Mashroom: End-User Mashup Programming Using Nested Tables

Guiling Wang[1], Shaohua Yang[1,2], Yanbo Han[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences
Beijing, 100190, P.R. China
{wangguiling, yhan}@software.ict.ac.cn
[2]Graduate University of Chinese Academy of Sciences
Beijing, 100190, P.R. China
yangshaohua@software.ict.ac.cn

ABSTRACT
This paper presents an end-user-oriented programming environment called Mashroom. Major contributions herein include an end-user programming model with an expressive data structure as well as a set of formally-defined mashup operators. The data structure takes advantage of nested table, and maintains the intuitiveness while allowing users to express complex data objects. The mashup operators are visualized with contextual menu and formula bar and can be directly applied on the data. Experiments and case studies reveal that end users have little difficulty in effectively and efficiently using Mashroom to build mashup applications.

Categories and Subject Descriptors
D.2.13 [**Reusable Software**]: Reuse models; H.5.m [**Information Interface and Presentation**]: Miscellaneous.

General Terms
Design, Human Factors, Languages

Keywords
Mashup, End-user programming, Spreadsheet, Nested table.

1. INTRODUCTION
The involvement of end users has been an essential driving force of the Web-related development. Recently, mashup has become a trend, which allows non-professional users to build Web applications by combining functionalities offered by more than one websites to deal with situational and ad-hoc problems. Software tools called mashup editors for constructing the new type of integrated applications, such as IBM Damia [1], Yahoo Pipes [2], Microsoft Popfly [3], Intel MashMaker [4,5] and CMU Marmite [6], have boomed. Consider the situation in which a user wants to get the state-of-the-art information of on-play movies and the related movie reviews, but seldom can an existing web site provides an integrated view of that information from various web sites. Rather than having to copy, paste and edit the movie list and the movie review list in a separate document, the mashup editors make it easy for end-users to build an application that can extract, combine the information, and generate the integrated view without the tedious manual work.

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

Popular mashup editors like IBM Damia, Yahoo Pipes, Microsoft Popfly employ flow-chart-like formalisms. However, studies [6] show that the concept of data flow is the main barrier for accomplishing a mashup task. Users are often puzzled by the alignment of inputs and outputs and the particular ordering of operators. In order to alleviate the difficulties of end users in understanding the flow-chart-like formalisms, we adopt the idea from spreadsheet programming, which supports visualization and direct-manipulation of data. The problem is that the traditional two-dimensional spreadsheet paradigm, such as MS Excel, does not allow direct manipulation of complex objects, such as RSS/Atom and XML or JSON results returned from RESTful Web Services. Once such data is imported and represented in a spreadsheet, they become a collection of atomic values in cells. All the operations are applied on the cells, and it is not convenient to manipulate or compose a complex object.

In this paper, we propose Mashroom, a mashup tool with a novel programming model. The key innovation of Mashroom lies in that it takes the nested table as the data structure and formally defines a set of visual mashup operators to offer a spreadsheet-like programming experience. Case studies are made, showing that Mashroom can provide a new way for end users to build the common mashups effectively and efficiently.

The rest of this paper is organized as follows. Section 2 explains what have influenced the Mashroom design. Section 3 presents the Mashroom programming model including the definition of the application structure, the data model and the mashup operators. Section 4 introduces a trail application. Section 5 evaluates Mashroom with experiments. Section 6 reviews related works. Finally, the paper concludes in Section 7.

2. FOUNDAMENTALS OF MASHROOM PROGRAMMING
In this section we explain how inspiration is drawn from nested table, spreadsheets, and end-user programming, and sketch the overall design philosophy of the Mashroom programming.

2.1 Uses of Nested Table
The nested relational model was first proposed in 1977 [7], also called NF^2(non-first normal form). It allows relations to have relation-valued attributes and is one of the most adopted data model for representing semi-structured web data. It has been successfully utilized in web data extraction applications [8] and has been implemented directly in some modern DBMSes, such as Oracle. The reason is that the nested relational model is simple, intuitive, and expressive enough to represent the semi-structured data commonly found in Web pages [9].

We adopted the nested relational model as the basic data model and designed the Mashroom editor's graphical user interface (GUI) based on the nested table out of the following reasons:

- Compared with the traditional 1NF relational model, the nested relational model is intuitive and closer to the real world because a whole complex object is not distributed over several different relations in the nested relational model. This model has a strong foundation both in query algebra and in query optimization with years of research and industry practice.
- In the nested relational model, data is easy to be represented as nested tables, which are understandable by end users.

For developing mashups, a set of high-level and task-oriented mashup operators and corresponding graphical mashup language are needed. There exist the following potential challenges:

- For end users, data query for mashups should be organized in such a way that human visual consumption can be facilitated. There are some research works focusing on graphical query languages based on the nested table such as QSByE [10] and GXQL [11]. However, the existing graphical query languages are designed strictly in a low-level database query manner. They lack the critical feature of building the dataflow into an individual operator. For example, the loop operation can't be described in a black-box and hidden from end-users. And data dependency can't be supported intuitively.
- For mashup creation, not only the data query operators are needed but also operators that facilitate data visualization on a view (e.g., a map, table, timeline, etc.) or asynchronous notification such as sending an E-mail are also needed.

2.2 Uses of Spreadsheets

Spreadsheets have achieved remarkable success in allowing non-programmers to represent complex data and perform certain computation tasks. One of the key advantages of spreadsheets is their "low entry barrier" programming paradigm. It requires little time and skill before end users are rewarded by simple but functioning programs that model their problems of interest. As noted, spreadsheets suggest that "a limited set of carefully chosen, high-level, task-specific operations and a strong visual format for structuring and presenting data are key characteristics for user programming environment" [12]. If mashup editors inherited the key characteristics of spreadsheets, such as low entry barrier, mixing values with expressions, carefully chosen operations and strong visual format, they would help end users in reducing the complexity and improving user experience in building mashups.

Specifically, Mashroom borrows the following features from spreadsheets:

- In addition to visual menu and direct manipulation of data, formula bar allows users to manipulate data by flexibly editing a formula.
- Loop execution is described by selecting a range of cells intuitively.

To adopt spreadsheets programming for building mashup, we introduce several modifications to the traditional spreadsheet programming paradigm.

- The traditional spreadsheet consists of a two-dimensional array of cells. It is unsuitable for displaying nested table. We designed a view structure for the nested table to display the nested relational data, and besides, we make some modification on the screen layout (see Section 3.4.1).
- We defined a set of mashup operators on the nested relational data model. Using the mashup operators, users can import a service as a table, drag and drop one column onto the other for merging, invoke and link another service directly on a range of rows in an iterative manner, and so on.

2.3 Learning from End-User Programming

Research and practice on end-user programming started as early as the 50's of the last century [13]. Programming by Example or Programming by Demonstration, as an important research thread in end-user programming, has been shown to help end users create programs without coding [14,15]. These systems let the user demonstrate the desired program by going through the steps on an example, then generate an instantiable operation sequences or infer the underlying application logic from the demonstration process.

Following the idea of "Programming by Example", as illustrated in Figure 1, at build-time, Mashroom lets users demonstrate the mashup logic by querying and composing the example nested table, then generate the mashup script through parameterizing the user instructions. At run-time, the mashup script can then be re-applied on other web resource instances.

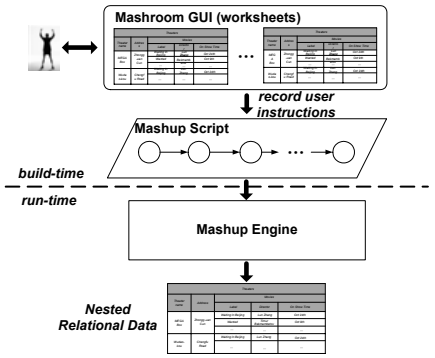


Figure 1. Programming by Example in Mashroom

3. MASHROOM PROGRAMMING MODEL

3.1 The Structure of a Mashup Application

The structure of a mashup application is depicted in Figure 2. The basic concepts of the mashup application structure include Web Source, Data Service, Composite Data Service, Presentation View and Mashup View. A mashup application, or a web/mobile widget, is a Data Service associated and configured with a presentation view for presenting the underlying data.

Web Source: Web Sources are the information resources on the Web in the format of HTML, RSS/Atom or RESTful Web Service.

Data Service: Web Sources become Data Services through encapsulation. A Data Service can be denoted as a tuple *DS = <id, name, uri, encoding, params, schema, desc>*, where *id* is the identification, *uri* is the web access address of the Web Source, *encoding* is the encoding format of the underlying data, *params* is the input parameters, *schema* is the data schema of the service's

output represented as the nested relational model and *desc* is the description of the data service. A Data Service wraps a Web Source which can be various formats and provides a uniform data model (Section 3.2) and a uniform data access interface. When a data service is associated and configured with a presentation view, it becomes a widget or mashup application.

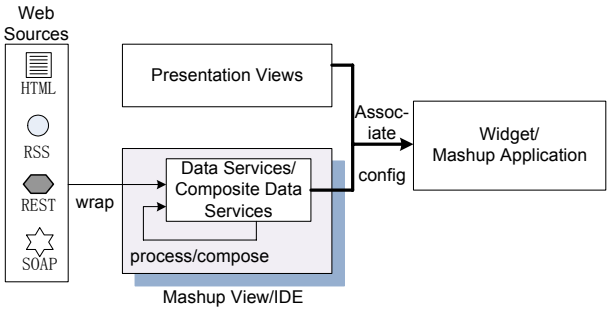


Figure 2. The Structure of a Mashup Application

Composite Data Service: Data Services can be processed or/and composed into a new Composite Data Service. A Composite Data Service can also be associated and configured with a presentation view to form a widget or mashup application. A Mashup view is a user interface (UI) for editing a mashup. Data Services can be processed and composed into a Composite Data Service using the mashup view. There are various ways to design a mashup view. For example, the mashup view of Yahoo! Pipes is designed based on a flow-chart-like graphic UI. In Mashroom, the mashup view is designed based on nested table UI.

Presentation View: A Presentation View is used to present the output data of a (Composite) Data Service. Data Services can have various presentation views, such as list view and table view for multiple-record data, map view for geography data and gallery view for photography related data.

3.2 Data Model

In Mashroom, we use the nested relational model as the underlying data model of Data Services. For the definition of nested relational model, please refer to paper [16]. In the nested relational model, data is represented as nested tables. Figure 3 shows an example nested table. The "***Theaters***" relation specifies a list of theaters located in a certain region. It has two atomic attributes ("***Theater name***" and "***Address***") and a sub-relation named "***Movies***". The sub-relation "***Movies***" describes the movies that are on show at each theater.

Nested table offers simple and intuitive access to underlying data sources. For the example above, data related to movies is represented directly as a sub-relation of "*Theaters*", rather than distributing over another distinct table. Thanks to the recursive algebra as the foundation of query language for nested table [16], developers can access and manipulate the sub-relation data directly, without having to restructure the un-nested relation. For example, the recursive selection σ *(Theaters(Movies*$_{Label="Waiting\ in\ Beijing"}$*))* can be used directly to access theaters where a certain movie is on show.

Theaters					
Theater name	*Address*	*Movies*			
		Label	*Director*	*On Show Time*	
MEGA Box	*Zhongguan Cun*	*Waiting In Beijing*	*Lun Zhang*	*Oct 24th*	Nested Table Instances
		Wanted	*Timur Bekmambetov*	*Oct 9th*	
		...	...	...	
Wudao-kou	*Chengfu Road*	*Waiting In Beijing*	*Lun Zhang*	*Oct 24th*	
		...	...	...	

Nested Table Schema

Figure 3. An Example Nested Table

In Mashroom, the underlying data of a mashup application is also called **MashSheet**, which consists of a set of **worksheets**. A worksheet is in fact a nested table, in which each column represents an atom attribute or a sub-relation, and each row represents a tuple. An atom attribute of a worksheet can be one of six types: *text*, *textlink*, *img*, *imglink*, *video*, *videolink*.

Different from the traditional spreadsheet where "cell" is the first-class object, MashSheet takes "column" as the first-class objects. The syntax of Mashroom formula is built from column references, operators (defined in section 3.3), and constant values. Because the schema of a nested table can be equivalently viewed as a tree, the syntax of column references is built from the path expression on the schema tree corresponding to the nested nature of the data model. For example, a formula *theaters/movie/director* specifies the "*director*" column of the relation "*movies*" which is a sub-relation of the relation "*theaters*".

3.3 Data Mashup Script and Operators

In Mashroom's mashup editor, a set of actions can be applied on a group of worksheets. Mashroom records the sequences of user instructions in a construct called **Data Mashup Script,** which describes the generalized logic of how Data Services can be processed and composed into a Composite Data Service. After the user finishes data services processing and composition, the script is built into a Composite Data Service for later instantiation with other parameter values.

A Data Mashup Script *script* can be modeled as a dataflow graph in a data flow computation way. The dataflow graph is a directed acyclic graph of nodes and edges. The nodes consume and produce data tokens along the edges. In a similar way, *script* can be represented as a directed acyclic graph of operators (*ops*) as nodes and variables (*vars*) as edges. Each *script* is associated with a subset of *vars* that are global input parameters. Therefore, a script can be represented as a tuple *script* = <*vars*, *ops*>. Where each operator in *ops* can be represented as a tuple *op* = <*actor*, *actorIn*>, where actor is an encapsulation of the computation that computes on nested tables from a set of input variables. The output variables of the script or operator are all atomic attributes, relation or sub-relations of the MashSheet, so it is not necessary to specify the output variables for each operator.

Table 1 shows the entire set of operators in Mashroom and their corresponding operations of the nested relational model. Detailed definitions of the operators are as follows:

Import

The *Import* operator inserts a new worksheet into the current MashSheet by importing a Data Service. When applying this operator, data from the specified service is fetched dynamically and displayed in a worksheet with a unique ID.

Table 1. Operators in Mashroom

Types	*Operators*	*Nested Relation Model Operation*
Worksheet Creation	*Import, CreateSheet*	*New, Insert*
Worksheet Data Manipulation	*Filter, Sort, HeaderTruncate, TailTruncate*	*Selection(σ)*
Worksheet Schema Manipulation	*DeleteColumn, RenameColumn, Nest/Unnest*	*Delete,Update, Nest(η),Unnest(μ)*
Worksheet Cleaning	*AddFunction, MergeInstance*	*Insert, Selection(σ)*
Worksheet Composition	*Merge, Fuse, LinkService*	*Union(∪),Join(⋈)*
Worksheet Export	*Sink*	—

This operator is defined as a formula: *import(sid, mapping, …)* , where *sid* specifies which data service to import and a *mapping* indicates a parameter assignment denoted by a tuple *<param, style, ref>*. An *import* operator can take more than one *mapping* according to the number of parameters of the specified service. The fields of the *mapping* tuple are described as follows: 1). *param* indicates the name of the parameter of the imported service to be assigned; 2). *style* can be set as one of the two constant variables: CONSTANT and MASHUP_PARAM, which indicate the manner of parameter assignment. 3). *ref* has different semantic meaning that depends on the value of *style*. When mapping *style* is CONSTANT, *ref* represents a constant value. When mapping *style* is MASHUP_PARAM, *ref* represents a parameter reference of the target composite Data Service.

CreateSheet

The *CreateSheet* operator is to create a new worksheet by copying a sub-relation or an atomic attribute from one of the current worksheets. The corresponding formula for this operator is: *createSheet(col)*, where *col* can be a sub-relation or an atomic attribute. For example, *createSheet(theaters/movies)* will create a new worksheet which presents the *movies* relation.

Filter

The *Filter* operator is to filter a worksheet with certain condition. It can be defined as a formula like this: *filter(col, conditions)*, where c*ol* can only be the name of a relation or sub-relation, *conditions* is a group of condition expressions defined on the atomic attributes of this sub-relation. This operator is equivalent to the recursive selection operation of the nested relational model.

Sort

The *Sort* operator is to sort tuples in a worksheet according to values of a certain attribute. It can be defined as a formula like this: *sort(col, order)*, where c*ol* can only be the name of atomic attribute, *order* indicates the sorting order: ascending or descending. This operator corresponds to a recursive selection operation with an "order by" clause for the nested relational model.

HeaderTruncate/TailTruncate

The *HeaderTruncate* and *TailTruncate* operator are to truncate the unwanted instances defined as a formula *header(count) and tail(count),* where *count* is the number of rows that the user wants to keep back. It can be very helpful when there are too many rows in a worksheet and when one wants to focus on the rows at the front or at the back.

DeleteColumn/RenameColumn

The *DeleteColumn/RenameColumn* operators are to delete a column from a worksheet or rename it. It can be defined like this: *delete(col)* and *rename(col, name)*, where *col* can be an atomic attribute or sub-relation, *name* is new name of the specified column. This operator is equivalent to the "delete" operation and "update" operation of the nested relational model.

Nest/Unnest

The *Unnest* operator is to convert a nested relational model into a traditional relational model and the nest operator packs the original relation into a nested form. The *Nest* and *Unnest* operator can be expressed by a formula *nest/unnest(col)*, where *col* is a relation or sub-relation of a worksheet.

AddFunction

The *AddFunction* operator allows users to apply the arithmetic, statistical and string functions to the data of the MashSheet and create a new column with the computation results. This operator is defined as a formula built from column references, row references, arithmetic functions (e.g., "+", "-", "*", "/", "*int*", "*mod*", "*abs*"), statistical functions (e.g., "*sum*", "*avg*", "*count*"), string functions (e.g., "*copy*", "*upper*", "*lower*", "*replace*") and constant values.

MergeInstance

The *MergeInstance* operator selects the distinct instances by grouping an atomic attribute. It is defined as a formula like this: *mergeInstance(col)*, where *col* can only be an atomic attribute.

Merge/Fuse

The *Merge* operator merges two worksheets. The corresponding formula like this: *merge(a, b, <a.X, b.Y>, …)*, where both *a* and *b* can only be a relation or sub-relation, *X* and *Y* are atomic attributes of relation/sub-relation *a* and *b*. This operator is equivalent to the recursive union operation of the nested relational model. If the names of the atomic attributes from two sub-relations are not the same, a dialog is popped out and let the users map the attributes.

The fuse operator is defined as a formula *fuse(a, b, <a.X, b.Y>, …)*, where *a, b* can only be a relation or sub-relation and *X, Y* have the same semantics as they are in *Merge* operator. This operation not only merges *a, b* (the same semantics as they are in *merge* operator), but also merges the instances of *a* (the same semantics as they are in *mergeInstance* operator).

LinkService

The *LinkService* operator imports a Data Service in a different way from the *import* operator stated above. Here the Data Service is dependent on the current worksheet. The *LinkService* operator can be defined as a formula like this: *linkservice(sid, mapping,…)*, where *sid* is the service unique identification, *mapping* is a tuple *<param, ref, style>*, *param* is the parameter name of the imported Data Service, *ref* is the value of this parameter, *ref* is the value of this parameter. Here the *style* can be of CONSTANT and TYPE. For the CONSTANT style, *ref* should be a constant value. For the TYPE style, *ref* should be a reference to one of the atomic attributes of the current sub-relation. It specifies that the value of the atomic attribute will be assigned to the input parameter of this Data Service for invocation. When the operator is executed, the service is invoked directly on a range of rows in an iterative manner. After the operation is executed, the output of the Data Service is linked into this worksheet.

For example, a movie search site like http://shenghuo.google.cn provides a movie list for a given city. A movie review service provides movie reviews for a given movie with the name as the

input parameter. The *LinkService* operator would get the reviews for each movie in the movie list and join with the movie list in an iterative way. The illustration is given in Figure 2 and will be discussed later. Without this operator, separate join would have to be done after invoking the movie review service for each given movie.

Sink

The sink operator is defined as a formula *sink(option, mapping, …)* , where *option* indicates which format the user is going to export the current mashup as, *mapping* is a tuple *<attr, type>,* where *attr* is the atomic attribute of the worksheet, *type* is the element type of the output data. For example, the user can display a nested table with "longitude" and "latitude" attributes on an interactive map simply with just a formula like this: *sink("map", <longitude, lng>, <latitude, lat>)*. Mashroom supports the MashSheet to be exported as a CSV file, a list/table view, an interactive map, an image gallery, an email message or a SMS message. The advanced users also can create a new kind of sink template.

3.4 User Interface

Mashroom is currently implemented as a Firefox extension as shown in Figure 4. We make use of a toolbar icon to enable users to discover the Web Sources while browsing the Web pages. Mashroom validates the web sources as RSS, Atom, RESTful or HTML Web Sources. For RSS, Atom, RESTful Web Services, users can directly save them in the Data Service list. For HTML Web Sources, Mashroom provides a "screen scrapping" interface for users to label which attribute (column) they want to extract (like Dapper [18]). The implementation details of this interactive HTML wrapper will be discussed in other papers of ours [19]. We make use of a sidebar to enable users to manage the list of Data Services and Mashup Scripts.

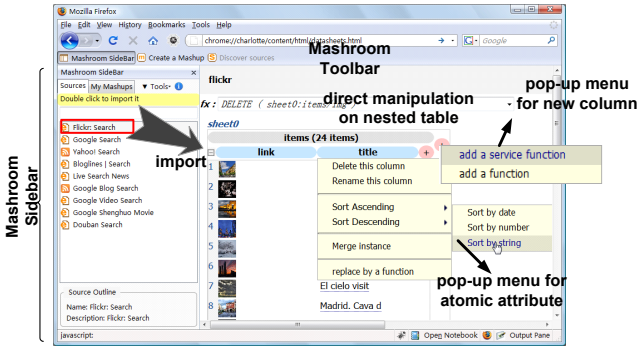


Figure 4. The Interface of Mashroom

3.4.1 Screen Layout

In traditional spreadsheet applications, the entire window is used to present one worksheet. Users can navigate other worksheets by tab clicking. Considering the browser-related user habit, we design our worksheets layout in a different way. In our design, worksheets are vertically aligned on the scrolled browser window and users can scroll the browser window to view those worksheets unseen in the current screen. Consider there may be many worksheets, they can be collapsed and extended to save the presentation space. When a new worksheet is imported, by default, this new worksheet is arranged automatically at the bottom of the current worksheets. Therefore, users can understand the worksheet arrangement in a consistent way. In addition, worksheets can be narrowed by dragging the right edge so that several worksheets can also be horizontally aligned for operating convenience.

3.4.2 Operator Visualization

There are different ways to express the operations. Users can import a Data Service by double-clicking the item in the service list. Clicking on different types of column will pop up different menus. The *DeleteColumn, RenameColumn, Filter, Sort, MergeInstance, LinkService and AddFunction* operators can be triggered though the pop-up menu. The pop-up menu is showed in Figure 4. Figure 5 (a) shows the *LinkService* menu item, the "*mapping*" construction dialog and the result worksheet. Similar with *LinkService,* users can describe the *AddFunction* operator by a calculator-like dialog.

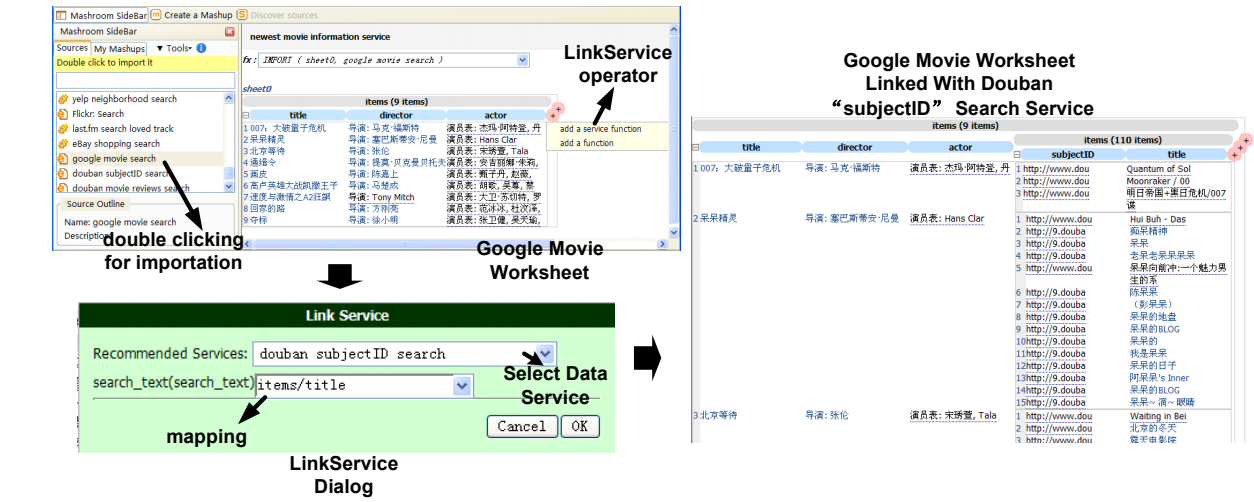
The *Merge* and *CreateSheet* operator is adequate to be triggered through direct drag and drop manipulation instead of through pop-up menu. Figure 5 (b) illustrates how the drag and drop manipulation work. Users can drag the "*items*" column on "*sheet1*" and drop it on the "*items*" column on "*sheet0*" for the *Merge* operation, or drag the "*link*" column to the blank rectangle for the *CreateSheet* operation.

We discover that not all operators can be expressed by contextual menu or drag/drop manipulation. It is partly because some operators are very similar in their semantics. For example, the *Fuse* operator is very similar with *Merge* operator in semantics. And it is very difficult to deign different drag/drop manipulation for them. Some operators are more complicated, and it is not convenience to be expressed by contextual menu or drag/drop manipulation. As shown in Figure 5 (c), we design a formula input box to display or edit the operators. We use SIMILE AJAX library from MIT [17] in implementing the pop-up menu and the column drag & drop utility.

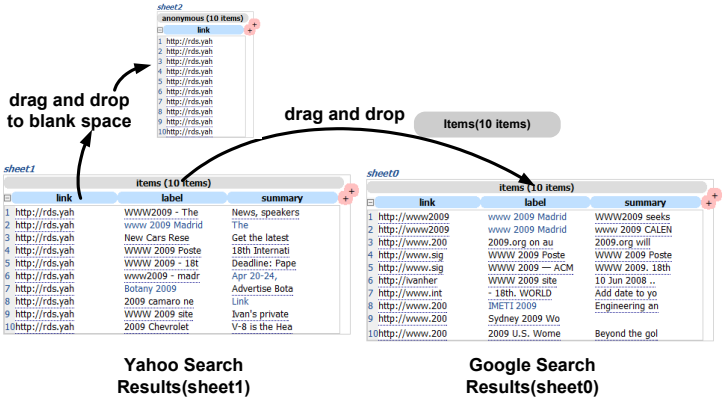
4. Trial Application

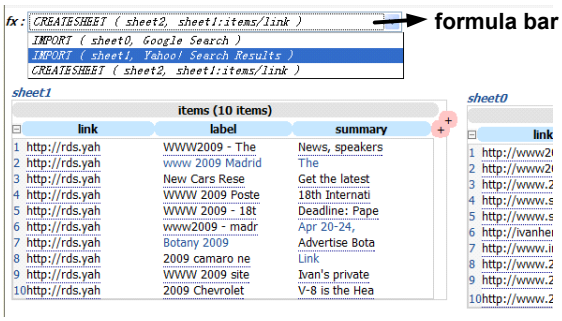
Newest Movie Information Service is a good candidate mashup application for demonstrating Mashroom capabilities. Usually, users need to collect the movie show information and the related movie reviews from various websites and publish the information. We believe that there are many web integration applications similar to this scenario which combines data from different web sites to give an integrated view.

In this scenario, a user launches the mashup to get the newest movie information. Figure 6 illustrates this sample mashup. At first, the movie show information from shenghuo.google.cn and imdb.cn should be integrated into the mashup application. In Mashroom, *"googlemovieDS"* and *"IMDBDS"* Data Services are created by labeling two sample Web pages from two Web Sources. Then, the two Data Services are imported into the editor as two worksheets (step 1 and 2). Then, they are merged into one by dragging the column (*items*) of worksheet and dropping on the other (step 3).



(a) *LinkService* menu item and the poped up dialog





(b) drag and drop to merge two worksheets or create a new worksheet **(c) formula bar**

Figure 5. Operator Visualization

The next task is to integrate the movie reviews from douban.com (a popular Chinese web 2.0 site for movie and book review). This can be done by searching the related movie reviews by the name on the merged movie list. However, here the user comes across a common issue in building such mashups. The user can't search Douban by the name of a movie for the reviews. However, the user can get the reviews once she has the *"subjectID"* (the internal number for each movie in Douban). Therefore, in order to get the movie reviews, the user has to find this *"subjectID"*. And then triggers the *LinkService* operator (step 4) to invoke the *"MovieSearchDS"* Data Service for *"subjectID"*. Note that the *title* attribute is a required input for the *"MovieSearchDS"* Data Service.

Although we can search for "*subjectID*" by the name of a movie, the result she gets (after step 4) is a list of *"subjectID"* that matches the name approximately instead of the precise *"subjectID"* matches the name exactly. This issue can be overcome by filtering the search result by the name. For example, the user can filter out the inexact movie using the formula *Filter*(*G/MovieSearchResults/title, contains, G/title*), where *G/title* is the *"title"* column of the worksheet, and *G/MovieSearchResults/title* is the *"title"* column of the movie review search results table embedded in the worksheet. This is what Mashroom has done in step 5.

Till step 5 the user eventually gets the exact *"subjectID"* of the newest movie list. So in step 6, *"MovieReviewsDS"* Data Service is linked in the worksheet by triggering "*LinkService*" and setting *"subjectID"* as its input value.

Once the user gets the result worksheet, she can associate the result with a presentation view by triggering the "*sink*" operator. Here the user associates the result with a "*list*" view. This output can be used to be started up every day to generate the newest movie information and be published on a web site.

Figure 6 also illustrates the relationship between Mashroom data flow and the nested relational query. Note that the "*LinkService*" operator is followed up by a join between the service invocation result and the relational data on the current worksheet.

Table 2 gives the serialized Data Mashup script in XML format in Mashroom implementation. The Mashroom's mashup engine parses the XML representation into a sequence of instructions. It then executes the instructions (currently in a serialized way) and emits the result MashSheet. To save space, the 16 byte universally unique identifier was abbreviated as "xx".

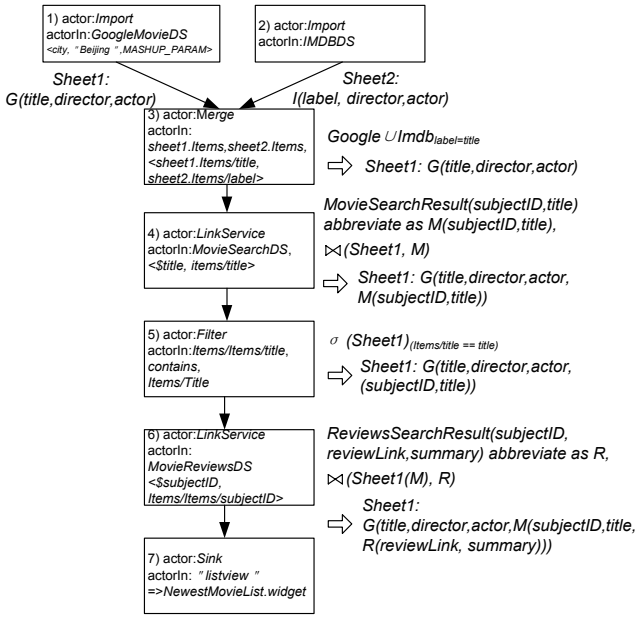


Figure 6. Data Flow Representation of a Data Mashup Script

Table 2. Data Mashup Script Description

```
<?xml version="1.0" encoding="GBK"?>
<mashup id="xx" name="movie mashup" description="" encoding="GBK">
  <params><param name="city" label="city">Beijing</param></params>
  <script>
    <import ref-service="googlemovieDSId" ref-datasheet="sheet1" id="xx">
      <mapping id="xx" param-name="city" style="MASHUP_PARAM" ref-mashup-
param="city" /> </import>
    <import ref-service="IMDBDSId" ref-datasheet="sheet2" id="xx"></import>
    <fuse id="xx">
       <typepair ref-from-type="sheet1:items/title" ref-to-type="sheet2:items/label" />
         ……
    </fuse>
    <serviceFunction ref-type="sheet1:items" ref-service="MovieSearchDS"
id="xx">
      <mapping id="xx" param-name="search_text" style="TYPE" ref-
atom="items/title" />
    </serviceFunction>
    <filter ref-type="sheet1:items" id="xx">
        <and><atom atom-key="items/items/title" rop="CONTAINS" ref-to-
type="items/title" /></and>
    </filter>
  <serviceFunction ref-type="sheet1:items" ref-service="MovieReviewsDS"
id="xx">
     <mapping id="xx" param-name="subject" style="TYPE" ref-
atom="Items/Items/subjectID" />
  </serviceFunction>
    <sink> <option>listview</option> </sink>
  </script>
</mashup>
```

5. Evaluation

To demonstrate the contributions of this paper, we conducted a set of experiments that highlight the expressivity and usability especially for end users.

To come up with a comprehensive evaluation, we first selected 10 popular mashups from the Yahoo! Pipe community (those have high "clone" number), two mashup directories (programmableWeb.com and mashupAwards.com) and a set of other typical mashups. Then we qualitatively studied the mashup creation process in Mashroom. For each mashup, we gave the URLs address for each of the Web Sources.

We also assigned category names to describe what types of mashup applications can be built using Mashroom. Following the "Mashup Patterns" research by Jeffrey Wong et.al [20], we found that there are four interesting sub-categories of mashups in the "Aggregation" category. These sub-categories can be divided into 6 Mashroom worksheet manipulation and composition patterns (M&C patterns for short).

We conducted a user study with 5 groups of people to measure the average time to build a mashup. The users are familiar with spreadsheets. In order to make them understand the Mashroom way of building a mashup, we gave them an example for each sub-category of mashup. The average time to build each mashup was recorded. The results are given in Table 3.

Aggregation for Collection (same kind of Web Sources). These mashups combine the same kind of Web Sources such as news, videos and pictures from different websites into one single worksheet and represent this worksheet as a Web page, a feed, or on a map.

"Aggregated News Alerts" sets up a persistent search at Bloglines, Google Blog Search, Microsoft Live News etc. Mashroom provides an easy way to build such a mashup. The query term parameter of the first imported Data Service should be configured as the parameter of the mashup, so that the parameter of the following imported Data Service can be then mapped to it. The user drags the worksheets' header columns and drops them on the first worksheet for *"merge"* operation. The mashup "Google and Yahoo News" and "Top Videos" also fall into this category. The process about how the Data Services are aggregated has a common pattern called "similarity aggregation without dependency" pattern as Figure 7(a) shows.

Aggregation for Comparison (same kind of Web Sources). Some of the mashups of aggregating different web sites especially in the e-business domain not only combine the data together but also compare them clearly. For instance, the "Book Price Comparison" mashup compares the price of the same book from dangdang.com and amazon.cn shopping web sites. Different from the "similarity aggregation without dependency" M&C pattern, after the Data Services are imported into the worksheet area, the user creates a new column labeled with both the price and the source using the "*AddFunction*" operator. Without this operation, it is not clear to differentiate the prices from different web site.

Focus View or Data Analysis (single Web Source). These mashups don't combine different kinds of web sites. They give a focus view for a large web site or do data analysis work for data from a single web site. For instance, the mashup "Focus View of YouTube Video" aims to list YouTube videos of a certain category or tag. "eBay Price Watch" aims to find eBay items within a certain price range. Different from the "similarity aggregation without dependency" M&C pattern, it is not necessary to map the attributes when the user triggers the *"merge"* (or *"fuse"*) operator because the worksheets have the same schema.

Aggregation for Collection (different kinds of Web Sources with dependency). These mashups combine different kinds of web sites and generate an integrated view that a single web site can't afford to provide. The Data Services are dependant on each other. For instance, in the "Neighborhood Pictures" mashup, the *"neighborhood name"* parameter of the imported "Search Flickr" Data Service is the output of the "Yelp Neighborhood Search" Data Service. Users can express the Data Service dependency by triggering the "*LinkService*" operator, and selecting one of the columns as the new imported Data Service's input parameter.

Table 3. Selected Mashups, Patterns and Experiment Results

CATEGORY	M&C PATTERN	MASHUP	WEB SOURCES	DESCRIPTION	*TIME (minutes)*
Aggregation for Collection (same kind of Web Sources)	"similarity aggregation without dependency" pattern	Google and Yahoo Search Results	http://www.google.com/ http://www.yahoo.com/	Search Google and Yahoo, then merges the results together parameters: Keyword	*< 1*
		Aggregated News Alerts	http://www.bloglines.com/search http://search.live.com/news http://blogsearch.google.com,etc	Setup a persistent search at Bloglines, Google Blog Search, MSFT Live News, etc. parameters: Keyword	*< 2*
		Top Videos	http://video.baidu.com http://video.google.com	List the most popular videos at Google Video and Baidu Video web sites.	*< 2*
Aggregation for Comparison (same kind of Web Sources)	"similarity aggregation with comparison" pattern	Book Price Comparison	http://book.dangdang.com/ http://www.amazon.cn/mn/searchApp	Compare the price of the same book from different shopping web sites parameters: book search keyword	*< 2*
Focus View or Data Analysis (single Web Source)	"focus view or analysis" pattern	eBay Price Watch	http://rss.api.ebay.com/ws/rssapi	Find eBay items within a certain price range parameters: Keyword, max price, min price	*< 2*
		Focus View of YouTube Video	http://www.youtube.com	List of YouTube videos of a certain tag or category parameters: tag or category	*< 2*
Aggregation for Collection (different kinds of Web Sources)	"aggregation with dependency" pattern	Neighborhood Pictures	http://api.yelp.com/neighborhood_search http://www.flickr.com/search/	Find out the neighborhoods and their related pictures	*< 2*
		My tracks on eBay	http://ws.audioscrobbler.com/2.0/?method=user.getlovedtracks&user={userid}&api_key={key} http://open.api.ebay.com/shopping?callname=FindItems	Find out the auctions on eBay of the user's loved tracks parameters: userID, api_key	*< 3*
	"search subjectID first" pattern	Newest Movie Info	http://shenghuo.google.cn http://api.douban.com/movie/subjects http://api.douban.com/movie/subject/{subjectID}/reviews	The movies on show and the related reviews.	*< 4*
	"aggregation with dependency" & "search subjectID first" pattern	All-around info about the movies on show	http://shenghuo.google.cn http://api.douban.com/movie/subject/{subjectID}/reviews http://zh.wikipedia.org/;http://www.flickr.com/search/	Reviews, pictures and wikipedia links of a loved movie	*< 4*

A common mashup we encountered is to search the related information of a subject (for example, reviews of a movie) by its name. However, we often come across a common issue in building such mashups. We can't get the information directly by the name of the subject. So we should search the "subjectID" first and then use it to search the related information. The detailed process has been introduced in Section 4. This "search subjectID first" M&C pattern is depicted in Figure 7(d) at the right.

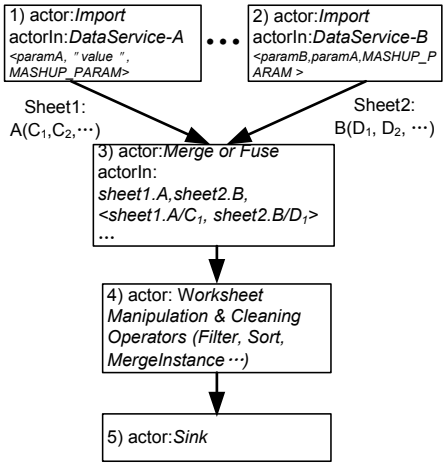
The "All-around info about the movies on show" mashup combines the movie reviews, the corresponding pictures and hyperlinks together. This mashup follows the hybrid M&C pattern of "aggregation with dependency" and "search subjectID first".

Comparing to the flow-chart-like programming experiences, operations are applied to the data directly, and some complex control operations are hidden from users. As a result, the mashup operations are simplized apparently. Take the "sort" operation as an example, to sort a feed by one item attribute, a "sort" module needs to be imported and connected to the source using Yahoo! pipes. While in Mashroom, users can sort the source directly by clicking on the column's pop-up menu item. The "*LinkService*" operator is another example. It encapsulates a loop that invokes a data service for all values of a column. And what's more, the learning curve of building a mashup application is shortened. Users need not to understand how to build up a sequence of operators at the beginning. They play an operator directly on the data and can see the changed data immediately. They can try to build a simple mashup application with little guidance at the beginning. Once the examples are given for each mashup pattern, they are able to complete the more complex mashup creation work without difficulty.

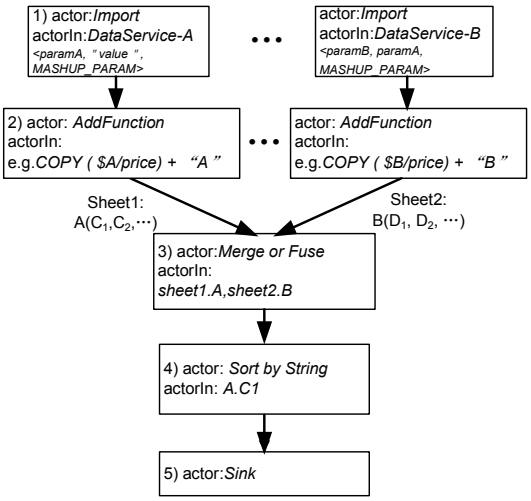
In summary, to verify the expressiveness of Mashroom, we described what mashup applications can be built using Mashroom. Our user tests showed that end users can use Mashroom to build the above categories of mashup applications effectively and efficiently.

6. RELATED WORKS

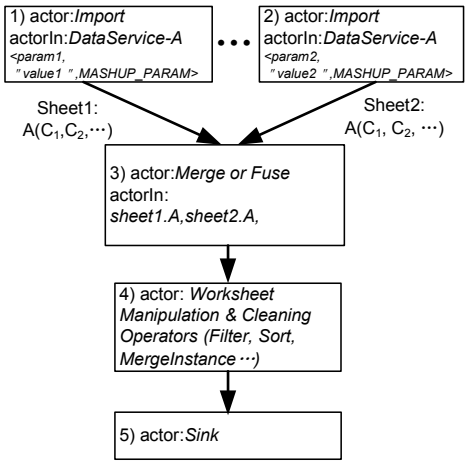
The programming model adopted in mashup editors generally falls into one of four categories: **1). flow-chart-like programming**. Data services are processed in a manner similar to flow-chart. Yahoo Pipes, Microsoft Popfly, IBM Damia and Marmite fall into this category. The flow-chart-like mashup editors often provide a set of flow-chart-based graphical operators and adopt a flow-style orchestration specification. **2). spreadsheet-like programming**. Data services are processed in a manner similar to spreadsheets. SpreadMash [21] and C3W [22] fall into this category. **3). tree-based programming.** Data services are combined based on a tree structure. The old version of Intel MashMaker [4] falls into this category. **4). browser-centric programming.** This kind of programming doesn't change the user experience of using a general browser. This programming model enables users to build mashups while browsing the web sites. It doesn't provide specific mashup editor, but provide mechanisms for users to trigger the mashup operations in the context of browsing. Ubiquity [23] and d.mix [24] fall into this category.



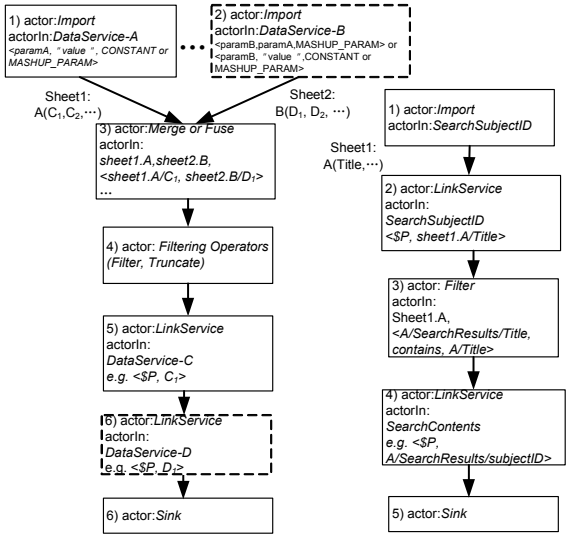
(a) "similarity aggregation without dependency" pattern



(b) "similarity aggregation with comparison" pattern



(c) "focus view or analysis" pattern



(d) "aggregation with dependency" pattern (left), "search subjectID first" pattern (right)

Figure 7. Worksheet Data Manipulation & Composition (M&C) Patterns in Mashroom

As discussed at the beginning of this paper, spreadsheet-like programming is more adequate for end users compared with flow-chart-like programming. But the traditional two-dimensional spreadsheet can't process and present the complex data in a nested table. Though the tree-based programming overcomes the disadvantages of the traditional spreadsheet programming by supporting more complex data, it is not convenient to define operations conducted on several nodes. Mashroom programming model takes the nested table as the graphical data structure and combines it with spreadsheet style programming.

C3W features with its ability to clip some input and result elements from a Web page to form cells on a spreadsheet. But unlike C3W, Mashroom first encapsulates unstructured and semi-structured web data resources as uniform services and import services to form blocks of cells on a spreadsheet-like worksheet. Therefore, once the Web pages are encapsulated as services in Mashroom, it can be reused and combined with other services without considering about how to extract the items from a Web page using XPath as in C3W.

SpreadMash also supports the complex data object by extending the traditional spreadsheet. One of the main characteristics of SpreadMash is that it provides reusable "data widget" to implement the importation, presentation and composition of the complex data objects. SpreadMash allows users to define the layout and space dependency of the complex data objects on a worksheet. So a lot of work in SpreadMash is to define the formula syntax and semantics to express the data layout and space dependency. Different from SpreadMash, Mashroom provides each complex data object a default layout. The layout and space dependency of data objects are meaningless in Mashroom. Considering the nested table is only a temporary view during the mashup building process, and not the view of the ultimate mashup application, we believe that the default layout is enough for the users.

Based on the above analysis, we come to a conclusion that Mashroom programming model goes beyond the other categories for end users, and, even compared with the other spreadsheet-like programming model, Mashroom programming model offers superior characteristics in some aspects.

7. CONCLUSIONS

For the past two years, Internet-related mashups have been boomed as a potential candidate for the next market opportunity and have become one of the biggest buzzwords in the web application area. However, comprehensive development tools, frameworks and programming models are lagging behind. In this paper, we have provided a programming model for building mashups by end users. We have discussed the design philosophy, the abstraction of mashup applications, and the implementation and evaluation issues associated with end-user mashup programming. The contributions are that we combine the nested table with the spreadsheet-like programming. The innovation has been verified to be effective by the experiment and case study results.

8. ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China under Grant No.60573117, the National Basic Research Program of China under Grant No.2007CB310804, the China Postdoctoral Science Foundation under Grant No.20080430528, and Huawei Technologies Co., Ltd. We thank Zhuofeng Zhao and Haifang Fu for their support. We also thank Hongshen Liao, Yanyan Cheng and Guang Ji who helped in developing Mashroom.

9. REFERENCES

[1] Simmen, D. E.; Altinel, M.; Markl, V.; Padmanabhan, S.; Singh, A. Damia: data mashups for intranet applications, SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM, 2008, pp. 1171-1182.

[2] Yahoo Pipes, Inc. http://pipes.yahoo.com/, 2008.

[3] Microsoft Popfly, http://www.popfly.com/, 2008

[4] R. Ennals and D. Gay. User-friendly functional programming for web mashups, In Proceedings of the 12th International Conference on Functional Programming (ICFP), Freiburg, Germany, 2007, pp. 223-234.

[5] Robert J. Ennals , Minos N. Garofalakis, MashMaker: mashups for the masses, Proceedings of the 2007 ACM SIGMOD international conference on Management of data, June 11-14, 2007, Beijing, China

[6] J. Wong and J. I. Hong, Making mashups with marmite: towards end-user programming for the web, In Proc. of the 2007 conference on Human factors in computing systems (CHI), San Jose, California, USA, 2007, pp. 1435-1444.

[7] Makinouchi, A. A consideration of normal form of not-necessarily-normalized relations in the relational data model. In Proceedings of the 3rd VLDB Conference (Tokyo), 1977, pp. 445-453.

[8] Laender, A. H.; da Silva, A. S.; Golgher, P. B.; Ribeiro-Neto, B.; Evangelista Filha, I. M.; Magalhaes, K. V. The Debye environment for Web data management. Internet Computing, IEEE 2002, 6, (4), pp. 60-69.

[9] Embley, D. W.; Campbell, D. M.; Jiang, Y. S.; Liddle, S. W.; Lonsdale, D. W.; Y-K Ng.; Smith, R. D. Conceptual-model-based data extraction from multiple-record Web pages. Data Knowl. Eng. 1999, 31, (3), pp. 227-251.

[10] Filha, I.M.R.E., Laender, A.H.F., and Silva, A.S.D. Querying Semistructured Data By Example: The QSByE Interface. In Proceedings of Workshop on Information Integration on the Web. 2001, pp. 156-163.

[11] Qin Z., Yao B. B., Liu Y., and McCool M. D., A Graphical XQuery Language Using Nested Windows, In Proc. of 5th Int. Conf. WISE, Brisbane, Australia, 2004, pp. 681-687

[12] B. A. Nardi,; J. R. Miller, The Spreadsheet Interface: A Basis for End User Programming, HP Labs Technical Reports, http://www.hpl.hp.com/techreports/90/HPL-90-08.pdf, 1990

[13] Myers, B.A., Ko, A.J., and Burnett, M.M. Invited research overview: end-user programming. In Proceedings of CHI Extended Abstracts. 2006, pp. 75-80.

[14] Cypher, Allen, ed. Watch What I Do: Programming by Demonstration, MIT Press, Cambridge MA, 1993.

[15] Lieberman, H. (Ed.) 2001. Your Wish is My Command: Programming by Example. San Francisco: Morgan Kaufmann.

[16] Colby, L. S. A recursive algebra and query optimization for nested relations. SIGMOD Rec. 1989, 18, (2), pp. 273-283.

[17] AJAX-SIMILE, http://simile.mit.edu/ajax/, 2007

[18] Dapper: The Data Mapper. http://dapper.net. 2008

[19] Shaohua Yang, Guiling Wang, Yanbo Han. Grubber: Allowing End-Users to Develop XML-based Wrappers for Web Data Sources. The Joint International Conferences on Asia-Pacific Web Conference (APWeb) and Web-Age Information Management (WAIM), Suzhou, China. 2009, pp. 645–650.

[20] Wong, J.; Hong, J. What do we "mashup" when we make mashups? WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering, ACM, 2008, pp. 35-39.

[21] Woralak Kongdenfha, Boualem Benatallah, Régis Saint-Paul, Fabio Casati. SpreadMash: A Spreadsheet-Based Interactive Browsing and Analysis Tool for Data Services, CAiSE, 2008, pp. 343-358.

[22] J. Fujima, A. Lunzer, K. Hornbaek, etc. Clip, connect, clone: combining application elements to build custom interfaces for information access, In Proc. of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST), 2004, pp. 175-184.

[23] Ubiquity. http://labs.mozilla.com/projects/ubiquity/. 2008

[24] Hartmann, B.; Leslie, W. u.; Collins, K.; Klemmer, S. R. Programming by a sample: rapidly creating web applications with d.mix, UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology, ACM, 2007, pp. 241-250.