The Design and Build of a Simple Personal Finance System, Focused on Budgeting and Expenditure Analysis

Claudius de Moura Brasil
BSc Computing Project Report
Birkbeck College, University of London

May 2018

1 Abstract

Personal finance systems exist in abundance nowadays, from open source to proprietary ones. They all tend to revolve around a basic common theme: providing accurate information about an individual's income and expenditure. Beyond this, they tend to vary in which features are implemented. The system designed and built for this project focuses on the use of the bookkeeping principle of double entry and the concept of pattern matching to find effective ways to cagetorise a user's expenditure, and provide them with relevant financial information to assist in decision making.

Contents

2 Introduction

Vaasen et al. (2009 cited Boczko, 2012, p. 8) suggests that an accounting information system's main purpose is to provide information to internal and external stakeholders. Although this refers to accounting systems for business, it could be argued that the same concept could be applied for personal finance systems – except that, in this case, the main stakeholder would be the individual using the system (that is, the user). In fact, one of the most widely known accounting systems available in the market, Quicken™, was conceived around the idea that there should be more efficient and less tedious ways to organise one's personal financial information than doing it manually (Quicken Inc., 2017). This project has been developed based on these ideas.

The system has been modelled after the principle of *double entry bookkeeping*, from the accounting domain, which states that "money is never created or destroyed – it merely moves from one account to another" (Fowler, 1997, Section 6.2). More specifically, double entry is the principle which ensures that every transaction always affects two accounts, one being credited (Out) and one debited (In). An account, for the scope of this project, refers either to a category created by the user, or to the user's *cash book* – the contents of their bank account plus any manual entry which they make. In bookkeeping, each account can be classified as *asset, liability, income* or *expenditure*. Whether the account increases or decreases will depend on which of these categories it falls under: *debits* will increase *assets* and *expense* accounts, and *credits* will increase *liability, capital* or *income* acccounts (Wood et al., 2004, pp. 18-19).

This report documents the work of the project. Each chapter delineates a specific aspect of the development lifecycle, which is in line with the development process listed in 3. Chapter 4 identifies the identified requirements which were used as motivation for the system to be developed.

Analysis of requirements has been incorporated in the design section (Chapter 5). This is also where the first mention of patterns can be seen.

3 Development Method

For this project, an approach similar to that adopted by Bennett et al. (2010, p. 77) regarding methodology has been employed, where no specific named methodology is espoused, but concepts of object-oriented analysis and design were applied, in an iterative and incremental fashion, using UML. More details about which concepts were used and the methodologies which originated them can be found in the following subsections.

3.1 The use of Universal Modelling Language (UML) constructs

UML is a modelling language created with the intention of providing system architects, software engineers and developers with a common set of modelling tools, with a defined syntax, which would help them better analyse and design software-based systems, and to model business and similar processes (OMG, 2015, p. 43). It defines several constructs which have been employed throughout this report in order to model the specifications of the system, such as:

Use Case diagrams As a useful, high level tool to document users' requirements (Bennett et al., 2010, p. 138), use case diagrams have been used to develop the requirements model of the system.

Activity diagrams

Class diagrams

Sequence diagrams

3.2 Requirements Capture Methods

Due to the nature of the system being for personal rather than commercial use – that is, by individuals rather than business entities – the usual fact finding techniques do not apply specifically well. However, the closest match identified to the techniques utilised has been with '*Knowledge Acquisition*'. This relates to the process of capturing knowledge from an expert (Bennett et al., 2010, p. 150). In this particular case, though perhaps not qualifying as an expert, the author's qualification and experience in accounting and bookkeeping was used to capture the main requirements.

3.3 Analysis and Design patterns

Where appropriate, analysis and design patterns will be used, either in full or with some modifications.

Fowler (1997, Section 1.3) defines a pattern as “an idea that has been useful in one context and will probably be useful in others”. This project will therefore attempt to utilise patterns where appropriate in order to prove this concept, and as an attempt to make use of the experience already acquired in the domain (or domains) in question.

The concept of *domain* here is being used, as defined by Evans (2004, p. 2), as the “activity or interest of its user” – the “subject area to which the user applies the program”.

Analysis patterns will be used “when trying to understand the problem” domain (Fowler, 1997, Section 1.1).

4 Requirements

This project is building a system to find a solution for a problem. One of the first steps in order to do this is to try to understand what the problem is – that is, try to map the requirements of the software. Bennett et al. (2010, pp. 140-142) categorises requirements as being of three types:

Functional Requirements The system's functionality – what it is expected to do.

Non-functional Requirements How well the system delivers its functionality. These requirements are related to the performance, scalability, availability, recovery time, security, and others.

Usability Requirements These relate to how effectively, efficiently and satisfactorily users can achieve their goals in the existing system. User interfaces can play a big part in meeting these requirements.

These definitions will be employed when trying to classify the requirements and model the problem domain. The initial iterations will be focused more on the functional and usability requirements, paying some attention as well to specific non-functional requirements such as performance and security.

Any personal accounting system should be able to provide accurate and relevant summaries of an individual's financial status. In order to do this, the user needs to be able to supply the system with the necessary data so that it can be analysed and properly converted into knowledge.

It seems fair to infer that nowadays most of a user's financial transactions happen in ways that can be listed electronically (usually via their bank or credit card statements) – a study by Payments UK (2017), for example, indicates that there has been a rise in debit card payments over the past few years, and that the volumes of this type of transaction is likely to be higher than that of cash payments by the year 2021. Therefore, an assumption has been made that the users will require means of uploading a list of their financial transactions into the system.

The system created for this project intends to do just this. Its main feature, however, will be to allow the user to categorise expenditure based on patterns in the entries' descriptions. However,

There must also be a feature to allow the user to view summaries of the income and expenditure over a period of time, as well as one to forecast budgets for future periods based on the "financial behaviour" analysed.

4.1 Functional Requirements

Use case diagrams are UML constructs which were developed by Jacobson et al. (1992, cited Bennett et al., 2010, p. 154). The use case diagram on Figure 1 is used to illustrate the functional requirements identified for this project:

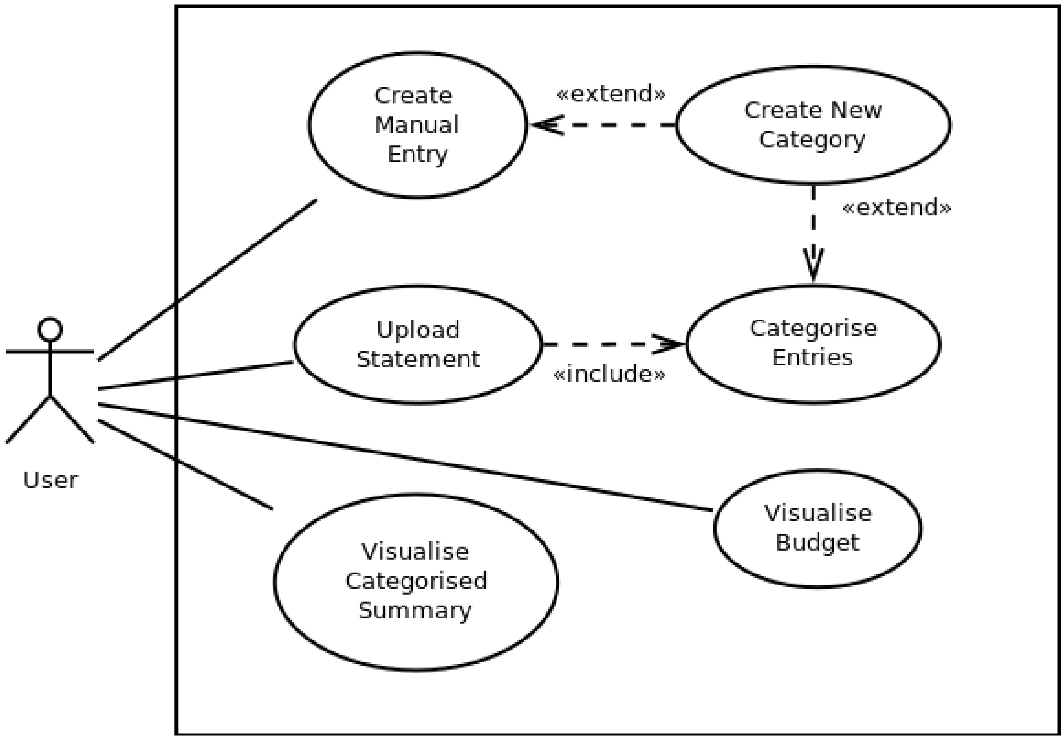


Figure 1: Use Case Diagram

Table 1 lists the descriptions for the use cases listed above. The first iteration had them all in a single table, but at subsequent ones more details were identified

Use Case	**Description**
Upload Statement	The user must be able to upload a list of their financial transactions, most likely their bank or credit card statements, in a suitable format, and all entries should be categorised based on specific patterns
Create Manual Entry	The user should be able to create a manual entry for income or expenditure, include a date, amount and description, and either choose an existing category for it or create a new one in the process
Visualise Categorised Summary	The use must be able to visualise a summary of their income and expenditure over a period of time
Visualise Budget	The user must be able to visualise a budget for future periods based on their income and expenditure data already entered

Table 1: Use Case Diagram

The wireframe below (Figure 2) was created to better illustrate the *Manual Entry* requirement from the point of view of the user. It shows an example of an entry for a laptop and a licence for a proprietary operating system, which can then be broken down among different categories. The user has the option to use the percentage or the amount boxes in order to provide a breakdown, and they can also add new lines if more than one is required – the example shows two lines, but the default would be one:

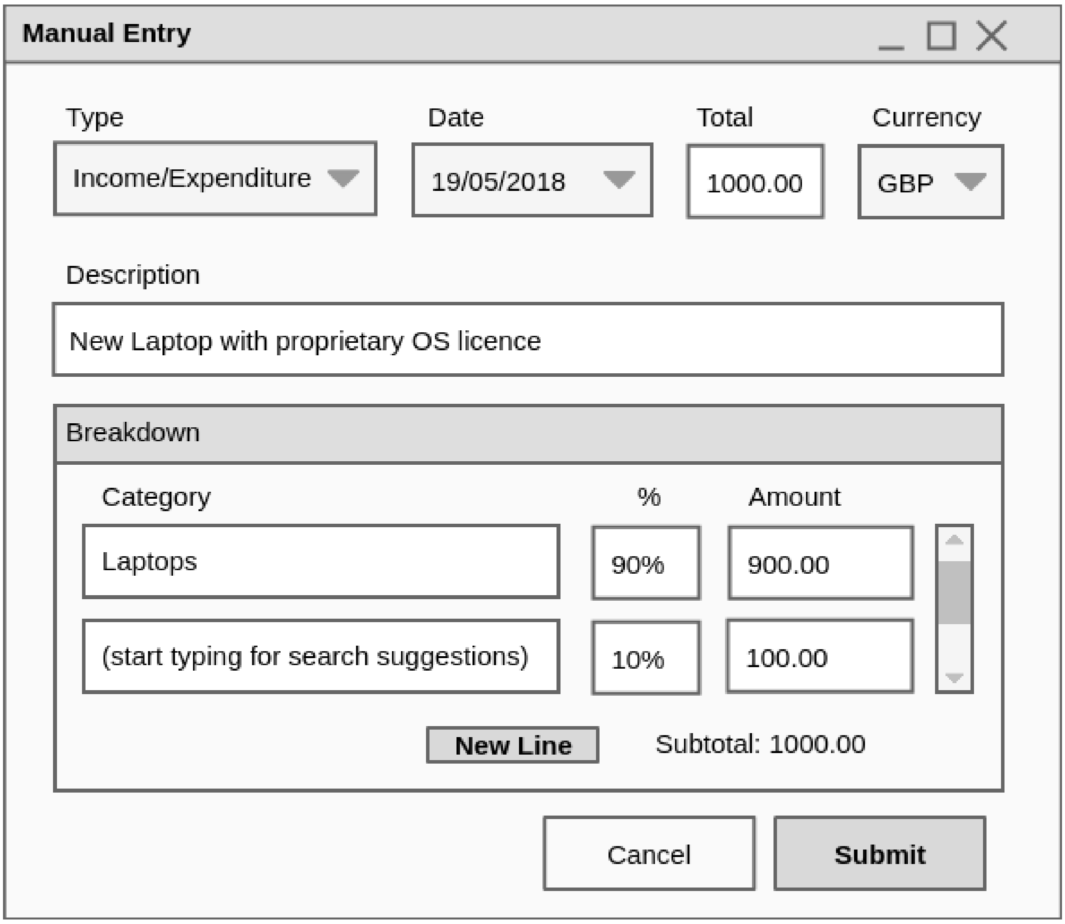


Figure 2: User interface wireframe for *Create Manual Entry* use case

And in order to better understand the relationship between *Upload Statement* and *Categorise Entries*, the activity diagram below (Figure 3) was developed:

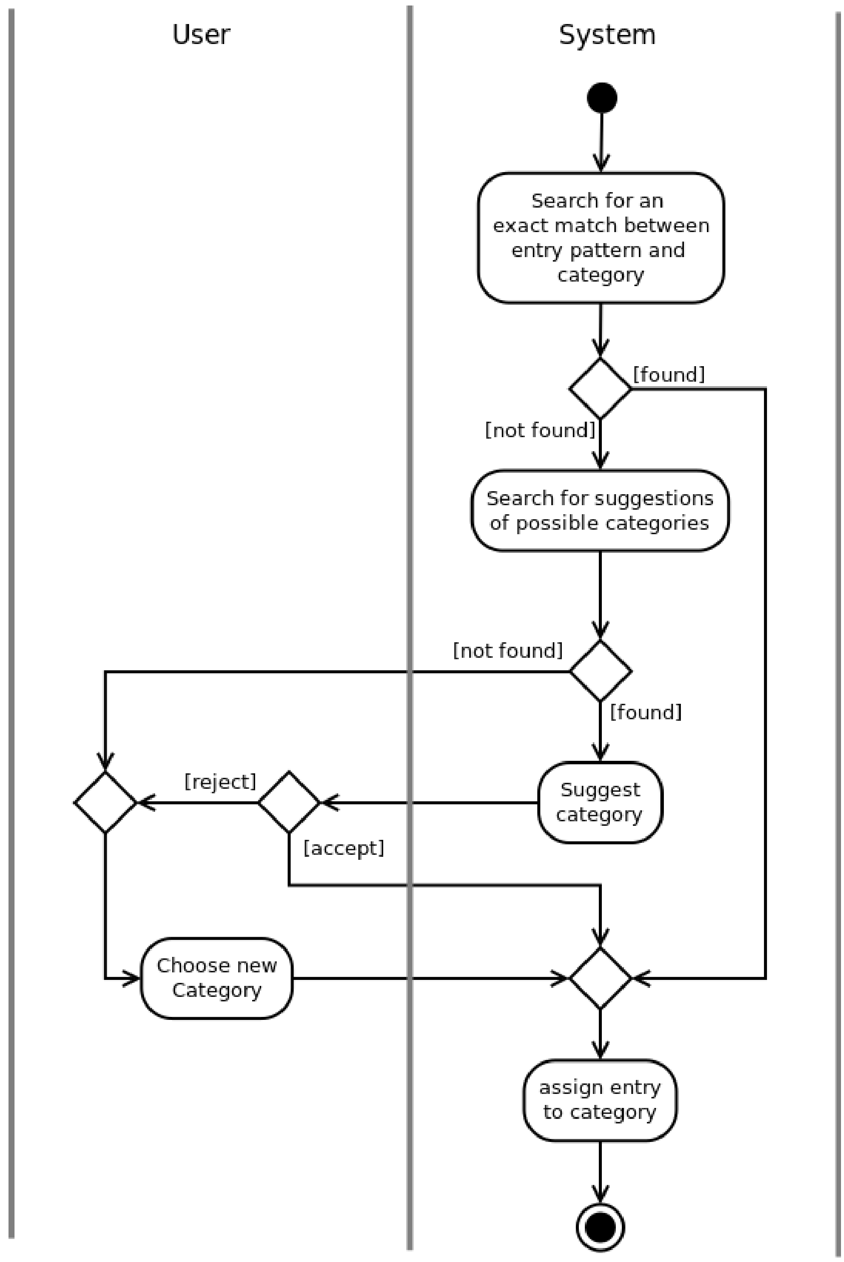


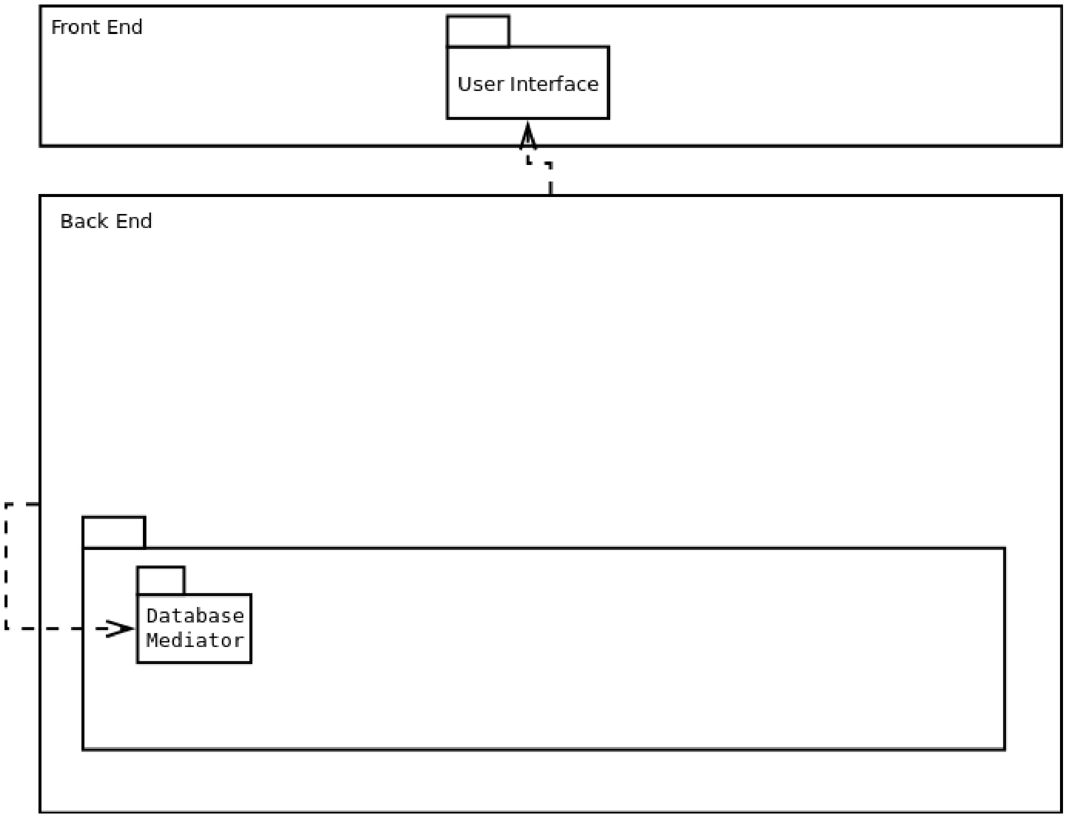
Figure 3

4.2 Non-Functional Requirements

Use cases and use case diagrams are an appropriate tool to document functional requirements, but not non-functional ones (Jacobson et al., 1999, cited Bennett et al., 2010, p. 153). Therefore, a separate list has been kept in order to document the non-functional requirements, where they exist.

4.3 Package Diagram

The diagram below provides a general idea of what the system should look like at a high level:



5 Analysis and Design

Seeing that this is a system dealing with finance, it would make sense to treat the categories as if they were accounts. And, in order to make sure to imbue this system with knowledge acquired by more experienced programmers, it makes sense to make use of patterns.

It is also useful at this point to make a distinction between the types of classes used to model the domain between three possible kinds: the first are the classes which model the interaction between the system and its actors – these are called *boundary classes*; the second kind are those classes which model information and/or behaviour or some concept or phenomenon – these will be called *entity classes*; and lastly, there are those classes which model transactions, coordination, control and sequencing of other objects – which are known as *control classes* (Jacobson et al., 1999, cited Bennett et al., 2010, pp. 198-201).

The first analysis patterns which seem appropriate are the *Account* pattern, used to create the *Category* entity class, and the *Quantity* pattern for the *Amount* entity class (Fowler, 1997, Sections 6.1 & 3.1):

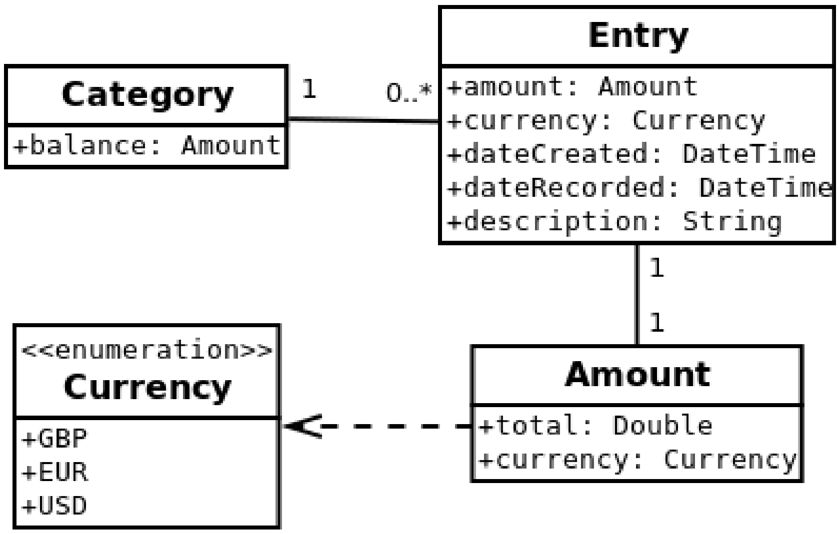
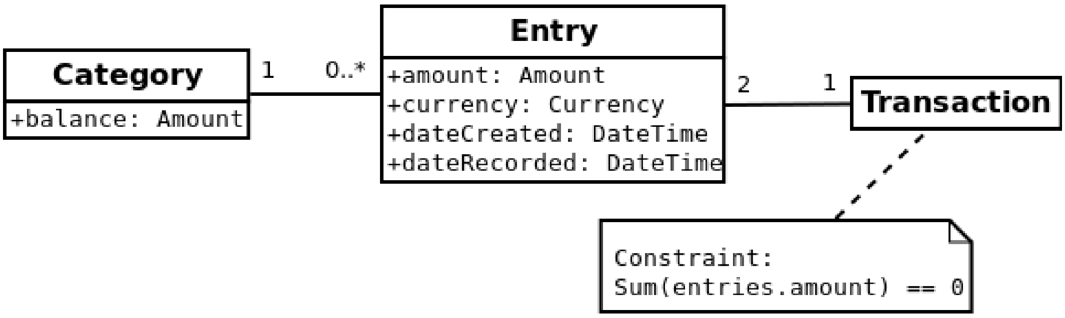


Figure 4

As implied by the diagram above, the *Category* class will be associated with instances of the *Entry* class, and will keep track of the balance made up of the sum of *Amount*'s of each entry. This is done so that the only way to change the total of a category is by adding positive or negative entries to it – for example, to indicate a credit to a category, a negative entry can be added to it.

Another design choice which can be observed in Figure 4 is that the *Amount* class also possesses an attribute for currency. This has been designed so as to allow for the possibility of extending the design to keep track of transactions in multiple currencies, although it was not a specific requirement. Initially, there will only be a single default currency which shall be set at runtime.

The next step is to provide a way for these entries to be added to categories. For this to happen, there needs to be a constraint in to ensure that double entry happens every time a change needs to be made to a category. One of the ways to achieve this is to apply the *Transaction* pattern (Fowler, 1997, Section 6.2):



After having determined the analysis patterns which shall be employed, it makes sense to dive into a deeper analysis of the use cases described in Chapter 4. At this point the objective will be to start modelling classes based on concepts or things found in the problem domain. This will be done in the following subsections.

5.1 Create Manual Entry

The *Create Manual Entry* use case, which allows a user to input financial transactions individually using a specific interface, can be modelled as follows (Figure 5):

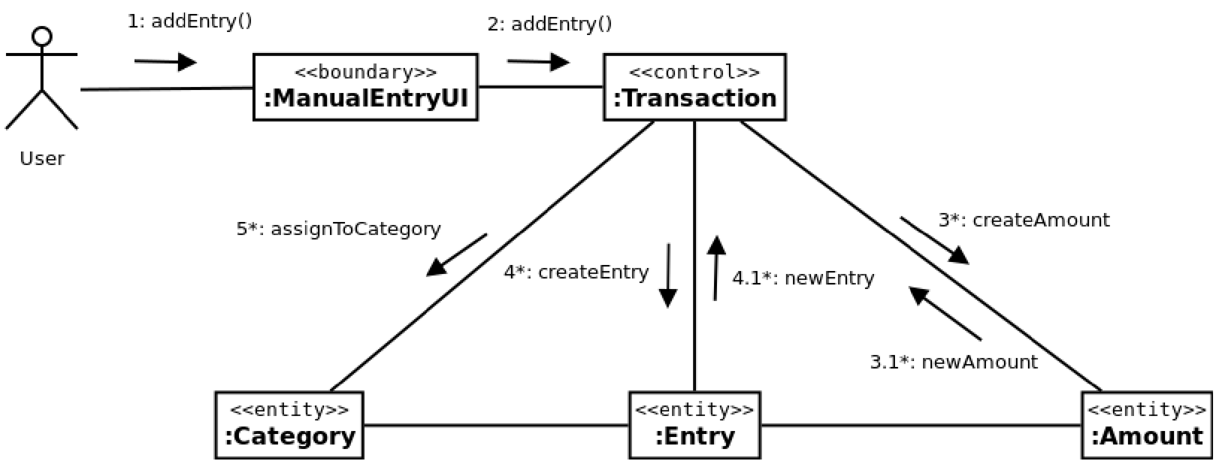


Figure 5: Communication Diagram for *Create Manual Entry* use case

As the diagram above indicates, the *Transaction* class is responsible for the creation of new instances of *Entry* and *Amount*, which then get assigned to the *Category* instances involved in the transaction. This process assumes that all categories already exist, and a separate diagram will be created for the creation of a new category.

6 Reflections

6.1 Expert Systems

Originally, the author did not know about expert systems when the idea for this project was conceived. However, during the literature search and review, the idea for these systems was found, and many of the patterns of what an expert system does and what this system is supposed to do were identified to be similar. This led to the conclusion that the project has the potential to become an expert system, even if just with a budgeting tool. The expert knowledge being provided by it for its first iteration includes:

- Double entry bookkeeping;
- Budgeting by category.

It achieves the above by separating the inference engine, which is the tool responsible for knowing how to apply double entry to transactions, from its knowledge base which is the information input by the user – for example, if the user tells the system a manual entry is income, the system will know to debit the cash book, and credit the category in question.

Brown et al. (1990) declare that the heuristics used by a financial planning system can be interpreted as a “rule of thumb” to be applied to a problem which will normally result in a correct solution for it. In the same article it is also stated that “an expert system is most commonly and most effectively used as an advisor to a human decision maker”. I this is considered as the measure by which classify an expert system, then the budgeting tool alone would place this system into it.

6.2 Use Case Templates

Originally, no template was used to document the use cases. The intention was to provide better ones at a later iteration, perhaps by researching the ones mentioned by Bennett et al. (2010, p. 157), but unfortunately there was not enough time, so the little there was had to be dedicated to the software itself.

References

Bennett, Simon, McRobb, Steve, and Farmer, Ray (2010). *Object-oriented systems analysis and design using UML*. Fourth Edition. McGraw Hill Higher Education. ISBN: 978-0-07-712536-3.

Boczko, Tony (2012). *Introduction to Accounting Information Systems*. Pearson Custom Publishing.

Brown, Carol E, Nielson, Norma I, and Phillips, Mary Ellen (1990). "EXPERT SYSTEMS FOR PERSONAL FINANCIAL PLANNING." In: *Journal of Financial Planning* 3.3.

Evans, Eric (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Fowler, Martin (1997). *Analysis patterns: reusable object models (OBT)*. Kindle Edition. Addison-Wesley Professional.

OMG (2015). *Unified Modeling Language Specification*. Version 2.5. URL: http://www.omg.org/spec/UML/2.5/PDF (visited on: 16 Dec. 2017).

Payments UK Management Ltd. (2017). *UK Payments Markets Summary*. URL: https://www.paymentsuk.org.uk/files/puk-uk-payment-markets-2017-summarypdf-0 (visited on: 18 Dec. 2017).

Quicken Inc. (2017). *About Us*. URL: https://www.quicken.com/about-us (visited on: 17 Dec. 2017).

Wood, Frank and Robinson, Sheila I (2004). *Book-keeping and Accounts*. 6th Edition. Pearson Education.