Bill Organiser Portal: A Case Study on End-User Composition

Agnes Ro, Lily Shu-Yi Xia, Hye-Young Paik and Chea Hyon Chon

CSE, UNSW, Sydney, Australia (`agnesr, lilyx, hpaik, cheac@cse.unsw.edu.au`)

Abstract. Whilst Web services can be composed by technical developers using a language such as BPEL, there is no easy way for non-technical end users to take advantage of these services. The advent of Web 2.0 and mashups has brought about the notion that content from different sources can be brought together by the user themselves to create a new service. Inspired by such ideas, we propose a lightweight end-user service composition paradigm, namely; *Stones*, *Stories* and *StoryBoard*. A Stone is a representation of a commonly performed task or operation that can be used to construct a Story. *StoryBoard* provides an intuitive drag-and-drop style user environment in which the Stories are created, validated and run. We demonstrate the concept through an implementation of a case study on bill management.

1 Introduction

The number of E-Commerce systems has grown dramatically over the last few years and now becoming a fundamental part of many businesses. However, with the large amount of information and services available on the Web, it is sometimes difficult for regular users[1] to piece it all together. Due to this problem, information and services portals (e.g., Yahoo, Expedia) are becoming increasingly popular.

With Web Service technology[7], it is possible to integrate business functionality into the one portal. This creates a potential service aggregation environment from which end users could greatly benefit. However, most portals stop short of providing their customers with the ability to aggregate services on-demand. They may support a simple, pre-defined workflow (e.g., book flight, rent a car, then book accommodation), or enable personalised configuration of individual services (e.g., Weather service for Sydney), but end users cannot 'compose' the service functionality on offer as a need arises.

Although service composition has been a well-discussed topic for the researchers and developers of Web services, the tools and methodologies for enabling end-user service composition have been largely ignored.

Mashups enable users to aggregate and filter information from more than one source at one convenient location. Intuitively, the concepts represented in Mashups could be applied, not only to data but also to Web services. If business

[1] We refer to them as end-users in this paper.

had their operations exposed through Web services, end users could utilise these services and engage them into a process to suit the individual's needs.

In this paper, inspired by Mashups, we propose an end-user service composition paradigm, namely; *Stones*, *Stories* and *StoryBoard*. A Stone is a representation of a commonly performed task or operation that can be used to construct a Story. *StoryBoard* provides an intuitive drag-and-drop style user environment in which the Stories are created, validated and run.

The paper is organised as follows: Section 2 discusses background and related work, followed by a description of the case study in Section 3. The end-user service composition framework is detailed in Section 4. Sections 5 and 6 discuss evaluation and concluding remarks, respectively.

2 Related Work

Many professional Web service developers can rely on the languages (e.g., BPEL[1]) and tools (e.g., Oracle Process Manager[11]) for service composition tasks. However, the same level of support has not been given to end users.

There has been an effort to simplify BPEL (e.g., Simple Service Composition Language[5]) or provide a guided-assistance along the composition process by modelling user preferences, past experience or service dependencies[6, 13]. However, we argue that none of the approaches is intuitive to end users in that they do not hide all the complexity of underlying technology. End users still have to understand the concept behind the tools or learn a language to be able to compose even a simple process.

Mashups[8, 9, 3] opened up easy access to data silos which were previously only accessible through a Web site. Major IT companies now provide so-called "end user oriented" mashup tools (e.g., Yahoo! Pipes, Microsoft Popfly) enabling the users to compose and organise existing information sources.

However, current Mashup tools and their applications are focused on accessing, manipulating data and composing data flows (e.g., filtering, merging, sorting data feeds). To use the tools effectively, the users need to know, not only how to 'program', but also how to use the different Web APIs from all services[4].

Our project is inspired by Mashup techniques in that we would like to empower the end-users with intuitive tools that allow them to create 'service' from existing 'services' as freely as their personal needs dictate, and also facilitate software reuse in mass by sharing such services with others. The core idea behind the Stones and Stories in this paper is to provide an intuitive and lightweight Web service composition environment for the users to define and execute repetitive tasks by "mashing-up" available Web service operations, without realising the complexity behind.

We would like to note that a technique commonly known as 'Web Scripting' seeks to achieve similar goals[14, 2]. For example, Koala[14] is a system that records the sequence of user actions during a Web browsing session. The sequence can be automatically repeated (i.e., playback) later in the future. Koala's main aim is for sharing commonly performed business process with could be shared with co-workers. However, Web scripting strictly applies to Web page browsing activities (e.g., requesting for a particular URL, filling in forms in the input boxes), not Web service operations.

3 Case Study: Bill Management Issues

To demonstrate the lightweight end-user composition concept, we take bill payment management as a case study.

The average consumer household has bills for water, electricity, telephone and gas. Some have additional bills for Internet, pay TV, health insurance, and car insurance. If the consumer owns their own home they receive bills for council rates and strata levies, otherwise they make rental payments. Individuals in the household receive bills for mobile phones, newspaper subscriptions, magazine subscriptions, and education/tuition fees. The bills may arrive monthly, quarterly, annually, and may have payment options via Post BillPay, direct debt, BPAY, Bill Express, or credit card.

Managing these bill payments is undeniably a substantial task which takes up a considerable amount of an individual's time and effort. Bills must be paid on time using an accepted payment format and paper bills need to be sorted and stored appropriately after payment. Each payment option requires the consumer to supply customer reference numbers and bill reference numbers which may change dynamically with every bill or may remain the same, depending the issuer of the bill or the payment option. It is clear that the repetitive tasks involved in the process of bill payment should be handled by a bill management system.

We implement a system named Billing Organiser Portal (BOP). In BOP, for example, a consumer may create a custom *Story* which retrieves all outstanding bills, pay them with a particular credit card and receive the receipt in an email.

Although currently there are existing systems that assist in bill management, there is currently no solution to integrate the entire bill management process from the biller to a consumer's financial service account, not to mention the ability to compose new functionality from existing services. In the following, we describe our experience with designing and building BOP.

3.1 Overview of Bill Organiser Portal

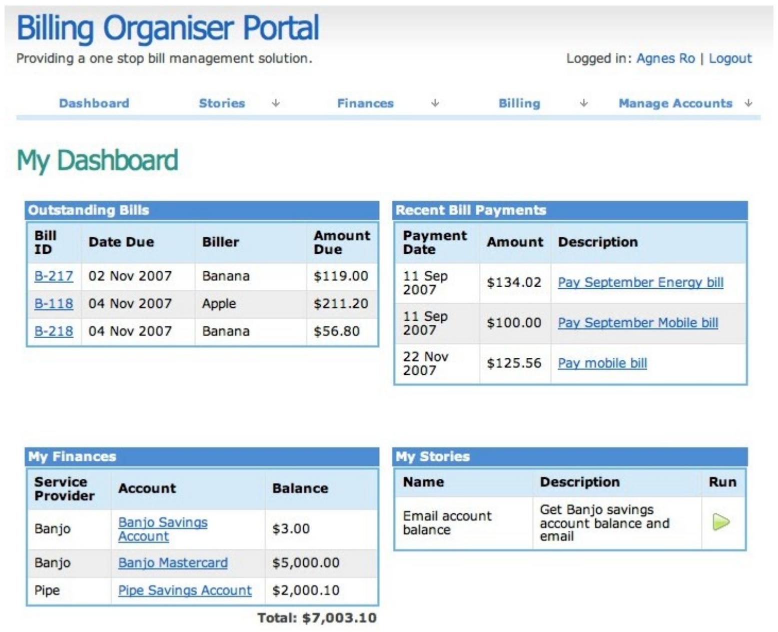
BOP uses Web services to aggregate business functionality such as the issuing and paying of bills for all bill providers as well as the displaying and managing of funds for all financial accounts at a single location customised for the individual consumer. In addition, BOP uses this portal as an avenue to explore the use of a simple, intuitive user interface for end-user Web service composition.

BOP acts as a single contact point for all businesses to interact with a specific consumer. Consumers who log on to BOP will be able to see bills issued by all of their registered billers as well as the funds available in all of their registered financial accounts. The consumer can then directly make payments for their bills from their chosen financial accounts.

Due to the repetitive nature of these bill management tasks, they are perfect candidates for Web service composition by the end user. Once a consumer has composed their own personal Web service process as *Stores*, they can run the process in one simple step, schedule the process to run at a certain point in time, or even share the process with other consumers who may wish to perform similar tasks.

3.2 Implementing the Web Application Module

The Web application module of BOP has been designed based upon the Model-View-Controller design pattern, standard for most Web applications today. The module is responsible for managing users and their billing and financial service accounts. Various Java-based application frameworks such as Hibernate (for object/relational persistence and query service), Spring(for managing dependency injection among Java objects) and WebWork (for effectively managing User Interface issues) were utilised. A typical look and feel of the application is shown in Figure 1. Full implementation details can be found in [12].



Billing Organiser Portal
Providing a one stop bill management solution.
Logged in: Agnes Ro | Logout
Dashboard
Stories
Finances
Billing
Manage Accounts

My Dashboard

Outstanding Bills

Bill ID	Date Due	Biller	Amount Due
B-217	02 Nov 2007	Banana	$119.00
B-118	04 Nov 2007	Apple	$211.20
B-218	04 Nov 2007	Banana	$56.80

Recent Bill Payments

Payment Date	Amount	Description
11 Sep 2007	$134.02	Pay September Energy bill
11 Sep 2007	$100.00	Pay September Mobile bill
22 Nov 2007	$125.56	Pay mobile bill

My Finances

Service Provider	Account	Balance
Banjo	Banjo Savings Account	$3.00
Banjo	Banjo Mastercard	$5,000.00
Pipe	Pipe Savings Account	$2,000.10

Total: $7,003.10

My Stories

Name	Description	Run
Email account balance	Get Banjo savings account balance and email	▶

Fig. 1. A typical user interface: BOP Dashboard Screen

3.3 Implementing the Web Services Module

For implementation purposes, we have defined two separate interfaces for Billing and Financial Service. External businesses that interact with BOP are expected to adopt the interfaces. It would be ideal if the interfaces for such operations were defined by an industry acknowledged Web services standards body. However the generalisation difficulties in standards definition for such industry specific functionality mean that such standards are not in existence[10].

The following snippet shows the interface definition that Billing Service providers must implement in order for BOP to interact directly with their system. Evidently, billing providers must provide the critical functionality, for example, to view individual consumer bills online (e.g., `getOutstandingBills()`). Similarly, there is also an interface required to be implemented by Financial Service providers.

```
public interface BopBillingService {
 public String authenticate(); // Returns xhtml form to be displayed for authentication
 public Bill[] getOutstandingBills(); // Returns a list of Bill Objects
 public String[] getPaymentMethods(); // Returns a list of Payment methods accepted by the biller
 public String getBillerCode(); // Return the BPAY Biller code of the biller
 public AccountDetails getAccountDetails(); // Return the biller's account details
 public String payBillViaCC(); // Pay a bill with the given credit card details
}
```

4 End-User Composition in BOP

Stones and Stories: A Stone is a representation of a commonly performed task or operation that can be used to construct a Story. Each Stone has a set of defined input types it can take in to process for the `run()` method (Figure 3). The output type for each Stone is the output type returned by the `run()` method. For our case study implementation, we defined eight implementations of the Stone interface.

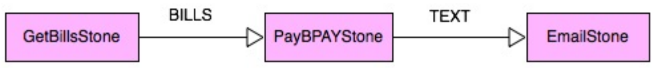


Fig. 2. Connecting Stones through Input and Output

A Story is a sequence of Stones. In order for a Story to be valid, the output type of a Stone must be an input type for the following Stone. For example, the `GetBillsStone` returns a `BILLS` type and the `PayBPAYStone` accepts `BILLS` as its input for processing and so on (Figure 2).

StoryBoard: StoryBoard is the end-user composition environment in which Stories are created. Giving the user the power to compose Web service composition in a simple and user friendly interface, proved to be a difficult task. The

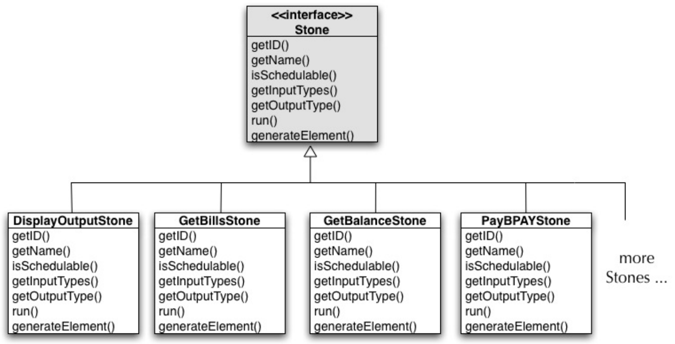


Fig. 3. The Design of Stones

final design is a product of several cycles of usability testing. We concluded that a drag/drop approach using Javascript, where Stones could be 'dragged' and 'dropped' onto a *Storyboard* would be the most intuitive design.

StoryBoard displays available Stones at the bottom-half of the screen. The top-half of the screen is the composition area into which the Stones are placed. StoryBoard also adds a simple validation support for the user. To ensure a valid composition is constructed by the user, only the Stones that can be placed on the next step in the Storyboard are highlighted and enabled (i.e., draggable). The other Stones are grayed out and disabled as illustrated in Figure 4. Also, the user is not allowed to skip a step (i.e., leave an empty step in between Stones). When a Stone requires inputs from the user, a popup window is automatically displayed when the Stone "clicks" into its place (Figure 5).

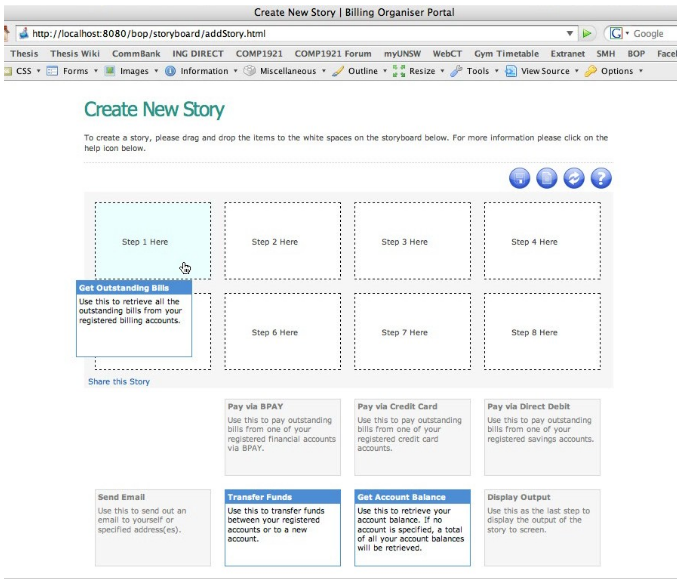
The actual content of the Story stored is a string of XML, containing the sequence of Stones to be executed as well as the parameters for each Stone. The schema definition for the Story content is not shown in the paper for space reasons. The `StoryManager` class is responsible for the processing and management of Stories. It relies on the `StoryDAO` class to store/retrieve/update Stories to the database, as well as the `StoryParser` class to parse the Story content from XML to Object form and vice-versa.

Once a Story is created, it can be made visible to other users in the system. A shared Story is treated like a template. Such a template contains a sequence of Stones, but no input parameters are associated with each Stone. When a shared Story is imported by a user, the user needs to update the parameters appropriately (e.g., credit card detail) before running the imported Story.

5 Evaluation

5.1 User Evaluation

Throughout the design of the system, we conducted usability testing with users and improved the design and functionality based on the feedback. This cycle was


Create New Story | Billing Organiser Portal
http://localhost:8080/bop/storyboard/addStory.html
Google
Thesis Thesis Wiki CommBank ING DIRECT COMP1921 COMP1921 Forum myUNSW WebCT Gym Timetable Extranet SMH BOP
CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
Step 1 Here
Step 2 Here
Step 3 Here
Step 4 Here
Step 6 Here
Step 7 Here
Step 8 Here
Share this Story

Create New Story

To create a story, please drag and drop the items to the white spaces on the storyboard below. For more information please click on the help icon below.

Get Outstanding Bills Use this to retrieve all the outstanding bills from your registered billing accounts.

Pay via BPAY Use this to pay outstanding bills from one of your registered financial accounts via BPAY.

Pay via Credit Card Use this to pay outstanding bills from one of your registered credit card accounts.

Pay via Direct Debit Use this to pay outstanding bills from one of your registered savings accounts.

Send Email Use this to send out an email to yourself or specified address(es).

Transfer Funds Use this to transfer funds between your registered accounts or to a new account.

Get Account Balance Use this to retrieve your account balance. If no account is specified, a total of all your account balances will be retrieved.

Display Output Use this as the last step to display the output of the story to screen.

Fig. 4. For usability purposes, a toolbar at the top right corner was added for saving the Story, loading a shared Story, refreshing the storyboard and Story help. This figure shows 'Create New Story' screen.

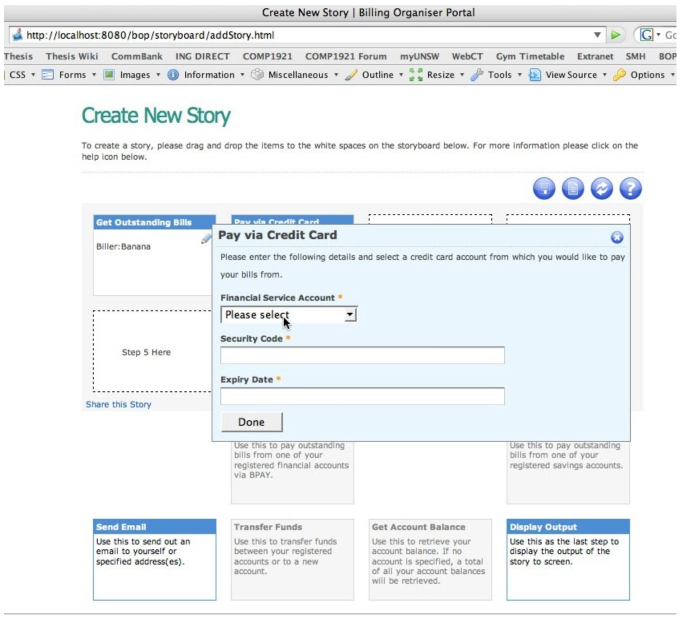
repeated a number of times before we finalised our design. Initially, most users seemed to have difficulty in creating a Story and adding a new billing account. The user interface design that was substantially improved via user evaluations was the create story screen. Ratings range from 2 to 5 as improvements were continually made in between evaluations.

In the final evaluation, total of 10 users were asked to perform the evaluation survey. Results are summarised in Figure 7 and 8. We asked the users to rate consistency, ease of navigation, access to help, intuitiveness, visibility of system status, aesthetic and minimalist design, user control and freedom and error handling. All criteria scored minimum of 3.8/5 or higher, except for user control and freedom which rated 3.4. This indicated that the system's lack of support for undoing user mistakes. Majority of the users indicated that they would use the system if publicly available and agreed that such system will be very useful for their daily lives.

5.2 Developer Reflection

The current implementation of BOP is a full working system, that effectively demonstrates the concept of end-user composition in a business portal.

One of the major weakness in our system, is admittedly security. We are well aware that the information passed throughout the system and to the service


Create New Story | Billing Organiser Portal
http://localhost:8080/bop/storyboard/addStory.html
Thesis Thesis Wiki CommBank ING DIRECT COMP1921 COMP1921 Forum myUNSW WebCT Gym Timetable Extranet SMH BOP
CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
Get Outstanding Bills
Pay via Credit Card
Step 5 Here
Share this Story
Done
Send Email
Transfer Funds
Get Account Balance
Display Output

Create New Story

To create a story, please drag and drop the items to the white spaces on the storyboard below. For more information please click on the help icon below.

Biller: Banana

Pay via Credit Card

Please enter the following details and select a credit card account from which you would like to pay your bills from.

Financial Service Account *: Please select

Security Code *

Expiry Date *

Use this to pay outstanding bills from one of your registered financial accounts via BPAY.

Use this to pay outstanding bills from one of your registered savings accounts.

Use this to send out an email to yourself or specified address(es).

Use this to transfer funds between your registered accounts or to a new account.

Use this to retrieve your account balance. If no account is specified, a total of all your account balances will be retrieved.

Use this as the last step to display the output of the story to screen.

Fig. 5. A Stone automatically displays input boxes for required inputs

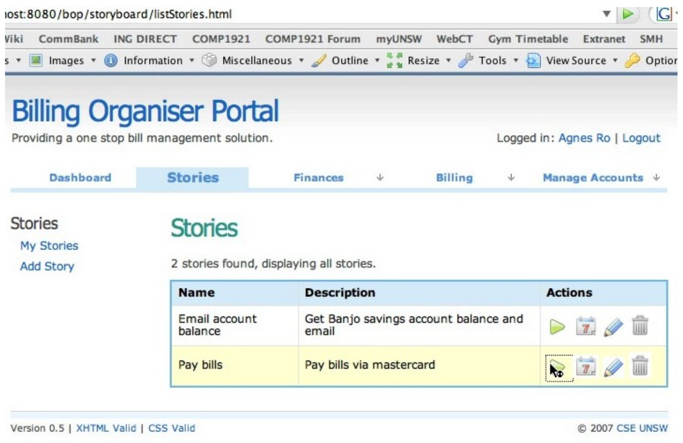
providers is highly critical and a target for abuse. For this case study, we have only applied minimal security measures such as encryption of passwords.

With the integration of multiple business services in BOP, it is difficult to maintain the ideal ACID properties for transaction management. For a system that primarily deals with finances being credited/debited between billers and financial service providers, it is vital that in the failure of such a transaction, the entire process is rolled back and all systems are back in it's original state. Although this issue has been identified, it is not something that our implementation currently supports.

6 Conclusion

Web services and the advance of SOA have allowed businesses to redesign their internal systems into modularised services that they can expose to other services in their domain as well as to the world. Whilst these Web services can be composed by technical developers using a Web service composition language such as BPEL, there is no easy way for non-technical end users to take advantage of these services. This project is a study of the practicability of a simple, intuitive user interface allows end users to create their own Web service composition using services from various different sources together in a manner which is appropriate for their own personal needs.

The development of the Story/Stone Web service composition module of this paper proposes an abstraction of Web services from different sources which



ost:8080/bop/storyboard/listStories.html

Viki CommBank ING DIRECT COMP1921 COMP1921 Forum myUNSW WebCT Gym Timetable Extranet SMH

s Images Information Miscellaneous Outline Resize Tools View Source Optior

Billing Organiser Portal

Providing a one stop bill management solution.

Logged in: Agnes Ro | Logout

Dashboard Stories Finances Billing Manage Accounts

Stories
My Stories
Add Story

Stories

2 stories found, displaying all stories.

Name	Description	Actions
Email account balance	Get Banjo savings account balance and email	
Pay bills	Pay bills via mastercard	

Version 0.5 | XHTML Valid | CSS Valid

© 2007 CSE UNSW

Fig. 6. The user can click on the Run icon to execute a composed Story

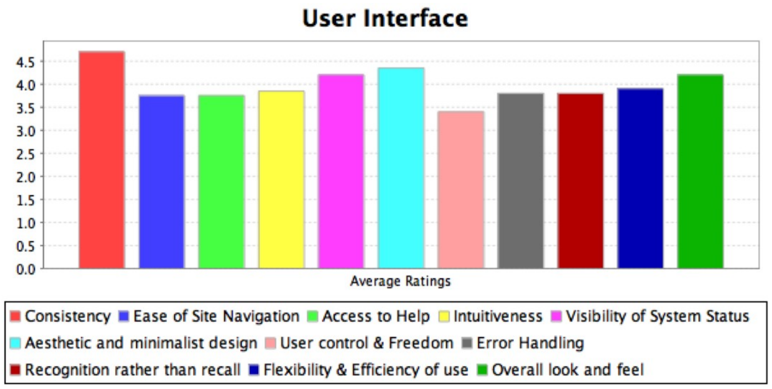


Fig. 7. User Interface Evaluation

allows the end user to compose their own composite services without knowing about the low level details of such an endeavour.

References

1. BPEL. Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/specification/ws-bpel.
2. Hupp D and Miller R. Smart bookmarks: Automatic Retroactive Macro Recording on the Web. In *Proc. of ACM symposium on User Interface Software and Technology*, pages 81–90, 2007.
3. Huynh D, Karger D, and Miller R. Exhibit: Lightweight Structured Data Publishing. In *Proc. of IntConf on World Wide Web*, pages 737–746, 2007.

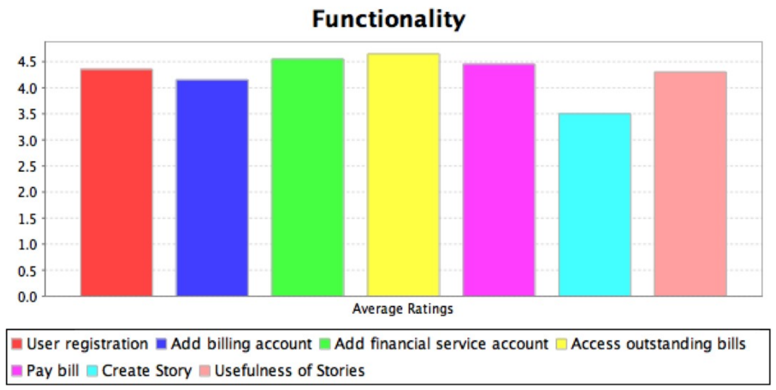


Fig. 8. Functionality Evaluation

4. Di Lorenzo G, Hacid H, Paik H, and Benatallah B. Mashups for Data Integration: An Analysis, 2008. School of Computer Science and Engineering, University of New South Wales, Technical Report 0810, http://cgi.cse.unsw.edu.au/∼reports/.
5. Gavran I, Milanovic A, and Srbljic S. In *Proc. of 7th Workshop on Distributed Data and Structures*, 2006. Santa Clara, CA.
6. Han J, Han Y, Jin Y, Wang J, and Yu J. Personalized active service spaces for end-user service composition. In *Proc of IEEE International Conference on Services Computing (SCC'06)*.
7. Huhns M and Singh M. Service-Oriented Computing: Key Concepts and Principles. *Internet Computing*, 9(1):75–81, 2005.
8. Maximilien M, Wilkison H, Desai N, , and Tai S. A Domain Specific Language for Web APIs and Services Mashups. In *Proc. of IntConf on Service Oriented Computing*, pages 13–26, 2007.
9. Diaz O, Perez S, and Paz I. Providing Personalized Mashups Within the Context of Existing Web Applications. In *Proc. of IntConf on Web Information Systems Eng.*, pages 493–502, 2007.
10. Zimmermann O, Milinski S, Craes M, and Oellermann F. Second Generation Web Services-Oriented Architecture in Production in the Finance Industry. In *Proc. of IntConf on Object Oriented Programming Systems Languages and Applications*, pages 283–289. ACM Press, 2004.
11. Oracle. Oracle BPEL Process Manager. www.oracle.com/appserver/bpel_home.html.
12. Agnes R and Xia L. End User Web Service Composition for a C2B Portal, 2007. School of Computer Science and Engineering, University of New South Wales, Thesis, http://www.cse.unsw.edu.au/∼hpaik/pdf/3100983.ThesisB.pdf.
13. Bova R, Paik H, Hassas S, Benbernou S, and Benatallah B. On embedding task memory in services composition frameworks. In *Proc of 7th International Conference on Web Engineering*. July 16-20, Como, Italy, Springer, pp.1-16.
14. Lau T. Social Scripting for the Web. *Computer*, 40(6):96–98, 2007.