic currently supports the Yahoo! geo-coder and geocoder.us for automatic geo-coding. A developer may use other geo-coders by enhancing the wrapper COM component that interacts with geo-coders, or develop a new COM component, or alter the XSLT style sheet that generates the document `markers.xml`. We recommend using COM wrappers (or other similar technologies) because script engines restrict a script to interacting with data inside a sandbox, potentially limiting the extent and nature of integration with geo-coders. Also, scripts embedded in XML queries can be slow (partly because they are interpreted).

Structure of the Sidepad document: A developer may use a Sidepad document structured quite different from that shown in Figure 9. In that case, he may need to alter the XSLT style sheets that generate the XML mash-up data, and possibly the style sheet that generates the document `index.html`. He may also need to change the XSLT style sheet `markerInfo.xslt` that is used to prepare contents for the information window popped up when a user clicks on a marker.

Structure of the XML mash-up data: A developer may change the structure of the mash-up XML data by changing the appropriate XSLT style sheets. He may also need to change the JavaScript source code.

User interface: A developer may wish to use a different layout for the UI. To do so, he needs to change the document `index.html`. He may also alter the XSLT style sheet that generates that HTML document if he intends to generate many mash-ups with the same UI. The current JavaScript source code looks for three specific elements (using specific element IDs) in the HTML layout. The developer may freely change the HTML layout as long as the three specific IDs are associated with HTML elements.

Mapping tool: Mash-o-matic generated mash-ups can use the Google Maps API and Yahoo! Maps Simple API [15]. A developer may use other mapping tools such as Virtual Earth [14] by changing the JavaScript source code. He may also need to change data structures and the UI.

The Mash-o-matic implementation we have described generates map-based mash-ups, but mash-ups do not need to be map-based. For example, the mash-up *My10Wishes.com* [30] displays information obtained using the Amazon Web Services [2]. Developers may use Mash-o-matic to generate mash-ups not based on maps, but it is likely they will need to change the Sidepad and XML structures, the XSLT style sheets, and the JavaScript source code. That is, Mash-o-matic serves only as an example of a mash-up generating tool in this case. However, the conceptual description of Mash-o-matic offered in Section 2.1 applies to any mash-up.

5.3 Example Customization and Savings

We briefly compare the Portland Metro Food Markets mash-up we produced for the ODA to the PSU campus-map mash-up to help appreciate the ability to customize the mash-up production process and the ability to customize the mash-ups generated.

We developed the ODA mash-up in about 16.5 hours. Table 2 shows the estimated time spent to develop this mash-up. In comparison, we built the PSU mash-up in 5.5 hours. However, the ODA mash-up involves much more data and has richer features. It plots more than 150 grocery stores distributed over eight geographical regions in the Portland metropolitan area.

The data for the ODA mash-up was gathered using a script to process MS Word documents. The script generates an XML document that Sidepad would have generated had we manually populated a Sidepad document. The simulated Sidepad document includes only type, name, and address information for landmarks.

The UIs of the two mash-ups are quite different. Much of the changes in the UI were accomplished using configuration values, and by changing the placement of UI elements in the document `index.html`. We customized the XSLT style sheet used to display information windows for markers.

Both mash-ups use the same bi-level query to generate mash-up data and the same JavaScript code to display data. That is, we did not spend any time to develop transformations or to develop the code to display markers when developing the ODA mash-up. Also, we automatically geo-coded all but two addresses.

Table 2: Estimated time to produce the ODA mash-up

Activity	Time (hours)	Remarks
1. Clean input data	8.0	Select and aggregate data
2. Script to gather data	2.5	
3. Customize `index.html`	2.0	Customize UI
4. Customize `markerInfo.xslt`	1.0	
5. Customize icons	3.0	
Total	**16.5**	

5.4 Role of SI

Using SI to prepare mash-up data can be beneficial, but it is not necessary. Using SI gives a mash-up developer two key benefits: He can superimpose appropriate structures over existing information, and extract only the necessary information from data sources. An SA such as Sidepad provides the former benefit, and a bi-level query processor provides the latter.

The use of Sidepad in collecting and organizing data for a mash-up also deserves some discussion. We have used Sidepad to create a "schema by convention": We have used an item's name as attribute's names, and the item's comment text as that attribute's value. In associating a mark with an item, and using it when the comment text is empty, we have effectively extended the domain of an attribute to include the text excerpt retrieved from the mark. We have used groups to conveniently collect attributes (name and value pairs). That is, we have used a group as a *named tuple*, with the group name being the tuple name. In addition, we have used a hierarchy to distinguish between marker groups and landmark groups: Outermost groups are always marker groups (with the exception of the group named `Config`), and these groups always contain landmark groups. Because Sidepad does not limit the number and names of items inside a group, this schema by convention, allows attributes to be added to or removed from tuples. Also, tuples do not have a fixed schema. For example, the marker groups in Figure 9 have different schemas.

Despite its advantages, Sidepad might not be the right SA for collecting and organizing information for some mash-ups. For example, if the ability to freely locate items and groups spatially is not needed, a simple hierarchical structure (such as that a file system supports over directories and files) might suffice. Also, based on our experience developing the PSU campus-map mash-up, Sidepad documents with information for over 50 landmarks can be unwieldy. However, using a different SA does not adversely affect Mash-o-matic because our bi-level query processor can work with any SI represented as XML.

Mash-up developers do not need to use an SA at all. They can use the JavaScript source code, the HTML document, and the XML structures of a Mash-o-matic generated mash-up as a template for their mash-ups. However, they would need to manage and validate the HTML and XML documents *manually*, or using means other than Mash-o-matic. A developer can use the bi-level query processor even if he does not use an SA because the processor can also work with traditional XML data, or he can use any traditional XML query processor.

6. RELATED WORK

The Yahoo! Maps Simple API [15] provides a REST-based service to display markers on a map hosted in an HTML page. To display landmarks, a developer submits landmark information in the GeoRSS [4] format. (Mash-o-matic is able to generate GeoRSS documents.) The service then returns an HTML page containing a map with markers for each landmark. However, the format of the resulting HTML page is fixed by the service. Yahoo! Maps does provide a JavaScript API, which gives the developer control over the user interface. Mash-o-matic can be customized to work with this API (as described in Section 5.2).

Several web sites are available for interactive map building. For example, Mapbuilder.net [6] and Platial.com [9] allow users to place markers on a map by pointing at a location on a map (or by specifying a street address). They allow users to label markers and attach notes to them. Platial.com also allows attaching video clips. Both systems allow collaborative map creation and maintenance. In contrast to Mash-o-matic, both systems fix the structure of data associated with maps and markers. They also fix the UI. The systems are *not* mash-ups because they do not gather data from existing sources; users enter data which the systems then display on maps.

Three representative systems presented in past ACM symposia on Document Engineering [1] related to Mash-o-matic are: wVIEW, the Universal Parsing Agent (UPA), and the harmonization and annotation services in the VIKEF framework.

The wVIEW system [39] generates web applications given content, navigation, and presentation models expressed in XML. It uses XSLT to process mapping rules within and among the models, and to generate the web applications themselves. A generated web application is in the form of XML documents and related scripts. In Mash-o-matic, a Sidepad document provides the content model for a mash-up, and the mapping rules are evaluated using an XML query language (although we have used only XSLT in this paper). Mash-o-matic does not use navigation and presentation models, but those models can be introduced easily. The wVIEW system does not extract information from different sources, but doing so is not its stated goal.

The UPA [40] provides a means to specify and extract document fragments using regular expression and natural language patterns. The extracted fragments can be labeled automatically and output as XML, which can then be transformed using XSLT style sheets. However, the UPA does not retain a reference to the source of an extracted fragment. Using the UPA is similar to programmatically generating marks (and possibly labeling) when collecting data for a mash-up. The XML output from UPA is similar to bi-level information, but it is not exactly *bi-level* because the absence of references blurs the line between the added label and the extracted data. However, UPA's XML output can potentially be transformed into the format a mash-up requires.

The VIKEF framework [28] uses natural-language processing algorithms to automatically identify document fragments for semantic annotation. The fragmentation procedure identifies *all* possible fragments of a given document and attaches semantic annotations to the fragments. The annotations, the text of document fragments (that is, excerpts), and the references to document fragments are stored in the Resource Description Framework [13] (RDF) format. The RDF data may then be queried in a manner similar to bi-level querying. It is conceivable that the semantic annotations might be used to identify information such as names and addresses, and potentially be transformed to mash-up data.

For brevity, we omit work related to SI and bi-level queries as they are covered in our past publications [25, 31, 36, 37].

7. CONCLUSION

Mash-o-matic illustrates information-management aspects of a new class of applications. It demonstrates how a variety of technologies, components, and information can be integrated to produce non-trivial data-driven web applications.

Mash-o-matic is freely available for non-commercial use. At the time of this writing, the third-party components used in developing Mash-o-matic are free, at least for non-commercial use. We urge readers to consult individual vendor's licensing policy before employing Mash-o-matic.

In future, we like to consider ways to make Mash-o-matic easier to use. For example, developers might benefit from an SA that can facilitate collection of information for a large number of landmarks (of the order of hundreds).

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. This work was supported in part by the National Science Foundation grants 0511050 and IIS-0534762.

9. REFERENCES

[1] The ACM Symposium on Document Engineering. http://www.documentengineering.org.

[2] Amazon Web Services. Amazon.com. http://www.amazon.com/gp/aws/landing.html.

[3] geocoder.us. Locative Technologies. http://geocoder.us.

[4] GeoRSS: Geocoded RSS Basics. http://worldkit.org/doc/rss.php.

[5] Google Maps API. Google. http://google.com/apis/maps.

[6] Mapbuilder.net. http://www.mapbuilder.net.

[7] Mapki. http://www.mapki.com.

[8] Mashup. Wikipedia. http://en.wikipedia.org/wiki/Mashup.

[9] Platial.com. http://www.platial.com.

[10] Portland Citysearch. IAC/InterActiveCorp. http://portland.citysearch.com.

[11] PSU Campus Maps and Building Floor plans. PSU AECS. http://www.fap.pdx.edu/floorplans.

[12] PSU Dining Services. PSU Dining Team. http://www.psudining.com/.

[13] Resource Description Framework. W3C. http://www.w3.org/RDF.

[14] Virtual Earth Standard Control. Microsoft Corporation. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/VEMCSDK/HTML/Introduction.asp.

[15] Yahoo! Maps Web Services. Yahoo! Inc. http://developer.yahoo.com/maps.

[16] Yahoo! Maps Web Services - Geocoding API. Yahoo! Inc. http://developer.yahoo.net/maps/rest/V1/geocode.html.

[17] Yahoo! News. Yahoo! Inc. http://news.yahoo.com.

[18] XML Path Language (XPath) Version 1.0. 1999. http://www.w3.org/TR/xpath.

[19] XSL Transformations (XSLT). 1999. W3C. http://www.w3.org/TR/xslt.

[20] Atlas of Science Literacy. 2001. Washington DC: American Association for the Advancement of Science and the National Science Teachers Association.

[21] Scalable Vector Graphics (SVG) 1.1 Specification. 2003. W3C. http://www.w3.org/TR/SVG.

[22] Comparing Geocoders: Ontok Geocoder, geocoder.us, Teleatlas and Yahoo Geocoder. 2005. Ontok Geocoder. http://www.ontok.com/geocode/compare.

[23] XQuery 1.0: An XML Query Language. 2005. W3C. http://www.w3.org/TR/xquery.

[24] Brockschmidt, K. Inside OLE 2. 1994: Microsoft Press.

[25] Delcambre, L., *et al*. Bundles in Captivity: An Application of Superimposed Information. In proceedings of ICDE 2001. 2001. Heidelberg, Germany. p. 111-120.

[26] Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures. 2000 http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[27] Garrett, J.J. Ajax: A New Approach to Web Applications. http://adaptivepath.com/publications/essays/archives/000385.php.

[28] Jacquin, T., Fambon, O., Chidlovskii, B. A web-based document harmonization and annotation chain: from PDF to RDF. In proceedings of the 2005 ACM symposium on Document engineering. 2005. Bristol, UK.

[29] James, W. OnNYTurf. http://www.onnyturf.com/subwaymap.php.

[30] Krasilshik, L. My10Wishes.com. http://bowgett.com/my10wishes.

[31] Maier, D., Delcambre, L. Superimposed Information for the Internet. In proceedings of WebDB 1999. 1999. Philadelphia, PA. p. 1-9.

[32] Marshall, B., *et al*. Knowledge Management and E-Learning: the GetSmart Experience. In proceedings of 3rd ACM and IEEE Joint Conference on Digital Libraries (JCDL-2003). 2003. Houston, TX.

[33] Murthy, S. Sidepad User Guide. 2005. http://sparce.cs.pdx.edu//apps/Sidepad/userguide.

[34] Murthy, S. Portland Metro Food Markets. http://sparce.cs.pdx.edu/mash-o-matic/oda-1.1.

[35] Murthy, S. Portland State University Campus Map. http://sparce.cs.pdx.edu/cmap.

[36] Murthy, S., Maier, D., Delcambre, L. Querying Bi-level Information. In proceedings of 7th International Workshop on the Web and Databases. 2004. Paris, France.

[37] Murthy, S., Maier, D., Delcambre, L., Bowers, S. Putting Integrated Information in Context: Superimposing Conceptual Models with SPARCE. In proceedings of First Asia-Pacific Conference of Conceptual Modeling. 2004. Dunedin, New Zealand. p. 71-80.

[38] Novak, J.D., Cañas, A.J. The Theory Underlying Concept Maps and How to Construct Them. http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf.

[39] R. de Andrade, A., Munson, E.V., Pimentel, M.G. A document-based approach to the generation of web applications. In proceedings of 2004 ACM symposium on Document engineering. 2004. Milwaukee, Wisconsin, USA.

[40] Whiting, M.A., *et al*. Enabling massive scale document transformation for the semantic web: the universal parsing agent. In proceedings of 2005 ACM symposium on Document engineering. 2005. Bristol, UK.