<table>
<thead>
<tr>
<th><strong>Title:</strong></th>
<th>End user Web Service Composition for a C2B Portal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors:</strong></td>
<td>Lily Xia, Agnes Ro</td>
</tr>
<tr>
<td><strong>Degree:</strong></td>
<td>Software Engineering</td>
</tr>
<tr>
<td><strong>Submission Date:</strong></td>
<td>29th October 2007</td>
</tr>
<tr>
<td><strong>Supervisor:</strong></td>
<td>Helen Paik</td>
</tr>
<tr>
<td><strong>Assessor:</strong></td>
<td>John Sheperd</td>
</tr>
</tbody>
</table>
Thesis B Report

Contents

• 1 Introduction
• 2 Background
  ° 2.1 E-Commerce
  ° 2.2 Web Service Composition
  ° 2.3 Bill Management Issues
  ° 2.4 Existing Systems
• 3 BOP
  ° 3.1 Overview
  ° 3.2 Requirements
  ° 3.3 System Design
    - 3.3.1 Web Application Design
      - 3.3.1.1 Data Layer
      - 3.3.1.2 Presentation Layer
        - 3.3.1.2.1 Navigation
        - 3.3.1.2.2 User Interface
      - 3.3.1.3 Business Layer
        - 3.3.1.3.1 Business Classes
        - 3.3.1.3.2 Web Services
    - 3.3.2 Web Service Composition Design
      - 3.3.2.1 Data Layer
      - 3.3.2.2 Presentation Layer
      - 3.3.2.3 Business Service Layer
  ° 3.4 Implementation
    - 3.4.1 Web Application Implementation
      - 3.4.1.1 Application Frameworks
      - 3.4.1.2 Web Services Registry
      - 3.4.1.3 Authentication
      - 3.4.1.4 Third Party Web Services
    - 3.4.2 Web Service Composition Implementation
  ° 3.5 Testing & Evaluation
• 4 Project Management
• 5 Conclusion
  ° 5.1 Future Work
• 6 References
• 7 Appendix
  ° 7.1 Assumptions
  ° 7.2 Screenshots
  ° 7.3 Evaluation Survey
  ° 7.4 Glossary
  ° 7.5 Detailed Class Diagrams
  ° 7.6 Stone Definitions
  ° 7.7 BOP Gantt Chart
  ° 7.8 Existing Bill Aggregators
  ° 7.9 Time Tracking
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 to piece it all together. Due to this problem, portals and mashups are becoming increasingly popular today.

Portals create a convenient environment for users, providing a single point of interaction between businesses. With Web Service technology, it makes it possible to integrate business functionalities into the one portal, which consumers could greatly benefit from. One such system could be a bill and personal financial management system, where both the billing and finance companies could have their business services integrated into one portal. Such a system would be able save a great amount of time and effort, consumers spend on bill management. 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.

Mashups enable users to accumulate and filter information from more than one source at one convenient location. Theoretically, this concept could be applied, not just to data but also to web services. If business had their operations exposed through web services, consumers could utilise these services and engage them into a process to suit the individual’s needs. For example, a consumer may like to create a process (i.e. web service composition) to make all his/her bill payments from a particular credit card and receive the receipt numbers in an email. However, this opportunity of web service composition for the end user is currently not available.

Primarily, our thesis attempts to provide an example implementation of how users can benefit from a system that provides users the ability to compose their own web service composition and also business to business service integration. Additional to the implementation, we are able to asses the issues and problems faced when such a system is built. To narrow down our scope, we have chosen the context of personal bill and finance management for our system, Billing Organiser Portal (BOP).
2 Background

This section provides background information on the core concepts and topics of our thesis. This includes e-Commerce, web service composition, current bill management issues and the existing systems in these areas.

- 2.1 E-Commerce
- 2.2 Web Service Composition
- 2.3 Bill Management Issues
- 2.4 Existing Systems
2.1 E-Commerce

There are three business models in the area of e-commerce:

- Business to business (B2B) e-commerce in the form of supply chain management systems.
- Business to consumer (B2C) e-commerce prevalent in online shopping and business websites.
- Consumer to consumer (C2C) e-commerce found in online trading systems of which the most common is eBay.

The fourth obvious model, consumer to business (C2B), is currently almost non-existent. In virtually all interactions between consumers and businesses, it is the consumer who has to go to log on to the business's website with a username and password specific to that website in order to access services provided by that business.

This project proposes a web services portal which provides C2B e-commerce capabilities built on the integration of business web services from all the organisations that an individual consumer conducts business with. The high-level idealisation is that through this single portal a consumer can perform all of their banking, financial trading, utilities management, and business and personal contact needs. Some simple utilisations of the wide ranging opportunities that such a portal would offer are:

- A single place where a consumer that moves house has to update their new address. This new address can then be propagated through to all the businesses that the consumer maintains address information with. The post office can be informed to forward all the consumer's mail to the new address. The consumer's utility accounts can all be transferred automatically to the new address. The efficiency and cost benefits to both businesses and the consumer would be considerable.
- A single place where a consumer has to inform businesses of a lost wallet. The business processes of card access cancellations and issuance of new cards can then be initiated for all businesses that the consumer does business with. The efficiency of such a system and the reduced risk of loss are obvious benefits to all parties involved.

Such a consumer portal is possible with web services technology and the widespread business adoption of service oriented architecture.
### 2.2 Web Service Composition

The Web, originally contemplated for human use is now evolving with modular services, enabling reuse by other systems or services themselves. Web services typically share business logic, data and processes through an interface across a network. The technology promotes interoperability and enables data to be shared between different machines or systems independent from the platform or framework of the machine/system. One of the chief advantages of web services and indeed of any modular system is that these exposed services or units of business logic can be assembled to create more complex business processes or operations; this is known as web service composition.

For developers, Web services provides the ability to reuse existing business processes to construct more comprehensive processes, without having to worry about the underlying implementations of the services. Languages such as the Business Process Execution Language (BPEL) exist to enable analysts to construct business processes or flows, without having to worry about implementation details at all (provided the services required exist!).

For both the developer and business analyst, web service composition supports extensive re-usability of existing functionality for processes that may be performed repetitively. However, this opportunity is not currently given to the end user. As an average internet user, we often perform repetitive tasks (or even a set sequence of tasks) over the web. For example, checking your account balance and then paying your bills. If the power of web service composition was provided to the average user, they would be able to benefit from web service composition to reduce the number of repetitive tasks performed. This is similar to the rational behind the Web 2.0 movement; allowing the end user to become an active participant in the construction of the Web without inundating them with the technical details of systems development.

Giving the user the ability to construct their own web service composition would further promote the reuse of existing web services. If the compositions were stored, the user could then invoke it manually or schedule it to run periodically. This could potentially save the user a lot of time and effort.
2.3 Bill Management Issues

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.
2.4 Existing Systems

**Mashups**

Mashups provide users a convenient way to aggregate information from more than one source. A good example of such a system is Yahoo Pipes. It mainly enables users to combine and manipulate RSS feeds into a 'pipe' which can also be shared between other users.

**IBM's Koala Project**

Koala[22] is a system that can record and automate processes performed in a web browser. It lets you make a recording as you perform the procedure and play it back in the future. Koala's main aim is for sharing commonly performed business process with could be shared with co-workers.

**Bill Aggregators**

Currently there are many exisiting solutions available which assist in bill management, however most of these systems enact as "bill aggregators". We will refer to them as bill aggregators as they only provide the simple functionality of accumulating a user's bills and enabling the user to pay them from one place. They do not incorporate personal finance management (e.g. online banking) for multiple finance institutions or the ability to pay a bill from the site by interacting directly with multiple financial service providers. Each bill aggregator has it's own set of features as well as limitations, which we have evaluated in Appendix 7.8.
This section documents the specification, implementation, and evaluation of our system, Billing Organiser Portal.

- 3.1 Overview
- 3.2 Requirements
- 3.3 System Design
- 3.4 Implementation
- 3.5 Testing & Evaluation
3.1 Overview

Bill Organiser Portal (BOP) is a C2B bill management portal built on a foundation of web services and business integration. BOP is a conceptual case study of the idealised portal described in Section 2.1 of this document where all business interaction with a single consumer is conducted through a single 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. As far as we are aware this is the first case study where web service composition by end users has been investigated.

Diagram 3.1A Interactions between BOP and Service Providers

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, 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. The following sections detail the design and implementation of the system as well as an evaluation of the final outcome.
3.2 Requirements

The full set of BOP requirements are outlined below. Requirements have been given priorities between 1-4, with 1 being the highest priority and 4 being the lowest. For the purposes of the project, scheduled implementation will concentrate on the requirements with priority 1. Implementation will then move on to lower priority requirements as time permits. The priority 1 core requirements demonstrate the essential working elements of the consumer portal.

These requirements have been based upon several assumptions, which have been included in Appendix 7.1.

Functional Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
<th>Priority</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-01</td>
<td>The portal will allow the end user to create a user account.</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-02</td>
<td>The portal will allow the end user to register billing &amp; financial service accounts.</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-03</td>
<td>The portal will allow the end user to remove billing &amp; financial service accounts.</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-04</td>
<td>The portal will authenticate the end user with registered billing &amp; financial service accounts directly with the service providers. (No additional login should be required after login to BOP)</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-05</td>
<td>The portal will allow the end user to login/logout.</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-06</td>
<td>The portal will allow the end user to view unpaid bills provided by the billing accounts registered.</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-07</td>
<td>The portal will allow the end user to view previously paid bills provided by the billing accounts registered.</td>
<td>2</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-08</td>
<td>The portal will allow the end user to pay a bill by immediate transfer of funds from</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>FR-09</td>
<td>The portal will allow the end user to pay a bill by scheduling a transfer (in the future) of funds from their financial service account(s).</td>
<td>2</td>
<td>✔</td>
</tr>
<tr>
<td>FR-10</td>
<td>The portal will allow the end user to edit/remove a scheduled bill payment transfer.</td>
<td>2</td>
<td>✔</td>
</tr>
<tr>
<td>FR-11</td>
<td>The portal will allow the end user to set-up automatic payment of bills from their financial service account(s).</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FR-12</td>
<td>The portal will allow the end user to edit/remove an automatic bill payment set-up.</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FR-13</td>
<td>The portal will record bill payment transactions.</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>FR-14</td>
<td>The portal will allow the end user to view/search bill payment transactions.</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>FR-15</td>
<td>The portal will allow the administrator to register/remove web services from producers.</td>
<td>2</td>
<td>✔</td>
</tr>
<tr>
<td>FR-16</td>
<td>The portal will allow the administrator to ban and blacklist fraudulent producers.</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
| FR-17 | The portal will allow the end user to set preferences for email notification on the following events:  
  • Notice of new bill  
  • Reminder of bill payment (Including number of days before due date)  
  • Payment of bill | 3 |
<p>| FR-18 | The portal will allow the end user to pay bills in another person’s name. | 4 | ✔ |</p>
<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
<th>Priority</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-19</td>
<td>The portal will allow the end user to transfer funds between their FSAs immediately.</td>
<td>2</td>
<td>✔</td>
</tr>
<tr>
<td>FR-20</td>
<td>The portal will allow the end user to schedule the transfer funds between their FSAs.</td>
<td>3</td>
<td>✔</td>
</tr>
<tr>
<td>FR-21</td>
<td>The portal will allow the end user to edit/remove scheduled fund transfers.</td>
<td>3</td>
<td>✔</td>
</tr>
<tr>
<td>FR-22</td>
<td>The portal will allow the end user to set-up automatic transfer of funds between their FSAs.</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FR-23</td>
<td>The portal will allow the end user to edit/remove automatic fund transfers.</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FR-24</td>
<td>The portal will allow the end user to compose web services to create a Story</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>FR-25</td>
<td>The portal will allow the end user to edit/remove a Story.</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>FR-26</td>
<td>The portal will allow the end user run a Story</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>FR-27</td>
<td>The portal will allow the end user to create and share Stories</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>FR-28</td>
<td>The portal will allow the end user to schedule a Story to run periodically.</td>
<td>2</td>
<td>✔</td>
</tr>
</tbody>
</table>

### Non Functional Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
<th>Priority</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFR-01</td>
<td>The portal will allow the end user to configure their dashboard using BOP plugins.</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>NFR-02</td>
<td>The portal will support a plugin subsystem. Plugins will include:</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
• FSA balance summary
• Outstanding bills summary
• Scheduled bill payments summary
• Most recent bill payments summary
• Individual FSA transaction summary
• Scheduled transfers summary (FSA transfers)
• Theme (Layout, colour, font, font size)

NFR-03  The portal will support a drag and drop (of individual web services) UI for the constructing of a Story. This will make it easier for the user, who shouldn't have to fully understand the concept of web service composition.  2  

Requirements Completion

The implementation of BOP has resulted in the completion of almost 75% of initial requirements, with 100% of core priority 1 and priority 2 requirements met. The requirements not completed were all lower priority requirements related to functionality enhancements within the portal for the convenience of the user such as automatic notifications, automatic payments and transfers, and customisation of the look and feel of the portal. These non-core requirements were left out solely due to time constraints on the project. Neither the core functionality of the portal nor the purpose of the project which was to demonstrate the C2B portal concept and the end user web service composition initiative were affected by their non-completion.
3.3 System Design

System design for BOP has been documented in two sections, where design for the Web Service Composition module of the system has been separately documented from the main web application design.

- 3.3.1 Web Application Design
- 3.3.2 Web Service Composition Design
3.3.1 Web Application Design

BOP has been designed based upon the MVC design pattern, standard for most web applications today. The three applications layers and their interactions are illustrated in the diagram below. The business layer (Controller) will be responsible for communication between the presentation (View) and data layers (Model), separating the responsibilities of managing data, business logic and providing information to the user interface.

- 3.3.1.1 Data Layer
- 3.3.1.2 Presentation Layer
- 3.3.1.3 Business Layer

Diagram 3.3A BOP Application Layers
3.3.1.1 Data Layer

The following is an Entity Relationship Diagram (ERD) of the BOP system illustrating the data layer of the system.

Assumptions:

- payments are always made from one and only one FS account
- one WSDL per provider

A User can have many Accounts which are either Billing or Financial Service Accounts. Both Billing and Financial Service Accounts subclass the Account entity, represented by the isa disjoint relationship.

Each Account is linked to a single Service Provider which encapsulates the service details which BOP can use to call and perform operations on the Account. Bills are sent to a particular Billing Account which are
retrieved from the linked Service Provider. These Bills can be paid from a user's Financial Service Account. The Payment entity is responsible for the record of a Bill payment from a Financial Service Account.

It is important to note that the Bill and FS Account attributes will not be obtained from the BOP data store, but from the Biller's and FS' data store via web service calls. As there is no direct ERD notation for this kind of 'derived' attribute, the diagram has been illustrated as shown above.

**Changes from Thesis A**

1. The Role entity was introduced by using Appfuse to setup the initial web application. We have decided to keep this entity to differentiate between the two types of users in the system; Administrators and BOP users.
2. The multi-valued attribute allowedPaymentMethod has been removed from the BillingAccount entity and replaced with a web service call to the biller. It was concluded that allowed payment types can change and such information should be directly retrieved from the billing service provider.
3.3.1.2 Presentation Layer

This section presents the design of the portal interface and navigation. The basic features of the graphical user interface and the main screen flows and interactions are depicted.

- 3.3.1.2.1 Navigation
- 3.3.1.2.2 User Interface
3.3.1.2.1 Navigation

The following diagram shows the navigational structure of BOP. Navigation throughout the site is achieved by a series of tabbed menus across the top of the screens and a sub-menu down the left-hand side of the screen as per the screenshots in Appendix 7.2.
3.3.1.2.2 User Interface

This section demonstrates the screen designs for the BOP system. Tabs across the top have been used for site navigation between the following main entry points:

- Dashboard
- Stories
- Manage Accounts
- Billing
- Finances

Additionally a menu for each of the tabs will be provided on the left hand side of the screen as illustrated below:

![Diagram 3.3.1.2.2A Outstanding Bills Summary](image1)

The above is a sample screen design of a user's outstanding bills summary. From this view, the user will be able to easily view or pay a bill and perform their bill management as required.

![Diagram 3.3.1.2.2B Manage Financial Service Accounts](image2)

The above is a sample screen design of a user's outstanding bills summary. From this view, the user will be able to easily view or pay a bill and perform their bill management as required.
A separate tab for account management will be provided where the user can add/remove their FS or Billing accounts.

Diagram 3.3.1.2.2C Financial Service Accounts Summary
The above screen design is a sample view of a user's FS accounts summary. From this view, the user will be able to easily view and manage their financial accounts without having to login to the respective online banking site individually.

Diagram 3.3.1.2.2D Dashboard
The above screen is the dashboard, where a short summary of the user's billing and finances can be easily viewed. It will be the first screen user's will see upon login.

The remaining screenshots can be found in the Appendix 7.2.
3.3.1.3 Business Layer

This section provides the overall design of the BOP business layer. This layer includes the business classes which will be responsible for business logic as well connection to the FS and biller providers by web services.

- [3.3.1.3.1 Business Classes](#)
- [3.3.1.3.2 Web Services](#)
3.3.1.3.1 Business Classes

The following is a list of the business classes which will enact as facades performing business logic and validation required. They will the main 'control' of the system, responsible for passing/processing information between the presentation and data layers. For a detailed class diagram on the relationships between the business classes see Appendix 7.5.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserManager</td>
<td>The User Manager will be responsible for general user management. This includes creating, updating, user login authentication and checking for the availability of usernames.</td>
</tr>
<tr>
<td>RoleManager</td>
<td>The Role Manager will be responsible for managing user roles in the BOP system.</td>
</tr>
<tr>
<td>AccountManager</td>
<td>The Account Manager will be responsible for managing both billing and financial service accounts. New accounts will be authenticated with the service provider before being registered in BOP as an 'approved' account. Accounts will be removed from a user by disabling the account, but account data will not actually be deleted for history purposes.</td>
</tr>
<tr>
<td>ServiceProviderManager</td>
<td>The ServiceProviderManager will responsible for storing/retrieving Service Provider information.</td>
</tr>
<tr>
<td>BillManager</td>
<td>The Bill Manager will be responsible for retrieving the bills (via web service calls) for a user's Billing Account.</td>
</tr>
<tr>
<td>PaymentManager</td>
<td>The Payment Manager will be responsible for processing a Bill payment (via web service calls) from a Financial Service Account. It will also provide search capabilities for bill payments.</td>
</tr>
<tr>
<td>TransferManager</td>
<td>The Transfer Manager will be responsible for the processing of fund transfers (via web service calls) from a financial services account to another.</td>
</tr>
</tbody>
</table>
The following is a detailed class diagram for the UserManager demonstrating the relationships between classes, acting as a facade. Actions from the presentation layer will call the UserManager interface and the implementation will call the correct operations in sequence, applying business logic and validation where required. Other business classes will be applied in a similar fashion.
3.3.1.3.2 Web Services

As two types of service providers are identified to interact with BOP, two separate interfaces have been defined for Billing and Financial Service providers. These interfaces are vital to BOP to allow for the authentication and authorization of the end user, retrieval of bills and account balances as well as payment of bills and fund transfers.

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. For implementation purposes we have defined the interfaces for such functionality which external businesses that interact with BOP are expected to adopt.

End user authentication and authorisation must be supported by both billing and financial service providers for external business functionality from within BOP. However different providers have different authentication mechanisms, which must also be supported upon new account registration in BOP.

Examples of the different authentication methods by service providers

In order to cater for this, it was decided that the authenticate method would return a String consisting of an XHTML form which could simply be embedded in BOP. The form would be submitted to the service provider, authenticating the user and establishing a common federated ID between the provider and BOP. This would all happen in the back-end, oblivious to the user and once complete the browser would simply be redirected to BOP. Next time the user logs in to BOP, they will not have to provide authentication details to the service provider again, but internally BOP will pass on the federated ID that was generated during the initial authentication process.

The following code is the interface that Billing Service providers must implement in order for BOP to interact directly with their system. Evidently, billing providers must provide the critical functionality to view individual consumer bills online i.e getOutstandingBills. BOP will support the viewing of bills in any format the bill providers provide the bills in, including document, HTML and PDF formats. Bill providers must also provide BOP with details about individual bills such as due dates, total amounts, and acceptable bill payment methods which have traditionally been provided to end users via the actual bills.

BopBillingService.java

```java
package com.bop.ws;

public interface BopBillingService {

    // Returns xhtml form to be displayed for authentication
    public String authenticate(String token, String accountId, String accountName, String accountNumber);

    // Returns a list of Bill Objects
    public Bill[] getOutstandingBills(String token, String federatedId);
}
```
// Returns a list of Payment methods accepted by the biller
public String[] getPaymentMethods(String token);

// Return the BPAY Biller code of the biller
public String getBillerCode(String token);

// Return the biller's account details
public AccountDetails getAccountDetails(String token);

/**
 * Pay a bill with the given credit card details. For demonstration purposes we assume that all the
 * payment processing. Ideally in the real world we wouldn't have this method and BOP would simply
 * invoke a 'Payment Gateway Processor'.
 */
public String payBillViaCC(String token, String billId, Float amount, String creditCardName, String
creditCardNumber, String expiryDate, String securityCode);

}
3.3.2 Web Service Composition Design

Web Service Composition for the end user is a fundamental component of our thesis and is documented separately from the main web application. The Web Service Composition module also follows the standard MVC design pattern but we refer to the business layer as the business service layer in this component.

- 3.3.2.1 Data Layer
- 3.3.2.2 Presentation Layer
- 3.3.2.3 Business Service Layer
### 3.3.2.1 Data Layer

The following is an ERD illustrating the relationship between Stories and Users. Stories that do not belong to a User indicate that they can be *shared* between Users.

![ERD Diagram](image)

#### 3.3.2.1A Story Entity Relationship Diagram

The schedulable field stores whether or not the story has any tangible, persistent outcome when scheduled to run. So a Story which gets a user's current outstanding bills and displays them on screen would not be schedulable but a Story which gets a user’s current outstanding bills and pays them would be schedulable.

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 shown below.

**Story.xsd**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.bop.com"
 xmlns="http://www.bop.com"
 elementFormDefault="qualified">

  <xsd:element name="Story">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Stone" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Stone">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Input" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="StoneIdType" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:simpleType name="StoneIdType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="getBillsStone"/>
      <xsd:enumeration value="payBPAYStone"/>
      <xsd:enumeration value="payCCStone"/>
      <xsd:enumeration value="payDDStone"/>
      <xsd:enumeration value="emailStone"/>
      <xsd:enumeration value="transferStone"/>
      <xsd:enumeration value="getBalanceStone"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```
<xsd:enumeration value="displayOutputStone"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:element name="Input">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="Value" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
### 3.3.2.2 Presentation Layer

This section outlines the presentation layer for the Web Service Composition module of BOP.

Giving the user the power to compose web service composition in a simple and user friendly interface, proved to be a difficult task. 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. To ensure a valid web service composition is constructed by the user, only the Stones that can be placed on the next step in the Storyboard are highlighted and enabled (draggable). The other Stones are grayed out and disabled as illustrated in the diagram below.

![Diagram 3.3.2.2A Create New Story Screen](image)

For usability purposes, a toolbar at the top right corner was added for the following operations:

- saving the Story
- loading a shared Story
- refreshing the storyboard
- Story help

Implementing the 'Create New Story' screen proved to be the most challenging UI screen. For more details please see 3.4.2.1 User Interface.
3.3.2.3 Business Service Layer

This section outlines the core components of the Web Service Composition module's Service Layer.

Stones

A Stone is a representation of a commonly performed task or operation that can be used to construct a Story. The `run()` method on the Stone may simply involve delegating to other business classes (managers) to process it's data. So far there are 8 implementations of the Stone interface, some of which involve invoking web services.

![Diagram 3.3.2.3A Stones Class Diagram](image)

Each Stone is responsible for generating its own XML via the `generateElement()` method.

Each Stone has a set of defined input types it can take in to process for the `run()` method. The output type for each Stone is the output type returned by the `run()` method. In order for a sequence of Stones 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. Input and output types for each Stone can be found in Appendix 7.6. For more information on how Stories are run, please read the section below.

![Diagram 3.3.2.3B Sample Story Sequence](image)

StoryManager

The Story Manager is responsible for the processing and management of Stories within BOP. It depends on the `StoryDao` to store/retrieve/update Stories to the database, as well as the `StoryParser` to parse the Story content from XML to Object form and vice-versa.

`runStory()`

When a Story is run, it's content is first parsed into a list of Stone objects. These Stones are wired the appropriate resources (such as managers or utilities classes) and invoked in order. Assuming that the Story is a valid sequence, the output of each Stone's `run()` method is passed in as the input to the following Stone. Due to this assumption of a valid sequence of Stones, it is important that Stories are validated upon creation.

`getTemplates()`

Templates is an internal term for "shared stories". Hence `getTemplates()` simply retrieves all the Stories that can be shared between users.
Diagram 3.3.2.3C StoryManager Class Diagram
3.4 Implementation

This section documents how BOP was implemented, what technologies were considered and used, as well as the engineering challenges faced during development.

- 3.4.1 Web Application Implementation
  - 3.4.1.1 Application Frameworks
  - 3.4.1.2 Web Services Registry
  - 3.4.1.3 Authentication
  - 3.4.1.4 Third Party Web Services
- 3.4.2 Web Service Composition Implementation
3.4.1 Web Application Implementation

- 3.4.1.1 Application Frameworks
- 3.4.1.2 Web Services Registry
- 3.4.1.3 Authentication
- 3.4.1.4 Third Party Web Services
3.4.1.1 Application Frameworks

Various application frameworks were used to implement BOP, which were chosen based upon the existing development skills we had with the frameworks.

**AppFuse**

AppFuse[23] is an open source project that enables the fast and efficient start up of a web-based application. The project helped us largely in the initial development of the BOP application by providing a quick way to set up a web application skeleton with basic features such as login, logout and user signup. It also provided the flexibility to choose our preferred build tool, Ant, web framework, persistence framework as well as database, Postgres.

Initial project setup was one day for all our preferred frameworks and database integrated and running. Appfuse saved us much effort and time from developing a web application from scratch and installing the various frameworks individually and provided a good build script and rich unit testing environment which we could easily adapt our system into. Internationalization, site themes and basic user authentication such as ‘remember me’ features were added bonuses that came with the application.

**Hibernate**

Hibernate[24] is an object/relational persistence and query service. Its purpose is to provide an easy way to map between object-oriented classes and a traditional relational database. It hides the details of data persistence related tasks from the developer and abstracts database query and retrieval tasks to a higher level. In having some basic development experience with the Hibernate persistent framework, we chose it over other frameworks such as JPA and iBATIS.

Most of the mappings for entity bean properties were quite straight forward and easy to do.

**BillingAccount.java**

```java
/**
 * @hibernate.property column= "account_name" length= "200" not-null= "true"
 */
public String getAccountName()
{
    return accountName;
}
```

Mapping entity bean relationships proved to be a bit more difficult but was achieved upon further examination into the lower level documentation.

**User.java**

```java
/**
 * @hibernate.set cascade= "save-update" lazy= "true" inverse= "true" where= "account_status != 'disabled'"
 * @hibernate.key column= "user_id"
 * @hibernate.one-to-many class= "com.bop.model.BillingAccount"
 */
public Set getBillingAccounts()
{
    return billingAccounts;
}
```

**Spring**

Spring[25] is an open source application framework for the Java platform. It provides a framework for implementing industry best-practice solutions to common problems and has become an alternative and replacement for the Enterprise JavaBean (EJB) model. Spring is a large framework with lots of functionality, for the purposes of BOP we mainly utilised Spring’s Inversion of Control container to manage Java objects and handle dependency injection. Dependency injection occurs whereby the
container passes objects by name to other objects via JavaBean setter methods. Objects and their dependencies are defined in XML files as follows:

```xml
<bean id="addAccountAction" class="com.bop.webapp.action.accounts.AddAccountAction"
    scope="prototype">
    <property name="userManager" ref="userManager"/>
    <property name="serviceProviderManager" ref="serviceProviderManager"/>
    <property name="accountManager" ref="accountManager"/>
    <property name="accountTypes">
        <list>
            <value>Billing Account</value>
            <value>Financial Service Account</value>
        </list>
    </property>
</bean>

<bean id="accountManager" class="com.bop.service.impl.AccountManagerImpl">
    <property name="accountDao" ref="accountDao"/>
</bean>

The above tells the Spring container to inject references to the UserManager, ServiceProviderManager, AccountManager, and a static list AccountTypes when it creates an instance of the AddAccountAction. The AccountManager similarly has a dependency on the AccountDAO which would in turn have its own dependencies.

Whilst this dependency injection was extremely useful in abstracting functionality a number of problems were encountered when Java objects which were not container managed had dependencies on beans which were container managed. One such case was the implementation of Stones in the web service composition module of the application. Since the stones are created via XML they are not container managed, however they require a number of manager classes to run eg. to get bills for the current user requires the UserManager, ServiceProviderManager, and BillManager. After trying a number of different approaches unsuccessfully, the problem was finally resolved by accessing the Spring bean factory through the servlet context and calling `autowireBeanProperties()` with AutowireCapableBeanFactory.AUTOWIRE_BY_NAME on the stones.

**StoryManagerImpl.java**

```java
public Object runStory(Story story) throws StoryParseException, RunStoryException {
    WebApplicationContext context = WebApplicationContextUtils.getWebApplicationContext(servletContext);
    AutowireCapableBeanFactory factory = context.getAutowireCapableBeanFactory();

    Object obj = null;
    List<Stone> stones = storyParser.parse(story.getContent());
    for (Iterator iterator = stones.iterator(); iterator.hasNext();)
    {
        Stone stone = (Stone) iterator.next();
        factory.autowireBeanProperties(stone, AutowireCapableBeanFactory.AUTOWIRE_BY_NAME, true);
        log.debug("about to run stone " + stone);
        obj = stone.run(obj);
    }

    if(stones.size() > 0 && stones.get(stones.size()-1).getOutputType() == Stone.StoneIOType.DISPLAY)
    {
        return obj;
    }

    return null;
}
```
Quartz

Another case of non-container managed objects requiring Spring dependency injection capabilities was the incorporation of the open source job scheduling system Quartz[26] into BOP. Quartz is capable of scheduling jobs with highly configurable triggers and has manages job persistence, transactions and clustering. Any Java that implements the simple Job interface can be scheduled with Quartz and BOP uses it to schedule Stories that the user wants to run at specific intervals. Quartz itself manages the job object instances and so is incompatible with Spring, the solution used to autowire Stones cannot be applied to jobs because Quartz creates the jobs and does not have access to the servlet context. This problem was partially resolved by passing the dependencies to the job via Quartz's JobDataMap, however this is not an ideal solution as it renders the jobs non-persistent because the dependent objects are not serializable. As yet, an ideal solution has not been found although a workaround to the problem would be to manage job persistence manually within BOP and bootstrap the saved jobs to be scheduled upon application startup.

ScheduleStoryAction.java

```java
Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
JobDetail jobDetail = scheduler.getJobDetail(storyId, Constants.STORY_JOB_TRIGGER_GROUP);
if  (jobDetail == null)  // New job
{
    jobDetail = new JobDetail(storyId, Constants.STORY_JOB_TRIGGER_GROUP, RunStoryJob.class);
    jobDetail.getJobDataMap().put("storyManager", storyManager);
    scheduler.scheduleJob(jobDetail, trigger);
} else
{
    trigger.setJobName(storyId);
    trigger.setJobGroup(Constants.STORY_JOB_TRIGGER_GROUP);
    scheduler.rescheduleJob(storyId, Constants.STORY_JOB_TRIGGER_GROUP, trigger);
}
```

WebWork

WebWork[27] is an open source Java web-application framework for building reusable UI templates for common web-application development tasks such as form controls, internationalisation, dynamic form parameter mapping to JavaBeans, and robust client and server side validation. For BOP WebWork provided action invocation control allowing requests, results and any chaining of actions to be easily and structurally defined.

xwork.xml

```xml
<action name= "addStory" class= "addStoryAction" method= "doInput" >
    <result name= "input" >/WEB-INF/pages/storyboard/addStory.jsp</result>
</action>

<action name= "doAddStory" class= "addStoryAction" >
    <result name= "success" type= "redirect" >/storyboard/listStories.html</result>
    <result name= "error" >/WEB-INF/pages/storyboard/addStory.jsp</result>
    <result name= "cancel" type= "redirect" >/storyboard/listStories.html</result>
</action>
```

The above action mapping tells WebWork that when a user tries to access a URL mapped to the addStory action (ie. when the user goes to [http://someserver/bop/storyboard/addStory.jsp](http://someserver/bop/storyboard/addStory.jsp)) the addStoryAction class should be invoked with a call to the method doInput() which sets up all the drop-down lists and other fields on that page for the user to access. Next when the user tries to add the story the form submit button will invoke the doAddStory action which again maps onto the addStoryAction but this time the default execute() method will be called. When the action returns a success result code the user will be redirected to the listStories action. If the action returns an error result code the user is directed back to the addStory.jsp page which will display any validation errors to the user.

```xml
<action name= "editServiceProvider" class= "editServiceProviderAction" method= "doInput" >
    <interceptor-ref name= "adminCheck" />
    <result name= "input" >/WEB-INF/pages/editServiceProvider.jsp</result>
</action>
```
In addition to the action mappings described above WebWork also invokes any defined interceptors before invoking the action itself as shown in the editServiceProvider action above. The adminCheck interceptor checks that the user has administrative privileges before allowing the action to take place. This type of aspect oriented design allows for the separation of concerns, the action itself does not have to contain any user privileges logic, it is protected by the interceptor designed for that purpose.

**payBill.jsp**

```html
<div id="bpayFields" style="display:none">
<ww:textfield label="%{getText('bill.biller.code')}
name="billerCode" id="billerCode"
value="%{billerCode}" cssClass="text large" required="true" readonly="true" />
<ww:select name="bpayAccount" label="%{getText('billing.from.account')}
list="fsAccounts" listKey="id" listValue="accountDescriptionDisplay" value="bpayAccount"
required="true" headerKey="-1" headerValue="%(getText('please.select'))"/>
</div>
```

The above creates a WebWork text field automatically populated with the value returned by a call to `getBillerCode()` on the action. The text field has a label corresponding to the `bill.biller.code` property defined in ApplicationResources.properties as below. This allows for internationalisation with different properties files for different languages. The second WebWork tag creates a select field populated with the list of Financial Service Accounts returned by `getFsAccounts()`. The value displayed to the user is the text returned by `getAccountDescriptionDisplay` and when the user selects a value from the drop-down list `setBpayAccount()` will be called on the action with the value corresponding to `getId()` on the selected Financial Service Account. In this way, WebWork simplifies much of the user interface design and interaction.

**ApplicationResources.properties**

```properties
# -- bills --
bill.id=Bill ID
bill.biller.code=Biller Code
bill.date.due=Date Due
bill.date.paid=Date Paid
bill.amount.due=Amount Due
bill.amount=Amount
bill.min.amount=Min Payment
```

What the user sees is shown below.
Other than the use of WebWork tags described above, the user interface for the BOP web application is based mainly on Velocity macros and Java Server Pages (JSPs). JSPs allow dynamic content to be generated for web pages by allowing access to server-based resources such as JavaBeans. Velocity is a Java-based template engine that provides a powerful template language to reference Java objects and is an alternative to JSPs, although we have used the two in conjunction. Velocity macros are used for email templates and stone/story outputs.

**Velocity & JSP**

Other than the use of WebWork tags described above, the user interface for the BOP web application is based mainly on Velocity macros and Java Server Pages (JSPs). JSPs allow dynamic content to be generated for web pages by allowing access to server-based resources such as JavaBeans. Velocity is a Java-based template engine that provides a powerful template language to reference Java objects and is an alternative to JSPs, although we have used the two in conjunction. Velocity macros are used for email templates and stone/story outputs.

**getBillsStoneEmail.vm**

```velocity
BOP Bill Summary

#if($items)
#if($items.size() > 0)
The following bills are outstanding.

#foreach($bill in $items)
Account: $bill.serviceProvider.providerName Bill: $bill.billId Due Date: $bill.dueDate Amount Due: $bill.amountDue
#end
#else
You have not bills outstanding.
```

**Pay Bill**

**Bill Details**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biller:</td>
<td>Apple</td>
</tr>
<tr>
<td>Bill ID:</td>
<td>B-112</td>
</tr>
<tr>
<td>Min Payment:</td>
<td>$21.93</td>
</tr>
<tr>
<td>Amount Due:</td>
<td>$21.93</td>
</tr>
</tbody>
</table>

**Payment Details**

- **Payment Method**: BPAV
- **Biller Code**: 111111
- **From Account**: Please select
- **Amount**: 21.93
- **Description**: 


---

**Velocity & JSP**

Other than the use of WebWork tags described above, the user interface for the BOP web application is based mainly on Velocity macros and Java Server Pages (JSPs). JSPs allow dynamic content to be generated for web pages by allowing access to server-based resources such as JavaBeans. Velocity is a Java-based template engine that provides a powerful template language to reference Java objects and is an alternative to JSPs, although we have used the two in conjunction. Velocity macros are used for email templates and stone/story outputs.

**getBillsStoneEmail.vm**

```velocity
BOP Bill Summary

#if($items)
#if($items.size() > 0)
The following bills are outstanding.

#foreach($bill in $items)
Account: $bill.serviceProvider.providerName Bill: $bill.billId Due Date: $bill.dueDate Amount Due: $bill.amountDue
#end
#else
You have not bills outstanding.
```
Thank you for using BOP for all your bill paying needs.
3.4.1.2 Web Services Registry

Web services are built on a foundation of three standards: Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). For the implementation of BOP the specifications for SOAP and WSDL were integral to the success of the project. Both bill providers and financial service providers had to expose their internal business functions as BOP-standard web services with corresponding WSDL descriptors and SOAP over HTTP binding was used for web service invocation throughout the portal.

The prospect of integrating the third and final specification, UDDI, into BOP was investigated with the open source projects JUDDI[29] and UDDI4J[30]. JUDDI is an open source Java implementation of the UDDI specification that unmarshalls incoming UDDI requests, invokes the appropriate UDDI function and marshalls correct UDDI responses. We successfully deployed JUDDI on Apache Tomcat and examined how it could be integrated into BOP. As JUDDI provided us with our own web services registry we needed BOP to be able to generate the correct requests to the registry. UDDI4J is a Java class library that provides an API to interact with an existing UDDI registry. Integration of UDDI4J into BOP involved a simple inclusion of the library "uddi4j.jar" and setting up the configuration parameters (host and port number) to point to our JUDDI instance.

It was decided at this point not to proceed with the integration of UDDI into the implementation of BOP due to the time resources required to master the complete UDDI specification in order to use the UDDI4J API. However it can be concluded that integration of a UDDI registry with BOP is quite feasible which would eliminate the need for BOP to manage any Service Provider data internally. BOP could query the registry for providers which match the given BOP billing service/financial service interface via a tModel query as per the UDDI specification and the service providers themselves would be responsible for adding/updating/retiring their web services.
3.4.1.3 Authentication

Authentication and authorization are critical components of any substantial software system. For most systems, however, the scope of the problem lies within the organisation itself. Authorized parties are known to the organisation and only an authentication policy and access control policy needs to be put in place to allow authorization to occur. Authorization for BOP is a more difficult endeavour because the authorization must take place across organisational domains. The same consumer must be authorized to view their bills from multiple bill providers and transfer funds into and out of multiple financial service accounts. All of these businesses must also authenticate the end user against the corresponding user identifier in their own systems at the same time, the point where the end user signs on to BOP.

A number of existing single sign-on solutions and standards are available for authorization situations similar to that faced by BOP including Java Open Single Sign-On (JOSSO)[19], Open Single Sign On (OpenSSO)[20] and the Security Assertion Markup Language (SAML)[21] standard. JOSSO is a pluggable centralised user authentication solution which provides web services for asserting user identity using SOAP over HTTP. OpenSSO is based on the Sun Java System Access Manager and relies on cookies to store session tokens to identify users against internal user profiles. SAML 2.0 is an OASIS standard which can be used to establish permanent privacy-preserving federation between trusted parties.

SAML Investigation

For BOP, it was originally determined that the ability to establish persistent identifiers between external businesses and BOP as defined in SAML 2.0 was an appropriate implementation solution. SAML requires that BOP and the external businesses have a pre-existing trust relationship based on a Public Key Infrastructure (PKI). After which it allows for a permanent pseudo-identity to be created linking the BOP end user and the business user identifier. It does not require the external business to change its existing authentication method and allows for BOP to establish end user identity permanently with all the user’s billing and financial service accounts.

Thus, a number of authentication solutions for BOP incorporation the SAML standard was attempted. This included a full implementation of the SAML standard, the use of an implementation of the SAML standard in another project, and the use of a single sign on solution which incorporated the SAML standard. The endeavors are documented in the sections below.

Java Architecture for XML Binding (JAXB)

The first attempt involved an implementation of the SAML standard from scratch. The OASIS standards committee provided all the XML schemas necessary for the implementation of SAML. Java Architecture for XML Binding (JAXB) is a Sun Java API for processing XML content using Java objects by binding its XML schema to Java representation. The JAXB Reference Implementation Project[31], part of Project Metro from the Glassfish community, is an implementation of JAXB that we used to generate Java classes from the SAML schemas. However, it was quickly determined that the SAML standard was too large for us to implement. Even an implementation for the specific authentication scenario required in BOP would be complex enough to warrant a separate thesis project.

Shibboleth and Enterprise Sign On Engine (ESOE)

Thus, the second attempt involved searching for a library implementation of SAML that could be easily incorporated into the BOP project; much like JUDDI and UDDI4J could have been done (see 3.4.1.2 Web Services Registry). Shibboleth[32] is a standards-based, open source middleware software which provides single sign on across or within organizational boundaries. The Shibboleth software implements the SAML standard and appeared to be well documented and simple to integrate with other projects such as BOP. Unfortunately, only the SAML 1.1 implementation was supported, while the authentication scenario required by BOP is described by the SAML 2.0 standard.

The Enterprise Sign On Engine (ESOE)[33] is a Queensland University of Technology open source implementation of the SAML 2.0 standard. Having obtained the most up-to-date version of the source, it was clear that the project was still a work in progress. We contacted the developers behind the project to determine whether their implementation was appropriate for our purposes and found that they had been using the implementation only for the exchange of XML documents at that time, although plans for the expansion of the project for more extensive authentication processes did exist. Since the project source was not stable at the time, ESOE was considered risky and insufficient to incorporate into BOP.
OpenSSO

The final attempt at using the SAML standard for BOP's authentication involved using the OpenSSO project for authentication. The OpenSSO project is based on the code base of Sun Java System Access Manager and supports the authentication standards Liberty Alliance Project Identity Federation Framework (Liberty ID-FF), Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF), SAML 1.0, SAML 1.1, and SAML 2.0. Two versions of Access Manager were deployed and interaction between them was set up for authentication between an identity provider IDP (in BOP terms this would be a financial/billing service provider) and a service provider SP (in BOP terms this would be BOP itself). However this interaction was completely independent of BOP, to enable BOP to use Access Manager as its authentication system would require the implementation of a BOP module able to communicate with Access Manager. Again it was determined that this would consume too many resource from the BOP project so the use of SAML for end user authentication was abandoned. Using Access Manager for authentication into BOP would also require that the service providers all use Access Manager for user authentication which is not a viable assumption.

BOP Authentication

In an effort to still keep end user authentication with third party service providers abstracted to a level where different service providers can have different authentication methods. It was determined that each service provider would provide their own authentication form much like a portlet into BOP. The authenticate method of the BopFinancialService and BopBillingService (see 3.3.1.3.2 Web Services) returns a XHTML form from the service provider which allows each service provider to specify their own authentication requirements. The implemented service providers Banjo and Apple illustrate two different authentication methods which are incorporated seamlessly into the Add New Account functionality of BOP.

Authentication method from Banjo service provider

![Banjo Authentication Method](image)

Authentication method from Apple service provider

![Apple Authentication Method](image)
The submit buttons on the above forms generate requests directly to each service provider without interference from BOP. Once the service provider has authenticated the user against their own internal user identifier they generate a unique federated identifier for the user and inform BOP that the user has been authenticated with them and the federated identifier which was generated to identify that user. In a deployed environment BOP and the service provider would already have an established trust relationship via public key encryption so both BOP and the service provider can trust requests and responses for authentication to each other (in our implementation, this is represented via the token argument passed in all interactions between the parties). If there was a problem validating the end user's credentials against the service provider's internal database, BOP is likewise informed and the end user is prompted for the valid credentials. Once a federated identifier is established, the user will no longer be prompted for their credentials when accessing information from the service provider. In the back-end, BOP will pass on their federated identifier to the service provider, thereby establishing permanent federation and allowing single sign on to occur from within BOP. The diagram below summarises the interactions between the two parties for end user authentication to occur.

Authentication between BOP and third party service providers

1. End user tries to add a new billing/financial service account in BOP
2. BOP makes web service call authenticate passing the user's account name and number
3. Service provider sends back XHTML authentication form required to validate this particular user (this also allows the same service provider to provide different authentication challenges to different users)
4. BOP embeds the form and displays it to the user
5. User fills in their credentials and submits the form
6. BOP passes the credentials to the service provider
7. Service provider validates the user, generates and stores a federated ID, then redirects back to BOP
8. BOP stores the federated ID returned and allows the user access to service provider's functionality
3.4.1.4 Third Party Web Services

In a production environment it is expected that each of the service providers (billers and financial service providers) will provide their own implementation of the BOP interface (defined in Section 3.3.1.3.2 Web Services). However, for demonstration purposes, we have implemented two of each service provider. Using Axis, we are able to develop simple implementations of the BOP interfaces.

**Billing Service Providers**

The two Billing service providers we implemented were named after fruits:

- Apple
- Banana

Our implementations involved connecting to an underlying database that held bill and billing account information. Most billers would already have this sort of information in their existing systems and would simply need to provide the web services to extract this information.

One problem we encountered was the way we had to handle credit card payments to bills. Ideally in a production system, BOP would communicate these credit card payments to a Payment Gateway Processor, however as a thesis implementation we did not feel it was crucial to replicate this behaviour. Instead we decided to add a web service for credit card processing on the biller’s interface. This extra web service was only added for demonstration purposes.

**Financial Service Providers**

The two Financial service providers we implemented were named after CSE computer labs:

- Banjo
- Pipe

Similarly to the billers, we setup database tables to hold sample account information and balances. Account balances were updated upon transfers and bill payments, however we did not implement a mock clearing house which would facilitate sending the funds to the billers and thereby enabling bills to be properly cleared and accounts to be properly updated. It was determined that it was non-essential to implement and mimic a financial service provider’s entire business functionality to demonstrate the core of our thesis.
3.4.2 Web Service Composition Implementation

Many challenges were faced during the UI implementation of the Web Service Composition module. This section outlines the technologies used, the problems encountered and how they were resolved.

**User Interface Design**

Even with a drag/drop design, we discovered that users had difficulty in creating a web service composition. With our initial design, users found it difficult to determine which Stones could be dragged next onto the storyboard. By simply changing the styling of Stones, we found that it made a large improvement. Grayed out stones blended well into the background making it evident that they were 'disabled' and the brighter blue of the Stones made it clear that they could be used.
Additionally, users seemed to be unaware that input parameters had to be specified on particular Stones before the Story could be saved. Users claimed that the set options link to specify the parameters was not obvious enough. We tackled this problem by ensuring that the set options window was immediately opened upon a drop of a Stone on the storyboard.

**Javascript**

Javascript was a requirement to make our drag/drop UI design possible. We investigated several drag/drop utility libraries and decided to use the YUI (Yahoo User Interface) library. It was the most well documented library (at the time of implementation) and supported the drag/drop framework required for our design. It also came with other utility modules such as the Event module, which we found handy and easy to use.

In addition to YUI, we used Matt Kruse's Javascript libraries for the popup windows and input validation.

Although existing libraries were used for general functionality, custom Javascript still had to be written for the validation of the user's web service composition. In particular, Stone highlighting, enabling/disabling had be re-calculated each time a new Stone was dropped on the Storyboard. This proved to be a difficult task as the more validation checks were added, the more it impacted the drag/drop performance and the more it lagged. Logic for the validation checks had to be revisited and optimized to ensure appropriate user response times to the drag/drop operations.

**DWR**

"Direct Web Remoting (DWR) allows Javascript in a browser to interact with Java on a server and helps you manipulate web pages with the results." [34]

DWR was used for two main reasons:

1. to dynamically load the Input/Output types and names from the Stone object classes
2. to load and save shared stories without reloading the browser

As the Stone interface was designed to expose all the Stone information (such as I/O types and names), it seemed very redundant to hard code these values in the Javascript where Stone information was needed. Instead, we used DWR to dynamically generate Javascript based on the Stone Java classes. The DWR configuration specifies which Java classes and which methods of these classes can be accessed via Javascript:

```html
<create creator= "new" javascript= "getBalanceStone">
<param name= "class" value= "com.bop.storyboard.GetBalanceStone" />
<include method= "getId" />
<include method= "getName" />
<include method= "getOutputType" />
<include method= "getInputTypes" />
</create>
```

This created a Javascript object `getBalanceStone` which could be referenced by including the generated Javascript file. Upon loading of these dynamically create Javascript objects, the required Stone information was extracted and stored into global variables. These global variables could be accessed from our custom Javascript functions to display Stone names and check the I/O types of the Stones.

During creation of a new story, a user is given the ability to load a shared a story or optionally to share a story of their own. While this functionality is present, it is not the main focus of the Story creation screen. AJAX support by DWR enabled the remote calling to the server to load and save shared stories, without having the browser to reload. The following function is invoked when the 'load a shared story' icon is clicked. It populates a popup window's contents with all the shared stories by making an AJAX call.

```javascript
/**
 * Loads templates from the database via AJAX and displays the output in the given div.
 */
function loadTemplates(divName)
{
    try
    {
```

---

 Thesis B Report  Page 48
var xmlHttp = new XMLHttpRequest();
xmlHttp.onreadystatechange = function()
{
    if(xmlHttp.readyState==4)
    {
        document.getElementById(divName).innerHTML = xmlHttp.responseText;
    }
}
xmlHttp.open("POST", "/bop/storyboard/templates/loadTemplates.html", true);
xmlHttp.send(null);
}
catch(e)
{
    alert("Error has occurred while trying to load templates: " + e);
    return false;
}
3.5 Testing & Evaluation

Testing

Three different types of testing have been performed during the development of the BOP system.

Unit testing
This is a suite of JUnit tests which have been written and run periodically during development. They typically test an individual function or procedure of the classes that were implemented.

Functional testing
This is a suite of JWeb Unit acceptance tests that have been written and run periodically during development. They typically mimic/test user scenarios and will also ensure that any incremental changes do not affect previous implementations.

User Acceptance testing
This involved user evaluations of the final implementation of BOP. As bugs/improvements were found during the evaluation, they were fixed and implemented before the next user’s evaluation. The criteria that has been used for these user surveys are listed in Appendix 7.3 and the results have been documented below.

User Evaluation

Survey Results

In total, 10 users were asked to perform the evaluation survey. The results have been summarised in the charts below.

From the graph, it is evident that User Control & Freedom was rated the lowest, indicating the system’s lack of support for undoing user mistakes. Additionally, many users commented that more user help was required.
Most users seemed to have difficulty in creating a Story and adding a new billing account. The screen 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. Users stated that once the link to add a new billing account was found, it was very easy the next time to add more accounts. Hence the higher rating for adding a financial service account.

**Suggested Improvements**

Some of the improvements to the system as a result of direct user suggestions include:

1. A number of users indicated that the link to add a new billing account was difficult to locate. The link is located under the Manage Accounts tabs, named Add New Account, while most users first looked under the Billing tab. Hence a user suggestion to add a second link under Billing was adopted. The link Add Billing Account was added and the equivalent link for adding financial service accounts was also added.

2. One user appropriately pointed out that some stories which only display output to the user should not be scheduled because the function they perform is meaningless without a user.
Hence a scheduleable attribute was added to the Story class and BOP was modified so that the functionality to schedule stories is only available for those stories that have a tangible outcome.
3. One user suggested that the pay bill screen should have the outstanding amount of the bill as the default payment amount which was adopted in BOP.
4. The add story screen was made more intuitive by changing the colour scheme and making stone options popup upon dropping each stone as described above.

**Developer Evaluation**

This section outlines the developer evaluations of BOP, including the weaknesses of the system.

**Web Service Composition & Business integration**

The current implementation of BOP is a full working system, that demonstrates the uses of user web service composition and business integration through a portal. Although we were unable to implement all the requirements specified in Section 3.2, we were able to complete the requirements with the higher priorities. We are satisfied with this level of achievement as it meets the principal aims of our thesis, but still believe there is always room for improvements (see Section 5.1 Future Work for details).

**Frameworks & Technologies**

BOP has been implemented using the latest frameworks and technologies used commonly in production systems. This demonstrates that building such a system for a production environment is quite feasible. As developers we were also able to learn about other existing frameworks/technologies and broaden our engineering experience and skills.

**User Interface**

As developers, one of our weakest development skills lay in designing and implementing user interfaces. The implementation of BOP (especially the add story screen) has encouraged us to develop and improve such skills. We initially started with very basic Javascript knowledge and almost no knowledge of DWR/AJAX. We have now reached the point where we can comfortably use complex Javascript libraries and make simple remote calls to the server via DWR.

We are very pleased with the current UI and believe it provides a simple and consistent, user-friendly interface. In particular the add story screen was revised many times and has gradually received higher ratings (in user evaluations).

**Security**

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 providers is highly critical and a target for abuse. As the system is a 'one-stop', convenient place where a user can manage all their finances, unauthorized users could potentially get access to a user’s entire savings.

Another main security hole for potential hackers is when information is passed between the different business domains i.e. BOP and the service providers. Without security, hackers could possibly eavesdrop on the conversations and even alter the information passed between BOP and the service providers.

Due to the fact that our thesis implementation is not built for a production environment, we have only applied minimal security measures to our system. User passwords in BOP have been encrypted, protecting the most obvious security breach, where a database hacker could obtain user logins.

**Transaction Management**

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 its original state.
Although this issue has been identified, it is not something that our implementation currently supports. With the large scope of such a feature and time constraints, we concluded that it wasn't a high priority for our chosen area of study.
4 Project Management

This section outlines how we planned and managed the progress of our thesis implementation.

Planning

During thesis A, we created a Gantt Chart, planning to complete most of the BOP implementation by Week 9. As with most projects, we weren't able to work precisely to the schedule and ended up completing most of the BOP functionality by Week 11.

In addition to the gantt chart, we also scheduled weekly tasks. At the beginning of each week, we calculated how much time we could spend on thesis that week and came up with a prioritised list of tasks. This method proved to be very effective and was inspired by the XP practices commonly used during software development.

Monitoring

All of our thesis documentation, planning, references, tracking and active todo lists are located on a wiki hosted at http://stout.cse.unsw.edu.au:8080/. Tracking our progress in general was done by using a wiki page (see Appendix 7.9). At the end of each day we would post up our completed tasks and hours taken to complete these tasks. This helped us identify which tasks took longer/shorter than expected and make better estimations on future task allocations. We have spent over 200 hours on thesis B to date.

Version Control

Subversion was used as our version control system for BOP. At the end of each major feature implementation, BOP was tagged. This was to ensure that we could rollback to a working copy easily, if ever required. Fortunately, we were never in the situation where a rollback was required.

Implementation

Continuing with the XP programming practices, we adopted pair programming throughout the beginning stages of BOP implementation. We continued pair programming throughout any implementation that involved new frameworks, techniques, or major modules of BOP. Once the major architecture of BOP was established we programed separately from approximately Week 5 onwards and paired only when the benefits of pair programming was deemed necessary.
5 Conclusion

The assimilation of the Internet into people's everyday lives has surpassed the expectations of all society. Email and online content have changed the way people communicate, receive news, and find information and entertainment. Businesses are increasingly finding the Internet an invaluable tool in reaching suppliers, business partners, and customers. However the average user has very little involvement in the creation of content on the Web and very little say about how content is provided and presented to them. The advent of Web 2.0 and mashups has brought about the notion that content from various different sources can be brought together by the user themselves to create a new service for their own purposes that was not envisaged by the sources of that content. If the end user were able to truly take control of how they interact with not only static content but services provided by businesses online, it would be possible for them to create their own personalised portal where businesses come to them in a new C2B e-commerce architecture.

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 thesis 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. It is also a study of the challenges facing the idea of a C2B end user portal were all services provided by businesses to an individual can be amalgamated into a single point of contact for that one individual. Major challenges to both these ideals are the interrelated themes of privacy, security and identity management, as well as the ability to abstract different types of services into a common, high level and manageable abstraction.

The implementation of BOP as a platform to illustrate the challenges of this abstraction and the authentication issues of the portal was chosen because the bill management domain is one which involves many different businesses; many of which are financial institutions which require a high level of security and different modes of authentication. The domain is also one which is common and familiar to the everyday end user and poses many areas of inefficiency for the end user which could be addressed by the advances in technology which are currently under development.

As an engineering challenge this thesis provided an opportunity for its developers to use the latest software engineering tools, frameworks, and methodologies to develop a web portal which tries to address some of these issues. The development of the Story/Stone web service composition module of this project provides a powerful abstraction of web services from different sources which allows the end user to compose their own composite services without knowing about the low level details of such an endeavour. This thesis proves, with the use of a simple drop and drop user interface, that it is definitely feasible for end users to be involved in web service composition and in the creation of tailored online business services for individual consumers.
5.1 Future Work

There are a number of areas where further investigation and system implementation is required. As mentioned in Section 3.5 Testing & Evaluation, the issues of security and transaction management were not dealt with in the implementation of BOP. These are two areas crucial for any plans of deployment. There are many security measures such as encryption, error detection, digital signatures/certificates which would need to be employed in an online system such as BOP. Many standards for web services security such as WS-Security exist also and would need to be adopted. In addition, the authentication procedure for BOP would need to be amended to follow an industry accepted standard, whether it be SAML or the adoption of a separate single sign on system.

Ideally, BOP should support all the ACID transaction properties due to the nature of data handled in the system. The Spring framework supports container managed transactions, which could be used to delegate transaction management for processes within BOP. However, atomic transactions for processes spanning over different business domains i.e. between BOP and the service providers, would require transaction manager agents to appropriately rollback processes over multiple application domains.

A further improvement to BOP would involve the integration of a web services registry for managing individual service providers. As mentioned in Section 3.4.1.2 Web Services Registry, the integration of JUDDI and UDDI4J into BOP would not be a complex undertaking. The ability of BOP to dynamically query a registry for providers which adhere to the BOP defined interfaces for financial service providers and billers would allow the separation of concerns between BOP and the service providers. Access points to services providers as well as versioning of BOP interfaces could be easily modified and maintained. The benefits of dynamic service selection based on quality of service heuristics could also be investigated.

The final and most interesting area of improvement to the system would be to extract the web service composition module of BOP into an extensible, re-usable library framework which could be plugged into any existing system with modularised services and/or web services. Such work would involve setting up a method of configuration such as an XML definition of services and their Stone implementations. With such a library the concept of end user web service composition would be completely realised.
6 References


Thesis B Report  Page 57
[19] Java Open Single Sign On, last viewed 28 October 2007,
<http://www.josso.org/>

[20] OpenSSO, last viewed 28 October 2007,
<https://op ensso.dev.java.net/>

9 October 2006, SAML 2.0 Technical Overview, OASIS

[22] IBM Koala Project, last viewed 28 October 2007,

[23] AppFuse, last viewed 29 October 2007,
<https://appfuse.dev.java.net/>

[24] Hibernate, last viewed 29 October 2007,
<http://www.hibernate.org/>

[25] Spring, last viewed 29 October 2007,
<http://www.springframework.org/>

[26] Quartz, last viewed 29 October 2007,
<http://www.opensymphony.com/quartz/>

[27] WebWork, last viewed 29 October 2007,
<http://www.opensymphony.com/webwork/>

[28] Velocity, last viewed 29 October 2007,
<http://velocity.apache.org/engine/devel/>

[29] JUDDI, last viewed 29 October 2007,
<http://ws.apache.org/juddi/>

[30] UDDI4J, last viewed 29 October 2007,
<http://uddi4j.sourceforge.net/>

[31] JAXB Reference Implementation Project, last viewed 28 October 2007,
<https://jaxb.dev.java.net/>

[32] Shibboleth, last viewed 28 October 2007,
<http://shibboleth.internet2.edu/>

[33] Enterprise Sign On Engine, last viewed 28 October 2007,
<http://esoeproject.org/>

[34] Direct Web Remoting, last viewed 28 October 2007,
<http://getahead.org/dwr/>

Addison-Wesley Professional
7 Appendix

- 7.1 Assumptions
- 7.2 Screenshots
- 7.3 Evaluation Survey
- 7.4 Glossary
- 7.5 Detailed Class Diagrams
- 7.6 Stone Definitions
- 7.7 BOP Gantt Chart
- 7.8 Existing Bill Aggregators
- 7.9 Time Tracking
## 7.1 Assumptions

The following table is a list of assumptions made for the BOP system.

<table>
<thead>
<tr>
<th>ID</th>
<th>Assumption</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-01</td>
<td>BOP will not be responsible for banning or blacklisting fraudulent end users.</td>
<td>The end user will contact financial service providers to stop transactions on their accounts in the event of fraudulent transfers.</td>
</tr>
<tr>
<td>A-02</td>
<td>BOP will not provide receipts</td>
<td>The financial service provider will provide receipts when payments are made.</td>
</tr>
<tr>
<td>A-03</td>
<td>BOP will allow viewing of historical bills only to the extent that billing account providers support.</td>
<td>It is not feasible for BOP to keep records of all end user historical bills.</td>
</tr>
<tr>
<td>A-04</td>
<td>BPAY will be provided by the FSAs as a web service.</td>
<td>BPAY is already integrated into banks.</td>
</tr>
<tr>
<td>A-05</td>
<td>Account numbers will not contain spaces.</td>
<td>Different financial service providers may have different spacing or formats for account numbers. To eliminate any inconsistencies, all account numbers will be passed around without spaces.</td>
</tr>
<tr>
<td>A-06</td>
<td>Service providers will adopt BOP interfaces.</td>
<td>To enable the portal to function, a coherent interface must exist that all service providers must adopt. For this implementation, that interface is assumed to be the BOP interface.</td>
</tr>
<tr>
<td>A-07</td>
<td>Service providers all have web facing authentication methods into their systems.</td>
<td>For users to access business functionality online, a web authentication system must exist.</td>
</tr>
</tbody>
</table>
7.2 Screenshots

Diagram 7.2A Manage Stories Screen

Diagram 7.2B Add New Story Screen
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.

Diagram 7.2C Share Story Screen
Create New Story

To create a new 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.

Diagram 7.2D Load Story Screen

Billing Organiser Portal

Add New Account

Account Type
Billing Account

Bill Provider
Apple

Account Name
John Smith

Account Number
12345678

Account Number
Apple Mobile Account

Next>> Cancel

Diagram 7.2E Add New Billing Account Screen
Diagram 7.2F Edit Profile Screen

Diagram 7.2G Search Payments Screen
7.3 Evaluation Survey

BOP Walkthrough

Please go through the following steps:

1. Create a user account
2. You receive bills from a company called Apple and have the following account details:
   
   Bill Provider: Apple  
   Account Name: Cyrus Judge  
   Account Number: 1234567890  
   Account PIN: 1234

   Please register a billing account in BOP, using the information above.
3. You have a Savings Account with a bank called Banjo and have the following account details:
   
   Financial Service Provider: Banjo  
   Account Name: Cyrus Judge  
   Account Number: 2345678901  
   Client Number: 13572468  
   Password: cyrus

   Please register a financial service account in BOP, using the information above.
4. You owe Tracey Miller $20 and would like to transfer funds from your Banjo savings account (the 
   account registered in step 3). Please transfer the funds using Tracey’s details provided below.
   
   Account Name: Tracey Miller  
   Account BSB: 112891  
   Account Number: 34567890  
   Amount: $20

5. View your outstanding bills
6. Pay a bill using the Banjo savings account you registered earlier
7. You are lazy and don’t like doing step 6 every time you want to pay a bill. You want to simply pay all 
   your outstanding bills using your Banjo savings account and email the payment receipts to yourself. 
   Create a story to do this (i.e. pay outstanding bills via BPAY and email yourself the receipts)
8. Run the story you have just created. You may now want to check your email to see if you received 
   the payment receipts.
9. Logout

Questions

Please answer the following questions, where:

- 1 = very poor
- 2 = poor
- 3 = average
- 4 = good
- 5 = excellent

If you answer less than average for any of the questions, please suggest how it could be 
   improved.

User Interface

1. Consistency - How well did the system use the same colours, fonts, terminology and positions for 
   recurring functionality?
   
   1 2 3 4 5

2. Ease of site navigation - How easily were you able to navigate through the site via menus, links, 
   logical screen flows?
   
   1 2 3 4 5

3. Access to help functionality - How well did the system provide appropriate contextual help?
   
   1 2 3 4 5
4. Intuitiveness - How easy was it to understand the functionality of buttons, links, menus by their wording?
   1  2  3  4  5
5. Visibility of system status - How well did the system inform you about what is going on, through appropriate feedback within reasonable time?
   1  2  3  4  5
6. Aesthetic and minimalist design - How relevant was the information presented for the task you were undertaking?
   1  2  3  4  5
7. User control and freedom - How easily were you able to undo your mistakes?
   1  2  3  4  5
8. Error handling - How well did the system provide meaningful error messages?
   1  2  3  4  5
9. Recognition rather than recall - How well did the system save you from having you to remember information from one part of the system to another?
   1  2  3  4  5
10. Flexibility and efficiency of use - How well did the system tailor to frequently used functions?
    1  2  3  4  5
11. Overall look and feel - What was your overall impression of the site?
    1  2  3  4  5

**Functionality**

1. How easy was it to register online from the BOP homepage?
   1  2  3  4  5
2. How easy was it to add a billing account?
   1  2  3  4  5
3. How easy was it to add a financial service account?
   1  2  3  4  5
4. How easy was it to access your outstanding bill(s)?
   1  2  3  4  5
5. How easy was it to pay a bill?
   1  2  3  4  5
6. How easy was it to create a story?
   1  2  3  4  5
7. How useful is the story functionality?
   1  2  3  4  5
8. Would you use the system if it was publicly available?
   Yes  No

**Improvements**

1. How would you improve the system?
2. Any other comments?

**Thank you**

Thank you for participating in our survey.
### 7.4 Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billing Accounts</td>
<td>Accounts that the end user holds with companies that bill the end user, for example; telephone, electricity, water and gas billing accounts.</td>
</tr>
<tr>
<td>BOP</td>
<td>Billing Organiser Portal</td>
</tr>
<tr>
<td>End User</td>
<td>A person who will use BOP</td>
</tr>
<tr>
<td>FS Accounts</td>
<td>Financial Service Accounts. Accounts that the end user holds with financial producers, for example; credit card, savings, cheque and brokerage accounts.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SSO</td>
<td>Single sign on</td>
</tr>
<tr>
<td>Stone</td>
<td>A modularised task or function (such as a web service call) which can be one step in a Story.</td>
</tr>
<tr>
<td>Story</td>
<td>A sequence of operations (could include web service calls) that the user has constructed and stored in BOP. It enables users to easily run a sequence of tasks performed repetitively.</td>
</tr>
</tbody>
</table>
7.5 Detailed Class Diagrams
### 7.6 Stone Definitions

<table>
<thead>
<tr>
<th>Stone Name</th>
<th>Input</th>
<th>Output</th>
<th>Email Template</th>
<th>Display Output Template</th>
<th>Parameters Required for run() Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBillsStone</td>
<td>None</td>
<td>Bills</td>
<td>getBillsStoneEmailTemplate</td>
<td>getBillsStoneDisplayOutputTemplate</td>
<td>PARAM_USER_ID, BILLER_SP (optional), ServiceProviderManager, ServiceProvider, BillManager</td>
</tr>
<tr>
<td>PayBPAYStone</td>
<td>Bills</td>
<td>Text</td>
<td>payBillStoneEmailTemplate</td>
<td>payBillStoneDisplayOutputTemplate</td>
<td>PARAM_FS_ACCOUNT, PAYMENT_DESCRIPTION, AccountManager, BillManager</td>
</tr>
<tr>
<td>PayCCStone</td>
<td>Bills</td>
<td>Text</td>
<td>payBillStoneEmailTemplate</td>
<td>payBillStoneDisplayOutputTemplate</td>
<td>PARAM_FS_ACCOUNT, PAYMENT_DESCRIPTION, PARAM_CC_EXPIRY_DATE, PARAM_CC_SECURITY_CODE, AccountManager, PaymentManager</td>
</tr>
<tr>
<td>PayDDStone</td>
<td>Bills</td>
<td>Text</td>
<td>payBillStoneEmailTemplate</td>
<td>payBillStoneDisplayOutputTemplate</td>
<td>PARAM_FS_ACCOUNT, PAYMENT_DESCRIPTION, AccountManager, PaymentManager</td>
</tr>
<tr>
<td>EmailStone</td>
<td>Text, Bills, Amount</td>
<td>None</td>
<td></td>
<td></td>
<td>PARAM_EMAIL_TO, PARAM_EMAIL_CC (optional), PARAM_EMAIL_BCC (optional), PARAM_EMAIL_SUBJECT, PARAM_EMAIL_TEMPLATE (optional), MailUtils</td>
</tr>
<tr>
<td>TransferStone</td>
<td>None</td>
<td>Text</td>
<td>transferStoneEmailTemplate</td>
<td>transferStoneDisplayOutputTemplate</td>
<td>PARAM_USER_ID, PARAM_FS_ACCOUNT, PARAM_BSB, PARAM_ACCOUNT_NUMBER, PARAM_ACCOUNT_NAME, PARAM_AMOUNT, PARAM_FROM_DESCRIPTION, PARAM_TO_DESCRIPTION, AccountManager, TransferManager</td>
</tr>
<tr>
<td>GetBalanceStone</td>
<td>None</td>
<td>Amount</td>
<td>getBalanceStoneEmailTemplate</td>
<td>getBalanceStoneDisplayOutputTemplate</td>
<td>PARAM_USER_ID, BILLER_SP (optional), ServiceProviderManager, ServiceProvider, BillManager</td>
</tr>
</tbody>
</table>
7.7 BOP Gantt Chart

The following gantt chart shows the original scheduled plan for thesis B. The project plan involved the completion of the BOP project implementation by Week 9, Session 2 of the UNSW academic calendar which left a substantial period of time for user acceptance testing via end user surveys and the modification of the system from feedback received. As stated in Section 4 Project Management, the implementation ran over the schedule by two weeks, but enough slippage was allowed for in the plan to enable the system to be successfully completed.
7.8 Existing Bill Aggregators

**Acresis - PayOnce**

In a nutshell, PayOnce [1] is a bill payment service with a linked credit facility. The company Acresis automatically pays the bills (for a user's registered billers) and the user is then sent a bill from Acresis. The main objective of this system is an "all-in-one" bill payment service as the user is sent a single monthly bill and not have to manage multiple bills.

**Advantages**

- a user will no longer have to worry about payment due dates
- a user only needs to pay one bill (the one sent from Acresis)
- ability to pay bills in another person's name

**Disadvantages**

- interest rates are quite high
- usual bills are still sent through mail by billers, which may be confusing to the user to differentiate which bills are managed and paid by Acresis or not.
- the number of supported billers is small

**Maybank2u**

Maybank2u [6] is a one-stop financial portal where a user can manage their Maybank finances along with bill payments. It is essentially an online banking service with bill management support.

![Maybank2u Interface](image)

**Advantages**

- accounts balance summary provided
- can schedule bill payments
- more secure and reliable as bill payment transactions go directly through the bank

**Disadvantages**

- accounts balance summary only for Maybank accounts or Maybank credit cards
- doesn’t provide the functionality to view the bill itself

**MyCheckFree**

MyCheckFree [2] provides a free online billing and payment service where you can receive, view, and pay your online bills from one personalized site. Bills can be individually paid using a checking or money-market account, or major credit cards where accepted by the biller.

**Advantages**

- can schedule payments
- can receive/view bills online (also by email) rather than by mail
- Ability to pay bills in another person's name
- a large number of billers are supported

**Disadvantages**

- limited in payment methods - some people may not have a checking or money-market account
- fees apply for same day or credit card payments
- takes 2 days to process a payment
PayTrust - MyBillCenter

PayTrust [3] is a fee paying internet bill delivery system that enables users to receive, review, pay and organize all bills online in one convenient location. Users can register a biller at MyBillCenter and PayTrust will be responsible for retrieving the bills (either electronically or by mail). Paper bills will be scanned and posted on MyBillCenter.

Advantages

- can schedule payments (including automatic payments)
- all billers are supported
- can receive/view bills online (also by email) rather than by mail
- can make payments to anyone (not just pay a bill)
- can access bank account balance

Disadvantages

- only works with transaction accounts that provide check writing privileges
- takes up to 3 business days to process an electronic payment and up to 5 business days to process a cheque payment

Xpress Bill Pay

Xpress Bill Pay [4] is a free online bill management system that enables users to pay bills via credit card, debit card, or transfer funds directly from a banking account using electronic funds transfer. Billers are required to upload bills to the Xpress Bill Pay's servers and download an activity file to import the payments made into their system.

Advantages

- can schedule payments (including automatic payments)
- can receive/view bills online (also by email) rather than by mail

Disadvantages

- billers are forced to upload bills and download/import payment activities into their system
- not for the average, everyday person. Marketed at:
  - Municipal Government - utility payments and other city fees
  - Utility Companies
  - Rental Properties

HSBC EasyView

EasyView [5] is not really a bill aggregator but brings all personal online account information together and presents it on a single summary page. This includes information from online bank accounts, investment firms, online shopping sources, your travel information, news sources and e-mail. This system has been included in our investigation as it is a good example of the various information a portal can gather.

Advantages

- can view account balances from multiple financial institutions
- aggregates a lot of information from various sources

Disadvantages
• no bill management support
• functionality is limited to only retrieval and display of information
• still need to login into Internet banking to pay bills
## 7.9 Time Tracking

<table>
<thead>
<tr>
<th>Date</th>
<th>What we did...</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon, 2 July</td>
<td>• setup BOP using <a href="#">appfuse</a> • setup BOP idea project</td>
<td>4</td>
</tr>
<tr>
<td>Tue, 3 July</td>
<td>• alter <a href="#">User table and user sign up/edit profile UI</a> • fix dao unit tests • fix ID generation for Postgres • create <a href="#">Account tables</a></td>
<td>7</td>
</tr>
<tr>
<td>Mon, 9 July</td>
<td>• create SP, <a href="#">Payment tables</a> &amp; dao classes • implement manage accounts tab • manage accounts &gt;&gt; billing accounts display • implement <a href="#">AccountManager</a></td>
<td>10</td>
</tr>
<tr>
<td>Tue, 10 July</td>
<td>• manage accounts &gt;&gt; fs accounts display • view billing/fs account • add new billing/fs account • implement <a href="#">SPManager</a> • logout position • add account tabs &amp; remove logout, file upload tabs</td>
<td>9</td>
</tr>
<tr>
<td>Sun, 15 July</td>
<td>• delete account • handle action errors • validation</td>
<td>4</td>
</tr>
<tr>
<td>Mon, 16 July</td>
<td>• meeting with Helen • install JUDDI + UDDI4J • install Open Federation</td>
<td>7</td>
</tr>
<tr>
<td>Sun, 22 July</td>
<td>• investigation on: • SAML • WS-Federation • ESOE • Open Federation + samples</td>
<td>6.5</td>
</tr>
<tr>
<td>Mon, 23 July</td>
<td>• generated SAML java objects using JAXB • started samI4J - authnRequest and handleAuthnRequest</td>
<td>9</td>
</tr>
<tr>
<td>Mon, 6 August</td>
<td>• meeting with Helen • implemented Banjo (FS) WS + mini webapp • added BOP WS interfaces</td>
<td>8</td>
</tr>
<tr>
<td>Wed, 8 August</td>
<td>• bop authenticationServlet</td>
<td>3</td>
</tr>
<tr>
<td>Mon, 13 August</td>
<td>• spring mvc investigation</td>
<td>4</td>
</tr>
<tr>
<td>Date</td>
<td>Tasks</td>
<td>Hours</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Wed, 15 August</td>
<td>• authentication response • account summary - fs + billing • Apple (Billing) WS + webapp</td>
<td>6.5</td>
</tr>
<tr>
<td>Mon, 20 August</td>
<td>• Meeting with Helen • Apple getOutstandingBills • getPaymentMethods • getBillierCode • payBillViaCC/DD/BPAY • view outstanding bills • view single bill • started pay bill</td>
<td>9.5</td>
</tr>
<tr>
<td>Wed, 22 August</td>
<td>• work more on pay bill • work more on view bill UI</td>
<td>2.5</td>
</tr>
<tr>
<td>Mon, 27 August</td>
<td>• finish pay bill • add view html/pdf bill • search payments • transfer funds UI</td>
<td>8</td>
</tr>
<tr>
<td>Wed, 29 August</td>
<td>• finish transfer funds • send mail method implemented • spec out stories</td>
<td>5.5</td>
</tr>
<tr>
<td>Mon, 3 September</td>
<td>• Story class data layer • Stone interface + abstract stone • StoryParser</td>
<td>7</td>
</tr>
<tr>
<td>Thu, 6 September</td>
<td>• extract getBalance to manager</td>
<td>1</td>
</tr>
<tr>
<td>Fri, 7 September</td>
<td>• autowire stones in StoneManager • implement run methods of GetBillsStone &amp; PayBPAYStone • added YUI toolkit to BOP • initial storyboard UI</td>
<td>7</td>
</tr>
<tr>
<td>Mon, 10 September</td>
<td>• extract storyboard css • stop stones dropping on top of each other • initial inputs screen for getBillsStone • transfer receipt email</td>
<td>7</td>
</tr>
<tr>
<td>Wed, 12 September</td>
<td>• fix picker styles • finish pickers for getBills and payBPAY stones</td>
<td>2</td>
</tr>
<tr>
<td>Mon, 17 September</td>
<td>• change stone inputs to have a set of inputs • email templates for email stone • highlight targets upon drag • other stones UI • stone popup input screens</td>
<td>6.5</td>
</tr>
<tr>
<td>Date</td>
<td>Tasks</td>
<td>Hours</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| Wed, 19 September| • finished off the popup input screens  
• added error checking on popup input screens  
• stop stones dropping on invalid targets                                    | 3     |
| Wed, 26 September| • list stories  
• delete story  
• run story  
• get balance stone  
• load template  
• save template  
• save story popup                                                             | 6.5   |
| Sun, 30 September| • finished add story action  
• code refactor  
• dwr (remove hard coded stone I/O types)  
• thesis B contents  
• write up evaluation survey                                                    | 5.5   |
| Mon, 1 October   | • meeting with helen  
• put in quartz (scheduling story)  
• tried to figure out ww tooltips  
• start display output stone                                                     | 5     |
| Wed, 3 October   | • got quartz working  
• finish display output stone  
• added more story tests, sample data                                           | 3     |
| Mon, 8 October   | • finish scheduling  
• added data for user evaluations  
• fixed random little bugs  
• started fixing up storyboard                                                  | 5.5   |
| Wed, 10 October  | • fixed storyboard so stones are highlighted  
• storyboard help added  
• load template fixed  
• random little bug fixes  
• started apple configuration (urls etc)                                         | 5     |
| Sun, 14 October  | • report writing                                                                                               | 5.5   |
| Mon, 15 October  | • user evaluations  
• fix storyboard styling  
• dashboard  
• bill history                                                                | 7     |
| Wed, 17 October  | • continue with dashboard  
• added icons  
• fixed decimal formatting  
• field validation errors  
• don't allow scheduling for display only stories  
• add story validation                                                          | 6     |
<table>
<thead>
<tr>
<th>Date</th>
<th>Task Description</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun, 21 October</td>
<td>• show stone popup window upon drag drop</td>
<td>5.5</td>
</tr>
<tr>
<td>Mon, 22 October</td>
<td>• report writing</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>• more user evaluations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• fix up things according to user evaluations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• fix storyboard highlighting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• small bug fixes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• added banana and pipe SP</td>
<td></td>
</tr>
<tr>
<td>Wed, 24 October</td>
<td>• fix email/display templates for pay bills</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>• edit story</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• user evaluations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• small bug fixes</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>202.5</strong></td>
</tr>
</tbody>
</table>