Abstract

Spreadsheets, a popular productivity tool, has gained attention as a potential mashup development environment targeted towards end-users. In this paper, we present a general architecture of mashup tools for spreadsheets. We also present an analysis of the state-of-the art on spreadsheet-based mashup tools. The analysis result is used to guide our research in developing a lightweight semi-automatic mashup tool using spreadsheet paradigm.

Keywords: Web services, Mashup, Spreadsheet, End-user programming

1 Introduction

Mashup is a new application development method enabling users with little programming skills to create application by reusing and combining data, application functionalities and presentations from different sources. Recently, we witnessed a sharp rise of mashup tools and applications on the Web. In 2006 alone, there were hundreds of individual mashup tools released, Yahoo Pipes, Google Mashup Editor, Microsoft Popfly, IBM QEDWiki, just to name a few. Different mashup tools solve different problems so that when users are embarking onto the mashup journey, they need to pick the right tool for the right problem. For example, data mashup allows users to retrieve data from one or several data sources, process, mix the data and publish the result either as a feed or another data source. Process mashup allows users to automate processes by orchestrating services, forms and other resources in a workflow. Web page customization (i.e., presentation mashup) allows user to change web sites by removing elements, adding additional widgets or changing the user interfaces.

According to Fischler et al. (2009), spreadsheet is one of the six programming paradigms for mashup development, along with integrated development environment, scripting languages, wiring (or piping) paradigm, programming by demonstration and automatic creation of mashup.

With spreadsheet-based mashup tool, spreadsheet users (e.g., office workers, professional accountants) are provided an opportunity to access more information, making their job more efficient. Recently, some companies have already provided products for spreadsheet-based mashup tool (e.g., StrikeIron, Extensions). The motivations of using spreadsheet paradigm for mashup tools are elaborated as follow:

• First, spreadsheet is a widely used application with millions of users (Scalfidi et al. 2005). The popularity of spreadsheets can help increase the level of acceptance of any mashup tool built in the familiar environment.

• Second, spreadsheet is an intuitive data management, analysis and reporting tool with useful but simple-to-use functions, such as import, export, sort, visualize, etc. It could be considered as a suitable paradigm for data mashup since most of mashups are applications that reuse and combine data available from difference sources.

• Third, spreadsheet is a preferred programming environment for end users with the following features (Kandogan et al. 2005):
  - It offers a mixture of development environment and runtime environment facilitating immediate feedback to the users.
  - It supports incremental (i.e., step-by-step) development of an application.
  - It is more resilient and forgiving than most programming languages (i.e., an error in a cell only affects referring cells and does not affect the whole program causing a program crash).

In this paper, we present a qualitative survey on a set of spreadsheet-based mashup tools. We believe that understanding these mashup tools and other related issues will pave the way for research directions to effectively address some of the research problems (e.g., how users can take advantages of spreadsheet paradigm for developing mashups).

As an illustration of spreadsheet-based mashup, let us consider the following scenario: Mary, a student, is learning Spanish. To study a word, she uses text-to-speech service (TTS), translation service (TSL), spell-checking service (SPL) and online workbook service (LOG). Instead of invoking these services separately and manually link the results, she opens a spreadsheet program, inputs an English word, writes formulas to link these services so that the completed formulas will now automatically suggest spelling corrections, pronounce the word in Spanish, translate the word to Spanish, pronounce the word in Spanish then finally write the English and Spanish words to her...
online workbook. Obviously, the ability to quickly compose such services in spreadsheet enables her to create new applications that suit her needs and save her time, while interacting with a tool she is already familiar with.

![Diagram of a spreadsheet with formulas]

**Figure 1: Motivating scenario**

Mary’s program is incrementally built. The “Formula view” in Figure 1 illustrates Mary’s tasks. First, she specifies the location for user input by writing a label “User input” in cell A1 and the input word “Orage” (in incorrect spelling format) in cell B1. Then she invokes SPL service by putting a formula in cell B2 (i.e., =SPL(B1)) which takes the value of cell B1 as a parameter for SPL. She will get the spelling suggestion in cell B2 (i.e., “Orange”) as shown in Evaluation view. Similarly, she writes the formulas in cell B3, B4, B5, and B6 step by step to get the voice from TTS, TSL and LOG. The mashup application is enacted when Mary changes the value in cell B1. Spreadsheet paradigm will automatically trigger the evaluation of formulas in dependent cells and produce results.

The remainder of this paper is organized as follows. Section 2 presents a general architecture of spreadsheet-based mashup tools. Section 3 introduces a number of mashup tools using spreadsheet paradigm. In sections 4 and 5, we discuss four dimensions used in our survey and evaluate cells tools according to the dimensions. We conclude the paper and have a discussion on future work in section 6.

### 2 General Architecture of Spreadsheet-Based Mashup Tools

When reviewing frameworks, it helps to have implementation architecture in mind. We describe here a general architecture of the spreadsheet-based mashup tools. Through the architecture, the readers should be able to picture how a mashup application is built in a spreadsheet. As depicted in the Figure 2, the architecture comprises of four elements:

- **Spreadsheet Interface (1).** This element is a conventional spreadsheet interface which includes a grid of cells and a formula editor. Cells are capable of storing values and formulas, while formula editor allows users to build mashup applications by specifying composition logic and layout information. The spreadsheet interface plays a role as the development environment in a mashup tool. For example, in the Mary’s scenario, Mary writes her mashup application in spreadsheet interface using formula as illustrated in Figure 1.

- **Component Repository (2).** This element is a repository of all mashup components available in the tool. Users choose an external resource (e.g., web service, file, database or application) and create a component that allows them to interact with the resource within the tool. This component can be assigned a friendly name (i.e., alias) for the spreadsheet formula editor. For example, in the scenario of Mary’s, SPL is a component representing a SOAP-based service that suggests correct spelling of words. It is created based on the WSDL document of the spelling service.

- **Mashup Engine (3).** This key element is responsible for evaluating the formula (i.e., composition logic) and “wire” mashup components together. It operates in a centrally-mediated fashion and plays a role as a server to manage the execution flow among components. Since most of spreadsheet tools do not allow users to modify their formula evaluation mechanisms, mashup engine could be developed as an extension to spreadsheet evaluation engine. Mashup engine is also responsible for maintaining the formula evaluation context (i.e., the mashup result) and facilitate the reaction to cell modification by triggering the re-evaluation of dependent formula (i.e., upon a service invocation returns, the corresponding references need to be updated with the returning value). For example, in Mary’s mashup application, whenever she change the input value in cell B1 (e.g., change “Orage” to “Lemon” - both are in incorrect spelling format), the formulas in cells B2-B6 will be re-evaluated by the engine to create new results.

- **Wrappers (4).** Wrappers facilitate interoperability among resources which have different data formats (e.g., HTML, XML, RSS, etc.) or use different access protocols as a server (HTTP, SOAP-based, REST-based, etc.). For example, we need different wrappers to create components to correctly serve the Mary’s scenario, such as SOAP-based service wrapper for SPL service, REST-based service wrapper for LOG service or application specific wrapper for TTS service.

### 3 Spreadsheet-Based Mashup Tools

In this section, we introduce the spreadsheet-based mashup tools we studied for evaluation. For more detailed classification and description of these tools, readers are referred to (Hoang et al. 2009).

In these tools, the spreadsheet itself is a productivity tool which now equipped with functions to access external resources (e.g., services in the Web, files in desktop computer, databases, etc). Data from external resources are placed in cells, wired to each other to create a mashup application (e.g., output data of a service can be used in the input fields of another service).

We review the following tools: spreadsheet connectors for mashup engines, StrikeIron (Brauer n.d.), Extensio Extend (http://www.extensio.com/products/ExcelExtend.html), A1 (Kandogan et al. 2005), AMICO-CALC (Obrenovic & Gasevic 2008), Husky (Sribjic et al. 2007).

1. **Spreadsheet connectors for mashup engines.** JackBe Presto (7) IBM Mashup Center (8) and Kapow (9) are mashup engines which provide separate “spreadsheet connectors” , allowing Mi-
crosoft Excel spreadsheet users to consume existing mashup applications directly from their spreadsheets. These spreadsheet connectors allow the end-users to easily re-use already-built mashups (outside spreadsheets) in the spreadsheet environment. However, they are not fully-functional spreadsheet-based mashup tools in their own right.

2. **StrikeIron SOA Express for Excel and Extensio Extender for Microsoft Excel** are commercial data mashup tools using spreadsheet paradigm. The basic idea of these approaches is that they allow the data contained in the web services to be pulled in Microsoft Excel workbook, “live” in cells, and integrated directly by users while still take advantages of all the analytical powers and flexibility of the spreadsheet tool. StrikeIron and Extensio Extender use SOAPful web services to create mashup applications by “hooking” the output value of one service with the input parameter of another service.

3. **AMICO:CALC** is an OpenOffice Calc extension that let users configure and connect services through a spreadsheet interface. Users manually write formula to compose the services in order to create a mashup. The execution is based on Adaptable Multi-Interface COmmunicator (AMICO) middleware platform for component integration. In AMICO:CALC, variables are considered as services in order to be used in spreadsheet. Users are provided some pre-defined functions to read value from and assign value to variables (i.e., to get data from and post data to services). For example, the expression `Amico_Read("spelling – suggestion")` will receive a value when the spelling service finishes. AMICO provides a control panel tool where users can read description of variables exported by services, change their values or names.

4. **AI**, also known as Autonomic Task Manager for Administrators, is a research prototype from IBM. It facilitates a programming environment for system administrators. With AI, users not only use a spreadsheet-like environment with a task-specific language to access remote and heterogeneous systems but also gather, integrate status data and orchestrate control of different systems. AI extends conventional spreadsheets by:

   - Extending cell formulas to include calls to methods of cell objects.
   - Allowing cells to contain procedural code blocks whose execution is triggered by events in the sheet.

5. **Husky** is a service composition tool. It extends the spreadsheet paradigm enabling users to intuitively express composition logic through a spatial arrangement of component services within spreadsheet cells.

### 4 Evaluation

We have installed and tried the tools mentioned in the previous section. Borrowing lessons from application integrations, we believe that a spreadsheet-based mashup framework needs the definition of four basic elements: component model, composition model, development environment, and runtime environment. Here, we summarize and compare the features across four dimensions: component model, composition model, development environment and runtime environment.

#### 4.1 Component Model

This dimension determines the nature of components by means of the following properties:

##### 4.1.1 Component Data Model

A data model, as defined in (Ullman 1990), contains a notation for describing data and a set of operations used to manipulate that data. In our surveyed mashup tools, the data model extends spreadsheet data model to accommodate heterogeneous resources from different sources (e.g., web services, databases, files). Spreadsheet is a collection of cells organized in a tabular grid. Each cell is identified by its coordinates and contains either empty or atomic type values. Traditional spreadsheets have limited of atomic types, such as number, string or datetime. Some spreadsheets extend the atomic types so that the cells can contain complex types such as object or XML document. AMICO:CALC, a service invocation may return an...
XML document with complex structure (e.g., nested). In order to display the output, AMICO:CALC uses service adapters to linearize the complex structure to a set of variables, which can be easily mapped to spreadsheets cells and formulas. This transformation at the component data model level means that user does not have to deal with complex hierarchical structures.

In the object-based model, since the spreadsheet cell is extended to contain complex types, a complex XML document can be mapped to a single cell. For example, A1 extends conventional spreadsheets by allowing cells to contain arbitrary Java objects and formulas call to methods in cell objects. The following code snippet illustrates object-based data model used in A1:

```java
package com.ibm.a1.plugin.api.samples;
public class SampleComponent {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Cell C1 contains a string value represents an IP address. In cell C2, we use JMX (Java Management Extensions) binding to a SOAP service, namely IP2Country(). This web service has two operations: IP2Country() and Country2IP() which returns a country name of a given IP address and vice versa. In cell C3, we invoke IP2Country() operation of this web service and the result is returned as an XML document.

To illustrate the differences between two component data models, let us consider an example given in Figure 3 which shows an RSS feed of Google news service. The grid-based model is presented in Figure 3(a). The news, presented as an XML document, is mapped to the range of cells B1:C9. Figure 3(b) shows how the data could be presented using the object-based model. In which, the news data is mapped to cell B1. Each element of the news could be accessed by using “dot” notations in the mapping definition. For example, the “channel language” element can be put in cell B2 by using formula = B1.language.

### 4.1.2 Component Access Model

Components are entities that allow users to interact with external resources inside a mashup tool. Resources can have heterogeneous data formats and use different access protocols. In this dimension, we investigate different data formats and access protocols supported by the mashup tools.

Some of the common data formats available in component services could be HTML, XML, RSS, Atom Syndication Format, JSON (Javascript Object Notation), spreadsheet, text file or RDF (Resource Description Framework).

The access protocols could be HTTP, TCP, UDP, SMTP, REST, SOAP, RPC or user-specific application protocols.

Most of the tools we studied provide support for REST and SOAP protocols using XML data format due to the prevalence of REST- and SOAP-based services. Some tools allow users to “mash” with other resources such as plain files, database, web applications or even legacy applications. This could greatly expand component resources from which users can make richer and better mashup application. For example, AMICO:CALC supports various communication protocols (e.g., TCP, UDP, XML-RPC, Open Sound Control, HTTP, SOAP, SQL and some application-specific interfaces such as Sesame RDF, WordNet, etc) for interconnecting services with different interfaces.

### 4.1.3 Extensibility

Extensibility defines the ability of a mashup tool supporting users to define their own functionalities (e.g., creating components, defining new mashup operators). Normally expert users use a general purpose language (e.g., Java, C#) to achieve this extension.

For example, A1 allows users to extend the functionality of the tool to best suit their need by developing custom plug-in components. These plug-in components provide new interaction capabilities and functionalities and could be developed by creating new Java class extending A1Component class. The following code snippet illustrates an example of a user defined component named SampleComponent:

```java
public class SampleComponent {
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

After deployed, in A1 spreadsheet’s cell users can create “samples” object and invoke its methods.

Extensibility can also be achieved by modifying existing functionality provided by the tool. For example, AMICO:CALC provides a generic adapter for services using TCP. Users can modify (or parameterize) this adapter to facilitate accessibility for a specific application using TCP protocol (e.g., a game).

### 4.2 Composition Model

This dimension determines how components can be glued together through the following properties:

#### 4.2.1 Mashup Strategy

This dimension identifies the mashup strategy used in spreadsheet mashup tools. We classify mashup strategy into two categories:

- **Static**: The first category concerns the time when a mashup application is created. This could be at design time or run time. Static mashup takes place during the design time when users plan the logic and components to be used in the mashup application. Components are selected, linked together and finally compiled and deployed. This strategy is suitable with a scenario when components and services are stable (e.g., in term of availability) and rarely change (e.g., data structure).

- **Dynamic**: When the component providers frequently change the structure of data or business logic, static mashup is too restrictive and requires users to re-built mashup application adapting to the changes. Dynamic mashup may solve this problem by adapting to unpredictable changes at the run time. The web environment is highly dynamic where new services become available and change frequently. Ideally, mashup application should be able to transparently adapt to environment changes, and to adapt to customer requirements with minimal user intervention.
• The second category concerns the way users build mashup application which could be either manual or automatic. In manual mashup, users manually describe the composition logic (i.e., data flow and control flow) for mashup application. This makes the development process of mashup application tedious, error-prone and time consuming. On the contrary, automatic mashup does not require users to be involved in low-level development process. Instead, users only need to provide high level requirements or goals of mashup application. The mashup tool will automatically suggest or choose composition plan based on given context or semantic of application and components.

4.2.2 Orchestration Style

Orchestration describes the arrangement, coordination and management of components in mashup application. It contains business logic of mashup application specifically about execution order of components. There are three orchestration styles could be used in spreadsheet paradigm, namely flow-based, layout-based and event-based:

- Flow-based. In flow-based orchestration style, mashup applications are built by using natural data flow and control flow created by cell dependencies in spreadsheet (i.e., linking one service’s output into another service’s input fields). For example, as shown in “Formula view” in Figure 1 Mary’s scenario can be implemented by “hooking” output of SPL service in cell B2 with input field of TSL service in cell B4. This orchestration style is naturally supported by spreadsheet paradigm and easy to use. However, as discussed in [Hoang et al., 2009] it has limitations when users need to build a mashup application with complex control flow logic.

- Layout-based. In layout-based orchestration style, users are allowed to explicitly define the execution order of components in mashup application by organizing spatial position of components. For example, in Husky, the execution order of mashup application is defined by spatial organization of services in a grid of cells. Husky transforms the spatial organization of services’ events into their ordering in time. Time progresses from left to right and from top to bottom in cell blocks. A set of adjacent cells makes a sequence of events, while empty cells disjoin the workspace into temporally independent event sequences. Let consider Mary’s scenario implemented by using Husky as shown in Table 1 below: The execution flow is defined by a sequence of services in adjacent cells from C2 to C7. The conditional split defined in cell C3 will trigger the operation in cell C4 if the content of cell C2 is not empty (i.e., like “goto” or “jump” statement in some programming languages). The rest of the program is evaluated according to the following order: evaluate formula in C5, C6 and finally C7. By explicitly defining the control flow, this orchestration style gives the users the benefits of generality and extensibility. However, it may break the philosophy of spreadsheet programming and increase the learning curve.

- Event-based. In event-based orchestration style, the execution flow of mashup application is determined by user actions (e.g., button clicks, key presses, etc) or interactions with other services (i.e., the completion of an action in one service may cause an action in other service). For example, in the A1 system, each cell is an arbitrary Java object. Each object is associated with a listener which is designed to manage events. The listener explicitly triggers the execution of a code block based on a condition. There are two listener constructs: on() is used to trigger an operation based on events and when() is used to trigger an operation based on boolean expression. Mary’s scenario can be implemented in A1 as shown below in Table 2.

Services definitions are defined from cell B1 to

![Figure 3: Component data model](image)
on the button defined in cell B6. Based on the user-generated event (i.e., button clicked), the formula in cell B7 returns a spelling suggestion. Based on the value of cell B7 (i.e., the output of spelling service), two formulas in cell B8 and B9 are evaluated. Similarly, the execution of formula defined in cell B10 and cell B11 are based on events generated by the execution of formulas in cell B9 and B10, respectively.

4.2.3 Data Passing Style

This dimension identifies how the data is passed between components. There are two data passing styles:

- **Data Flow.** In this style, data is passed from one component to another component. For example, in Mary’s scenario, the data from output of SPL service in cell B2 is directly transferred to input field of TSL service in cell B4. This is the default data passing style of spreadsheet applications.

- **Blackboard.** In this style, data is read from and written to a shared data repository (e.g., variable). For example, AMICO:CALC uses variables (which encapsulate services’ data structure) to abstract the heterogeneity of services. Users are provided with various predefined functions (e.g., Amico_Read, Amico_Write, etc) to get data from and post data to services. The formula Amico_Read(“SPL”) will receive a value when the spelling services finished. Amico_Write(“SPL”, B1) will assign value in cell B1 to the variable SPL. This data passing style is used mainly in programming languages.

The dominant data passing style using in the evaluated tools is data flow. This could be explained by the fact that the data flow model is easy to use and exists in most of mashup tools (e.g., Yahoo pipes) and is naturally support by spreadsheet paradigm.

### 4.3 Development Environment

This dimension characterizes the development environment offered by a mashup framework through the following properties:

#### 4.3.1 Target Users

We identify three types of users in spreadsheet-based mashup tools: developer, skilled user and novice user. A developer, who is the most skillful user, should be familiar with programming, web technologies, different APIs as well as the usage of mashup tools. A skilled user has no programming skills but does have detailed functional knowledge about some technologies and a specific mashup tool. Novice user only has knowledge on the functionality of spreadsheet and be able to use some simple spreadsheet formulas and operations (e.g., make a cell reference, copy and paste the content of a cell).

Most of the tools target at spreadsheet users who have limited programming skills. For example, users of Extensio Extender do not need to spend too much learning effort in order to use the tool since the programming metaphor of the tool is similar to spreadsheet programming style. Some tools extend the spreadsheet paradigm to accommodate new orchestration style and require users to learn a specific formula syntax in order to create a mashup application. For example, in A1, the user needs to learn how to write procedural codes within a cell formula. Or users need to learn the rule for spatial arrangement of services as well as four special objects (namely, Clipboard, Queue, TokenCenter and BrokerCenter) in Husky spreadsheet in order to define flow control of service invocations.

#### 4.3.2 Search-ability

Search-ability expresses the ability of a mashup tool helping users in finding desired components (or even mashup application) created by others. There are three types of search:

- **Text/Keyword-based.** Users can search on the title, description or tags of components by specifying keywords.

- **Browsing.** The tool displays a list of components in a typical file-finder or folder-subfolder fashion.

- **Context-specific suggestion.** The tool provides (or suggests) the needed component to users depending on the users’ context without having to search.

Within the evaluated mashup tools, there is no tool that offers context-specific suggestion to the users. Having context-awareness could lead to the possibility of having a framework that relies on the context and incrementally guiding users in developing mashup applications. Browsing capability could make users feel conformable when they want to find desired components since it classifies components and services into different categories in a conventional manner. For example, StrikeIron provides browsing capabilities to its “Web service Marketplace” which has different categories (e.g., Communication, Financial, Marketing, E-Commerce, Utilities, etc). Text-base searching is useful when users know the name or description of components. AMICO:CALC provide the text-based searching capability for finding components in its middleware control panel.

#### 4.3.3 Community Features

One of the most important factors that could lead to the success of an end-user development tool is community support (Nardi 1993). Some of the desirable community support features are:

- **Sharing and reusing.** Components and mashup applications are created by one user but they can be run, copied or modified by other users.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Orange</td>
<td>Execute[A2] “GetSpell”</td>
<td>[A1]</td>
</tr>
<tr>
<td>3</td>
<td>If [C2]&lt;&gt;”” Then Set Clock To [C4]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Layout-based orchestration style in Husky
4.3.4 Software Engineering Aspects

This dimension concentrates on the software engineering aspects including the following aspects:

- **Debugging.** Users can locate and fix errors (including composition logic error, syntax error, etc).
- **Testing.** Users can investigate the quality of mashup application with respect to the context in which it is intended to operate.
- **Version control.** Users can manage the changes and multiple users can access and modify a mashup application.

The surveyed mashup tools mainly focus on offering functionalities for creating mashup applications. The overall support for software engineering aspects such as debugging and version control is very limited. There is only AMICO:CALC supports debugging capability. In AMICO:CALC, debugging is enabled through the use of middleware control panel for monitoring all the variables that are exchanged among services.

4.4 Runtime Environment

This dimension expresses how the results of mashup are delivered to users.

4.4.1 Mashup Engine

We identify two types of mashup engine could be used in the spreadsheet-based mashup tools: (i) mashup engine that is based on spreadsheet evaluation engine; and (ii) mashup engine that developed as a plug-in to spreadsheet application. On one hand, spreadsheet evaluation engine could be used for data mashup by using natural data flow created by cell dependencies in spreadsheet. The mashup application executed using this mashup engine must use spreadsheet programming metaphor (e.g., using formula, cell referencing, etc). For example, in StrikeIron, a very simple data mashup can be implemented by linking one service’s output into another service’s input fields. On the other hand, mashup engine can be developed separately from spreadsheet evaluation engine. The mashup application does not need to conform to spreadsheet programming metaphor. Tools which use layout-based and event-based orchestration styles usually have this type of mashup engine.

4.4.2 Execution Type

We identify three types of execution could be used in mashup tools, namely centralized, distributed and hybrid. Centralized execution is similar to the client-server paradigm. In this case, the server is the central scheduler that controls the execution of the component services in a mashup application. The distributed paradigm in contrast expects the services to share their execution context. Each component service has its own coordinator, which has to collaborate with the coordinators of the other services, to guarantee a correct ordered execution. Hybrid form of the distributed and centralized paradigms may be a coordinator that controls not only one but a set of web services (Benatallah et al. 2003).

All of surveyed mashup tools have centralized execution type. For example, the execution of mashup program in AMICO:CALC is based on a middleware named Adaptable Multi-Interface Communicator which is a centralized platform facilitates adaptation, abstraction, and mediation for diverse service interfaces. The middleware maintains a list of variables which encapsulate data structures used by services and can be easily mapped spreadsheet cells.

4.4.3 Exception and Transaction Handling

Exception handling is a collection of mechanism supporting the detection, signaling and after-the-fact handling of unusual events whether erroneous or not (Burnett et al. 2000). In programming languages, exception handling is considered as necessary part and exists in almost all of application development tools. However, existing spreadsheet mashup tools have very limited support for exception handling. In spreadsheet, exception handling is compatible with reasoning model of spreadsheet formula. For example, in Microsoft Excel, when an operation detects an exception, it will return one of seven possible error values (e.g., NULL, DIV/0, VALUE, REF, NAME, NUM, N/A). The error value model used in spreadsheet is different from the use of status flag in traditional programming language in the sense that it can be ignored so that the error may not cause the whole “spreadsheet program” to crash.

A transaction is an atomic operation which may not be divided into smaller operations. Transaction handling is a mechanism supporting the ACID (i.e., Atomic, Consistent, Isolated, and Durable) properties of transactions. In our surveyed mashup tools, transaction handling is not supported.

5 Summary of Evaluation

So far, we have discussed different dimensions using in this survey. In this section we discuss our analysis result and summarize it in Table 3.

The majority of tools have grid-based component data model. This design choice can be explained by the fact that using grid-based model there is no need to extend traditional spreadsheet data model to present data. The issues of how to map structured data to spreadsheet data model have been addressed by existing works. For example, Kongdenfha et al. (2008) provides a set of widgets for presenting complex data in spreadsheet paradigm namely content, repeater, hierarchical, inner operation, drag and drop interface for manually mapping data to target location in spreadsheet. Object-based data model is more sophisticated since it encapsulates object type in a cell. However, this data model requires more advanced knowledge from users and is more suitable with skilled users and developers than...
novice users. This is the reason of why there are only a few number of tools used object-based model.

In order to facilitate the mashup capability with a specific resource (e.g., a specific application), user need to write their own wrapper. Within reviewed tools, only A1 and AMICO:CALC allow users to extend Java class to write plug-ins/adapters for developing new functionality of the tool. This could not be suitable with all users since it requires users to have programming skill.

All of the analyzed tools use static and manual mashup strategies. Mashup programming using these strategies is tedious, error-prone and time consuming while the mashup application is incapable of adapting to requirement changes. This limitation leads to the need to have an automatic and dynamic mashup tool supporting users in creating mashup applications.

Flow-based orchestration style is used in most of the tools since it conforms to natural spreadsheet programming metaphor (i.e., cell referencing). Layout-based orchestration is rarely used in mashup tools since it may break the functional programming nature of spreadsheet (i.e., it introduces procedural programming notions within spreadsheet). Event-based orchestration is based on the Event-Condition-Action grammar notions within spreadsheet). Event-based programming metaphor (i.e., cell referencing). Layout-based orchestration is rarely used in mashup tools since it may break the functional programming nature of spreadsheet (i.e., it introduces procedural programming notions within spreadsheet). Event-based orchestration is based on the Event-Condition-Action form (i.e., < event > if < condition > then < action >) and is preferred programming model for programmers.

In summary, the reviewed mashup tools have provided many useful features for creating mashup applications. However there are plenty of rooms for further improvement.

6 Conclusion and Future Work

Mashup is an application development method which can be done in a lightweight manner to mix information and automate processes. There has been a plethora of mashup tools in many shapes and forms. In this paper, we analyze a number of spreadsheet-based mashup tools. We believe the spreadsheet environment has inherent advantages over other mashup environment due to its popularity and familiarity with the users.

We attempted to make a comparison among existing spreadsheet-based tools by finding common dimensions and characteristics. We hope that the analysis will be useful for the researchers and developers in this area to understand the architecture of spreadsheet-based mashup tools, the strengths and weaknesses of the current approaches.

The main limitation of our work is that we conducted a qualitative instead of quantitative survey, therefore, some of the observations could inherently be subjective. However, the results are only used to develop an initial understanding and create a foundation for further research. More rigorous surveys with quantitative measures are planned to add objectivity to the study.

Typically, components in our surveyed tools are manually created and composed at design time with user interaction. However, requirements and situations of the tasks in our daily life may change at any time so that a mashup tool need to be able to transparently adapt to environment changes, adapt to customer requirements with minimal user intervention. None of the reviewed tools can do this. This problem leads to an unsatisfied support to the users which can prevent them from being productive.

The goal of our future work is on alleviating aforementioned pain. We adopt the notion of context to be used in our framework facilitating lightweight semi-automatic mashup. Context is the information that characterizes the interactions between humans, applications and the surrounding environment. For example, location (e.g., geography coordinates, country, time), user information (e.g., name, address, email) or client context (e.g., hardware, software). Our future work will concentrate on achieving the following targets:

A Lightweight Programming Model

Mashup applications are often created manually by combining available components with certain predefined operators. With the increasing demand for components, operators and functionalities, the user’s task to efficiently select, combine and configure components becomes more complex, time-consuming and error prone even for experienced users.

The first target of our framework is to simplify the programming process of mashup. Mashup framework should leverage the spreadsheet model which combines functional and visual approach to deliver an intuitive environment with little or no learning barriers (Brad et al. 2004).

Semi-automatic Mashup

Most of the current mashup approaches use manual composition and full automatic support is still the target of ongoing research activity. Our second target is to provide a semi-automatic mashup framework with user intervention. In order to provide such a framework, users’ requests and components need to be stated in a way that rich semantic information can be incorporated when components are queried.

We try to embed intelligence into the manual process of mashup: service selection, binding and composition through the smart use of context. We intend to construct a user model that reflects user preferences, such as usage characteristic, interest, expertise and community model that measure the interest of users on a mashup or component. Based on these two models, our framework incrementally guides users on the process of building mashup application. In addition, when a requirement changed (e.g., interest changed) the system will automatically suggest new mashup plan to the users. Our research prototype will have two basic components: a composer and an inference engine. The inference engine stores the information about known component, user model and community model in its Knowledge Base (KB) and has the capability to suggest composition plans. The composer has a spreadsheet grid interface that enable users to incrementally build their mashup. The users start the composition process by selecting preferred profile and one component service. A query is sent to the inference engine to get the suggestion on possible mashup plans. The composers then get the results and display the plans for user to select.

References


Burnett, M., Agrawal, A. & van Zee, P. (2000), ‘Exception handling in the spreadsheet
Table 3: Evaluation results

<table>
<thead>
<tr>
<th>Data model</th>
<th>Excel connectors</th>
<th>StrikeIron</th>
<th>Extensible Extender</th>
<th>AMICO/AMLC</th>
<th>AI</th>
<th>Husky</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid-based object-based</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Component access model</td>
<td>Protocol</td>
<td>P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10</td>
<td>D1, D2, D3, D4, D5, D6, D7, D8, D9, D10</td>
<td>P1, P2, P3, P4, P6, P8, P9, P10, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10</td>
<td>P1, P2, P3, P4, P6, P8, P9, P10, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10</td>
<td>P1, P2, P3, P4, P6, P8, P9, P10, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10</td>
</tr>
<tr>
<td>Data format</td>
<td>Data format</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Extensibility</td>
<td>static</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Mashup strategies</td>
<td>dynamic</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Orchestration styles</td>
<td>manual</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Data passing styles</td>
<td>flow-based</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Target user</td>
<td>layout-based</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Search-ability</td>
<td>event-based</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Community features</td>
<td>dataflow</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Software engineering aspects</td>
<td>blackboard</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Execution engine</td>
<td>developer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Execution types</td>
<td>skilled user</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Transaction &amp; exception handling</td>
<td>novice user</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hybrid</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(+ means the dimension is directly supported; (-) means the dimension is not supported; N/A means the dimension is not related publication.

\[ p_1=HTTP; p_2=TCP; P_3=SMTP; P_4=REST; P_5=SOAP; P_7=RPC; P_8=application specific; P_9=ODBC; P_{10}=JDBC; \]
\[ d_1=HTML; d_2=XML; d_3=RSS; d_4=ATOM; d_5=JSON; d_6=XLS; d_7=RDF; d_8=Text; \]

Table 3: Evaluation results


