



SOEN 6471: Advanced Software Architectures

Winter 2014

BUDDI – Personal Budget

Team Members

Amer Al-Mahdawi (6411754)

Khalid Altoum (6413439)

Gustavo Barbieri Pereira (6273033)

Nikita Parashar (6766994)

Vishnu Vardhan Rajmohan (6688799)

Table of Contents

Team Members	1
Project Description.....	3
Project Scope	3
Glossary.....	4
1. Personas, Actors, and Stakeholders.....	5
1.1. Personas.....	5
1.1.1. Primary Persona.....	5
1.1.2. Secondary Persona (1)	5
1.1.3. Secondary Persona (2)	5
1.2. Possible stakeholders.....	6
1.3. Actors	6
2. Informal Use Case	7
2.1 UC-1: Maintain Budget Category	8
2.2 UC-2: Set up a Budget	10
2.3 UC-3: Maintain Account.....	11
2.4 UC-4: Maintain Account Type	13
2.5 UC-5: Record a Transaction.....	15
2.6 UC-6: Record a Scheduled Transaction.....	16
2.7 UC-7: Generate Report	17
Nouns and Verbs.....	18
UML Diagram	19
Used tools:	21
References	22

Project Description

Buddi[2] is a software developed for managing personal finance and budgeting. The goal of this application is to help people with little or no financial background. It can be run on almost any platform (Windows, Mac OS X, Linux, and many other operating system) which has java virtual machine installed (Java 1.6 or higher). It can be translated to any languages easily by using its own language editor which allows anyone to edit the translation of the program itself.

Buddi is an open source software developed by *Wyatt Olson*, released under GNU General Public Licenses, that allows anyone to access, modify, and add/remove features to the source code preferably. Its first release (version 0.9 Beta Release) was in May 2006 and its most current stable release is Buddi-3.4.1.11 which was released in August 2013.

In 2011 Buddi was the winner of *About.com Personal Finance Software (Mac) Award*, and also it has been nominated for the *About.com Reader's Choice Awards*.

There are thousands of similar softwares in the market, and most of them are expensive and difficult to understand and maintain. There are some free good tools, as Mint[4], AceMoney[1], MoneyDance[5], YNAB[7] that are listed on a Forbes research about Budgeting Software Options[3], but none of them is an extensible open source project.

All of the above inspires us to choose this software as a part of our project to fulfill the requirements of SOEN 6471/Advanced Software Architecture course.

Project Scope

Based on our main persona and the priority assessment of our use cases, we are focusing our project on Transactions, Budget Categories and Accounts. This is also the core of the business model of this application. Reports and visualization using graphics are not in our scope.

Glossary

Term	Definition
Account	A representation of the place where the money is coming from or going to.
Budget	An estimate of income amount or expense amount for a set period of time.
Budget Category	A classification for a Budget, that type can be either Income or Expense
Expense	The cost required for something; the money spent on something
Income	Money received, occasional or on a regular basis, into accounts.
N/A	Not applicable.
Priority	Importance of a use case or a user story, classified as Low, Medium, or High.
Transaction	A record of money movement, either income or expense.
Use Case	Describes the use of the system in terms of a series of interactions between the actors and the system that yield a value to a particular actor.
Use Case Diagram	A representation of a user's interaction with the system and depicting the specifications of a use case.
User	A person who uses Buddi system to control a Personal Budget.
User Story	Describes the use of the system by focusing on a single interaction between the user and the system that yields a value to the user.

1. Personas, Actors, and Stakeholders

1.1. Personas

1.1.1. Primary Persona

Natalie Desjardins is a 35 years old mother of two children. She holds a bachelor degree in Human Relations, and she works as a store manager at Dollarama. She takes care of her family finances, since her husband does not know how to deal with finances and always forget to pay his bills on time. She has a joint-account with her husband. She would love to have an application to help her control her family budget. She would like to easily take notes of expenses. She would love to have control of her husband spending habits and where their salaries are going to.

1.1.2. Secondary Persona (1)

Ashley is a mother of a teenager. She gives him weekly allowance to use as pocket money, but he asks for more before the weekend, and when she asks him where he spent his money, he forget to mention some details all the time. She wants to teach him how to manage his expenses, but she does not know the right way to do so. She knows that he likes to use electronic devices like his laptop and his smartphone. So she wants to give him an application to control his expenses. This way, she can see all the details of his expenses.

1.1.3. Secondary Persona (2)

Andy is a 25 years old, graduate student in Archeology, University of Toronto. He is comfortable with technology like email and office applications. He has no financial background. Each month, he tries to manage and control his expenses, but he fails on keep tracking of them. He does not know how to categorize and assign a budget for his expenses. He needs an application to help him with his finances without costing him a lot.

1.2 Possible stakeholders

Project sponsor, Development team, Free-lancer Developers and Contributors to Open-Source project.

1.3. Actors

User.

2. Informal Use Case

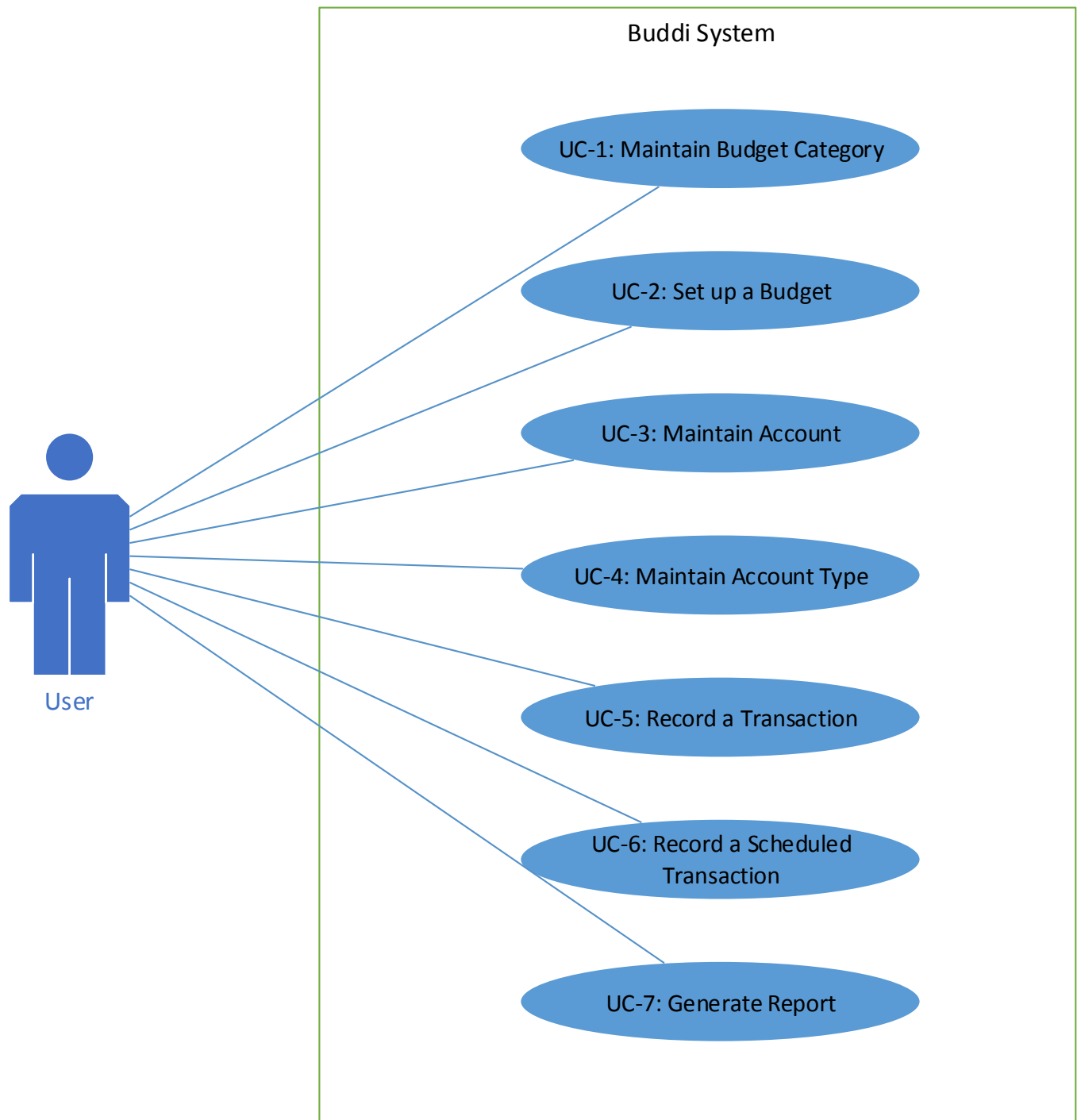


Figure 1: UML Usecase Diagram for Buddi System

2.1 UC-1: Maintain Budget Category

Use Case ID:	UC-1
Use Case Name:	Maintain Budget Category
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to add, update, and/or delete budget categories.
Description/Summary:	This use case allows the user to <u>maintain</u> budget categories information in the system . This includes <u>adding, modifying, and deleting</u> budget categories from the system . Each category has its own type (income or expense) , and has a period type (week, month, quarter, etc).
Preconditions:	N/A
Post-conditions:	New budget category is created with a unique name.
Basic Flow:	<p>This use case starts when the user wish add, change, and/or delete budget category information in the system.</p> <ol style="list-style-type: none"> 1. The system <u>requests</u> that the user to <u>specify</u> the function he/she would like to <u>perform</u> (either Add/Update/Delete a budget category) 2. Once the user <u>provides</u> the <u>requested</u> information, one of the sub flows is <u>executed</u>. If the user <u>selected</u> “Add a Budget Category”, the Add a Budget Category subflow is <u>executed</u>. If the user <u>selected</u> “Update a Budget Category”, the Update a Budget Category subflow is <u>executed</u>. If the user <u>selected</u> “Delete a Budget Category”, the Delete a Budget Category subflow is <u>executed</u>. <p>1.1 Add a Budget Category</p> <p>The system <u>requests</u> that the user <u>enter</u> the budget category information. This includes: category name, its type either income or expense, and its period type whether week, month, quarter, etc.</p> <ol style="list-style-type: none"> 1. User <u>enters</u> the required information values, and <u>requests</u> the system to <u>save</u> them. 2. Once the user <u>provides</u> the <u>requested</u> information, the system <u>validates</u> the uniqueness of the category’s name.

3. **System** stores the entered category's values, and notifies the **user** that the category has been created.

1.2 Update a Budget Category

1. The **system** displays existing **budget categories** and requests the **user** to select the **budget category** name.
2. The **user** selects the desired **budget category** name. The **system** retrieves and displays the **budget category** information.
3. The **user** makes the desired changes to the **budget category** information. This includes any of the information specified in the **Add a Budget Category** sub-flow.
4. Once the **user** updates the necessary information, the **system** updates the Budget Category information.

1.3 Delete a Budget Category

1. The **system** displays existing **budget categories** and requests the **user** to select the **budget category** name.
2. The **user** selects the **budget category** name. The **system** retrieves and displays the budget category information.
3. The **system** prompts the **user** to confirm the deletion of the budget category.
4. The **user** verifies the deletion.
5. The **system** deletes the budget category from the **system**.

Alternative Flows:

1.1.2a , 1.2.3a Budget category name already exists:

If, in the **Add/Update a Budget Category** sub-flows, the **user** enters a budget category name that is already existed, the **system** prompts the **user** to enter new **budget category** name, and the **system** continues to next step of main scenario.

1.3.4a Delete Cancelled

If, in the **Delete a Budget Category** sub-flow, the **user** decides not to delete the budget category, the delete is cancelled, and the Basic Flow is re-started at the beginning.

2.2 UC-2: Set up a Budget

Use Case ID:	UC-2
Use Case Name:	Set up a Budget
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to set up a budget.
Description/Summary:	The Set up a Budget use case <u>allows</u> the user to <u>set up</u> his own budget for a specific coming period. Each category has its own type (income or expense), and has a period type (week, month, quarter, etc.).
Preconditions:	At least an account and one budget category has been created.
Post-conditions:	New budget is set up for upcoming period.
Basic Flow:	<p>This use case <u>starts</u> when a user <u>wishes</u> to <u>setup</u> a budget either income or expense for upcoming period.</p> <ol style="list-style-type: none"> 1. System <u>requests</u> to <u>enter</u> the amount of money <u>planned</u> for each category in the specified period. 2. The user <u>enters</u> the required values, and <u>requests</u> the system to <u>save</u> them. 3. System <u>stores</u> the <u>entered</u> values, <u>calculates</u> the account balance and <u>displays</u> it to the user.
Alternative Flows:	<p>2a Insufficient funds:</p> <p>If the user <u>enters</u> amount more than what he/she has in his/her account, the system <u>prompts</u> the user to <u>re-enter</u> a valid amount, and the system <u>continues</u> to step 2 of main scenario.</p>

2.3 UC-3: Maintain Account

Use Case ID:	UC-3
Use Case Name:	Maintain Account
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to add, update, and/or delete his/her account.
Description/Summary:	This use case allows the user to <u>maintain</u> account information in the system . This includes <u>adding, modifying, and deleting</u> account from the system . Each account has its own name, type (cash, saving, credit card, etc.), and its starting balance .
Preconditions:	At least an account type has been created.
Post-conditions:	New account is created with a unique name.
Basic Flow:	<p>This use case starts when the user wishes to add, change, and/or delete account information in the system.</p> <ol style="list-style-type: none"> 1. The system <u>requests</u> that the user to <u>specify</u> the function he/she would like to <u>perform</u> (either Add/Update/Delete an Account) 2. Once the user <u>provides</u> the <u>requested</u> information, one of the sub flows is <u>executed</u>. If the user <u>selected</u> “Add an Account”, the Add an Account subflow is <u>executed</u>. If the user <u>selected</u> “Update an Account”, the <u>Update an Account</u> subflow is <u>executed</u>. If the user <u>selected</u> “Delete an Account”, the <u>Delete an Account</u> subflow is <u>executed</u>. <p>3.1 Add an Account</p> <p>The system <u>requests</u> that the user <u>enter</u> the account information. This includes: account’s name, type (cash, saving, credit card, etc.), and specify its starting balance.</p> <ol style="list-style-type: none"> 1. User <u>enters</u> the required information values, and <u>requests</u> the system to <u>save</u> them. 2. Once the user <u>provides</u> the requested information, the system <u>validates</u> the uniqueness of the account name. 3. System <u>stores</u> the <u>entered</u> account values, and <u>notifies</u> the user that the account has been <u>created</u>.

3.2 Update an Account

1. The **system** displays existing accounts and requests the **user** to select the **account** name.
2. The **user** selects the desired **account** name. The **system** retrieves and displays the **account** information.
3. The user makes the desired changes to the **account** information. This includes any of the information specified in the **Add an Account** sub-flow.
4. Once the **user** updates the necessary information, the **system** updates the **Account** information.

3.3 Delete an Account

1. The **system** displays existing **accounts** and requests the **user** to select the **account** name.
2. The **user** selects the **account** name. The **system** retrieves and displays the **account** information.
3. The **system** prompts the **user** to confirm the deletion of the **account**.
4. The **user** verifies the **deletion**.
5. The **system** deletes the **Account** from the **system**.

Alternative Flows:

3.1.2a , 1.2.3a Account name already exists:

If, in the **Add/Update an Account** sub-flows, the **user** enters an **account** name that is already existed, the **system** prompts the **user** to enter new **account** name, and the **system** continues to next step of main scenario.

3.3.4a Delete Cancelled

If, in the **Delete an Account** sub-flow, the **user** decides not to delete the **account**, the delete is cancelled, and the Basic Flow is re-started at the beginning.

2.4 UC-4: Maintain Account Type


Use Case ID:	UC-4
Use Case Name:	Maintain Account Type
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to add, update, and/or delete account type.
Description/Summary:	This use case allows the user to <u>maintain</u> account type information in the system . This includes <u>adding</u> , <u>modifying</u> , and <u>deleting</u> account from the system . Each account type has its own name, type (debit or credit).
Preconditions:	N/A
Post-conditions:	New account type is created with a unique name
Basic Flow:	<p>This use case <u>starts</u> when the user <u>wishes</u> to <u>add</u>, <u>change</u>, and/or <u>delete</u> account type information in the system.</p> <ol style="list-style-type: none"> 1. The system <u>requests</u> that the user to <u>specify</u> the function he/she would like to <u>perform</u> (either Add/Update/Delete an Account Type) 2. Once the user <u>provides</u> the <u>requested</u> information, one of the sub flows is <u>executed</u>. If the user <u>selected</u> “Add an Account Type”, the <u>Add an Account</u> Type subflow is <u>executed</u>. If the user <u>selected</u> “Update an Account Type”, the <u>Update an Account</u> Type subflow is <u>executed</u>. If the user <u>selected</u> “Delete an Account Type”, the <u>Delete an Account</u> Type subflow is <u>executed</u>. <ol style="list-style-type: none"> 4.1 Add an Account Type <p>The system <u>requests</u> that the user <u>enter</u> the account type information. This includes: account type name, type (debit or credit).</p> <ol style="list-style-type: none"> 1. User <u>enters</u> the required information values, and <u>requests</u> the system to <u>save</u> them. 2. Once the user <u>provides</u> the <u>requested</u> information, the system <u>validates</u> the uniqueness of the account type name. 3. System <u>stores</u> the <u>entered</u> account type values, and <u>notifies</u> the user that the account type has been <u>created</u>. 4.2 Update an Account Type <ol style="list-style-type: none"> 1. The system <u>displays</u> existing account types and <u>requests</u> the user to

	<p><u>select</u> the account type name.</p> <ol style="list-style-type: none"> 2. The user <u>selects</u> the desired account type name. The system <u>retrieves</u> and <u>displays</u> the account type information. 3. The user <u>makes</u> the <u>desired</u> <u>changes</u> to the account type information. This includes any of the information <u>specified</u> in the Add an Account Type sub-flow. 4. Once the user <u>updates</u> the necessary information, the system <u>updates</u> the Account Type information. <p>4.3 Delete an Account Type</p> <ol style="list-style-type: none"> 1. The system <u>displays</u> existing account types and <u>requests</u> the user to <u>select</u> the account type name. 2. The user <u>selects</u> the account type name. The system <u>retrieves</u> and <u>displays</u> the account type information. 3. The system <u>prompts</u> the user to <u>confirm</u> the <u>deletion</u> of the account type. 4. The user <u>verifies</u> the <u>deletion</u>. 6. The system <u>deletes</u> the Account Type from the system.
Alternative Flows:	<p>4.1.2a , 1.2.3a Account Type name already exists:</p> <p>If, in the Add/Update an Account sub-flows, the user <u>enters</u> an account type name that is already <u>existed</u>, the system <u>prompts</u> the user to <u>enter</u> new account type name, and the system <u>continues</u> to next step of main scenario.</p> <p>4.3.4a Delete Cancelled</p> <p>If, in the Delete an Account sub-flow, the user <u>decides</u> not to delete the account type, the <u>delete</u> is <u>cancelled</u>, and the Basic Flow is <u>re-started</u> at the beginning.</p>


2.5 UC-5: Record a Transaction

Use Case ID:	UC-5
Use Case Name:	Record a Transaction
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to record a new transaction.
Description/Summary:	This use case allows the user to <u>record</u> a Transaction into the system . The system <u>records</u> a transaction details which includes: date, description, transaction number, amount, and from/to.
Preconditions:	An account has been created and category types have been initialized.
Post-conditions:	<ul style="list-style-type: none">• New transaction is recorded.• The account balance is updated.
Basic Flow:	This use case starts when a user <u>wishes</u> to <u>record</u> a new <u>transaction</u> . The <u>transaction</u> information includes: date, description, amount to be <u>transferred</u> , the source of funds which could be either an account or an income category , and the destination account or expense category . The user <u>enters</u> the required values, and <u>requests</u> the system to <u>save transaction</u> . The system <u>stores</u> the <u>entered</u> account values, <u>updates</u> all the <u>affected</u> account(s) and/or budget(s), and <u>notifies</u> the user that the <u>transaction</u> has been successfully <u>recorded</u> .
Alternative Flows:	N/A

2.6 UC-6: Record a Scheduled Transaction

Use Case ID:	UC-6
Use Case Name:	Record a Scheduled Transaction
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to schedule a transaction.
Description/Summary:	This use case allows the user to record a Transaction into the system. The system records a transaction details as follows: date, description, transaction number, amount, start date, end date, interval and from/to.
Preconditions:	An account has been created and category types have been initialized.
Post-conditions:	New scheduled transaction is recorded.
Basic Flow:	<p>This use case starts when a user <u>wishes to record</u> a new <u>transaction</u>. The transaction information <u>includes</u>: date, description, amount to be transferred, the source of funds which could be either an account or an income  category, and the destination account or expense category, start date, end date and interval. The user <u>enters</u> the required values, and <u>requests</u> the system to <u>create</u> a new scheduled <u>transaction</u>. The system <u>saves</u> the scheduled transaction and <u>trigger</u> the <u>transfer of funds</u> when it <u>reaches</u> the cycle period. The system doesn't <u>make</u> any <u>changes</u> to the accounts(s)/budget(s) until it <u>reaches</u> the cycle period. When the <u>transfer of funds</u> are done, the system will <u>update</u> all the <u>affected account(s)</u> and/or <u>category(s)</u>, and <u>notifies</u> the user that the <u>transaction</u> has been <u>recorded</u>.</p>
Alternative Flows:	

2.7 UC-7: Generate Report

Use Case ID:	UC-7
Use Case Name:	Generate Report
Actor(s):	Primary Actor: User
Goal/Actor Goals:	A user wants to generate different representation (pie chart, histogram) of reports for a specified period of time.
Description/Summary:	This use case allows the user to view a graphical or tabulated report that includes all the details of the expenses, incomes, and transactions happen in a specific period, to verify how well s/he kept the planned budget.
Preconditions:	Transactions have been recorded.
Post-conditions:	A report of the specified period is displayed.
Basic Flow:	This use case starts when a user wishes to <u>generate</u> a report for a specific period of time. The system  <u>requests</u> the user to <u>enter</u> a specific period, and specify the <u>desired report</u> type. The user <u>enters</u> the <u>required</u> information values, and <u>requests</u> the system to <u>display</u> the report . System <u>displays</u> the <u>desired report</u>
Alternative Flows:	N/A

Nouns and Verbs

Nouns	Synonyms
User	
System	
Report	
Fund	
Category	
Account	
Account name	
Account type	
Budget	
Amount	
Description	
Income	
Source	
Transaction	
Scheduled Transaction	
Date	
Period	
Description	


Verbs	Synonyms
Generate	
Required	
Display	Notify
Desired	
Enter	Prompt
Has/Have	
Reaches	
Affected	
Transfer	
Trigger	
Create	Make, Add
Update	Modify, Change
Delete	Cancel
Record	
Include	
Save	
Store	
Enter	
Continue	
Restarted	
Verify	Confirm
Retrieved	
Calculate	

Request	
Allow	


UML Diagram

From the personas and the use cases model created in the previous sub-sections, the UML of the domain model of the underlying system has been driven as depicted in Figure 2.

Below are the details of the concepts included in this model:

Source  This is a super-class of two sub-classes that can be created by the User:

- **Account.** An instance of this class represents a location created to either store or owe money. The sum of the money of all Account's instances is the net worth. It has a name, balance of money, and it has an **Account Type**.
 - **Account Type.** Represent the type of the created type, like cash, saving, credit card, etc, and it has its own type, either income or expense.
- **Budget Category.** Instances of this class are created by the User to categorize the income and the the expenses, which can help to track the money. These categories can have a parent-children relationship in order to give more organizational flexibility. The User can set up her/his **budget** by specifying the amount of money assigned to each category at a specific time **period**, like weekly, monthly, etc.

Transaction. After creating the accounts, and setting-up the budget of a specific time period, instances of this class  are used to represent the transition of money from one account/budget category to another in a specific time.

Scheduled Transaction. Instances of this class are used to schedule transaction that happens in a regular basis on a specific period of time.

Report. Provides a way of reviewing the transactions and keep track of budget.



Used tools:

- Microsoft Word.
- Microsoft Visio (UML Diagram).

References

- [1] AceMoney: <http://www.mechcad.net/products/acemoney/>
- [2] Buddi : <http://buddi.digitalcave.ca/>
- [3] Forbes:<http://www.forbes.com/sites/moneywisewomen/2012/01/03/budgeting-software-options/>
- [4] Mint: <https://www.mint.com/>
- [5] MoneyDance: <http://moneydance.com/>
- [6] Simple Source Line Counter in Java for Java:
<http://www.codeproject.com/Tips/139036/Simple-Source-Line-Counter-in-Java-for-Java/>
- [7] YNAB: <http://www.youneedabudget.com/>