

## **Lincoln University Digital Thesis**

### **Copyright Statement**

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- you will use the copy only for the purposes of research or private study
- you will recognise the author's right to be identified as the author of the thesis and due acknowledgement will be made to the author where appropriate
- you will obtain the author's permission before publishing any material from the thesis.

# **Designing a Framework for End User Applications**

---

A thesis  
submitted in partial fulfilment of the requirements  
for Degree of Doctor of Philosophy  
in Software and Information Technology

at

Lincoln University

By

Yanbo Deng

---

Lincoln University

2013

# **Designing a Framework for End User Applications**

## **by Yanbo Deng**

End user developers (i.e. non-professional developers) often create database applications to meet their immediate needs. However, these applications can often be difficult to generalise or adapt when requirements inevitably change. As part of this thesis, we visited several research institutions to investigate the issues of end user developed databases. We found that different user groups in the same organisation might require similar, but different, data management applications. However, the very specific designs used in most of these systems meant it was difficult to adapt them for other similar uses.

In this thesis we propose a set of guidelines for supporting end user developers to create more flexible and adaptable data management applications. Our approach involves professional and end user developers working together to find a “middle way” between very specific and very generic designs. We propose a framework solution that allows the data model to have several co-existing variations which can satisfy the requirements of different user groups in a common domain. A “framework provider” (IT professional) will create the initial framework and data model. Configuration tools are then provided for a “framework manager” to easily customise the model to the specific needs of various user groups. The system also provides client toolkits and application generators to help end user developers (EUDs) to quickly create and customise applications based on the framework.

The framework approach was applied to a case study involving a Laboratory Information Management System (LIMS) for data on research experiments. We demonstrated that the framework developed could be successfully applied to several groups working in the same domain and could be extended to include new or changed requirements.

We also evaluated the framework through software trials at several research organisations. All participants successfully used the configuration tools to extend the LIMS framework within an average of 40 minutes. EUDs were also able to easily create basic applications within an average of 25 minutes. The overall feedback was that the framework approach was a useful and efficient way to create adaptable data management applications. More importantly,

participants were able to immediately see how the framework could be applied to their own laboratory data.

**Keywords:** End user development, software flexibility, data management, framework approaches, database evolution

# Acknowledgements

First, I want to acknowledge my supervisors, Clare Churcher and Walt Abell, for their support and guidance. They made significant efforts to provide detailed comments and suggestions for this research. Without them, I could not have completed this thesis.

Secondly, I would like to express my thanks to my external advisor, John McCallum (from New Zealand Institute of Plant & Food Research Ltd), for his contribution to the research. I would also like to acknowledge the support from a FRST sub-contract.

Thirdly, I would like to acknowledge all those who helped me with this research. I wish to thank all participants from: the Department of Applied Computing, Lincoln University; the New Zealand Institute of Plant & Food Research Ltd; the Faculty of Agriculture and Life Sciences, Lincoln University; the IT Department, AgResearch Ltd; Diversity Arrays Technology Pty Ltd, Australia; the Research School of Chemistry, Australian National University; the Department of Environmental Engineering, China University of Mining and Technology; the Department of Psychology, Peking University ; the College of Life Science, Peking University; and the Beijing Institute of Genomics (BIG), Chinese Academy of Sciences.

Special thanks go to Caitriona Cameron for improving my thesis writing skills, and Douglas Broughton for his excellent help in ensuring all services, resources and facilities were available for my study.

Finally, I would like to thank Aimei Xiong and Rong Shang for their encouragement and support.

# Table of contents

<b>Abstract .....</b>	<b>i</b>
<b>Acknowledgements .....</b>	<b>iii</b>
<b>Table of contents .....</b>	<b>iv</b>
<b>List of figures .....</b>	<b>ix</b>
<b>List of tables .....</b>	<b>xi</b>
<b>Chapter 1 Introduction .....</b>	<b>1</b>
1.1 End user development issues .....	1
1.2 Aims and scope of the research.....	1
1.3 Structure of the thesis.....	2
<b>Chapter 2 Flexible software development .....</b>	<b>3</b>
2.1 Flexible software development .....	3
2.1.1 Object Oriented (OO) techniques.....	3
2.1.2 Software components .....	5
2.1.3 Model View Controller (MVC) and Layered architecture.....	6
2.1.4 Software application generator.....	8
2.1.5 Object Oriented frameworks .....	9
2.1.6 Software Patterns.....	10
2.2 Flexible data storage .....	11
2.2.1 Schema evolution .....	11
2.2.2 Metadata model .....	12
2.2.3 Ontology-based system .....	13
2.2.4 Application and database co-evolution .....	13
2.3 Web services .....	14
2.4 Summary .....	15
<b>Chapter 3 End User Development .....</b>	<b>16</b>
3.1 End user development .....	16
3.2 Potential end user development tools.....	17
3.2.1 Spreadsheet.....	17
3.2.2 Reporting tools .....	17
3.2.3 End user databases .....	18
3.2.4 Web development.....	18
3.3 Application customisation and implementation.....	19
3.3.1 Approaches.....	19
3.3.2 Challenges .....	21

3.4	Evolution of end user applications and databases.....	22
3.4.1	Collaborative design to evolve applications.....	22
3.4.2	Database evolution tools for EUDs.....	22
3.4.3	Unresolved challenges.....	23
<b>Chapter 4</b>	<b>Proposed Research .....</b>	<b>24</b>
4.1	Exploratory case study.....	24
4.1.1	In-house software and end user development .....	25
4.1.2	IT professionals .....	27
4.1.3	End user developers .....	28
4.2	Research Methodology.....	29
4.2.1	Problem .....	30
4.2.2	Proposal.....	30
4.2.3	Implementation.....	31
4.2.4	Evaluation.....	31
4.3	Research case study.....	32
4.3.1	NZPFR LIMS .....	32
4.3.2	Problems and development challenges .....	33
4.4	Summary .....	34
<b>Chapter 5</b>	<b>Framework and System Design .....</b>	<b>36</b>
5.1	Review of problems and guidelines .....	36
5.2	Framework Rationale: Middle Way.....	37
5.3	Types of developers .....	39
5.4	Framework Design .....	40
5.4.1	Domain model design.....	41
5.4.2	Framework API.....	43
5.4.3	End User Development toolkit.....	44
5.4.4	End user application development .....	46
5.4.5	Framework configuration tool.....	47
5.5	Summary .....	48
<b>Chapter 6</b>	<b>Framework Implementation .....</b>	<b>50</b>
6.1	Responsibilities of framework providers .....	50
6.1.1	Domain model and persistent data implementation .....	51
6.1.2	API implementation .....	52
6.1.3	Summary .....	53
6.2	LIMS domain model implementation .....	54
6.2.1	Domain model.....	54

6.2.2	Database .....	57
6.2.3	Class and database schema generation .....	58
6.2.4	ORM files .....	59
6.2.5	Constraint files .....	61
6.3	Framework API implementation .....	64
6.3.1	Web service API for PFR .....	64
6.3.2	HibernateValidator .....	65
6.3.3	Hibernate DAO .....	66
6.3.4	Web service API configuration .....	68
6.3.5	Web service API methods .....	69
6.4	Summary .....	72
<b>Chapter 7 Framework development tools .....</b>		<b>73</b>
7.1	Overview of development tools .....	73
7.2	Framework development tools .....	75
7.2.1	Defining the configuration template .....	76
7.2.2	Framework code generator .....	76
7.2.3	Toolkit generator .....	77
7.3	Example code for end user applications .....	79
7.3.1	Inserting data .....	79
7.3.2	Inserting data across tables .....	80
7.3.3	Retrieving and updating data .....	81
7.3.4	Retrieving associated classes .....	82
7.4	End user application generator .....	83
7.4.1	Application generator .....	84
7.4.2	Generated user interface and code .....	84
7.5	Summary .....	86
<b>Chapter 8 Framework User Trials .....</b>		<b>87</b>
8.1	Framework Manager User Trials .....	87
8.1.1	Framework Manager Trial Task 1: Adapting Schema for Existing Samples .....	89
8.1.2	Framework Manager Trial Task 2: Extending Schema for New Samples .....	90
8.1.3	Framework Manager Trail Results .....	91
8.1.4	Interview feedback summary .....	92
8.2	EUD trials .....	94
8.2.1	End User Development Trial Task1: Generating application .....	95
8.2.2	End User Development Trial Task 2: Customising the application .....	96
8.2.3	EUD Trials Results .....	99



8.2.4 Interview feedback summary .....	99
8.3 Summary .....	100
<b>Chapter 9 Framework Evaluation .....</b>	<b>101</b>
9.1 Framework extension .....	101
9.1.1 Add base level and organisational level classes .....	103
9.1.2 Add new constraints .....	105
9.1.3 Extend the web service .....	106
9.1.4 Discussion of framework extension process .....	107
9.2 Framework generalisation .....	108
9.3 Summary .....	109
<b>Chapter 10 Future work .....</b>	<b>111</b>
10.1 Database and web services performance .....	111
10.2 Web service performance .....	112
10.3 System reliability .....	113
10.3.1 Ensuring consistent updates .....	113
10.3.2 Handle conflicting updates .....	114
10.4 Retrieving data .....	115
10.5 System authentication and authorisation .....	117
10.5.1 Authentication .....	117
10.5.2 Authorisation .....	117
10.6 Application generator .....	118
10.7 Different client application platforms .....	118
10.8 More complex data models .....	119
10.9 Summary .....	120
<b>Chapter 11 Conclusions .....</b>	<b>121</b>
11.1 Guidelines and the Middle Way .....	121
11.2 Framework evaluation .....	122
11.3 Final remarks .....	124
<b>References .....</b>	<b>126</b>
<b>Appendix 1: Interview questions for in-house development .....</b>	<b>135</b>
Questions for end user developers .....	135
Questions for professional developers .....	136
<b>Appendix 2: Framework manager instructions .....</b>	<b>137</b>
Introduction .....	137
Instruction A: Creating a new configuration file and database .....	138
Instruction B: Creating the toolkit and prototype applications .....	140
<b>Appendix 3: End user developer instructions .....</b>	<b>141</b>

Instruction A: Creating prototype VB applications for Excel .....	141
Instruction B: Accessing the generated VBA code .....	142
<b>Appendix 4: Framework manager user trial tasks .....</b>	<b>143</b>
Task 1: updating the existing Apple sample.....	143
Task 2: Defining a new Orange sample .....	144
Example Data (worksheet FrameworkTrial2) .....	145
<b>Appendix 5: End user developer trial tasks .....</b>	<b>146</b>
Task 1: Generating an Excel application .....	146
Example Data (ExcelDevelopmentTrial1) .....	146
Task 2: Customising the Excel application .....	147
Example Data (ExcelDevelopmentTrial2) .....	148
<b>Appendix 6: Interview questions for framework manager .....</b>	<b>149</b>
<b>Appendix 7 Interview questions for end user developers .....</b>	<b>150</b>

## List of figures

Figure 2-1 Class diagram for a simplified library management system .....	4
Figure 2-2 Two sub classes inherited from the Item class .....	5
Figure 2-3 Software components – LoanManager .....	6
Figure 2-4 Model View Contoller .....	7
Figure 2-5 An example of multi-layered architecture .....	7
Figure 2-6 Simplified metadata model for managing different types of objects.....	12
Figure 3-1 Imbalance of development tools and schema evolution tools .....	23
Figure 4-1 Specific attributes for different sample types .....	33
Figure 4-2 Specific cardinality constraints.....	34
Figure 5-1 Middle Way .....	37
Figure 5-2 Example of a Middle Way chosen for the plant samples .....	38
Figure 5-3 Framework design.....	40
Figure 5-4 Three level hierarchy shown for the PFR example.....	41
Figure 5-5 Example of a three level hierarchy for vehicle rental .....	43
Figure 5-6 End User Development Toolkit .....	45
Figure 5-7 Data management workflow .....	46
Figure 6-1 Framework Implementation.....	50
Figure 6-2 Three level design.....	55
Figure 6-3 Interactions between specific samples .....	56
Figure 6-4 Database tables .....	58
Figure 6-5 System build script .....	59
Figure 6-6 ORM file for Class Sample.....	59
Figure 6-7 ORM file for the entire Sample hierarchy .....	60
Figure 6-8 Adding a new individual level Apple Sample .....	61
Figure 6-9 Constraint file for base class Sample.....	61
Figure 6-10 Constraint file for the entire Sample hierarchy.....	63
Figure 6-11 Cardinality Constraint.....	64
Figure 6-12 Web service Implementation.....	65
Figure 6-13 workflow of validation logic .....	66
Figure 6-14 Workflow of insert data .....	67
Figure 6-15 object relational mapping.....	67
Figure 6-16 Work flow of data retrieving .....	68
Figure 6-17 Application configuration file.....	69

Figure 6-18 Example code of insertSample .....	70
Figure 7-1 Framework development tools.....	75
Figure 7-2 Framework configuration template.....	76
Figure 7-3 inserting data from spreadsheet into the database .....	79
Figure 7-4 inserting assay objects with associated samples .....	81
Figure 7-5 Example of retrieving a single sample into an Excel spreadsheet.....	82
Figure 7-6 Example of retrieving all samples for further processing .....	82
Figure 7-7 Example of retrieving an assay and its samples .....	83
Figure 7-8 Comparison between the Onion and Apple data entry applications.....	83
Figure 7-9 Application generator .....	84
Figure 7-10 Generated Excel data entry template .....	84
Figure 7-11 Generated code for loading Apple data .....	85
Figure 8-1 Generated client applications.....	90
Figure 8-2 Time spend on trial tasks .....	93
Figure 8-3 Application generator .....	95
Figure 8-4 Generated Excel data entry worksheet.....	96
Figure 8-5 Customised data entry worksheet .....	96
Figure 9-1 Framework base class extension.....	102
Figure 9-2 Example Reagent ORM file (base level) .....	104
Figure 9-3 Adding a new ReagentSet class .....	105
Figure 9-4 Example Reagent ORM file (organisational level).....	105
Figure 9-5 Constraint configuration file.....	106
Figure 9-6 Web service sample code.....	107
Figure 9-7 The framework approach applied to a vehicle rental application domain.....	109
Figure 10-1 Elapsed time to insert 50 sequences records.....	112
Figure 10-2 Transaction saving an Assay record in the database.....	114
Figure 10-3 Example code of retrieving and update data.....	115
Figure 10-4 Example code for retrieving subset of the data.....	116
Figure 10-5 Proposed Queryobject and Criteria class .....	116
Figure 10-6 Use a query object to retrieve records .....	117
Figure 10-7 Assay data management application generator.....	118
Figure 10-8 UI prototype for iPhone .....	119

## List of tables

Table 4-2 In-house development .....	25
Table 4-3 Software evolution issues.....	26
Table 6-1 Summary of the framework components .....	72
Table 8-1 Participants in the framework manager trial .....	89
Table 8-2 Apple sample configuration template provided to participants .....	90
Table 8-3 Participant defined attributes and related validation rules .....	91
Table 8-4 Participant defined configuration file for Task 2 .....	89
Table 8-5 The time spend on trial tasks.....	91
Table 8-6 Customised code for End User Development Task 2 .....	98
Table 8-7 Time spend on trial tasks.....	99
Table 9-2 Tasks needed to include a new entity in the end user applications .....	103
Table 10-1 Performance testing results .....	112

# Chapter 1 Introduction

There are a number of challenges involved with the development of data management applications, especially in small organisations. Small organisations may not have the budget to buy commercial off-the-shelf software nor the expertise to create specialist software from scratch. Open source applications, which can be adapted to meet specific needs, are a possibility. However, there is often a big learning curve to understand how to customise and use these systems effectively.

In many organisations, end user developers (non-professional developers) create small applications for their own use (or that of their immediate colleagues). One of the major disadvantages of end user developed applications (EUDAs) is that they are often focused on meeting immediate needs and not designed to be easily modified or extended in the future.

## 1.1 End user development issues

Scientific domains are a good example of where end user development often occurs. Many small-scale information systems (e.g. spreadsheets and small databases) have been developed by scientists acting as end user developers (EUDs) to manage the specialist data from their research. Basic EUDAs are often developed quickly and simply to meet current needs (e.g. to manage experimental records). However, these applications are often difficult to generalise or adapt when requirements inevitably change.

As part of this study, we visited several research institutions in New Zealand, Australia and China to investigate the issues involved in end user development. We found that different user groups in the same organisation might require similar, but different, database systems to support their research. However, the very specific designs used in most of these systems meant that it was simpler for EUDs to create separate systems when new requirements arose. While this may have met immediate needs, it caused long-term problems with data integration and application maintenance. There is a critical need to help EUDs develop applications that can be modified or extended to meet similar requirements in the same application domain.

## 1.2 Aims and scope of the research

In this thesis, we investigate ways to support EUDs to create flexible data management applications. A key part of this study is how to allow a basic data model to have several co-existing variations to satisfy the requirements of different end user groups in a common

domain. The results of this research should provide both guidelines and software tools to support end user developers to easily create and adapt data management applications. In addition, we will apply the guidelines and approach developed in a case study of a scientific organisation working in the genetic research domain.

### **1.3 Structure of the thesis**

Chapter 2 reviews the literature on software engineering techniques and summarises how these assist professional developers to design flexible applications to support changing requirements. Chapter 3 looks at studies relating to end user development that, in general, does not use sophisticated software engineering techniques. These studies cover approaches to educate and enable EUDs to create flexible applications.

Chapter 4 summarises our exploratory case studies of application development in the research laboratories we visited and discusses common issues that arise in EUDAs. It also describes the proposed research and suggests guidelines for developing flexible EUDAs.

Chapter 5 presents our approach to solving the flexibility issues in EUDAs. This involves cooperation between professional and end user developers to produce a framework based system to enable the easy creation and modification of applications. Chapters 6 and 7 describe the implementation of these ideas at the chosen case study research organisation.

In Chapter 8, we discuss the evaluation of framework via software trials with EUDs. Participants assessed both the concept of the system as well as the software tools created to support it. Chapter 9 contains a self-evaluation that extended the case study framework to explore how well the approach provides flexibility of design.

Chapter 10 discusses framework implementation issues and outlines future work to address these. Finally, the work is summarised and conclusions are drawn in Chapter 11.

## **Chapter 2 Flexible software development**

The Computer Society of the Institute of Electrical and Electronics Engineers (IEEE) (IEEE, 1990) defines software flexibility as “the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.” In order to improve software quality and minimise the effort required for software evolution, software flexibility and extensibility are high priorities when IT professionals design and implement software (Gamma, 1995). It is more cost-effective to spend extra effort on software flexibility and extensibility during the design and implementation stage (Sommerville, 2007) because software changes are more expensive to make after the software is developed. Using good software engineering techniques can maximise flexibility to effectively add new functionality to existing software.

Techniques are used by software engineers promote software flexibilities and adaptabilities. Software flexibility and adaptability are often overlooked by end user developers (non-professional developers); however, software engineering techniques can be also applied to the design and implementation of end user applications. This chapter presents a review of a number of related software engineering techniques. Section 2.1 reports how current software engineering techniques help in the development of flexible and reusable applications. Section 2.2 reviews current approaches to developing flexible data storage in order to ensure effective data persistence. Section 2.3 introduces generator-based development approaches to simplify schema evaluation processes and development efforts. Section 2.4 introduces communication techniques that allow data to be exchanged among different implementation platforms.

### **2.1 Flexible software development**

In order to increase software flexibility, software engineering techniques are used to develop flexible and reusable applications for effectively supporting software extension. We describe some of these techniques in this section.

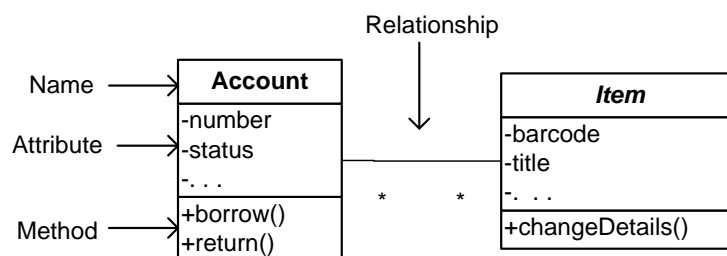
#### **2.1.1 Object Oriented (OO) techniques**

OO techniques are based on the idea of classes, which are templates for creating objects (an object is an instance of a class) (Booch, 1994). A class definition includes attributes and methods: the attributes store the object information and the methods perform data process actions based on the object information. If designed appropriately, it is possible to reuse and



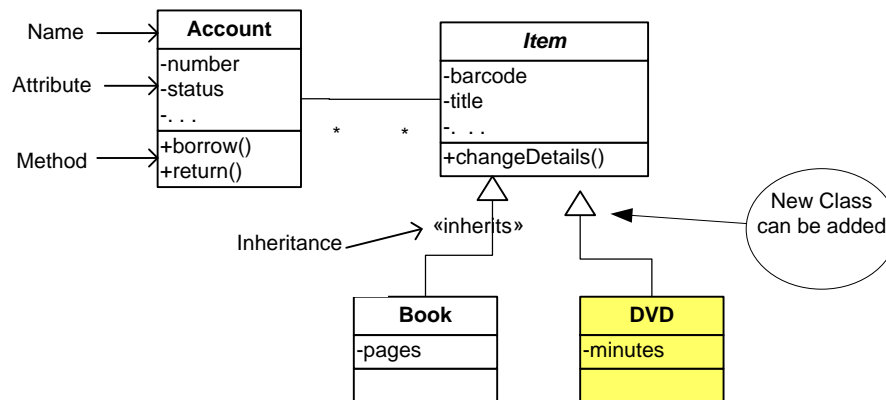
change particular objects to satisfy new requirements without affecting other objects in the system.

Figure 2-1 is a class diagram designed for a simple library system to manage library items and user account information. The library users need an account for borrowing library items, so the library system has account objects and item objects. Each account object has an account number and a status attribute. The class defines borrow and return methods. In Figure 2-1 the rectangles represent the two classes and the association/relationship (the line) between them. A particular account can borrow many library items (the \* at the right under the line), while a particular library item object can also be borrowed by many accounts (the \* at the left under the line). This is referred to as a many-many relationship (during different time periods).



**Figure 2-1 Class diagram for a simplified library management system**

Inheritance is an important OO concept to promote flexibility and reusability. The inheritance concept allows a software system to be “open for extension, closed for modification” (Meyer, 1998). This means that the new software functionalities should be able to be added to the existing systems without changing the existing applications. In Figure 2-2, a Book sub class is inherited from the Item class. This means the book subclass not only has all the attributes and methods of the generic Item but also has additional attributes and methods specific to a book. It is possible to add the additional subclasses based on the new requirements at later stages. A good example is shown in Figure 2-2, where a DVD class is added. This keeps a time (attribute) rather than the number of pages. The new class is quite independent of the existing Book class and does not affect any Book data or applications using the Book objects. OO design, particularly in the use of inheritance, minimises the impacts to the overall system when new requirements are introduced.



**Figure 2-2 Two sub classes inherited from the Item class**

### 2.1.2 Software components

As the number of classes in a software application increases, it can become difficult for developers to understand the whole system. A software component based approach can assist with reducing the complexity and enable the reuse of code. There are many definitions for the software components. In this thesis, a component is considered as a set of classes that provide common functionality through a well defined interface. More importantly, it works with other objects or components in order to perform more specific tasks based on the users' requirements. Well designed component interfaces also allow developers to extend or replace the existing components when new functionality is required (Sommerville, 2007).

A component is larger than an individual class but smaller than a software application (see Figure 2-3). If the developers (component users) understand the interface, they do not worry about the underlying implementation details and can quickly reuse the component in different applications.

The middle part of Figure 2-3 shows a LoanManager component. The LoanManager provides an interface to associate the item objects with user account objects in order to manage the book borrowing application logic. The component can then be used within an application (shown at the top of Figure 2-3) for tasks such as borrowing a book or DVD.

## Application

loanManager.borrowItems (new List (Book0001, DVD0001), Account001)

## Component

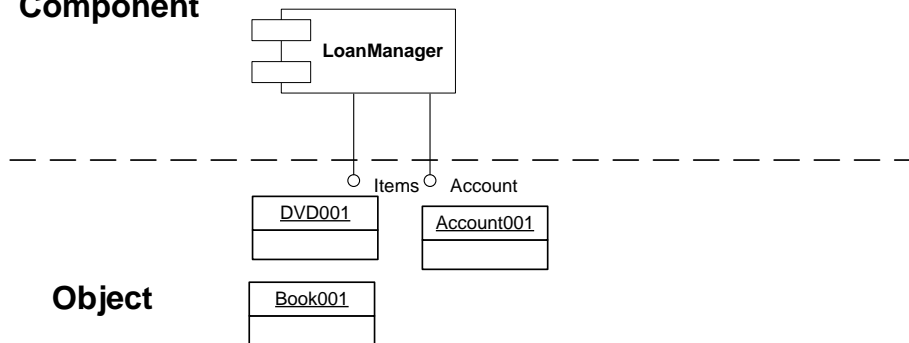


Figure 2-3 Software components – LoanManager

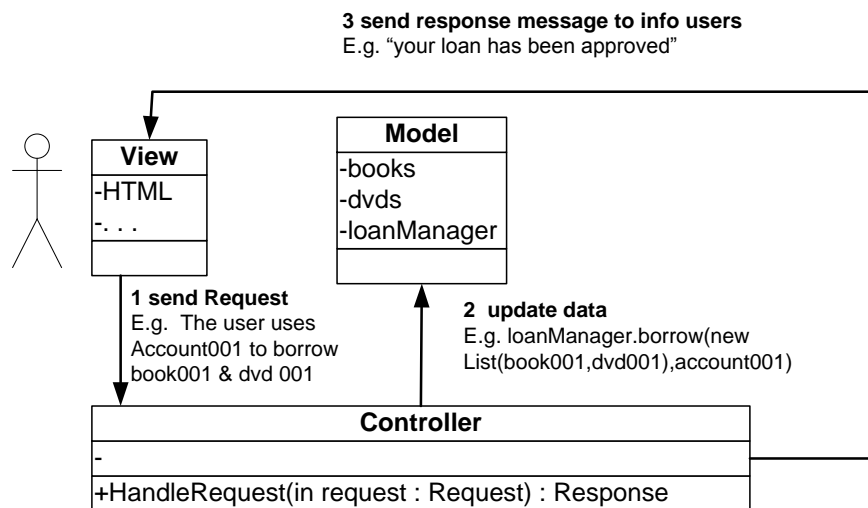
### 2.1.3 Model View Controller (MVC) and Layered architecture

Software application changes are mainly caused by alterations in requirements and environments. Model View Controller and Layered architectures are recognised techniques that make software flexible and maintainable. Both decouple the related functionality into its individual parts. This minimises the impact to the overall applications when a specific individual part changes (Evans, 2003) (Alur, Crupi, & Malks, 2003). The following content briefly describes these two architectures.

- **Model View Controller**

Model View Controller (Krasner & Pope, 1988) is a commonly used software architecture that separates the dependencies between the data model, presentation logic or user interface. An example is shown in Figure 2-4.

- The model includes the domain logic, data and methods (i.e. the Loan manager, Book and DVD objects).
- The view is the presentation logic that produces the UI to enable end users to retrieve and update the domain object data. For example, a HTML table lists all available books for users to borrow while a button is provided to borrow the books.
- The controller handles requests from the view (e.g. borrow a book), updates the domain objects (e.g. using the Loan Manager to assign the books to the user's account) and sends the responses to the view.

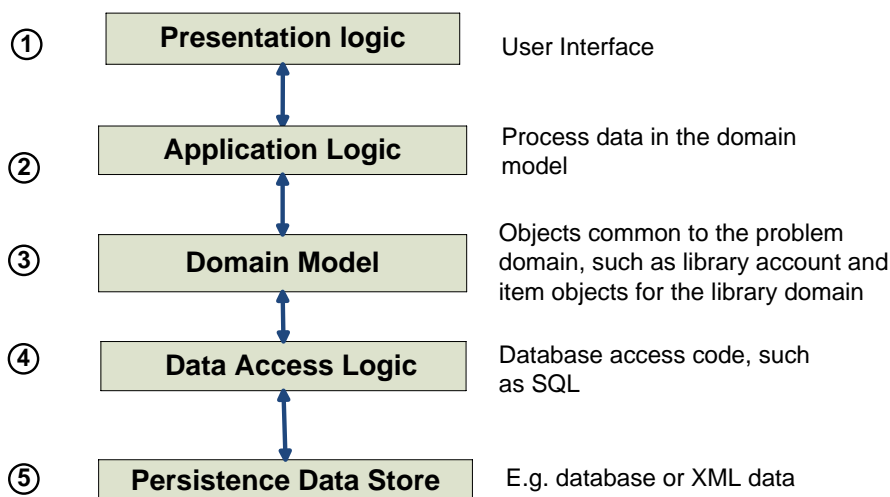


**Figure 2-4 Model View Controller**

This architecture increases system flexibility and reusability. The key goal of MVC is to separate the data model and the presentation logic. This separation supports multiple presentations of the domain data. When the requirements are changed (e.g. need a new windows form UI to represent books), the developers only need to add a new UI (View) and a controller for handling the inputs from the windows UI and querying the domain objects. The domain classes do not need to be altered. The MVC separation makes the software structure more flexible and the code reusable (Fowler, 2003).

- **Multi-layered architecture**

MVC separates the domain logic and the UI logic. The software can be further divided into many small units in order to decouple the complexity of the applications. For example, Figure 2-5 shows a multi-layered architecture based application (Fowler, 2003). In Figure 2-5, the information system is divided into five logical units: presentation, application, domain model, data access and data storage layers.



**Figure 2-5 An example of multi-layered architecture**

- 1. Presentation logic layer:** This is used to present information to users and receive input. (e.g. UI forms, text fields, buttons and data entry fields) For example, a web page or Windows Form can be provided for users to check items on loan in a library system.
- 2. Application logic layer:** This layer encapsulates specific data process tasks. First, this layer could be implemented to include common methods in order to retrieve and manipulate objects data in the domain model based on the requirements for the presentation layer (Fowler, 2003). In the layered architecture, the application layer often encapsulates common data process operations with an Application Programming Interface (API). For example, the LoanManager can be provided as an API to include methods for displaying or borrowing books or DVDs. The API implementation accesses the domain layers in order to use an account object for retrieving all its associated items.
- 3. Domain layer:** The domain layer encapsulates the logic for a particular area of interest such as a library, manufacturer, finance company or healthcare organisation. It consists of fundamental classes, business rules and methods to address specific problems (Evans, 2003). For our library example, this includes the definition of items and accounts as well as the rules that exist in the relationships between them. Other layers, such as application logic layer, data access logic and data storage, are all created based on the domain model. A good design of the domain layer (domain model) will make software applications more extensible and flexible (Fowler, 1997).
- 4. Data access layer:** The domain objects need to be stored in a persistent data store like a database. The data access layer maintains the persistence logic for how this is done. Some object databases, such as JADE (Clarke, 2010), can store the domain objects directly. In contrast, if a relational database is used, there is a mismatch between data represented in the domain object model and the relational data model. The data access layer is used to map between the domain objects and the tables in the relational database.
- 5. Persistence data store:** This is software that ensures domain object information is maintained. For example, databases and XML files are common persistence data stores to keep object information. Developers save and retrieve the domain object information in these data stores via the data access layer.

#### **2.1.4 Software application generator**

In order to accelerate software development and adaptation, application generators help developers to automate skeleton information. Developers define the data model, such as entities/classes, attributes and associations. The generated applications apply MVC (see Section 2.1.3) architecture: a database is generated based on the given data model, an HTML

view is generated to represent the data for end users, and a controller is generated to handle requests from the view as well as manipulate the model. As a result, a skeleton application can be generated to update and retrieve the information in the database. Developers then only need to focus on their own specific application's logic development. For example, Ruby on Rails (RubyonRails, 2009), Django (Django, 2008) and Spring Roo (Alex & Schmidt, 2009) are web application generators to aid in the development of database driven web sites.

Application generators help IT professionals to quickly develop and adapt applications. However, developers still need to design the domain data model in order to generate the rest of the applications. With a poor data model, the generator cannot produce flexible applications. In order to design a useful data model and application logic, developers require a strong knowledge of both the problem domain and of OO techniques. The next section introduces OO frameworks and software patterns that assist IT professionals to design and implement reusable applications.

### **2.1.5 Object Oriented frameworks**

In Section 2.1.1, we discussed how OO facilities, such as inheritance, allow software to adapt to changing situations (we added a DVD class at a later stage without affecting the existing code for Book objects). A framework provides an overall design that can be reused in different, but similar, situations (Fayad & Schmidt, 1997; Oscar & Dennis, 1995). Johnson and Foote said, “A framework is a semi-complete application that can be specialized to produce custom applications”(Johnson & Foote, 1988). This means that the framework not only provides a high level abstract class and interface design but also includes concrete subclasses that can be used directly in an application (Gamma, 1995). In this thesis, we use a framework to mean a comprehensive set of high level reusable abstract classes (or interfaces) and a library of concrete subclasses which developers can use directly in their applications.

In layered architectures, each layer might have its own framework. The frameworks and components are provided for general use and can be applied to many different applications. Large complex software applications are usually developed using many frameworks, such as UI, application and persistence frameworks (Fayad & Schmidt, 1997). For example, developers often reuse UI frameworks, such as Windows Foundation Class (WFC) and Mac OSX Cocoa (Stevenson, 2010) to develop GUI in the presentation layer. The Spring framework (Spring, 2009; Walls & Breidenbach, 2008) or Microsoft Enterprise library (Fenster, 2006; Microsoft, 2011c) can be used to manage application logic for completing common tasks, such as data validation, user authentication, system logging and transaction

management. In order to manage persistence logic in a data store, persistence frameworks like Hibernate and ADO.NET can be used to implement the data access layer to update or retrieve domain objects from the database.

Application, UI and persistence frameworks can be used in application development across many problem domains. However, it is difficult to create a standard data model that is valid across many different domains because, “No one had defined a schema sufficiently broad, deep and flexible enough to support transaction processing on all key business entities and serve as a master superset to all other data models deployed in heterogeneous IT environments” (Oracle, 2010).

Frameworks and commercial off-the-shelf (COTS) components are provided for some problem domains to improve productivity for users performing common tasks; these include Customer Relation Management or Enterprise Resources Planning systems. However, work will be needed to customise the data model and processes to meet individual requirements. For example, ERP components for a university would need to be customised to meet the requirements for student management, researchers and research funding. This customisation would be quite different for a commercial organisation. As a result, although frameworks and components can be provided for the domain model, developers still need to spend considerable time to customise and extend them according to the users’ specific requirements (Sommerville, 2007). Moreover, learning how to use an application framework can require a considerable effort. Even professional developers might need several months to become highly productive with a framework (Fayad & Schmidt, 1997).

### **2.1.6 Software Patterns**

Software patterns are provided to analyse, design and implement software. They provide useful approaches to common problems. Software patterns minimize the software changes for the application itself when new requirements are introduced. Patterns “help make software frameworks suitable for many different applications without redesign” (Gamma, 1995). For example, a factory pattern is a creational pattern that suggests code for creating objects and is easily amended as new types of object are introduced. The factory pattern could be used in our example of a library management application to create different types of items. The pattern allows programming code to be written in such a way that minimises changes when new types of items (e.g. magazines or CDs) are added. The patterns employ OO techniques to guide developers to extend the application functionality by subclassing without modifying the existing application structure.

Understanding and applying software patterns not only makes an application flexible but are also useful for IT professionals when modelling domain problems. In order to design a flexible domain model, IT professionals need to understand the problem domain and discover the key domain concepts and classes. Software patterns, such as data model patterns (Hay, 1996) and analysis patterns (Fowler, 1997) guide IT professionals to develop a high level domain model that can underpin a variety of software requirements within a particular problem domain. For example, the Quantity Pattern represents values for both their amounts and units. If we want to represent the size of a book in our library the Quantity Pattern suggests separating the value and dimension into different classes (i.e. a value of 100 and the number of pages). When we add DVDs we can easily represent their size as a value of 120 and a dimension of minutes. These patterns make the domain model more flexible to meet new requirements without the need for redesigning.

## **2.2 Flexible data storage**

A database schema (or a data model) defines the structure for storing information in a database management system. The schema is the key part of an information system for storing important data. An information system will also have processes for managing the data and carrying out tasks. Providing support for the evolution of a schema (changes to data types, attributes and relationships to meet new requirements) is a challenge, because the schema underpins the rest of the information system, e.g. the UI, data access and process logic (Curino, Moon, Tanca, & Zaniolo, 2008; Fowler & Sadalage, 2004).

### **2.2.1 Schema evolution**

When software requirements are changed, the domain data model and backend database need to be changed as well. As a consequence, all system dependent components must be changed to reflect the changes in the database schema. Roddick (1995) suggests schema modification for database systems should require the minimal changes to the existing database schema and applications (Roddick, 1995).

Many studies have provided solutions and tools for IT professionals to analyse schema evolution and its impact on applications. These tools support database IT professionals to analyse the relational database schema evolution in order to detect possible flaws in the database design (Curino et al., 2008) and evaluate the impacts of schema changes to current applications. Database tools (Hick & Hainaut, 2006) allow IT professionals to modify database schema and migrate data in order to support rapidly evolving databases, such as

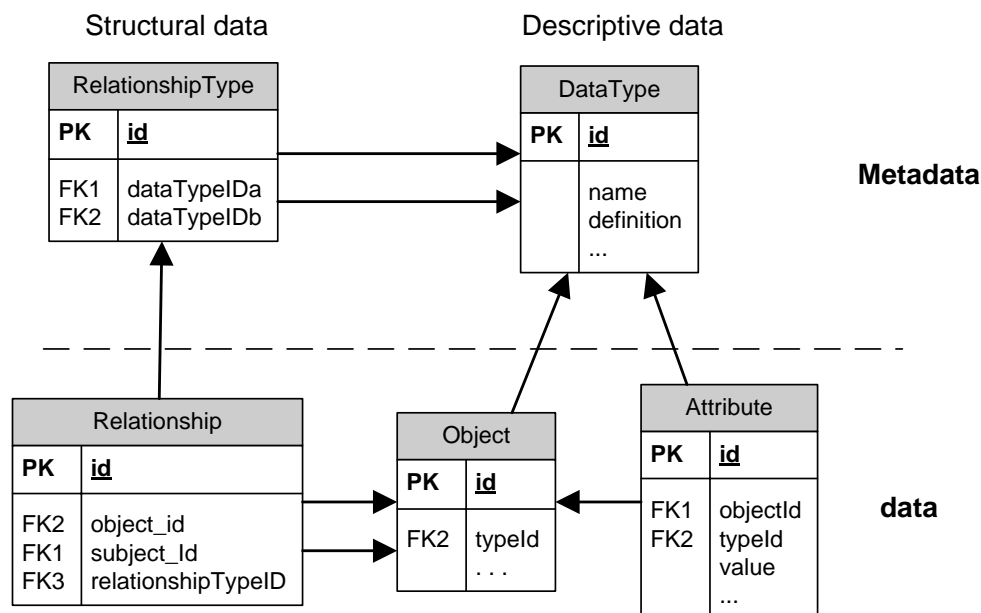


biological databases. Some evolution tools (Carlo Curino 2009) not only help IT professionals to evolve the databases but also allow historical data to be stored in a data archive. Therefore, end users can still access important information from the legacy databases. In addition, many researchers have suggested ways to improve Object Oriented database flexibility and manage schema evolution (Galante, dos Santos, Edelweiss, & Moreira, 2005; Rashid, 2001).

## 2.2.2 Metadata model

Some researchers (Mungall & Emmert, 2007) suggest that “stable schemas must be provided to support rapidly evolving schemas”. To design an evolutionary data schema, a metadata model and a runtime attribute pattern are often applied to the relational database implementation to provide greater flexibility (Wendl et al., 2007) (Diestelkamp & Lundberg, 2000; Mungall & Emmert, 2007) (GMOD, 2008). A metadata model can be implemented as a relational data schema. The purpose is so developers can add, modify or remove entities and attributes without changing other existing entities at runtime.

A metadata model based implementation includes tables to represent the key domain objects and their complex relationships. For example, Figure 2-6 shows the metadata which includes a RelationshipType and a DataType. The DataType table contains data defining the name of objects while the RelationshipType table defines how the objects are related to each other.



**Figure 2-6 Simplified metadata model for managing different types of objects**

The “real” data are stored in the Relationship, Object and Attribute tables. The Object table is used to store any types of data that are defined in the DataType table. The Relationship table stores the relationship between two objects based on the rules defined in the Relationship

Type, such as a relationship between two different types of domain object. The Attribute table stores the values of the attributes, which are defined in the Data Type table.

The benefit is that this design provides flexibility to support requirement changes after the software system is delivered. Returning to our library example, the Object table could be used to store any type of library item (e.g. books, DVDs, journals, CDs). Different types of items have different attributes; therefore, the Attribute table allows developers to add additional attributes. In addition, any new relationship between objects can be also added in the Relationship table via the Relationship Types. For example, some books (e.g. software development books) may include digital content, e.g. a CD with an example code. The design allows us to add the new relationship between a Book and CD object.

### **2.2.3 Ontology-based system**

Ontology is a technique to define a knowledge base for all existing object types, object attributes, and relationships (between individual objects) in a particular problem domain (Liu & Özsu, 2009). An ontology will also include all terms to describe the names of entities attributes and relationships.

In order to use the metadata model based database system according to different requirements, IT professionals need to provide the metadata, e.g. entity types, valid relationship types between individual objects and attribute types (see Section 2.2.2). In an ontology based database system, the ontology is the metadata. The ontology based system can also store a broad range of domain object data; new objects types and attributes types as well as their relationships can be added in the ontology (metadata level) to meet new requirements.

Developing and maintaining ontology is difficult and time consuming. It requires database developers, domain experts and ontology designers. The aim of these databases is to define all existing entities in a problem domain. For our library, for example, all types of items in a library (e.g. journals, music, anthologies) would be defined and include details such as how to catalogue them. An ontological approach is not suitable for small scale projects.

### **2.2.4 Application and database co-evolution**

Database schema evolution tools help database administrators to evaluate the impacts of relational schema changes based on existing OO applications and also help to migrate existing data to the new database schema. Schema evolution tools, metadata models and ontologies help IT professionals to evolve databases and manage data. However, schema evolution does not only affect the database structure but also impacts on all dependent software applications.

This is the focus of many studies in software application evolution, such as reengineering (Demeyer, 2008), system migration (Hainaut, Cleve, Henrard, & Hick, 2008) and architecture evolution (Heckel et al., 2008).

Database evolution and software application evolution are often studied separately (Lin & Neamtiu, 2009), with only a few studies investigating both evolution issues. Lin (2009) indicates that the key problem of schema evolution is due to having different schema for the persistent data and for the applications. An example is an OO application using a relational database to store data. The problems are most evident where the applications have hard coded references to the database schema. Changes to the database and applications must be carefully synchronized. Studies discussing the challenges of applications and databases co-evolution include (Bhattacharya & Neamtiu, 2010).

Some researchers have experimented with automating the adaptation of existing applications to be consistent with a newly evolved database (Cleve, 2010). However, the researchers argued that full automation of the application adaptation is unattainable so some work still needs to be done manually.

## **2.3 Web services**

Web services can improve application component flexibility and reusability. The World Wide Web Consortium (W3C) defines a web service as an industry standard to support distributed communication across diverse platforms via the network. For example, in our library example, the loan manager component could be published as a web service; the service can then be used by .NET applications or by Java applications. Web services play an important role in increasing software flexibility for different types of implementation.

A web service dramatically increases the flexibility of the software to handle new requirements. Many web services are provided for developers to address common requirements with minimal effort, such as language translation and weather forecasts. For example, Google Translate (Google, 2012) is a language translation web service that can be used in our Library system to support multiple languages. It dramatically reduces development effort for providing new functionality. IT professionals only need to reuse these services in their own applications to meet new requirements; they do not need to worry about how to implement them. As a result, web services can make it easier to include new functionality in applications.

## 2.4 Summary

If not designed with flexibility in mind, applications can be expensive or impossible to adapt or extend to meet new requirements. When changes are required such applications need to be redesigned and the redesign has impacts on the entire implementation, including persistence data stores, data access logic and the User. More importantly, all these aspects must be re-implemented.

The goal of flexible software development is to avoid needing an entire system redesign and so minimise the software maintenance costs. In this chapter we have discussed a number of flexible software development techniques that help produce flexible and reusable software applications. These techniques help software engineers design and implement applications that can be reused in a range of applications which address similar domain problems.

However, end user developers (non-professional developers) without extensive software design and implementation skills will have difficulty applying these techniques. The following chapters will consider how to create more flexible applications for end user developers.

## Chapter 3 End User Development

Chapter 3 describes end user development studies and related approaches for improving end user development flexibility. Section 3.1 provides a background of end user development. It also points out that end users often do not have the skills to design flexible databases and applications and find it difficult to evolve (extend and modify) their existing databases and applications to meet new requirements.

Section 3.2 reviews potential end user development tools and Section 3.3 discusses approaches that help end users implement and customise their applications to meet specific needs. Section 3.4 reviews approaches and tools that can help end users evolve their applications and databases. It also highlights that while developing flexible databases and applications is hard, evolving them to meet future requirements can even be harder.

### 3.1 End user development

There are a number of challenges involved in end user development for small organizations that tend not to be faced by large organisations. Large organisations may have the funds to consider commercial off-the-shelf (COTS) software to meet their data management needs. These organisations may also have the budget and resources to develop specific software from scratch. However, small organizations may not be able to afford the costs of purchasing COTS software.

Some reusable applications, such as COTS or open source applications allow customisation, but they may not be suitable for small scale projects. Apart from the cost, there is often a steep learning curve for non-professional developers to understand how to customise and use these systems effectively. As a result, many software development tools are provided to allow end users to act as non-professional developers and create their own applications.

Two types activities were identified by Lieberman, Paternò, and Wulf (2006):

- 1) Parameterisation and customisation allows users to alter the application's behaviour. For example, users could customise what content should be displayed in a web site based on their requirements, for example, customising the UI layout, modifying and deleting web page contents.
- 2) Program creation and modification allows end user developers (EUDs) to develop new applications for supporting their daily tasks or additional functionalities for existing

applications. For example, EUDs could develop new applications from scratch or add code to extend current applications for new requirements.

End user developed applications (EUDAs) are often created to perform a specific task for an individual group of users (Costabile et al., 2003). These are usually developed using one or more of following tools: databases, spreadsheets, templates, scripts and application generators. They may be used to store and process data for specialists groups of users. Due to limitations of time and development expertise and knowledge of suitable software engineering techniques, EUDAs are often very specific and unable to be generalised or extended when requirements inevitably change.

## **3.2 Potential end user development tools**

There are many potential end user development tools available to most organisations. EUDs use these tools to develop applications to meet their requirements.

### **3.2.1 Spreadsheet**

A spreadsheet is one of the most popular tools used by end users to develop data management applications. Spreadsheet products such as Microsoft Excel and Open Office Calc provide built-in programming languages that allow end users to manipulate and analyse data. EUDs use spreadsheets to store data, perform calculations and generate reports. Spreadsheets have been viewed as programming environments for non-professional developers for many years (Nardi, 1993). However, there can be problems with debugging errors and maintaining data integrity for EUDs.

Many studies have focused on these problems. In order to reduce spreadsheet development errors, Panko (2008) suggested providing software tools to help end users find development errors, for example, formula errors. Studies have been carried out to improve spreadsheet maintenance, such as tools for testing and debugging Excel formula errors (Burnett, Rothermel, & Cook, 2006) and improving spreadsheet data integrity (Cunha, Jo, Saraiva, & Visser, 2009).

### **3.2.2 Reporting tools**

There are many software tools to help end users analyse data with less effort. Tools such as Crystal Reports (Peck & ebrary Inc., 2004) and R (RDevelopmentCoreTeam, 2011) assist end users to perform data manipulation, analysis, visualisation and reporting. Most of these tools support different types of data sources, such as databases, spreadsheets and text files. More

importantly, some of the tools allow advanced users to develop additional functionality by programming.

### **3.2.3 End user databases**

Small databases, created with software such as Microsoft Access and Open Office Base, are often used by end user developers to store their daily work data. However, without strong data modelling skills, EUDs often create inflexible data models. These increase the risk of having to be significantly restructured when new types of data are introduced. Over recent decades attempts have been made to help end users improve database design. Approaches include teaching end users how to design databases (Ahrens & Sankar, 1993) and software tools developed to evaluate end user designs. Some researchers (Churcher, McLennan, & McKinnon, 2001) provide a conceptual data model approach to support data model extension and evolution for EUDs.

These approaches help end users to gain database modelling skills rather than assist them to create flexible and extensible databases. As a result, software tools are still required to help end users implement database.

### **3.2.4 Web development**

There are some web development tools suitable for end user developers to create database driven web sites, such as Microsoft Expression Web and Macromedia Dreamweaver.

However, these tools assume that the database design has been completed. Built in wizards are used to select what tables and fields are to be used, e.g. for creating a web form for viewing or entering data. However, these tools only provide predefined application logic, such as adding, removing and editing records. If more complex processing is required, EUDs have to add their own code, for example, by such as using SQL to generate specific reports.

In addition, the database design problems mentioned above can still occur (Rode, Howarth, & Pérez-Quñones, 2003; Rode, Rosson, & Quiñones, 2006). Many end user development studies focus on how to provide tools to create web applications for managing their databases. Although tools can be provided to simplify application development, many researchers report that EUDs do not create flexible databases due to their limited development skills. Segal (2009) points out that end user developers focus on modelling their team or individual data rather than general requirements for similar teams or institutions. This can lead to major problems when requirements change and the applications and databases must be updated.

### **3.3 Application customisation and implementation**

The lack of software flexibility causes problems when software requirements change. Some researchers (EUD-Net, 2003; Klann, Paternò, & Wulf, 2006) suggest that substantial adaptability should be an important goal of all EUDAs. Therefore, the adaptability of applications is an important criterion to measure EUDAs success.

#### **3.3.1 Approaches**

In this section we will discuss studies about how to support EUDs to create customisable applications based on their specific requirements.

- ***Reusable applications and databases***

Some professionally developed applications are designed for a generic purpose based on a specific problem domain and can be customised and extended based on customers' requirements (Sommerville, 2007). There are many good examples, such as payroll or personnel management applications in an Enterprises Resource Planning (ERP) system. These COTS applications are designed as a base system that supports business processes in a general way. The same base system can be reused and customised to create a family of applications for different companies who are carrying the similar data processing. For example, many ERP modules are configurable for different business processes and rules.

Many end users are looking for reusable applications (especially open source versions) and would like to customise the application for their specific needs. However, customisation often requires advanced database and programming skills. This issue been studied for many years (Mackay, 1990). For example,(Cushing et al., 2007) provided domain specific database templates for managing data. However, the study found that the database entities were too generic to be understood by end users. As a result, database customisation still needs to be carried out by IT professionals. In addition, it also has some limitations in supporting the definition of data field constraints and user friendly data entry.

- ***Component based approach***

Component based application development involves component creation and component assembling. Professional developers create components as well as define the interactions between components. End users use composition tools to assemble components together to form applications based on their specific needs. A number of studies have focused on helping end users to understand how components interact and how to assemble them to accomplish



particular tasks, such as assembling a document searching tool (Anders et al., 2004; Wulf, Pipek, & Won, 2008).

The advantage of this approach is that assembling predefined components can make creating end user applications quick. However, a lack of support for end users to extend existing components and integrate external components can limit the usefulness of this approach (Anders et al., 2004).

- ***Software shaping workshop***

Software Shaping Workshops (SSWs) help end users to extend and tailor application functionality. (Costabile et al., 2003) claim that SSWs allow applications to be extended and redesigned by end users after the software is completed by professional developers. SSWs are used in many application domains to help end users enhance application functionality. For example, a toolbox was provided for radiologists to create and share annotations for X-rays (Costabile et al., 2006) . Another example is tools provided for end users to perform basic content management tasks, such as tailoring web site contents and layout for different user groups.(Costabile et al., 2008)

- ***Software modification tool for end users***

Customisable applications can only be customised among predefined options (Letondal, 2006) . It is hard to meet rapidly changing data management requirement, such as new data queries and algorithms. A solution must be provided for non-professional developers to directly modify existing applications to include new data and functionality. In order to help end users to modify existing software functionalities, Letondal (2006) provided an end user development environment that helps users locate and modify methods (in the source code) from the UI.

Both SSWs and end user development environments can help end users customise and extend software applications. However these studies only allowed end users to create applications or customise functionality based on an existing data model. If the underlying data model needs to be evolved, it is still difficult for end users to modify the database and update exiting applications to reflect the database changes.

- ***Service based approaches***

Web services are a platform and language independent way to expose application functionality to other, external, applications. Common functions (e.g. database access) are often implemented as generic web services to simplify end user development. Web services

provide great flexibility for application interaction, data integration and data exchange in different programming environments.

There are some studies about using web services to provide software functionality for extending and customising end user applications. For example, a web service approach is used to provide database access methods in order to help EUDs create customised data entry applications in a familiar development environment, such as Excel (Deng, Abell, Churcher, & McCallum, 2007) (Deng, McCallum, Churcher, & Abell, 2009). Researchers discuss the potential of using web services to exchange data between end user and enterprise applications (Christian et al., 2008). In addition, some web services help end users tailor applications to their specific needs (Nestler, Namoun, & Schill, 2011) and allow workflow customisation (Wajid, Namoun, & Mehandjiev, 2011) .

- ***Web Mash-ups applications***

One of the aspects of Web 2.0 (O'Reilly, 2005) is that it provides a way for end users to participate in web application development, such as creating mash-up applications to combine data from many different sources. For example, Yahoo Pipes (Yahoo, 2011) provide web based development tools for creating mash-up applications. End users can simply select data from different sources (such as web sites, news feeds and web services) and create a new application with the data. Mash-up tools are designed to make it easy for end users to integrate data and filter data without programming skills. Many researchers (Cappiello, Daniel, Matera, Picozzi, & Weiss, 2011; Floch, 2011; Ghiani, Paternò, & Spano, 2011) have provided mash-ups tools for inexperienced users to create applications with minimal effort.

However, mash-up applications mainly provide data retrieval functionality. They do not easily support end users to populate or update their databases.

### **3.3.2 Challenges**

This section covered a number of approaches to help end users to develop and customise applications. However, there are still many difficulties facing EUDs. The biggest challenge is how to help end users understand and use customisation systems. A good example is discussed by Dittrich, Lindeberg and Lundberg (2006), who provided a prototype database with tailoring capabilities that allowed end users to customise data entities. However, the study reported that end users were not confident to make changes to the application and database; they preferred to leave the changes to IT professionals. Even though a system may be designed to be highly flexible, there still needs to be supporting tools as well as proper training for end users to make effective use of the system.

### **3.4 Evolution of end user applications and databases**

End user development studies reviewed provide approaches and tools to support more flexible end user applications. However, end users still need tools to evolve their applications for future needs. Some studies have focused on how to provide adaptable EU applications and databases.

#### **3.4.1 Collaborative design to evolve applications**

Some researchers suggest that end users should work with IT professionals as co-designers in order to evolve applications. Meta-design is an approach that suggests providing software tools to enable collaborative design between IT professionals and end users (G. Fischer, Giaccardi, Ye, Sutcliffe, & Mehandjiev, 2004; G. Fischer, Nakakoji, K., & Ye, Y, 2009). The tools enable all potential users to participate in the software design, such as specifying requirements and objectives as well as contributing ideas and feedback. IT professionals then design the applications based on these requirements. (Koehne & Redmiles, 2011; Zhu, Vaghi, & Barricelli, 2011) Eriksson (2008) provides tools for end users to specify their software tailoring requirements, i.e. what types of customisations to implement (by professional developers) in order to provide a system that allows for future evolution.

This approach is similar to agile software development methods in that end users and IT professionals work together as a team, refining requirements and modifying the system in iterations. However, this does not actually help end users learn how to evolve the applications, because IT professionals need to be involved through the entire development process.

#### **3.4.2 Database evolution tools for EUDs**

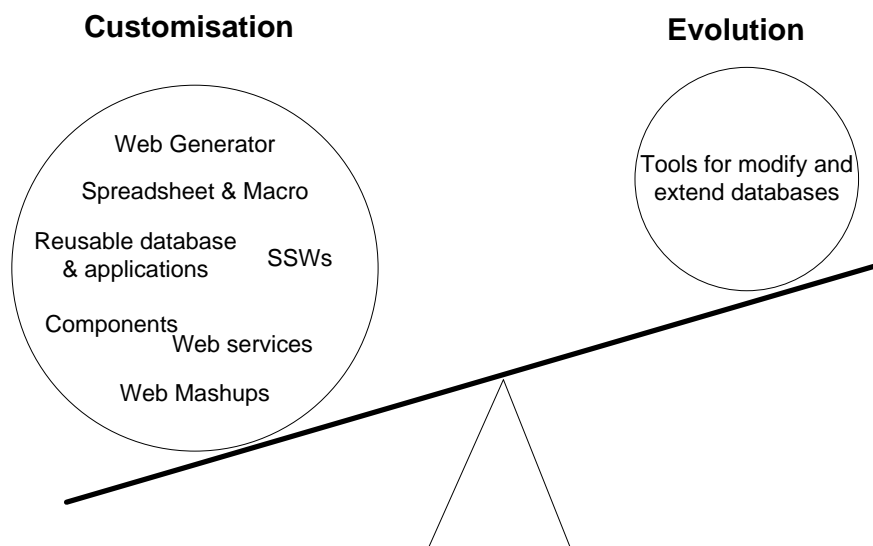
End user development and evolution problems have been investigated for many years. End users need evolution tools to easily extend and modify their databases and applications (Qing & McLeod, 1994). Evolution studies of end user developed databases are comparatively rare and only a few investigated what kind of tools should be provided for extending existing databases. For example, Diestelkamp and Lundberg (2000) provided tools for end users to modify and add database entities but there have been few evaluation and usability studies done for these tools.

These studies overlook how to support co-evolution of databases and applications (i.e. when a database is changed, how the corresponding applications need to be changed). These tools

only support database evolution process. In fact, the impacts of database changes on existing applications have not been resolved in these studies.

### 3.4.3 Unresolved challenges

Figure 3-1 illustrates the imbalance between tools for customising end user applications and tools for evolving these applications. The left side shows that many tools are provided for end users to create and customise applications and databases based on their specific requirements. In contrast, the right side shows that there are few studies that have been undertaken on helping end users to evolve (extend and modify) existing databases. It is a difficult problem to determine how best to support end users to create flexible databases and applications and evolve them when requirements change.



**Figure 3-1 Imbalance of development tools and schema evolution tools**

Further work is required to support EUDs to develop and evolve their applications and databases. In the next chapter, we look at some actual EUDAs examples from scientific research organisations. Later, we will investigate some potential solutions for end users to effectively evolve their applications and databases.

## Chapter 4 Proposed Research

In Chapter 3 we discussed that the lack of software engineering expertise means many end user developed applications (EUDAs) are very specific and unable to be generalised or extended when requirements inevitably changed. In order to investigate the prevalence and impacts of the EUDA issues, we undertook a set of visits to organisations that were developing software for in-house requirements. Section 4.1 summarises our exploratory case study of end user development issues and illustrates general software flexibility and reusability issues. Section 4.2 outlines our research objectives. In Section 4.3, we describe the research case study used to investigate the identified problems and issues in more detail.

### 4.1 Exploratory case study

We visited in seven research institutions in New Zealand, Australia and China (see Table 4-1). These institutions had several laboratories each; we selected laboratories at each place to investigate the issues of in-house and end user development. These laboratories have between 5 and 15 staff who carry out different types of research. All these laboratories produce a large amount of research data every day, thus, all require data management systems.

**Table 4-1 Summary of in-house development**

Institution	Description of laboratory	Size : Number of persons in each laboratory	In-house development	
			Professional development	EUD
A	An environmental engineering laboratory	8		√
B	Two psychology research laboratories	10		√
C	Two genetic research laboratories	15		√
D	Three genetic research laboratories	12		√
E	A genetic research laboratory	14	√	
F	A genetic research laboratory	10	√	
G	A chemistry research laboratory	12	√	

The first four institutions do not have professional development support, so scientists and laboratory technicians create their own applications. IT professionals develop applications for end users in the other three institutions.

We asked end user developers (EUDs) and IT professionals several questions about software development and evolution (detailed questions are listed in Appendix 1). We asked about software development approaches (e.g. developing from scratch or adapting open source software) and software maintenance/evolution questions (e.g. how they handle changing requirements).

#### **4.1.1 In-house software and end user development**

We started by asking whether the organisations used open source applications and what, if any, issues they have. Five institutions (see Table 4-2) have attempted using open source applications but none of them have successfully adapted open source applications for their specific needs.

Institutions with professional developers had tried to tailor open source applications for their specific requirements. However, most made statements such as:

*It requires huge amount of effort to learn how to adapt open source applications, so these applications cannot be adapted to meet all of the specific requirements in a short time.*

As a result, these institutions abandoned trying to adapt open source applications. They all developed applications from scratch using Java, Perl and PHP.

**Table 4-2 In-house development**

<b>Inst.</b>	<b>Type of development</b>	<b>Tried open source</b>	<b>Development tools</b>
A	End user	No	Access /Excel
B	End user	No	Excel
C	End user	Yes	Excel
D	End user	Yes	Access/Excel
E	Professional	Yes	JAVA
F	Professional	Yes	PERL
G	Professional	Yes	PHP

The two institutions undertaking EUD also tried to set up open source applications to manage their data. However, they were not able to do this without professional support. As a result, all EUDs preferred to use Excel and Access tools for their projects, software maintenance and evolution

We also asked questions about software evolution to explore how both groups handled changes to requirements. The general question is how they handled new requirements after the software was developed and any issues caused by changing requirements. All institutions mentioned they had experienced issues, primarily when changes were required to the data model, for example, in adding new entities, changing existing entities and data validation constraints (see Table 4-3).

**Table 4-3 Software evolution issues**

	<b>Inst.</b>	<b>Application</b>	<b>Issues</b>	<b>Current Approaches</b>
<b>EUDs</b>	A	Single table Access database (Access)	Changes to entities/constraints	Update the table Create a different database
	B	Excel data template	Changes to entities	Create a different Excel
	C	Excel data template VBA Macro	Changes to entities	Re-implement Excel Macro
	D	Excel data template VBA Macro Single table database (Access)	Changes to entities/constraints	Redesign Excel data template and Macro Create a different database
<b>IT Professionals</b>	E	JAVA web application Normalised database	Changes to entities/constraints	Update code and database
	F	PERL web Normalised database	Changes to entities/constraints	Update code and database
	G	PHP web Normalised database	Changes to entities/constraints	Update code and database

The IT professionals commented that they were confident to update their applications and databases to meet new requirements. Some EUDs have tried to adapt their own applications

for other colleagues to use. However, the EUDs found it very hard to adapt their specific applications for general use by other colleagues. Most of their applications need to be redesigned or re-implemented when new requirements were introduced. These issues encountered by both groups of developers are further discussed below.

#### **4.1.2 IT professionals**

Most IT professionals realise that if new requirements are introduced that affect the data recorded, they need to update the database schema. All IT professionals said that changing the requirements to the database is the critical issue. The reason is that the application must be updated based on the new database schema. The IT professionals' comments and answers are discussed and summarised below.

All the professional developers are confident that their databases are flexible and can handle many different requirements. They performed detailed requirement analysis and user trials before delivering the database to end users. One professional developer said that:

*We developed a chemistry inventory system which can be adapted for many chemistry research laboratories to manage chemical stores. If new fields, tables or constraints are required to include in the system, he can adapt the system within 10 hours.*

Another IT professional said that:

*Changing the database requires a lot of effort for a large information system, especially if new entities are introduced in the database. We will spend at least a week to update, test and deploy the new system.*

All IT professionals said that it is common for end users to change their requirements after software delivery. As a consequence, the developers have to spend considerable effort on software updates and extension. Three IT professionals recommended that:

*If end users change requirements quite often (e.g. adding/changing entities, attributes and validation rules), we suggest end users accumulate their requirements. We will update the system in a scheduled time (e.g. every quarter each year). We do not have time to immediately update the system when new requirements arise.*

However, this can cause delays in updating applications. For example, an end user biologist in the same institution said that:



*Sometimes we complain that the software update process is too slow, but we can understand that. We have limited IT resources to support software maintenance and extension.*

#### **4.1.3 End user developers**

We asked the EUDs similar questions as the professional developers about software development and evolution. Five EUDs preferred to use a non-normalised data schema, such as a spreadsheet or a single table database. They commented that it is much easier for them to include extra fields into the single database table. However, there are common data management issues for these EUDAs because of using non-normalised databases. These issues and EUD comments are summarised below.

##### ***New data requirements***

EUDs in two laboratories developed small Access databases to manage their data. A single table approach is commonly used even when data for different entities are required. For example, an environmental engineer (acting as an EUD) noted that

*Observation data sets are produced from different lab devices with different information (fields). It is very difficult to design a system to store all types of observation data.*

EUDs often only have basic database skills and encounter difficulties in developing an application to meet varying requirements. The single table solution is often used because EUDs can simply add new fields in the table to handle additional types of data.

However, there are many potential problems with the one table approach. One EUD said:

*It wastes database space and is hard to apply constraints to some of the fields which are not relevant to all the different types of data. Therefore, end users have to check the validity of the data entered because the system does not do this.*

##### ***Changes to the entities***

Changes to existing entities are a major problem that happens in all institutions. While some EUDs were able to develop databases and applications to perform specific tasks, these proved difficult to adapt for other colleagues who have slightly different requirements.

For example, a laboratory technician said that he developed an Excel data template and VBA code to manage and analyse the data for his laboratory. However, most of the VBA code needs to be updated when new data columns are added to the template. As a result, some laboratory staff commented that:

*We are still looking for a flexible database system which can be adapted by the non-professional developers to meet our specific needs.*

It was common that different users in the same institution required similar but slightly different data entities, attributes and constraints. In order to adapt an existing application for these specific needs, the data model usually needs to be amended. EUDs told us that:

*It is very hard to extend an existing system to handle multiple requirements, so we simply develop different systems to meet the new requirements. For example, some laboratories use a different database to manage the data for each of their projects.*

These database design problems occur in many institutions that deal with varying data requirements for researchers in the same general domain.

## **4.2 Research Methodology**

If not designed with flexibility in mind, applications can be expensive or impossible to update, extend or adapt. In practice, EUDAs often need to be completely redesigned to allow for extensions and adaptations. Unfortunately, application redesign can impact on the entire implementation, including data storage, data access logic and the user interface (UI). More importantly, this may all have to be repeated if requirements change again in the future. As a result, flexibility and adaptability is a critical issue of EUDAs.

An approach to resolve this sort of problem is the Design Research Methodology (Hevner, March, Park, & Ram, 2004) . This methodology has five stages which are:

1. Identification of a problem
2. Develop a proposal for an approach to resolve the problem
3. Outline a potential implementation of the proposed approach
4. Evaluation of the approach
5. Draw conclusions the effectiveness of the approach

The first four stages are summarised in the following sections below.

### 4.2.1 Problem

Current studies have looked at how to improve the flexibility and quality of end user applications. Chapter 3 discussed various approaches to allow an end user to:

- Customise and modify applications to meet a user's specific needs, such as software shaping workshops (Costabile et al., 2003) and tools for end users to modify existing database queries and operations (Letondal et al., 2006).
- Develop new applications using predefined software libraries, such as software components (Wulf et al., 2008) and web service based approaches (Christian et al., 2008, Deng et al., 2007).

However these approaches only allow end users to create applications or customise functionality based on an existing data model. If the underlying data model needs to be evolved, it is still difficult for end users to modify the database and update existing applications to reflect the database changes.

Compared to customisation, database evolution solutions are relatively rare. It is hard to support EUDs to evolve their database designs as they have usually been created to solve very specific problems rather than address data management requirements across an organisation. It is often impossible to evolve these databases for new requirements.

Although application design is reusable; it needs adequate supporting tools for users to adapt the databases and applications. Chapter 3 described how some researchers provided a prototype database with tailoring capabilities that allowed end users to customise data entities. However, evaluation studies show that end users do not feel confident to use the tools to update their applications. Improving the tools available and increasing the ability and confidence of EUDs to use them is critical to supporting flexible development.

### 4.2.2 Proposal

In order to assist EUDs to create more flexible and reusable databases and applications, it would be useful to develop a way to assist end users to translate specific requirements into a more generic design. We will look at approaches to provide support for EUDs to create applications that are flexible and extendable. An ideal solution to the flexibility problem should meet the following guidelines:

1. Have a data model that allows for modifications and extensions within a specified domain (e.g. recording of experimental data)

2. Allow end users to easily manage their data
3. Allow EUDs to easily create and extend applications for different requirements within the same domain
4. Allow modifications and extensions without affecting existing data and applications

In order to meet the guidelines, we are going to

- Investigate a domain framework approach (co-designed by professionals and end users) which will make database evolution and extensions within a domain easier.
- Provide simple mechanisms for knowledgeable end users to specify extensions and tools to automate the process.
- Provide easy to use libraries and application generation tools for EUDs to create and extend their applications.

#### **4.2.3 Implementation**

We will implement the proposed framework approach for a Laboratory Information Management System at Plant and Food Research (see details in Section 4.3). The proposed framework includes:

- An object oriented domain model with an associated database
- A web service API to expose common data persistence and validation logic
- A text based configuration templates to allow end users to customise and extend the model with minimal effort
- Development tools for end users to generate client applications

With the framework EUDs should be able to evolve their databases and applications when data requirements change.

#### **4.2.4 Evaluation**

We will carry out software trials to test how well the framework prototype supports EUDs to modify and extend databases and applications. The trial participants will include advanced EUDs who can extend the data model and EUDs who will create applications based on the data model. We will consider the trials successful if participants can correctly extend or modify existing databases and applications within a few hours. In addition, we will also look

at extending the framework for the domain to evaluate how much flexibility the approach provides and what potential impacts this would have on existing applications.

### **4.3 Research case study**

The New Zealand Plant & Food Research Institute (NZPFR) manages different types of plant and experimental data for different laboratories. It is representative of the other research institutions we visited.

NZPFR carries out Polymerase Chain Reactions (PCRs) experiments on crops such as onions, peas and potatoes. With an increase in experimental data, it has become necessary to have a Laboratory Information Management System (LIMS) for managing the data and automating the experimental workflow, such as setting up assays, generating instructions for instruments and recording experimental results. The current LIMS includes an Excel front end application and a back end database for storing laboratory research data for an onion research group.

NZPFR has attempted both the adaptation of open source applications and “from scratch” end user development. Initially, they looked at open source LIMS applications, such as Germinate (Lee et al., 2005), CaLIMS (NCICB, 2002, 2008) and AGL-LIMS (Jayashree et al., 2006). However, adapting these to NZPFR’s specific needs required considerable database and object oriented development skills. More importantly, NZPFR professional developers reported taking several months to master the complex models and code. As an alternative, EUDs at NZPFR have designed their own LIMS application.

#### **4.3.1 NZPFR LIMS**

Initially, EUDs in the Onion Research Group designed and implemented a database to manage their inventory of onion samples. They later added extra tables for the experiment setup details and results.

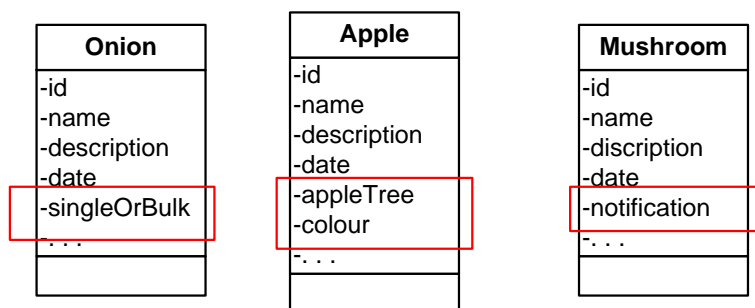
An Excel based front-end LIMS was created by the EUDs (with support from IT professionals) to manage assay (experiment) results and related data in the extended onion database. This included information about samples and primers (a type of reagent used in the experiments) for a particular assay set up, and specific assay parameters (such as temperature and volume). The LIMS application then generated instructions for a laboratory robot to prepare the sample material and reagents for the assays. After the experiment was completed, the application allowed users to record the assay results in the database.

The prototype application was created specifically for the Onion Research Group. For these original users, the application was successful in helping them efficiently manage samples, execute experiment workflow and record results. It was not long before other research groups expressed an interest in using the application to manage their own laboratory data. These groups requested additional data entities, fields and procedures, which would have required adding tables to the database and redesigning the front-end application.

#### 4.3.2 Problems and development challenges

Although the original database and application were flexible enough to manage samples and assays for the Onion Research Group, it was difficult to adapt to similar data for other research groups (e.g. for Mushroom or Apple data). The challenges faced by the PFR EUDs are summarised below:

1. **Additional entities and relationships:** Some research groups needed new entities (e.g. a specific type of reagent called a probe). Additional relationships were also required between the new and existing entities (e.g. between entity Probe and Assay).
2. **Different attributes:** Even for the same entity (e.g. Sample), different groups required different attributes for their specific requirements. Examples are shown in Figure 4-1.

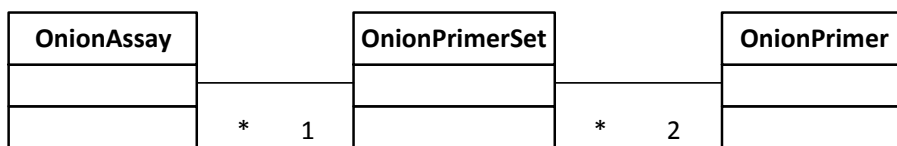


**Figure 4-1 Specific attributes for different sample types**

3. **Different validation rules:** Different groups required different rules to validate the same attributes. For example, Onion samples require that the name attribute starts with a specific prefix while Apple sample name have a fixed length. This means that validation rules have to be customised for each group.
4. **Cardinality constraints:** The limits on relationships between different entities can be different for different groups. For example, Onion assays are performed with a predefined set of two primers (see Figure 4-2). Other research groups require more primers to test an assay. This means that the original relationship between assay and primer needs to be changed to a more generic many-to-many relationship. While this

could have been implemented in the initial design, it wasn't required for the original Onion database. Once the database was implemented it was difficult to change the cardinality without affecting the database schema and front-end application.

#### Specific cardinality for onions samples



#### More generic cardinality for all samples



**Figure 4-2 Specific cardinality constraints**

Although the many-to-many approach provides a more generic cardinality, it does not support the original Onion Group's need for primer sets. A more flexible solution must be provided in order to accommodate those who use primer sets and those that do not, as well as to constrain the number of primers used for specific assays (e.g. a Mushroom assay must use three primers). Designing (and evolving) a schema to meet a wide variety of requirements is beyond EUD skills.

All the changes in requirements described above meant that the data model must be amended. This, in turn, affects the entire application, including the UI and database access code. Therefore, EUDs often find it much easier to create a new database and set of applications when confronted with new requirements. At the time we visited, the Apple and Mushroom Research Groups had each developed separate databases to manage their data. Creating different (but similar) databases for different user groups are common in research institutions. Not only can this waste a lot of time but also it makes data integration across the groups virtually impossible.

## 4.4 Summary

The literature review (Chapter 3), interviews and NZ PFR case study all illustrate that specialised EUDAs are often difficult to generalise or adapt as requirements change. In order to avoid the fragmentation and wasted effort from developing multiple databases to meet

slightly different requirements, a solution should be provided to help EUDs create more flexible applications.

In order to address these issues, we suggested guidelines for flexible EUDAs. We propose to devise a development method to support those guidelines and implement this for our case study (a LIMS system at NZPFR). Chapter 5 will investigate a framework approach to meet the guidelines. Chapter 6 and 7 will discuss the details of the framework implementation for our case study. Chapter 8 and 9 will report on an evaluation of the framework and possible extensions. Chapter 10 will look at future work and Chapter 11 will provide our conclusions.



## Chapter 5 Framework and System Design

Chapter 3 described the problems of end user developed applications (EUDAs) that are common in many organisations. A case study (the NZPFR database) was used to identify the difficulties of adapting specific EUDAs for wider use. In this chapter, we investigate an application framework solution to help end user developers (EUDs) develop more flexible and reusable data management applications.

### 5.1 Review of problems and guidelines

Flexibility is one of the biggest issues with EUDAs. Specialised EUDAs can often be designed and developed quickly and simply to meet specific needs. However, these applications are often difficult to generalise or adapt when requirements inevitably change. Scientific domains are a good example of end user development activities. Many small-scale information systems (e.g. spreadsheets and small end user databases) have been developed by end users (e.g. scientists and laboratory technicians) to manage their specialist data for personal use.

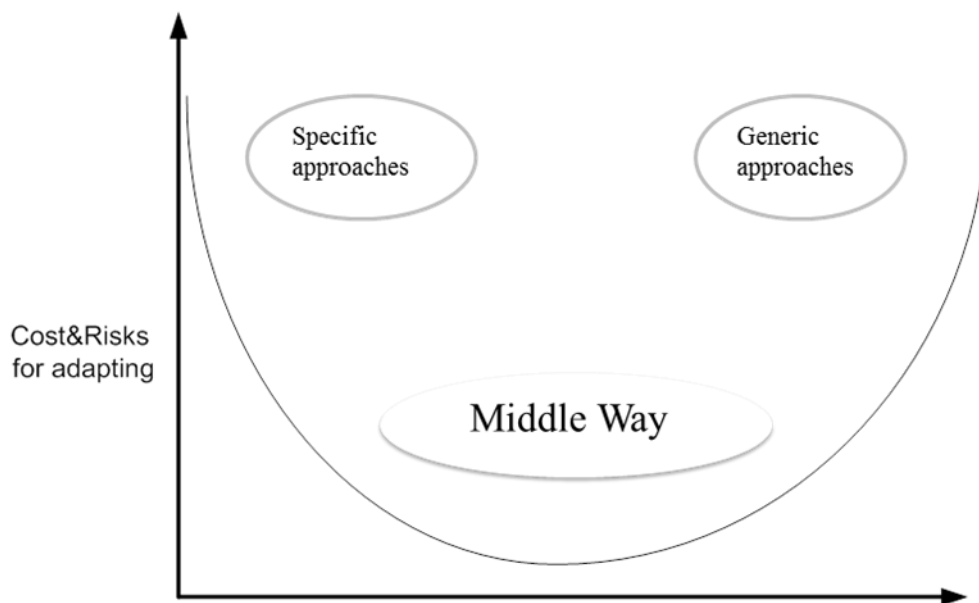
We visited 10 research organisations to investigate their flexibility issues with end user databases. We found that different user groups in the same organisation might require similar, but different data entities, attributes, relationships and constraints. This usually means the data model needs to be amended and end users often find it simpler to develop separate systems to meet the new requirements. This eventually causes problems with long term data integration and maintenance. A good example of this is the PFR database discussed in Chapter 3. A critical issue is how existing EUDAs can be reused to meet other similar requirements in the same application domain.

We investigate a framework solution to support EUDs in developing flexible data management systems. The key point of the guidelines (in Section 4.2.2) is that the solutions should have a data model designed for a specific domain that allows for modifications and extensions without affecting existing data. EUDs also need to be able to customise and extend the data model without breaking the structure and code of the existing applications. Another important guideline is to help EUDs create their own applications to manage their data. We look at what supporting tools should be provided for EUDs to allow them to easily create applications to manage their data.

## 5.2 Framework Rationale: Middle Way

Figure 5-1 illustrates the cost and risks for specific and generic development approaches. At the two extremes we have:

- **Specific:** A cheap and quickly developed application that meets the immediate needs of an individual but is unlikely to be amenable to adaptation.
- **Generic:** An application that can be adapted but will be time consuming and require considerable skills to develop, adapt and extend.

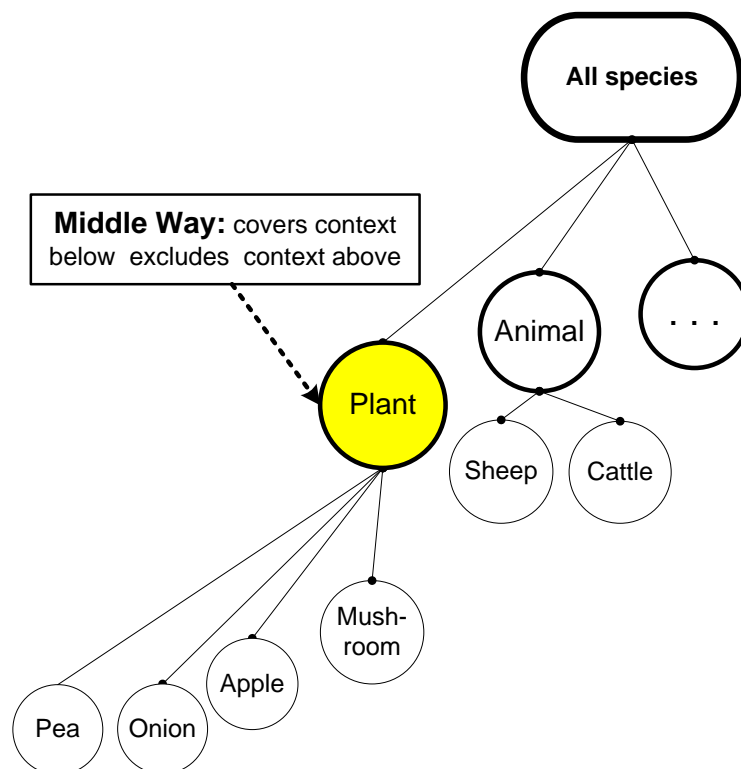


**Figure 5-1 Middle Way**

Highly specific applications can be developed quickly but can be very hard to adapt in the short term. This can mean users with similar problems may find it easier to develop a completely new database and application. However, this is not desirable for two reasons: 1) unnecessary repetition of development work, and 2) the fact that similar data is stored between two separate systems. This will cause long term management and data integration problems. In contrast, generic applications require a large quantity of resources to develop, and may not be affordable for small user groups. There can also be a steep learning curve to master how to adapt the generic applications to meet specific needs.

**Middle Way:** We suggest that the IT professionals work with EUDs to find a compromise between these two extremes (see Figure 5-1) in an early stage of the development (Deng, Churcher, Abell, & McCallum, 2011). The Middle Way is not a fixed point; it could be a

dynamic point in a particular range depending on the project scale, local context and level of application flexibility. As an illustration, we consider a database for storing plant samples in a Laboratory Information Management System (LIMS). A specific approach would be to store single plant sample data, such as a onion. In contrast, a generic approach could be a LIMS database for storing all species (e.g. a generic relevant database). Neither approach is desirable for end users. If EUDs would like to create a flexible application that can be reused to manage onion, mushroom, pea, potato and tomato data, IT professionals need to help them identify the Middle Way point, based on the plants of interest to the organisation.



**Figure 5-2 Example of a Middle Way chosen for the plant samples**

An appropriate Middle Way point should include all domain objects/entities of likely interest but exclude more generic objects/entities that are unlikely to be needed. A framework designed at the appropriate level will then only need small changes to be adapted for different specialist objects/ entities.

Figure 5-2 illustrates a Middle Way that covers all plant samples but excludes other species. A domain specific framework designed for this Middle Way point is likely to be suitable for a large number of laboratories dealing with plant data. Consequently, a very specific onion application might only need a minor change in design to make it available to colleagues working with mushrooms. This same method could be also applied to other application domains, such as a healthcare or car rental domains.

## 5.3 Types of developers

Our approach recommends providing a minimal application framework to support end user applications in a specific application domain. There are three types of developers involved in the system. This section describes the roles of each type of developer:

- **Framework provider**

Framework providers are IT professionals who will work with EUDs to find a compromise Middle Way between a very specific design and a very generic approach. A framework designed at the appropriate level will then only need few changes when adapted for different specialist data in a specific domain.

The framework should include a reusable domain model that can be easily extended for specific needs of a range of EUDs. The domain model includes the most important domain object abstract base classes with minimal attributes and validation rules, for example, class Sample and class Experiments for a genetic laboratory domain, or a class Vehicle and class Rental for a car rental domain. To persist the state of domain objects in the model, a corresponding data store needs to be implemented to represent the domain model. We also suggest that framework providers develop appropriate framework configuration tools to enable framework managers to easily reuse and extend the model to meet individual end users' needs.

The data access logic could be complex for EUDs, so framework providers should develop a generic framework application programming interface (API) that includes common data access and data validation methods. In order to make sure EUDs can effortlessly include the API methods in their commonly used applications (e.g. an Excel spreadsheet) a well-designed end user development toolkit should be provided to map the API methods into languages appropriate for the user (e.g. VBA). Once the toolkits are in place, EUDs only need to concentrate on the development of the user interface that will manage user input and system output between the interface and toolkit methods.

- **Framework managers**

With the framework configuration tools developed by the framework provider, framework managers (EUDs or in-house IT professionals) can extend the framework by subclassing the abstract base classes in order to meet the specific needs of a range of EUDAs. For example, adding a specific onion subclasses to meet specific onion object attributes and rules.

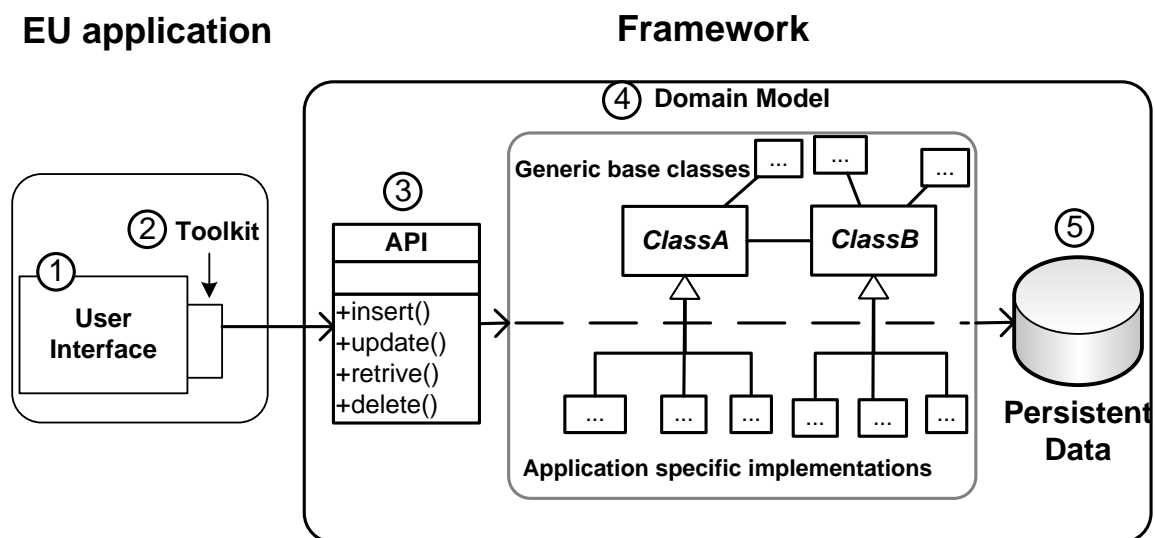
- **End user developers**

EUDs are able to use the development toolkits to develop their specific applications. Since the complex design parameters and persistence logic are encapsulated in the domain model and API, EUDs do not need to know how to design a flexible data store and write complex code to manipulate the data. By using the toolkits, EUDs can write only a few lines of code to include the data management functionality in their commonly used applications, such as Excel spreadsheets.

More importantly, the proposed framework approach does not require EUDs to change their programming habits or learn new software design skills. By using the framework they are able to focus on their specific problems rather than solving a generic problem for all related users. However, the resulting systems will be flexible enough to handle the additional requirements for a variety of end users.

## 5.4 Framework Design

The proposed system includes the framework and end user applications (see Figure 5-3). The framework includes Part 4, the Domain model base classes; Part 5, the corresponding data store, such as a relational database; and Part 3, the framework Application Programming Interface (API). In addition, the end user development toolkits in Part 2 are provided to simplify end user application development. EUDs only need to provide the user interface logic development in Part 1.

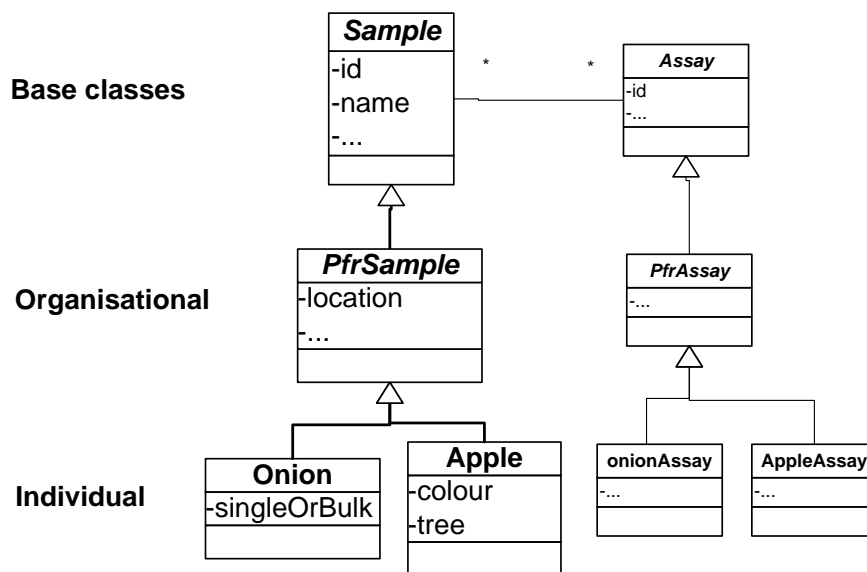


**Figure 5-3 Framework design**

The following sections describe each part of the framework in more detail:

### 5.4.1 Domain model design

The framework includes an abstraction of the domain model that can be implemented for specific needs of a range of EUDs. We propose to represent the domain model as a three level class hierarchy: base abstract classes, organisational classes and individual classes. The domain model will be represented in a data store, such as a relational database, to provide persistence for the state of the domain objects. In Figure 5-4 we illustrate this with an example of a domain model designed to store samples and assays (experiments) in a database for the PFR example (described in chapter 3)



**Figure 5-4 Three level hierarchy shown for the PFR example**

- **Base Classes:** These are developed by the framework providers. The base abstract classes represent collaborations and responsibilities for the fundamental domain objects/entities that are required. Framework providers should supply base classes that only have minimum attributes (e.g. Id and Name), minimum validation rules (such as object id and name are required) and generic relationships between classes (e.g. many-to-many).

The “Open for extension, closed for modification” principle (Meyer, 1998) is applied to the framework design. The framework manager is not able to modify the base classes. In order to meet specific requirements, base classes can be extended by subclassing in the lower levels: Organisational and Individual. The lower level classes inherit attributes, rules and associations from the base classes and can be customised to meet specific needs.

We propose to centralise the validation code in the API layer. The API is able to apply the validation rules defined in the domain model to check the domain objects.

- **Organisation Level:** The second level is the organisational level abstract classes. The organisational level classes not only inherit attributes, rules, associations from the base classes but also centralise common attributes and validation rules for all user groups/departments within a particular organisation.

Framework managers should work with the framework provider to customise organisational level classes and provide attributes common to the whole organisation. For example, the class PfrSample is an example of an organisational level class (see Figure 5-4). It includes a plant location attribute that is common in all sample records at PFR.

The validation rules defined at this level will be inherited by all individual level concrete classes, which make it possible to control and manage data from an organisational aspect. This level would be developed by the framework provider and then be maintained by the framework managers with the framework configuration tools provided.

- **Individual Level:** The third level is the individual level concrete classes. Since the base classes and organisational classes incorporate the most important design elements, the individual level only requires specific attributes and rules to meet the individual end user's requirements. In our example a singleOrBulk attribute has been added for the Onion class. The framework provider will provide an example individual class and then additional classes at this level can be developed and maintained by the framework manager using the configuration tools.

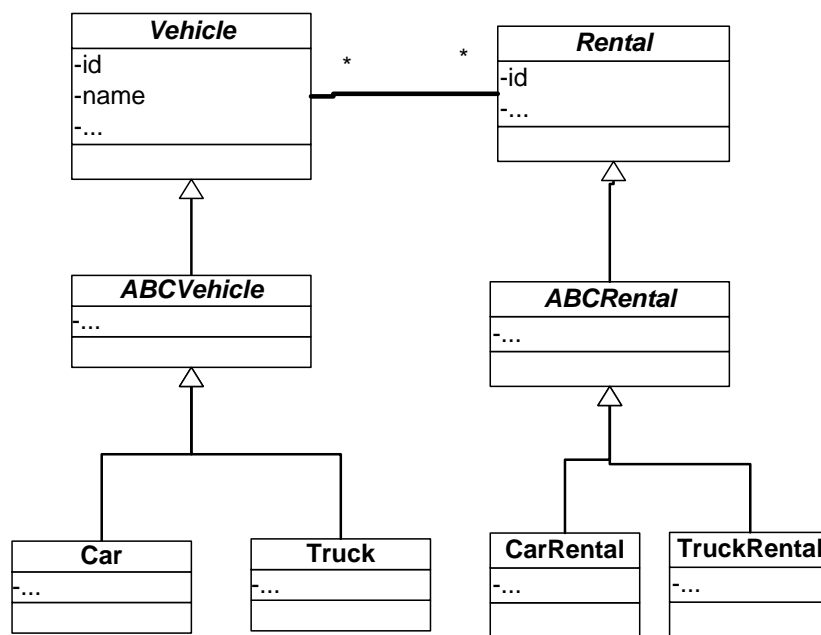
Figure 5-4 shows a many-to-many relationship between the Sample and Assay base classes. This design is proposed to maximise flexibility in the relationships between the base classes. The actual cardinalities of the relationship can be specified for the individual class objects. For example, some types of assays test only one sample (i.e. a one-to-many relationship). In this case, relationship validation rules can be added to constrain the inherited many-to-many relationship from the upper level. In order to implement association relationships between the Assay and Sample classes, the Assay class needs to include a mySamples collection attribute that stores all samples associated with a particular assay object. However, a validation rule can be applied at

the individual level to constrain the number of samples that can be used in a particular type of assay.

It is clear that this framework design provides support for non-technical people to extend the data model to meet their needs by adding new concrete classes. The framework predefines the high-level structure for the application domain, so framework managers only need to use the framework configuration tools (see Section 5.4.5) to amend or extend the example class to match their specific needs.

The framework approach can be applied to different application domains. The example above shows how the framework supports the management of samples and experimental data in the genetic research laboratory domain. The same framework approach can be applied to other application domains. For example, a domain model could be designed for the vehicle rental application domain to support managing car or truck rental information (see Figure 5-5).

Another example might be a domain model for the health care domain to manage patients and their observational information.



**Figure 5-5 Example of a three level hierarchy for vehicle rental**

## 5.4.2 Framework API

The data access logic can be difficult for EUDs, because it includes complex data access and validation codes. Therefore, framework providers should develop a unified API to facilitate data access and validation logic:



- **Data access logic:** It is important to encapsulate the domain model and the underlying data store details. The API assists EUDs to update and retrieve domain objects in the persistent data store. When using the API methods, the EUDs do not have to understand the details, for example, how the domain objects are mapped onto appropriate tables in a relational database. This simplifies end user application development.

The API treats any objects inherited from the same base class in the same way, greatly simplifying the logic. For example, an insertSample method takes a sample object (from the base class) as an input parameter. This means individual level sample objects (such as Onion and Apple) are able to be passed to the API. The API will check the individual level object types and store them appropriately in the persistent data store. For example, if using a relational database, each individual level class will have its own overriding methods to map the object to the appropriate table.

More importantly, as individual level classes are added to meet specific requirements, the API method signature (method name and number and types of parameters) will not change. Therefore, all existing applications stay the same even if new domain objects are introduced.

In addition, encapsulating the data management logic in the API prevents internal database changes (such as migrating to a different database product) from affecting existing applications. Such changes can be handled with changes to the API.

- **Data validation logic** We propose to centralise validation code in the API. Checks can be made for invalid attribute values based on the predefined rules in the domain model. The API can also check reference collection attributes, such as the mySample attribute in the Assay objects, to ensure cardinality rules are adhered to. Any invalid input data is detected and the method will pass back an error message.

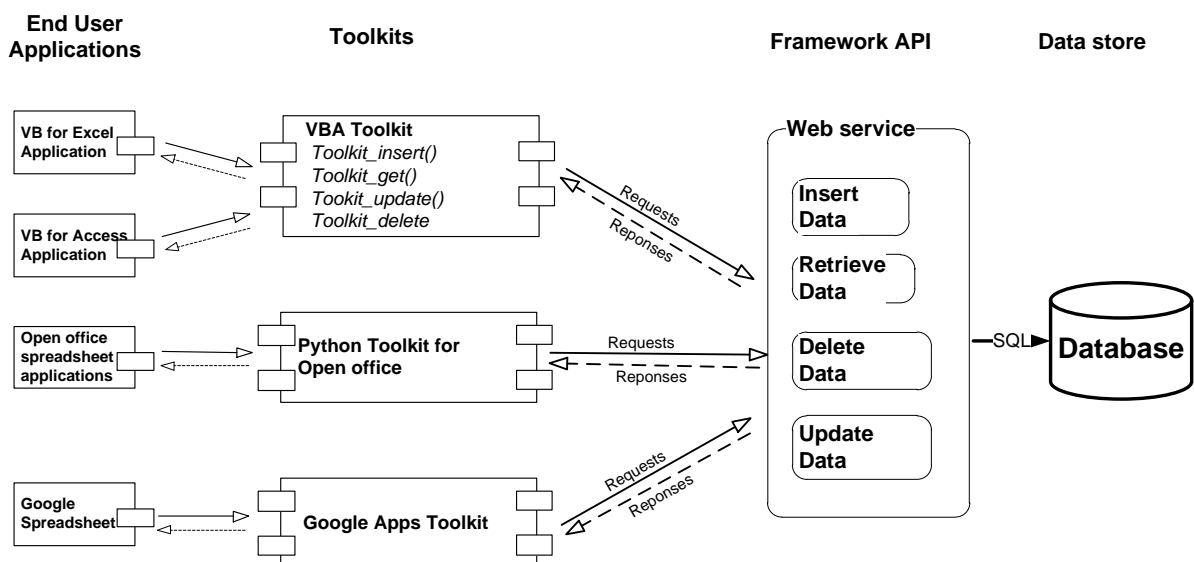
The API makes development of final end user applications simpler because it avoids encoding the common application logic in each separate end user application. Individual end user applications only need to include the code for the user interface logic.

### 5.4.3 End User Development toolkit

Without help, it can be difficult for EUDs to invoke the remote API methods (deployed in a server) from their commonly used applications or familiar development environment. The EUDs would need to have expertise in remote method invocation mechanisms, such as XML web services. In order to help EUDs to interact with the API, we recommend that framework

providers develop toolkits to simplify API invocation. A toolkit will include a set of native methods to call the remote API methods, such as inserting and retrieving data methods. With an appropriate toolkit, a EUD just needs to provide an interface and a small amount of code to interact with the toolkit, e.g. provide an input form and send the values to the toolkit.

Figure 5-6 illustrates an example that includes toolkits developed for end user development environments. For example, it shows a VBA toolkit that provides methods to facilitate passing data between Excel and an XML web service API. EUDs do not need to understand how to use the web service to manipulate XML requests and responses. The EUDs only need to provide the appropriate data from their Excel spreadsheet as parameters to the VBA toolkit insert method. This approach considerably reduces the expertise and effort required to develop end user applications.



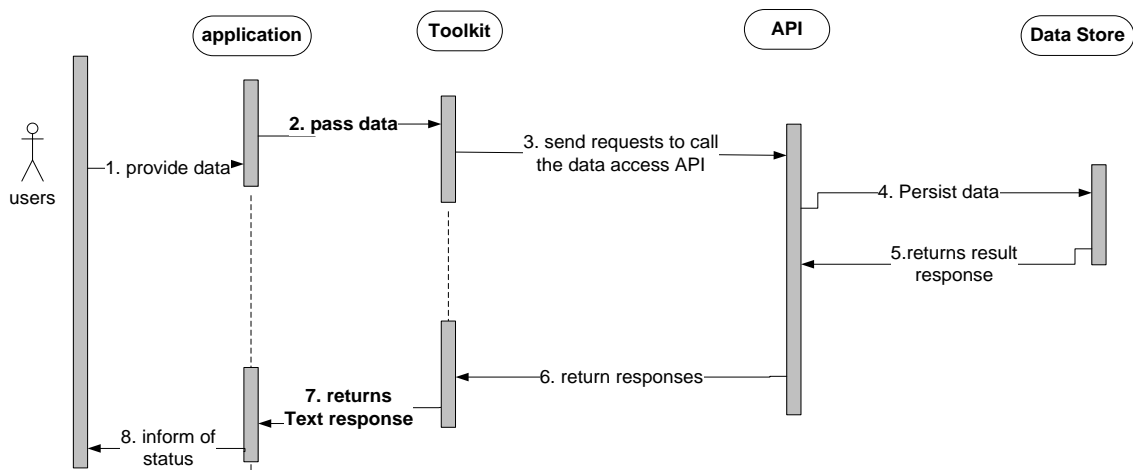
**Figure 5-6 End User Development Toolkit**

Toolkits can be provided for a number of languages and platforms, such as a Python toolkit for Open Office applications (Oracle, 2011) or a Google App Scripts toolkit for Google Spreadsheet applications (Google, 2011b). In some application domains, end users not only access data from their desktop applications but also from mobile applications. Therefore, framework providers might also provide appropriate toolkits for mobile operating systems, such as Android (Google, 2011a) or iOS (Apple, 2011). EUDs are then assisted to create their own mobile applications to access the database.

#### 5.4.4 End user application development

With the API and development toolkits provided, much of the complexity of developing a data management application is taken care of. Figure 5-7 illustrates a typical data entry workflow from an end user application to the data store via the API.

- The input data in the user's application (Step 1) are passed to the toolkit (Step 2).
- The toolkit transfers the data as parameters for a request (Step 3), which calls the appropriate data access methods of the API, e.g. the Insert data method.
- Next, the API persists the data in the corresponding data store (Steps 4 and 5).
- Whether the data transaction succeeds or fails, the API sends back response messages to the toolkit (Step 6).
- Finally, the toolkit parses the responses to a native data structure that can be displayed by the application (Step 7).



**Figure 5-7 Data management workflow**

Figure 5-7 shows the eight step process. The EUDs only need to deal with passing input data (Step 2) to the toolkit and returning system responses (Step 7) to the application. This means that only a few lines of code are needed to pass the relevant parameters to the toolkit for data loading.

Programming, by example, is an effective way for EUDs to write an application (Halbert, 1984) (H. Lieberman, 2001). Therefore, we suggest that framework providers supply sample end user applications to demonstrate how to use the toolkit methods. EUDs then study the examples in order to understand how to use the toolkit.

The framework API and development toolkits not only make application development faster and easier, but it is also easier for EUDs to customise and extend individual EUDAs. All EUDAs programmed using the same framework will have a similar structure: the applications only need to take data from the UI and pass it to the API to access the database. This means that with minimal changes, a data management application designed for a specific data object can be adapted for the different specialist types of data, such as Apple versus Onion in a LIMS domain or Cars versus Trucks for a vehicle rental domain.

#### **5.4.5 Framework configuration tool**

Section 5.4.1 described how the framework manager (EUDs or in-house IT professionals) can add new individual level classes to meet specific end user's needs. Adding new concrete classes requires framework managers to develop concrete classes with specific attributes and validation rules and to represent the new classes in the underlying data store, such as a relational schema. This requires extensive OOP and database development experience, which may be beyond the skills of many typical EUDs. Although in-house IT professionals might have the development expertise, it will take effort and time to manually update and maintain the framework as it evolves over time.

In order to help framework managers maintain and extend the framework with minimum effort, we suggest that framework providers supply configuration tools to automate the framework customisation and extension. For example, a framework configuration tool might include:

- **A framework configuration template:** A text based definition template (e.g. CSV format) is a relatively simple format for framework managers to understand and manipulate. A template could be provided to show how to specify metadata such as class attributes and validation rules. Framework managers can amend the text based definition to include their specific object attributes and validation rules.
- **A framework configuration tool:** This tool will read the user-defined configuration template and automatically generate individual level classes and code to map them to the data store. For example, a configuration file parser and database schema generator can be provided in order to automate the creation of individual level classes and matching database schema. The implementation details of the configuration tool are described in Chapter 6.

A configuration template and code generation facilities can dramatically reduce the effort needed by framework managers to customise and extend the framework.

## 5.5 Summary

In order to meet the guidelines we developed in Chapter 3, we propose a framework that will support EUDs to customise and extend data management applications. The framework comprises a domain model and a corresponding data store. An API and toolkits are supplied to simplify data access from end user software. Configuration tools are provided to maintain and extend the generalised data model for specific situations. The advantages of using the framework are summarised below for each of the guidelines:

- 1. Have a data model designed for a specific domain that allows for modification and extension**

The framework provider develops a framework comprising a domain data model and configuration tools. Using the configuration tools, the framework managers are able to customise and extend the domain model to meet a variety of data requirements in the same application domain.

- 2. Allow end users to easily manage their data**

The framework approach provided a single organisational store for end users to manage a variety of data sets. The framework provides client application toolkits for EUDs to create applications based on their familiar UIs, such as spreadsheets. More importantly, EUDs need only customise the generated applications in order to meet specific requirements.

- 3. Allow EUDs to easily create and extend applications for different requirements within the same domain**

Framework providers supply an API and development toolkits to minimise the end user's effort needed to develop an application. EUDs only need write relatively small amounts of code to include data access functionality in their commonly used applications.

As the framework provides a generalised base class, all end user applications have a similar structure. Data management applications can be quickly adapted for other user groups who have similar requirements.

#### **4. Allow modifications and extensions without affecting existing data and applications**

The “Open for extension and closed for modification” principle is applied to the framework design. This ensures all new requirements are addressed by adding new individual classes and tables without affecting those already in use. Existing applications will, therefore, be unaffected.

There are many software development tools and libraries able to be used to implement the type of framework we have proposed. Chapter 6 will describe the possible implementation techniques and explain how we implemented an example of the framework at the New Zealand Institute for Plant & Food Research (NZPFR).

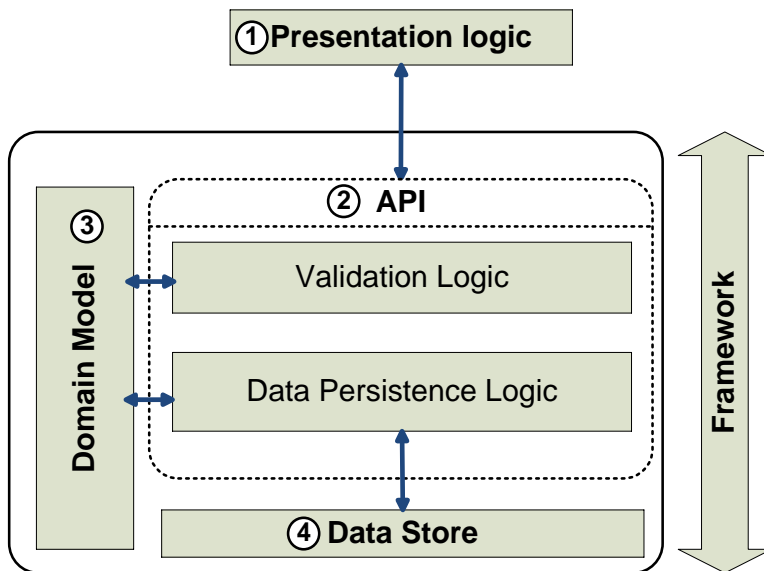
## Chapter 6 Framework Implementation

The proposed framework includes a domain model with an associated data store and an API to expose common data management methods. In this chapter, we describe the implementation details of the framework. Section 6.1 describes what framework providers need to implement and what techniques and software could be used for the implementation. Sections 6.2 and 6.3 illustrate a specific example we implemented at PFR for the LIMS case study (described in Chapter 4).

### 6.1 Responsibilities of framework providers

The framework includes three parts (shown in Figure 6-1): the domain model (Part 3) the corresponding persistence data store (Part 4) and an API (Part 2) to validate the domain objects and persist these domain objects in the data store.

With framework configuration tools, framework managers can subclass the domain object classes in order to meet specific end user's needs. Since, the API encapsulates the complex validation and data persistence logic, application development becomes relatively easy for end user developers (EUDs). EUDs only need to concentrate on Part 1, the development of the presentation logic.



**Figure 6-1 Framework Implementation**

This section outlines some alternatives for framework providers when implementing the framework and discusses some of the techniques and software required to do this efficiently.

### 6.1.1 Domain model and persistent data implementation

Framework providers first need to implement the most important part: the domain model and its corresponding data store.

- ***Domain model***

Framework providers identify the most fundamental domain objects in order to design and implement the domain model. We recommend framework providers visit different end user groups to collect their specific requirements and to think about the appropriate attributes, constraints and associations for the base classes and organisational level classes in the domain model. In addition, an individual level class should be provided for framework managers as an example. After the base classes and organisational classes are defined, the framework providers need to choose an Object Oriented Programming (OOP) language, such as Java or C#, to implement the domain model. The initial base classes and organisational level classes need to include the attributes and set/get access methods.

Data modelling tools can be used to quickly implement the domain model code. Data modelling tools, such as, Hibernate (Elliott, O'Brien, & Fowler, 2008; Hibernate, 2007), Eclipse Modelling Framework (Eclipse, 2011) and ADO.NET Entity Framework (Microsoft, 2011a) allow developers to define the class metadata (definitions of class, attributes, associations and inheritances). The modelling tools are then able to automatically generate the code for the domain classes based on the metadata. The set and get access methods also can also be automatically generated in the class implementation.

Validation rules also need to be defined for the domain objects and many software tools can be used to implement the validation logic. Possible tools include Microsoft Enterprise Library (Microsoft, 2011d) and Hibernate Validator (RedHat & Morling, 2011). These tools provide built-in constraints, such as Not Null, Min, Max or Regular expressions, which allow framework providers/managers to define these constraints inside the domain classes or in a constraint file. (These rules will be checked by the API, as discussed in Section 6.1.2.)

- ***Persistent data store***

Once domain model classes are implemented, a data store needs to be implemented to ensure that the domain objects persist. Different types of data store can be implemented, e.g. relational databases, object oriented databases or flat data files. In this section we use a relational database as an implementation example because the relational schema is commonly used to represent persistent data.



In order to represent the three level hierarchies in a relational model, a data mapping strategy needs to be deployed. Possible strategies include one table per class, one table per concrete class or one table per hierarchy mapping. All these have advantages and disadvantages and framework providers need to choose a suitable strategy for each of their projects. For example, the one table per class strategy is straightforward for mapping the domain classes to the database tables. Subclass tables have primary key associations to the superclass table so the relational model has a one-to-one association between the sub and super classes. This is suitable for extending the database to meet additional specialised needs because individual level tables can be added and updated without affecting the existing tables.

The modelling tools described previously can also be used to generate a corresponding data store, such as a relational database schema. The tools allow developers to define the Object Relational Database Mapping (ORM) definitions given a proper mapping strategy and can automate the schema generation based the defined ORM definitions.

### **6.1.2 API implementation**

The API is provided to keep presentation logic (see Part 1 of Figure 6-1) separate from the implementation of validation and data persistence logic (Part 2).

- ***Data validation logic***

Before inserting the domain objects into the database, the validation logic provided in the API needs to check the domain objects. The validation logic applies the defined validation rules to domain objects to check the domain object. Validation tools such as Microsoft enterprise library (Microsoft, 2011c) and Hibernate Validator (RedHat & Morling, 2011) provide generic methods to apply pre-defined rules to the domain objects. The rules may be defined either inside the domain classes or in separate constraint files. If any violation is detected, proper error messages will be returned from the method.

- ***Data persistence logic***

As a minimum requirement, the framework developers need to provide API methods to manage the abstract types, e.g. insertSample(Sample sample). EUDs only need to work with the variables of the abstract Sample class and the specific persistence logic is encapsulated in the API. For example, if a relational database is employed to store the persistent data, SQL queries will be provided in the API for inserting different types of sample records.

In order to simplify the SQL development for framework providers, Object-Oriented Mapping (ORM) software can be used manage the database access logic. For example, Hibernate for Java and ADO.NET tools for C#. Framework providers can use ORM tools to detect

concrete object types, such as Onion or Apple and then automatically generate the correct SQL to map the objects into the appropriate tables. The SQL is generated at runtime, so the method insertSample can be implemented to populate different types of samples, such as Onion and Apple.

- ***API method calling mechanism***

A suitable API method calling mechanism should be used to communicate between the presentation logic and the API. It should be language and platform independent, such as web services with a Simple Object Access Protocol (SOAP) (W3C, 2000) or a Representational state transfer (REST) protocol (Rodriguez, 2008). It should allow interaction between different presentation and development languages and platforms, such as Windows forms, HTML forms and other applications.

There are many web service development libraries that support web service implementations, such as Apache CXF (Sosnoski, 2010) and Window Communication Foundation (WCF)(Chappell, 2009). These tools support SOAP and REST web service implementations. For example, a SOAP web service can be provided to expose the API methods, and a Web Service Description Language (WSDL) (W3C, 2001) file can be provided to describe the web service details, such as method names, input parameters and output parameters.

- ***Development toolkit***

Language specific development toolkits can be provided to simplify API calls. Most development tools are able to access the WSDL file in order to generate native language specific methods, (Java, C#, VBA) to map to the web service methods. For example, a VBA toolkit could be provided to map the web service methods to an end user development environment such as Excel. EUDs can then use such a toolkit to include data management functionality in their Excel based data entry applications.

The provision of toolkits makes the application development relatively simple; EUDs need to focus only on presentation logic development (using a toolkit to interact with the API). They do not need to be concerned about writing specific validation and data access code to persist specific types of data, such as Onion or Apple.

### **6.1.3 Summary**

Section 6.1 discusses the features framework developers should implement to support end user database development: a domain model, a persistent data store and an API. We outline possible software tools to support framework implementation and alternatives to minimise the implementation effort for framework providers. In the following sections, we will focus on

discussing how we implemented the framework model, database and API for PFR users to manage a variety of laboratory data. The Client development toolkit will be discussed in Chapter 7.

## 6.2 LIMS domain model implementation

To develop the LIMS model for PFR (described in section 5.4.1), we visited several EUDs and laboratory staff at PFR in order to identify the most important domain objects. Laboratory staff set up assays (experiments) to test plant samples, so they need a database to manage the sample and assay records and their associations. The most important domain objects are genetic samples and assays from plants. In addition, we talked with LIMS professionals in several other institutions to ensure our framework was as general as possible within the LIMS domain. We then implemented our initial domain model and corresponding database at PFR.

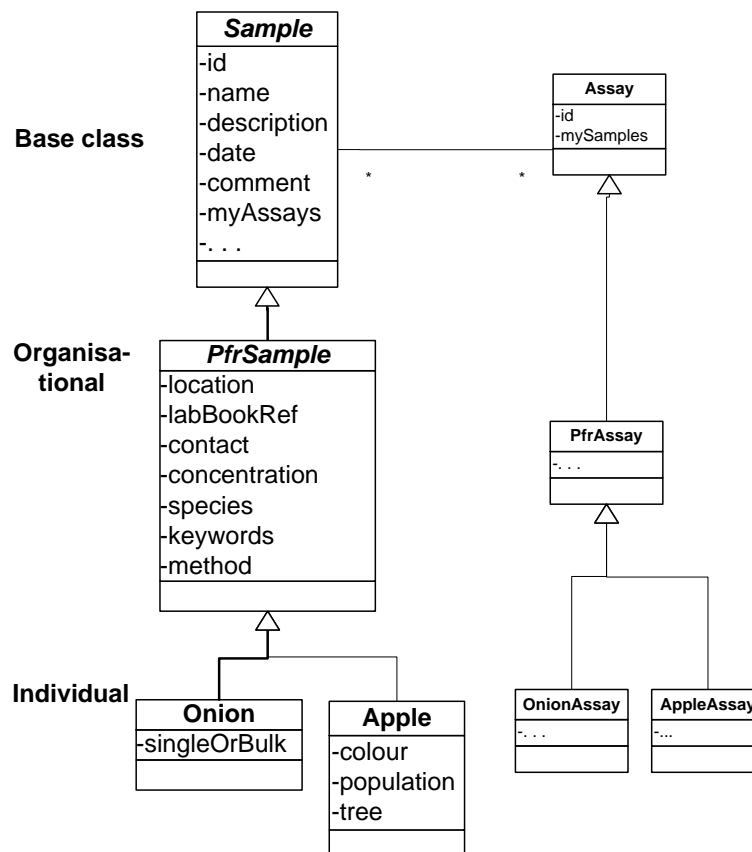
The details of domain model and database implementation will be discussed in Section 6.2.1 and 6.2.2, respectively. The system is implemented to automatically generate the entire domain model classes, validation constraints and database schema based on mapping files that will be described in more detail in Sections 6.2.3, 6.2.4 and 6.2.5. With the code generation tools, framework providers do not need to manually write code or SQL to implement the domain model and database tables.

### 6.2.1 Domain model

The initial domain model is shown in Figure 6-2:

**The base classes** (e.g. Sample and Assay) can be reused and extended in many different organisations in the same application domain. At PFR, the base Sample class includes:

- Minimal attributes, such as *Sample id, name, description, date, and comment*. This is the basic descriptive information common to all organisations in this domain.
- Minimal validation rules to enforce the required sample information, such as unique sample id and required sample names.
- Generic associations between classes (e.g. a many-to-many class association) In order to implement the many-to-many relationship between the base classes Sample and Assay, a *mySamples* collection needs to be added as an attribute of the Assay class, and *myAssays* was added as an attribute of Sample class. The cardinality can be validated by validation logic in the API.



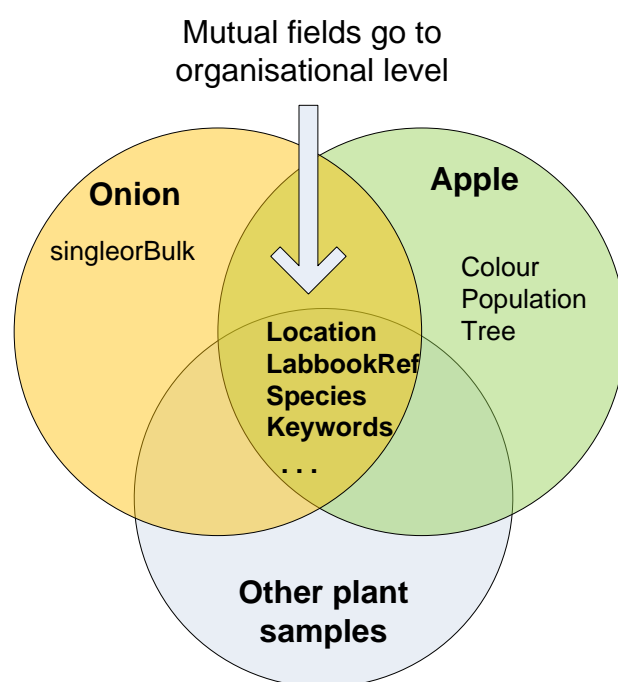
**Figure 6-2 Three level design**

**The organisational level** classes refine the generic design for a particular organisation. For example, we visited all user groups at PFR, who will use the framework to create their client applications, to determine the organisational level attributes and constraints. The PFR organisational level Sample classes include:

- Attributes common to all PFR users, such as *location*, *labbookReference*, *species*, *keywords*, and *method*. We worked with framework managers, EUDs and end users from different groups to find common fields in their existing data. The intersection of Sample fields for different groups at PFR (see Figure 6-3), should go into the organisational level. The remaining fields, such as *singleOrbulk* or *Tree* that are not applicable for all samples should go in the individual level.
- Constraints should be defined to ensure data records are complete and correct from the organisational view. For example, the species value must be provided for each sample record, so a constraint must be defined to ensure the field value is “not null”. In addition, the rules defined at this level must be applicable for all samples in the

organisation, because the validation rules are automatically inherited by all the individual level classes.

- Class behaviours common to individual classes can also be added to the organisational level. Currently, the framework developed for PFR is to help EUDs to create a flexible database and there are no specific behaviours defined in the domain model. In the future some specific methods could be added to provide comprehensive functionality. For example, a method could be developed to generate barcodes for samples. As a consequence, EUDs would not need to worry about adding code for barcode generation.



**Figure 6-3 Interactions between specific samples**

The organisational level design is very important. We highly recommend that the framework providers and framework managers work together to design it. Having an appropriate organisational level design early is essential because later changes to the organisational level data will likely affect all individual level classes and applications.

**The individual level** classes need to include only attributes and constraints specific to a particular type of data. Since most of the complex design has been included in the base classes and organisational level classes, the framework managers need to concentrate only on specific requirements in each individual level (e.g. Onion or Apple).

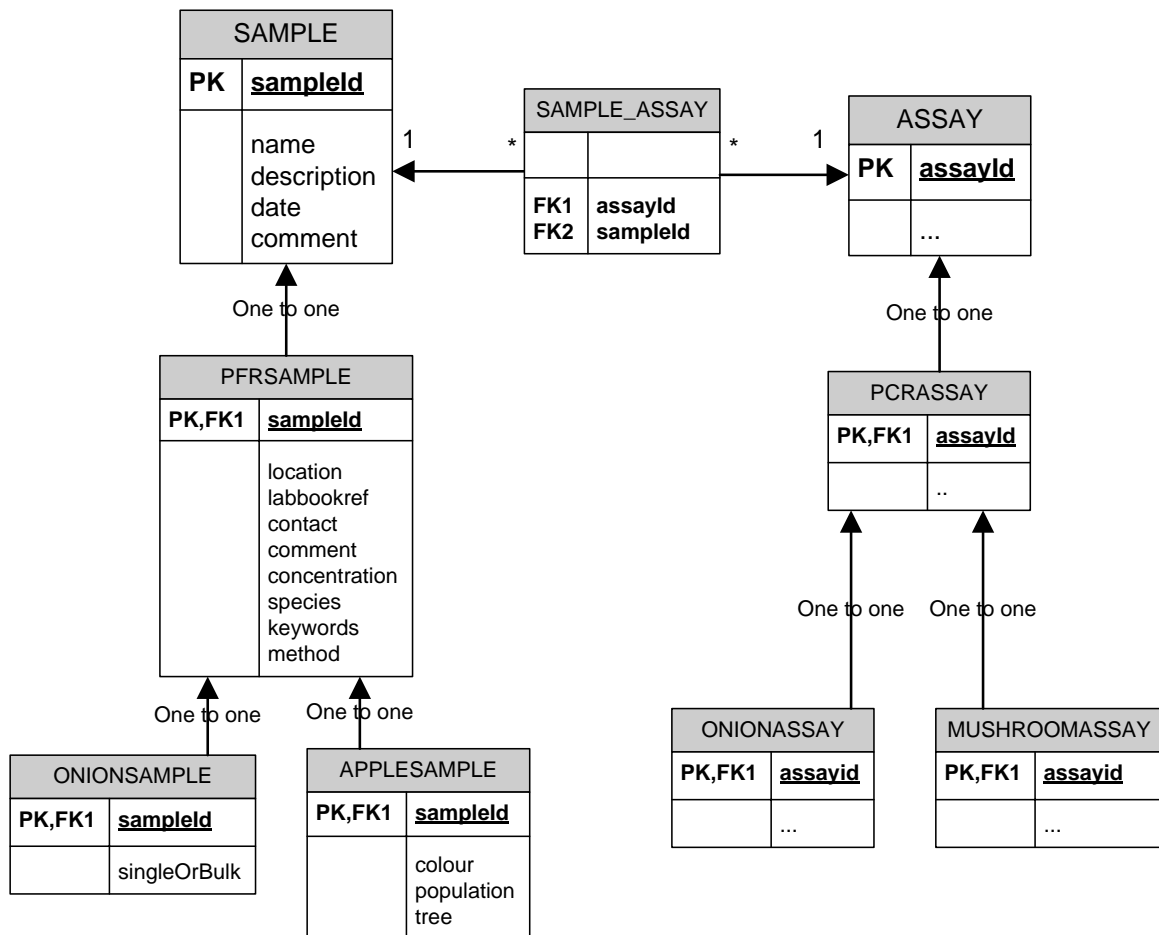
- Attributes need to be included in this level that are applicable to specific sample types. For example, all fields left over from the intersections illustrated in Figure 6-3 will be included in this level, e.g. the SingleOrBulk attribute of the Class Onion.

- Validation rules can be defined for the specific attributes. For example, a validation rule ensures the value of a colour attribute must either “green” or “red” in the Apple class. The validation rules inherited from the organisational level cannot be overridden by individual level classes as they have been designed to enforce data integrity from an organisational level.

Framework configuration tools (described in Chapter 6) will allow framework managers to add individual level classes without needing to know how to write class code for specific attributes and constraints.

### **6.2.2 Database**

We applied the one table per class strategy to directly map the domain object classes into database tables, as shown in . The benefit is that new classes/tables can be added without affecting existing ones. Individual level and organisational level tables have primary key associations to the framework and organisational level class tables, so there is a one-to-one association between each level. For example, shows a SAMPLE table that represents the base sample class, the PFRSAMPLE table represents the organisational level class and the ONION table represents a separate individual class. The relationship between these three tables is one to one (they shared the same primary key).



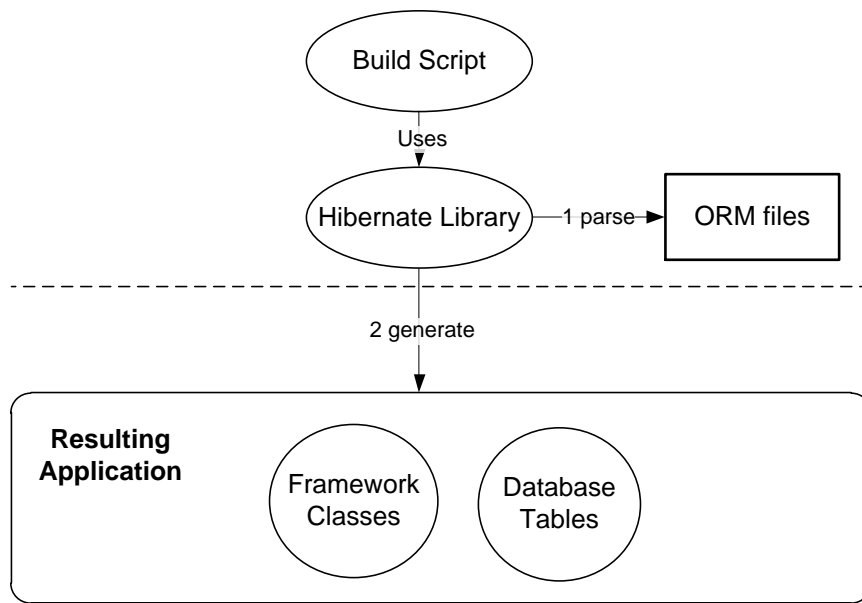
**Figure 6-4 Database tables**

In the database, an intermediate table **SAMPLE\_ASSAY** (see ) was added between **SAMPLE** and **ASSAY** to store all the relevant associations of sample and assay entities. The **SAMPLE\_ASSAY** table has a one-to-many relationship to the **SAMPLE** and **ASSAY** tables. To represent, this `sampleId` and `assayId` were added as foreign keys in the **SAMPLE\_ASSAY** table.

### 6.2.3 Class and database schema generation

The implementation of the framework requires ORM details to be defined in a configuration file. The ORM files define mapping details between the domain classes and relational database tables (Section 6.2.4 will describe it in more detail). A build script (see Figure 6-5) is developed (using Hibernate library) to parse the ORM files in order to generate the framework domain classes and associated database tables. The build script is able to generate domain classes code (Java) that include class attributes, constructors and access methods (get/set) for each attribute. After the domain class code generation is completed, the build

script generates SQL to automate database schema generation. This approach significantly reduces the coding effort of framework implementation.



**Figure 6-5 System build script**

The framework provides flexibility to handle new individual level classes. The framework and database can be extended by modifying the ORM files in order to meet additional requirements in the future. There is no need to write code or SQL to implement classes and tables, because the generation of the classes and database tables is fully automated.

## 6.2.4 ORM files

For our implementation at PFR an ORM file (see Figure 6-6) file defines the mapping between framework classes (object oriented model) and database tables (relational schema). The file maps Class Sample to Table SAMPLE.

```

<!--Base classes/tables-->
<class name="Sample" table="SAMPLE"> 1
    <id name="id" column="sampleId" type="int"/> 2
    <property name="name" column="name" type="string"/> 3
    <property name="description" column="description" type="string"/>
    <set name="myAssays" table="SAMPLE_ASSAYS"> 4
        <key column="sampleId"/>
        <many-to-many class="Assay" column="assayId"/>
    </set>
    . . .
  
```

**Figure 6-6 ORM file for Class Sample**



- **Class/Table mapping:** defines the mapping of the class (Sample) to the database table (SAMPLE).
- **Primary key mapping:** maps the sample attribute id as the primary key of SAMPLE table. This also defines the id attributes and type.
- **Class attribute/Table column mapping:** maps each class attribute to a table column with an appropriate data type.

Association mapping: maps the associations between classes (e.g. the assays and samples).

Collections or sets of objects are mapped onto the intermediate table. For example, in the Sample class the attribute myAssays represents a collection of assay objects associated with a sample object. The attribute myAssays is mapped onto the intermediate Table SAMPLE\_ASSAY (see Figure 6-6) between SAMPLE and ASSAY to store the combinations of sample and assay.

- ***Inheritance mapping***

Each ORM file defines the entire class hierarchy using the one table per class mapping strategy. Figure 6-7 shows “joined-subclass” is the syntax for implementing the one table per class mapping with inheritance between base, organisational and individual level classes, e.g. between classes Sample, PfrSample and Onion.

```
<!--Base classes/tables-->

<class name="Sample" table="SAMPLE">
  <id name="id" column="sampleId" type="int"/>
  <property name="name" column="name" type="string"/>
  . . .

  <!-- Organizational level classes/tables -->

  <joined-subclass name=" PfrSample" table="PFRSAMPLE">
    <key column="sampleId"/>
    <property name="contact" type="string"/>
    <property name="labBookRef" type="string"/>
    . . .

    <!-- Individual level classes/tables-->

    <joined-subclass name="Onion" table="ONIONSAMPLE">
      <key column="sampleId"/>
      <property name="singleOrBulk" type="string"/>

    </joined-subclass>

    . . .
  </joined-subclass>
</class>
```

**Figure 6-7 ORM file for the entire Sample hierarchy**

- **Adding new individual level classes**

A new individual level Class, such as Apple can be added by defining a new joined-subclass element (see Figure 6-8). There is no need to manually update the code and database schema. The system uses the ORM file to build a script that automates the implementation of individual level classes and tables. Creating the text (see Figure 6-8) is all a Framework Manager needs to do to extend the database for a new specialised sample.

```
<joined-subclass name="Apple" table="APPLESAMPLE">
  <key column="sampleId"/>
  <property name="colour" type="string"/>
  . . .
</joined-subclass>
```

**Figure 6-8 Adding a new individual level Apple Sample**

### 6.2.5 Constraint files

In order to simplify the validation logic implementation, we use the Hibernate library to help framework providers/managers define validation rules in constraint files (see Figure 6-9). Hibernate requires constraints to be defined in XML format, which is then processed by the API (described in Section 6.3.2) to perform the validation. The validations can be extended quickly without developing new application code. Framework providers/managers can add or modify rules in the XML file to address different requirements.

Figure 6-9 shows constraints defined in XML for the base class Sample

```
<!-- Base class constraints -->
<bean class="Sample">
  <getter name="name"> 1
    <constraint annotation="javax.validation.constraints.NotNull"> 2
      <message>ERR: Sample name is required</message> 3
    </constraint>
  </getter>
  . . .
</bean>
```

**Figure 6-9 Constraint file for base class Sample**

- **Getter Name:** defines which attribute is going to be checked (name in the Sample class in this case). The API will use the class get method (e.g. getName) to retrieve the attribute value to check.
- **Constraint:** defines the validation constraints, e.g. NotNull in order to make sure the end user has supplied the required sample name

- **Message:** a message can be defined here to inform end users a sample name is required.

- ***Constraint Inheritance***

The rules defined in the base classes will be inherited by all the individual level classes. For example, a Not Null constraint is defined to check the sample name in the base class; the constraint will be automatically inherited to individual level (via the organisational class) to make sure all samples names are required.

The organisational and individual level constraints are also defined in the same constraint file (see Figure 6-10). Validation rules in the organisational level should be applicable for all users within the particular organisation. For example, the sample location must be provided at PFR. A NotNull constraint can be defined at the organisational level in order to ensure all individual level samples have location information.

At the individual level (e.g. Onion Sample), all validation rules from the base and organisational level are inherited and specific validation rules can also be added, e.g. a constraint for attribute singleOrBulk (see Figure 6-10). A regular expression is used to check the value must either be Single or Bulk. In addition, more constraints can be added in the individual level to check different types of samples, e.g. Apple or Orange.

Constraints are combined if multiple constraints are defined for the same attribute at different levels. For example, if a specific constraint (e.g. the sample name must be between six and ten characters) is defined at the individual level, it will be checked in addition to the NotNull constraint defined from the base sample class.

```

<!-- Base class constraints -->

<bean class="Sample">
    . . .
</bean>

<!-- Organisational level constraints -->

<bean class="PFRSample">
    <getter name="location">
        <constraint annotation="javax.validation.constraints.NotNull">
            <message>ERR:Sample location may not be null</message>
        </constraint>
    </getter>
    . . .
</bean>

<!-- Individual level constraints -->

<bean class="Onion">
    <getter name="singleorBulk">
        <constraint annotation="javax.validation.constraints.Pattern">
            <message>ERR: the singleOrBulk must be
                Either single or bulk</message>
            <element name="regexp">Single|Bulk</element>
        </constraint>
    </getter>
    . . .
</bean>

```

**Figure 6-10 Constraint file for the entire Sample hierarchy**

- **Cardinality Constraints**

A many-to-many relationship between the Sample and Assay was implemented to maximise the database flexibility. In some cases, a particular assay only tests one plant sample, in other cases it might test more. In order to control the cardinality, a constraint can be defined to limit the number of elements in the mySamples collection of the assay class. The generic validation code will check whether the number of samples matches the predefined constraint rules. For example, the cardinality constraint in the class OnionAssay (shown in Figure 6-11) demonstrates how to restrict the number of samples in the OnionAssay Object. The constraint is used to make sure that OnionAssay must have at least one and not more than three associated samples.

```

<bean class="OnionAssay">
    . . .
    <getter name="mysamples">
        <constraint annotation="javax.validation.constraints.Size">
            <message>ERR:An assay can only test between 1 and 3
                samples</message>
        </constraint>
    </getter>
</bean>

```

```
<element name="min">1</element>
<element name="max">3</element>
</constraint>
</getter>
. . .
</bean>
```

**Figure 6-11 Cardinality Constraint**

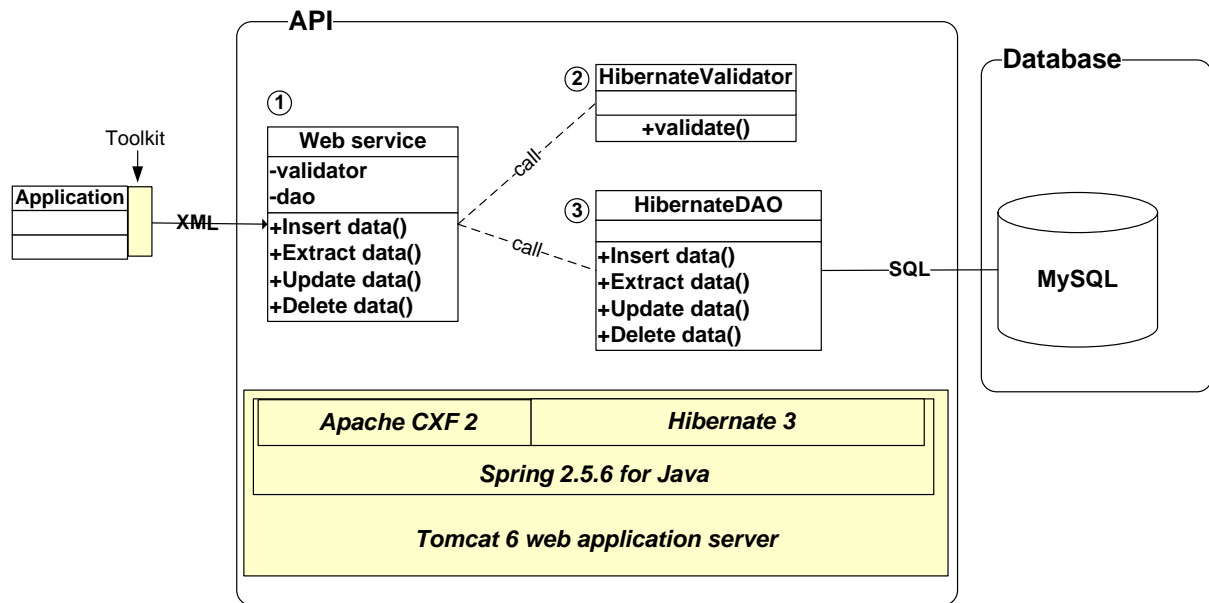
Domain objects and validation rules can be defined in ORM and constraint files but maintaining these files requires learning specific mapping syntax. In order to make the definitions even simpler, we provided a single text based configuration template to help framework managers customise their individual level domain objects and validation rules. The configuration template will be described in Chapter 7.

## 6.3 Framework API implementation

Section 6.1.2 suggests implementing the API as a web service to help EUDs easily manage the data and simplify including database management functionality in different client applications. The web service sits between end user applications and the database to carry out data validation and database access logic. In the following sections we describe how we implemented the API at Plant and Food Research (PFR).

### 6.3.1 Web service API for PFR

There are three types of abstract classes designed in the API: Web service, Validator and Data Access Object (DAO). We have implemented these abstract classes at PFR. The diagram Figure 6-12 shows object interactions. The web service (Part 1) uses Validator (Part 2) and DAO (Part 3) to manage the validation and persistence logic. The web service is able to provide the API functionalities for a various types of client applications, such as spreadsheets or mobile applications. We implemented the Validator and using the Hibernate Library, so they are named HibernateValidator and HibernateDAO.



**Figure 6-12 Web service Implementation**

The Apache CXF Java web service library (Hathi & Balani, 2008) was used to publish the web service on a Tomcat (Apache, 2011) web application server. The default web service implementation uses SOAP as communication protocol. A Web Service Description Language (WSDL) file is generated to describe the web service methods and input and output parameters.

The SOAP web service is an industry standard. Most software libraries, such as (the NET WSDL tool or Apache CXF WSDL tool) are able to read WSDL in order to generate a language specific code library to access the web service. Using SOAP with WSDL significantly reduces the effort of creating client application toolkits.

Spring (Spring, 2009; Walls & Breidenbach, 2008) is used in the system to manage the object lifecycle and dependencies between the web service and the DAO and Validator objects. For example Spring is used to initialise the HibernateDAO and HibernateValidator objects for web service object to use.

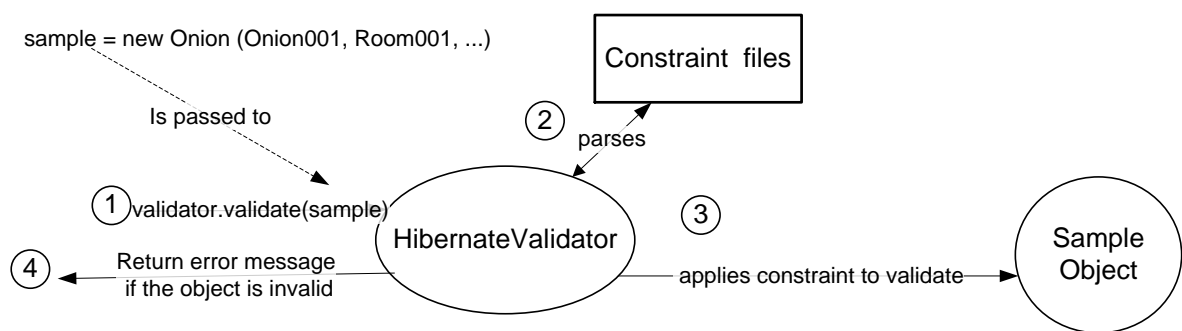
The system build script (described in section 6.2.3) not only generates the classes and tables, but also deploys these with all required libraries (Hibernate, Apache CXF and Spring) to the Tomcat web application server.

### 6.3.2 HibernateValidator

The HibernateValidator is implemented to centralise data validation code for different types of objects in the domain model. The web service uses the HibernateValidator to check individual level objects before calling the HibernateDAO to persist the objects.

The HibernateValidator provides a generic method to validate different types of domain objects, such as Onion or Apple. A validation workflow with four steps is shown in Figure 6-13 :

1. Takes a sample object as an input and determines its type.
2. Parses the constraint file in order to look up the predefined constraints.
3. Retrieves object attributes using object access method (e.g getName in Class Sample) and validates the attribute.
4. If an invalid object is detected (e.g. the sample name is not specified), the method will return the error message defined in the constraint file e.g. “ERR: Sample name is required”.



**Figure 6-13 workflow of validation logic**

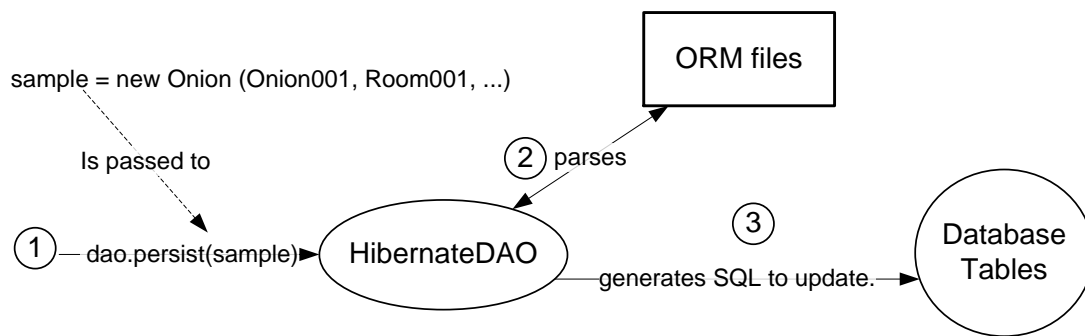
### 6.3.3 Hibernate DAO

The HibernateDAO provides generic database access methods to insert, retrieve and update a database. When the HibernateDAO is invoked to manage the persistence of domain objects, it reads the Object Relational Database Mapping (ORM) file and generates SQL scripts to transfer data between the domain objects and database tables.

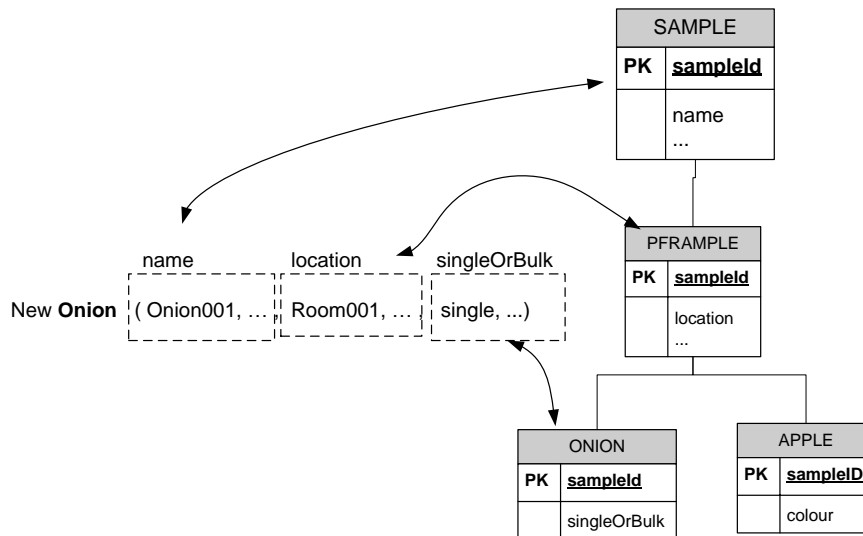
The ORM files control how the HibernateDAO carries out the database methods at runtime, so framework providers do not need to manually create the SQL to map attributes to columns. The HibernateDAO provides a generic method to store domain objects into the database.

There are three steps which are illustrated in Figure 6-14:

1. Determines the input parameter type, such as an Onion sample.
  2. Looks up the mapping details for an Onion sample in the ORM file,
  3. Generates SQL to store the Onion sample attribute values in the proper table columns.
- For example, Figure 6-15 demonstrates details of how attributes are mapped into appropriate columns of the Tables SAMPLE, PFRSAMPLE, and ONION respectively.



**Figure 6-14 Workflow of insert data**



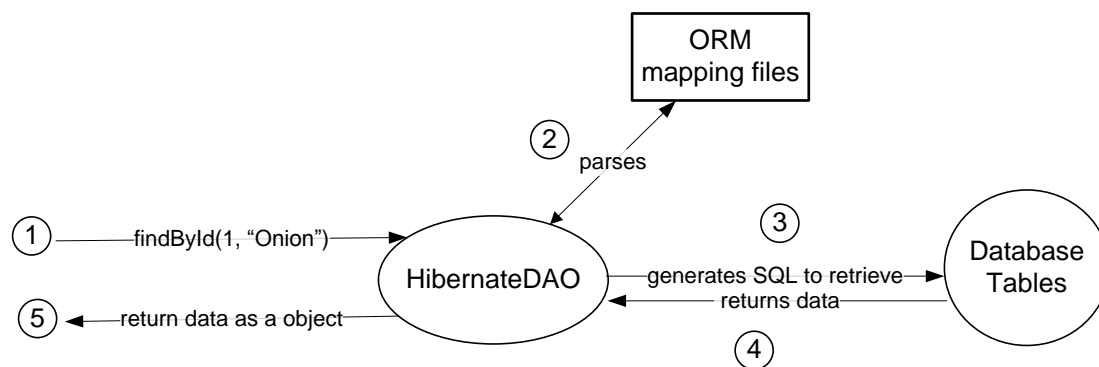
**Figure 6-15 object relational mapping**

The insert method also generates a sequence number as an id (sampleId in this case). The id will be used as a primary key in the database tables in the sample hierarchy.

The structure makes the database access code extremely flexible. Even if new individual level classes are added only the ORM file needs to be updated. The database access code does not need to be changed. For example, when a new type of sample class is added, the database access code can determine the class mapping details from the ORM file and generate the appropriate SQL. Framework developers do not need to provide extra code to handle newly added classes; they just need to add the mapping details to the ORM file.

The HibernateDAO also provides methods to retrieve data from the database. Figure 6-16 demonstrates how to use the HibernateDAO to retrieve an Onion sample from the database. For example, the HibernateDAO provides a FindById method. When the web service calls FindById, the HibernateDAO reads the ORM mapping files (Step 2) and generates SQL to retrieve the data from different tables (Step 3 and 4). After that, the HibernateDAO maps the data attributes into an Onion Object and returns this object (Step 5).





**Figure 6-16 Work flow of data retrieving**

There are four methods supplied in the system by DAO:

1. **findById(int id, string type)** is to retrieve objects by id and type, such as finding a onion sample by the sampleId.
2. **findbyType(String type)** returns a collection of objects by type, such as retrieving all onion samples. The code only needs to pass the type as the input parameter in order to return all samples of the type.
3. **findByExample(Object object)** takes an input object as an example, such as a onion, and executes a query based on the given object type and attributes. It returns matching persistent objects in an array list.
4. **findByQuery(String query)** returns a collection of objects that match criteria specified in the Hibernate Query Language (HQL). HQL is similar to SQL syntax, but the Hibernate makes it fully object-oriented so can deal with inheritance and collections. For example, a query to retrieve onion samples produced on a particular date might look like:

*findByQuery ("from OnionSample as sample where sample.date = '2010-5-28")*

This query not only returns matching onion sample objects including inherited attributes from base and organisational levels but also returns the myAssays attribute with a collection of associated assay objects.

Update and delete methods are also provided in the DAO. Once objects are retrieved from the database, these methods can be used to modify or delete the persist objects.

### 6.3.4 Web service API configuration

Spring is used to manage the life cycle and dependencies among the API objects, such as creating instances of the HibernateDAO and HibernateValidator objects for the web service to use. Therefore, framework providers do not need to write code to initialise these objects.

Instead, they can simply define an XML configuration file to control the initialisation of objects and the dependencies between them.

Figure 6-17 shows a Spring configuration file. A Bean is a Java Object; the Bean keyword is used to initialise a Java class object. The first Bean definition creates a HibernateDAO instance and the second Bean definition creates a HibernateValidator instance. The third Bean initialises the web service instance and sets it to use the validator and DAO object above.

Configuration file	Equivalent Java code
<pre> &lt;bean id="myDAO" class="HibernateDAO" &gt;     . . . &lt;/bean&gt;  &lt;bean id="myValidator" class="HibernateDAO"&gt;     . . . &lt;/bean&gt;  &lt;bean id="myWS" class="WebService"&gt;     &lt;property name="dao" ref="myDAO"/&gt;     &lt;property name="validator"                 ref="myValidator"/&gt;     . . . &lt;/bean&gt; </pre>	<pre> myDAO = new HibernateDAO();  myValidator = new     HibernateValidator();  myWS = new WebService(); myWS.setDao(myDAO); myWS.setValidator(myValidator); </pre>

**Figure 6-17 Application configuration file**

The implementation decouples the dependencies between the web service and database implementation details. The benefit is that new functionality can be added without changing the existing code. Currently, the HibernateDAO is implemented to map objects between the domain model and an SQL based relational database, such as MySQL or SQL server. If an object oriented database, such as JADE (Clarke, 2010; JADE, 2008), is implemented to store the domain objects, a JadeDAO can be implemented to manage the persistence logic.

Framework providers only need to alter the configuration file to swap from HibernateDAO to JadeDAO in the configuration file. The web service API is clearly very easily able to be extended for different storage models.

### 6.3.5 Web service API methods

Currently, the web service provides a set of common data access methods.

- **Insert data**

The web service `insertSample()` is shown in Figure 6-18. The web service calls the validator to check the sample attributes based on the predefined validation rules. If there are any violations, the proper error messages are returned to the end user applications.

The validator in Part 1 looks for violations within a single object. Other database validation (e.g. checking that a sample name is unique) requires existing data to be investigated. Part 2 shows how the DAO uses `findByExample` to see if a duplicate sample already exists in the database. If the sample data is valid, the web service method calls the DAO to save the sample object to the database.

```
public String InsertSample (Sample sample) {  
  
    //1. Validate domain objects based on the predefined validation rules  
    violations = this.validator.validate(sample);  
    if ( violations.size() > 0 ) {  
        return violations.iterator().next().getMessage(); }  
  
    //2. Use the dao object to check duplicate date in the database  
    else if (this.dao.findByExample(sample) {  
        return "ERR:" + "you cannot insert duplicate samples in the database,  
the sample name must be unique"; }  
  
    // 3. Using the dao object to save data  
    else {  
        this.dao.persist(sample);  
        return "The sample has been inserted"; }  
    ...  
}
```

**Figure 6-18 Example code of `insertSample`**

In order to create associations between assay and sample objects an `insertAssay` method was implemented in the web service. The method takes an assay object and a collection of sampleIDs as input parameters in order to store the assay object and associated samples in the database. The logic is similar to the `insertSample` method, but it needs to validate the sampleIDs to makes sure they are valid samples in the database. There are four steps:

1. The `insertAssay` method checks that each sample exists using the `sampleId`. If a null value is returned from the `findById` method for any sample in the collection parameter, the `insertAssay` method returns an error message to the client application.
2. Another database validation (e.g. checking that an assay name is unique) requires existing data to be investigated; the DAO uses `findByExample` to see if a duplicate

assay already exists in the database. The method checks the duplicate assay data in the database based on provided assay name.

3. The web service calls the validator to check the assay attributes conform to the predefined validation rules.
4. If the input assay object is valid, the web service uses the DAO to save the assay data in the database. The DAO is able to add sampleId and assayId as foreign keys in the SAMPLE\_ASSAY table in order to store the associations.

- ***Retrieve data***

Retrieval methods allow EUDs to retrieve existing data from the database for updating. These use the DAO methods (described in section 6.3.3) to extract data from database and return the data as objects to the client applications. Delete and update methods are also provided to manage the returned objects.

Currently, the web service uses the Hibernate DAO to provide two retrieval methods: findbyID and findbyType. Although the findbyQuery implemented in the DAO can be used to retrieve subsets of the sample object collections, it is still difficult for novice EUDs to write such queries. In Chapter 8 we will discuss how to provide a simpler alternative to retrieve subsets of the collections by specific criteria conditions.

- ***Updating data and deleting data***

The update and delete methods allow maintenance of existing objects in the database. Once retrieve methods return object data from the database, the update method can modify the object. Similarly, the delete method can remove the objects and their related associations, e.g. assays objects with their associated samples.

## 6.4 Summary

Sections 6.2 and 6.4 introduced the framework, database and web service implementations.

Table 6-1 summarises each component:

**Table 6-1 Summary of the framework components**

Name	Description
ORM file	Defines the mapping between framework classes and database tables.
Constraint file	Defines constraints for domain objects
Build script	Reads the ORM file to generate classes and tables.  Deploys all classes and tables and required libraries to the application server.
Data Access Object	Parses the ORM file to manage the persistence logic and mapping
Object Validator	Parses the constraint files to manage the validation logic
Web service API	Manages the DAO and validator to provide data management functionality
Spring Application configuration file	Defines the dependencies between application objects  Used by Spring library to manage application object initialisation

The framework was implemented to support EUDs to have a data store that allows for modifications and extensions. We also proposed the idea of framework development tools (discussed in Chapter 7) to allow framework managers to easily customise domain classes with minimal effort.

The web service API is developed for EUDs to manage the data. It encapsulates the database implementation and is able to generate SQL to manage the data persistence logic. This means that EUDs do not need to spend considerable effort writing SQL to update and retrieve information. With the client development toolkit (discussed in Chapter 7), EUDs need only to write a few lines of code to include the database functionality in their commonly used applications.

## Chapter 7 Framework development tools

Chapter 5 discussed the implementation of the framework for the Laboratory Information Management System (LIMS) at Plant and Food Research (PFR). The framework supports end user developers (EUDs) maintain a database that allows for modifications and extensions. A web service API is provided to supply common data management methods. As a result, EUDs need only to focus on UI development and can use the web service client development toolkit to manage their data. In this chapter, we will discuss the challenges for non-professional developers in customising and extending the framework. We also explain how we provide development tools to help framework managers and EUDs to extend and modify the framework to meet their specific needs.

### 7.1 Overview of development tools

In order to extend the framework, for example, by adding a new individual level sample type (such as Apple) with specific attributes and constraints, framework managers need to:

1. Add individual level class definitions in the ORM and constraint files.
2. Run the system build script to generate the class and table based on the ORM files.
3. Provide development toolkits to simplify application development by clients.

In order to complete the above steps, framework managers need to understand quite complex XML mapping syntax and Linux command skills (e.g. running build scripts in a command environment). Moreover, framework managers have to learn how to provide a new client development toolkit for a specific development application environment, such as mapping web service methods and individual level classes in a VBA toolkit for Excel. When new individual level classes are added to the framework, framework managers also need to update the toolkit to include the new classes. These require development expertise and skills (such as XML mapping and toolkit development) could be beyond many framework managers who are non-professional developers.

To investigate the ability of framework managers to customise and extend the framework, an informal pilot user trial was carried out to test how a potential framework manager could work with ORM and constraint files, and system build scripts. A genetic scientist (who is an experienced VBA EUD) acted as a framework manager. With assistance from a framework provider, the participant was able to define the ORM and constraint files to add a new sample

class and run the system scripts to generate the sample class and database table. However, the participant commented it that it could be difficult for non-IT professionals to maintain the complex XML mappings (for ORM and constraints) and that user friendly tools to customise and extend the framework would be useful.

The participant, however, was not asked to update the client application development toolkits to include the new sample class, because the participant did not have any web service development skills. Instead, the framework provider created the client application toolkit, and the participant used the toolkit to successfully develop a data entry application to manage onion samples in the database.

- **Framework development tools for framework managers**

In order to ensure the framework can be extended easily and quickly, we suggest that framework providers make available user friendly development tools to:

- a. simplify the framework customisation and extension
- b. automatically generate classes and tables
- c. automatically generate a client application toolkit to provide native methods and classes against the framework API
- d. automatically update the client application toolkit when new web service methods and individual level classes are added in the framework.

- **Example of an application and application generator for EUDs**

A key goal of the guidelines (Section 4.2.2) is to allow EUDs to easily create applications using the framework; therefore, we also suggest framework providers provide:

- a. Example applications that demonstrate how to use the client toolkits to develop end user applications.
- b. An end user application generator to automatically generate data management prototypes.

In order to demonstrate how to help non-IT professionals to extend and use the framework, we implemented the framework development tools, client applications and a prototype application generator at PFR. They are discussed in more detail in Sections 7.2, 7.3 and 7.4, respectively.

## 7.2 Framework development tools

At PFR, the framework development tools we provided included a framework configuration tool, a framework code generator and a VBA toolkit generator. The sequence diagram (Figure 7-1) shows the workflow of these development tools.

- Step 1: Initially, framework managers need to amend the configuration template to add specific data items and validation rules for the new individual level class (e.g. Apple) and pass it to the configuration tool.
- Step 2: The configuration tool processes the configuration template and converts it to the required ORM and constraint files.
- Steps 3-6: The code generator parses the ORM and constraint files and calls the system build script to generate classes and tables. The generator also automates the web service description language (WSDL) file generation. (WSDL includes the details of web service methods and individual level data types.)
- Steps 7-8: The VBA toolkit generator parses the WSDL file in order to generate the VBA development toolkit. (If a toolkit already exists, the toolkit generator only updates the toolkit to include new web service methods and individual level class types.)

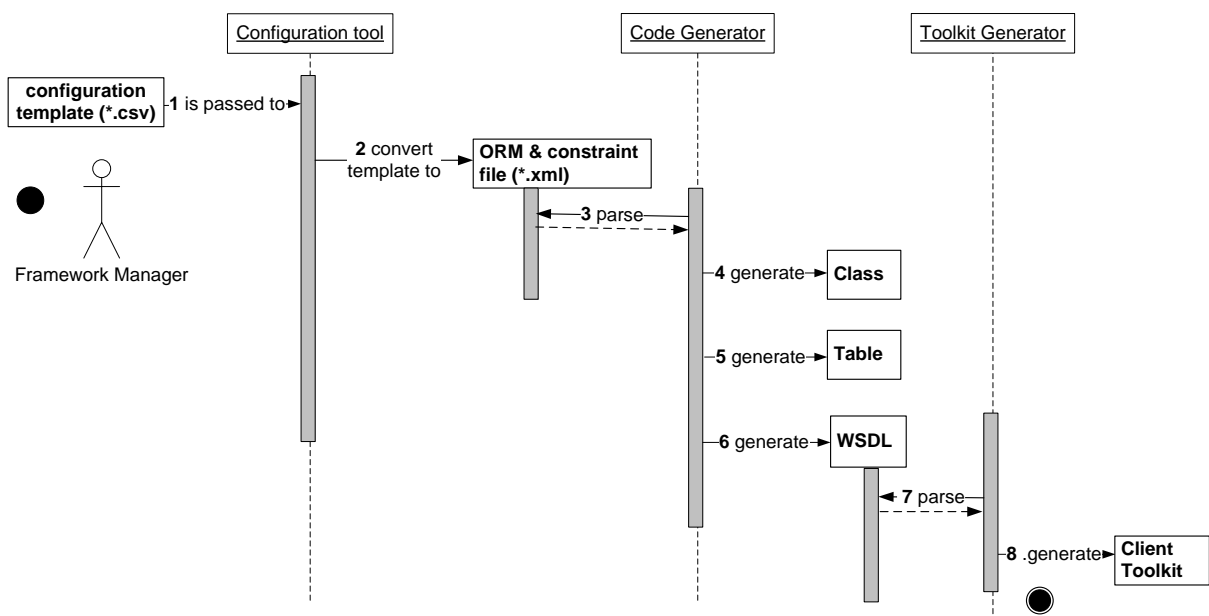


Figure 7-1 Framework development tools



Instructions (see Appendix 1) are provided to help framework managers define the configuration template and use these development tools.

### 7.2.1 Defining the configuration template

Figure 7-2 shows an example of a configuration template (Apple.csv) in Excel. Framework managers can refer to the instructions in order to understand how to modify the configuration template to address their specific requirements.

1	class	Apple	PfrSample	
2	field	measure	double	
3	rule	Min	5	measure must be higher or equal to 5
4	rule	Max	20	measure must be lower or equal to 20
5	field	colour	string	
6	rule	Pattern	red green	colour must be either red or green
7	field	population	string	

**Figure 7-2 Framework configuration template**

The keywords of the configuration template are summarised below:

- **Class:** Defines a new type of sample object and the higher level class it is inherited from, e.g. Row 1 shows how the *Apple* sample is defined, and its parent class at the organisation level is *PfrSample*.
- **Field:** Defines an attribute in the class. Row 2 shows how to define an attribute *measure* and set its type to *double* (i.e. a numeric value).
- **Rule:** Defines a validation rule for an attribute. Rows 3 and 4 mean that the input for the attribute *measure* must be between 5 and 20. An error message to help users to correct invalid input can be included in the next column of the rule.

Validation rules are optional and do not have to be defined for every attribute (e.g. there are no rules for the attribute *population*).

### 7.2.2 Framework code generator

A web based code generator is provided for framework managers to upload the configuration template to the server (that hosts the framework code). The code generator parses the ORM and constraint files and implements the classes and database tables (using the system build scripts). A WSDL file also is generated by the code generator to describe the web service methods and individual level classes. Once the generation process is complete, the database

and web service are ready to use. Moreover, if there are some errors in the configuration template, the tools will provide error messages to inform users before compiling and deploying the code.

### 7.2.3 Toolkit generator

A toolkit generator should be able to generate and extend a client's application development toolkit. Client development toolkits are language-specific and based on the EUD's programming environment. The generator approach can be applied to create different types of toolkit, such as VBA, PHP or Python toolkits, and based on user familiar development environment.

At PFR, EUDs are familiar with the Excel development environment, so we developed a VBA toolkit generator to include helper methods, individual level classes and native methods (called web service proxy) to access the web service API. The VBA toolkit allows EUDs to create and modify an Excel application without having to understand the complex domain model, database and web service API.

The VBA toolkit has two parts: a helper method class and a web service proxy. The initial toolkit only included a helper class to help EUDs develop Excel UI forms. Then, the VBA toolkit generator was provided to generate the web service proxy to represent a web service API and individual level classes as local VBA classes in the toolkit. Therefore, the framework managers can use the generator to update the VBA toolkits when new API methods and domain classes are added.

- **Helper class:** Helps EUDs develop UI. The framework provider needs to manually develop these methods based on specific requirements. For example, the current toolkit included a method: `addObjectsToControl`. This helps EUDs to populate objects, such as samples, into a VBA combo box. These helper methods can be used to add any type of object to a combo box. These methods do not need to be created by the generator, because these methods can be used for different types of domain objects and are not likely to be changed during the framework evaluation. Therefore, we manually developed a VBA toolkit skeleton with these helper methods. The VBA toolkit generator was then provided to map web service methods and individual classes into the toolkit skeleton.
- **Web service proxy:** The VBA toolkit generator is able to automatically generate the web service proxy that maps the web service methods and individual level classes into

the VBA toolkit skeleton. For the PFR implementation we used the Web Service Description Language (WSDL) utility (Microsoft, 2011b) and Visual Studio Tools for Office applications (Carter & Lippert, 2009) to generate the local web service proxy. For example, if the web service provides a method called *insertSample*, the toolkit will have a corresponding VBA method called *insertSample*. The toolkit method simply passes the objects to the web service for processing. All individual level domain classes (e.g. Onion and Apple) also are mapped as native classes in the proxy.

The process of client toolkit generation is completed at the client's side. For example, the EUDs only need to run the VBA toolkit generator; the VBA toolkit will be automatically generated to include helper methods, individual classes and web service native methods. This makes it easy for framework managers to develop and deploy the toolkit on the EUDs' computers. In the future, all related documentation (such as application development instructions) can also be included into the toolkit.

When new individual level classes and web service methods are added, the framework manager can simply run the VBA toolkit generator to update the web service proxy in the VBA toolkit. The VBA generator will check the WSDL file (includes all domain classes and API methods details) in order to see any newly added methods and classes. If new methods and classes are detected, the VBA toolkit generator maps these methods and classes into the VBA development toolkit. The framework managers do not need to manually update the toolkit.

## 7.3 Example code for end user applications

Once the client toolkit is installed, EUDs are able to create their applications. An example of code for inserting and retrieving data is discussed below.

### 7.3.1 Inserting data

A sample application to load records from Excel is provided. Figure 7-3 shows the Excel spreadsheet and example code. There are four steps to insert the sample data from the spreadsheet into the database.

1. Initialise an individual level object, such as an Apple object (shown in Line 2).
2. Collect the attribute values from the spreadsheet and assign them to the initialised object. For example, Lines 3 and 4 show how to copy the values from the spreadsheet to the Apple sample object.
3. Call the insertSample method and pass in the object for loading. All web service methods are encapsulated in a VBA class object called Service (Line 5).
4. The web service will process the request and send back a response message, which can be displayed to end users. The example code (Line 6) shows how to display the response message in a spreadsheet cell.

	A	B	C	D	E	F	O
1	name	description	entryDate	comment	contact	...	responseMessage
2	Appledna001	Apple DNA	2010-09-02		Tom		The sample has been inserted
3	...						

```
1 Sub insert()  
2   Dim sample as new Apple  
3   sample.name = Range ("A2")  
4   sample.description = Range ("B2")  
5   ...  
6   response = service.insertSample(sample )  
7   range ("O2") = response  
8 End sub
```

**Figure 7-3 inserting data from spreadsheet into the database**

The application example demonstrates the simplicity of creating client applications. Only a small amount of code is required to add spreadsheet data to the database. The EUD does not

have to develop validation and database access code, as this is performed by the web service and appropriate error messages are returned. The toolkit and web service make the application smaller and simpler. In addition, if end users would like to insert several samples, a loop statement can be used to read the values in every row and submit these records to the API for inserting.

### 7.3.2 Inserting data across tables

The method *insertAssay* stores associated sample objects with an assay object in the framework database. An example of an application that does this is shown in Figure 7-4. The spreadsheet provides data entry fields for end users to fill in with assay information. Two combo box controls allow users to choose two sample objects to be associated with an assay object.

There are five steps:

1. Initialise individual level Assay objects, such as AppleDna object (shown in Line 2).
2. Collect the Assay attribute values from the spreadsheet and assign them to the object.
3. Use the *findByType* method to retrieve onion samples from the database (Line 6) and Lines 7-8 add onion samples to the combo boxes using a helper method *addObjectsToControl*.

The toolkit provides the *addObjectToControl* helper method. It displays object names in a dropdown list (in the combo boxes). This method helps EUDs populate different types of domain objects into a combo box. To use the method, EUDs need to pass three parameters: a object collection, a target combo box control and a required attribute displayed in the dropdown list. EUDs can pass different parameters to display different attributes, such as the sample description.

4. Create an array list and add the selected objects to it Lines 10-12.
5. Finally, the *insertAssay* method is used to pass the Assay object and its associated sample objects to the web services (Line 14) and the returned message is displayed (Line 15).

	A	B
1	<b>Name</b>	OnionPcrAssay001
2	<b>PCR Lable</b>	PCR0001
3	<b>Description</b>	This is Onion PCR assay
9	...	
16	<b>Select Sample A</b>	onion001
17		
18	<b>Select Sample B</b>	onion002
19		Submit Assay
20		
21	<b>Response</b>	The assay is inserted

```

1 Sub insertAssay()
2   Dim assay As New OnionAssay
3   assay.name = Range("B1")
4   assay.lable = Range("B2")
5   ...
6   'initialise the Sample combo boxes
7   samples = service.findByType("Onion")
8   addObjectsToControl (samples, ComboBox1.Object, name)
9   addObjectsToControl (samples, ComboBox2.Object, name)
10
11 'create an array to include sample ids , and add two sample ids to the array
12 Dim sampleIds As New ArrayList
13 sampleIds.Add (ComboBox1.Value)
14 sampleIds.Add (Combo.Box2.Value)
15 ...
16 'insert assay and samples
17 response = service.insertAssay(assay, sampleIds)
18 Range ("B21") = response
19 End Sub

```

**Figure 7-4 inserting assay objects with assoiated samples**

### 7.3.3 Retrieving and updating data

An example of code to demonstrate how to retrieve objects from the database is shown in Figure 7-5. Line 2 shows how to retrieve a sample object with an id of 1. After the sample is retrieved from the framework database, the sample attributes can be displayed in an Excel spreadsheet (shown in Lines 3 -4).

1	Sub retrieveSample()
2	sample = service.findSamplebyid(1)
3	Range ("A2") = sample.name
4	Range ("B2") = sample.description
	...
5	End Sub

**Figure 7-5 Example of retrieving a single sample into an Excel spreadsheet**

Once an object is retrieved from the database, the updateSample method can be used to send the updated object back to the database. The syntax of this method is similar to insertSample but it requires that the object already exists in the framework database.

If it is necessary to retrieve all samples for further processing (such as bulk updating), EUDs can use the findByType method. The method requires the object type to be passed as an input parameter. Figure 7-6 shows how to use findByType to retrieve all Apple objects and display them in an Excel spreadsheet. Each loop is used to get each sample from the collection and display all the samples attributes in a separate row (shown in Lines 3-7).

1	<b>samples = service.findByType("Apple")</b>
2	row = 1
3	For Each sample In samples
4	Range ("A" & row)= sample.name
5	Range ("B " & row)= sample.description
	...
6	row = row + 1
7	Next

**Figure 7-6 Example of retrieving all samples for further processing**

### 7.3.4 Retrieving associated classes

It is a common requirement to retrieve data from associated classes. For example, after retrieving an assay object from the framework database, it may be necessary to retrieve samples associated with the assay. Figure 7-7 shows an example of this. The findAssayById method (Lines 2-3) is used to get a assay object and its associated samples. A loop is used to display information about each sample (Lines 5-10).

1	Sub retrieveAssay()
2	<b>assay = service.findAssaybyId(1)</b>
3	<b>samples = assay.samples</b>
4	Range ("A1") = assay.name
	Range ("B1") = assay.description
4	...
5	row = 1
6	For Each sample In samples
7	Range ("C" & row)= sample.name
8	Range ("D" & row)= sample.description
	...
9	row = row + 1
10	Next
11	End Sub

**Figure 7-7 Example of retrieving an assay and its samples**

## 7.4 End user application generator

Section 7.3 described example code that demonstrated how to use the toolkit to insert and retrieve data to and from the database. All applications using the toolkit will have a similar structure. Two major differences across client applications will be defining what individual level objects are used (Line 2 of Figure 7-8) and ensuring that each individual level object has different attributes (Line 6). However, the rest of code is almost identical; for example the code to set framework and organisational level attributes (Lines 3-5) and calling the insertSample method (Line 7). Because the applications all have a similar structure, it is possible to automate the generation of a skeleton application based on the individual level classes used.

	<b>Onion data entry application</b>	<b>Apple data entry application</b>
1	Sub insertOnion()	Sub insertApple()
2	Dim sample as new <b>Onion</b>	Dim sample as new <b>Apple</b>
	sample.name = "Onion001"	sample.name = "apple001"
3	sample.description = "onion sample"	sample.description = "apple sample"
4	sample.location = "Room1"	sample.location = "Room2"
5	...	...
	<b>sample.singleOrBulk = "single"</b>	<b>sample.color = "red"</b>
6	res = service.insertSample(sample )	res = service.insertSample(sample )
7	end sub	end sub

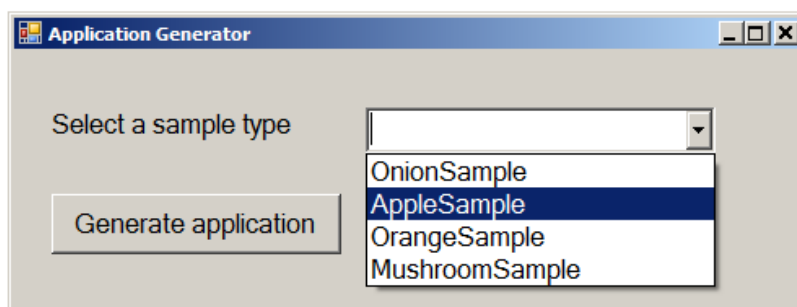
**Figure 7-8 Comparson between the Onion and Apple data entry applications**



### 7.4.1 Application generator

The application generator is provided to make end user application development even simpler. Framework providers can write a different generator to generate different types of applications, such as Office applications, web applications or smart phone applications. At PFR, we implemented an Excel application generator; Figure 7-9 shows the generator, which lists all individual level types, (such as Onion, Apple and Mushroom). An EUD only needs to select the type and a prototype data entry application is then automatically generated. In future, if end users require web based applications or smart phone applications to manage their data, the generator can be extended to generate different types of applications to meet specific needs.

The generator can dramatically reduce the development effort and learning curve to use the framework. EUDs can generate a prototype application in a matter of minutes. They do not need to learn how to use the toolkit methods until they need to customise the generated applications. At PFR, we provided instructions (see Appendix 2) to guide EUDs to use the application generator and help them understand the code generated.



**Figure 7-9 Application generator**

### 7.4.2 Generated user interface and code

The generated application includes an empty data entry spreadsheet with appropriate column headers, shown in Row 1 of Figure 7-10. The generator retrieves the domain class attributes from the VBA toolkit and these attributes are then used to generate the appropriate column headers. Users can then fill sample records under the header, but this can easily be adapted.

	A	B	C	D	E	F	O
1	name	description	entryDate	comment	contact	....	responseMessage
2							
3							
4							

**Figure 7-10 Generated Excel data entry template**

Users can fill in data entry fields in the spreadsheet and run the generated Macro in order to insert data into the framework database.

The insertApple Macro (Figure 7-11) is also automatically generated to load input data from the spreadsheet user interface to the database via the toolkit.

1. The Macro initialises an Apple object (Line 1), collects the attribute values from the worksheet cells and assigns them to the sample object (from Lines 5 - 8).
2. The Macro calls the toolkit insertSample method, passing the sample Apple object for loading to the database (Line 9).
3. The web service processes the request and sends back a response message to be displayed (Line10).
4. The Macro allows users to loads multiple records to the database; Line 4 shows the loop continues to process every row containing data.

Example code	Comment
<pre> 1 Sub insertApple() 2   Dim sample As New Apple 3   rownumber = 2 4   Do While Len(Range("A" &amp; rownumber).Formula) &gt; 0 5       sample.name = Range("A" &amp; rownumber) 6       sample.description = Range("B" &amp; rownumber) 7       sample.entryDate = Range("C" &amp; rownumber) 8       sample.comment = Range("D" &amp; rownumber) 9       ... 10      res = service.insertSample(sample) 11      Range ("O" &amp; rownumber) = res 12      rownumber = rownumber+ 1 13  Loop 14 End Sub </pre>	<p><i>create a new apple object</i></p> <p><i>start loading input data from row 2</i></p> <p><i>loop to read every row with a value in column A</i></p> <p><i>collect values from worksheet cells and assign them to the sample object</i></p> <p><i>call the insertSample function to insert the record into the database</i></p> <p><i>display the response message returned from the web service</i></p> <p><i>process the next row</i></p>

**Figure 7-11 Generated code for loading Apple data**

The application generator automatically generates the data entry application, which can dramatically reduce end user development effort. In the future, generating data retrieving and updating applications could also be provided to further simplify application development.

## 7.5 Summary

The framework provides a domain model that allows for modifications and extensions. In order to enable non-professional developers to customise and extend the model with minimum effort, we provide the framework configuration tool and code generator to assist framework managers to easily customise domain models based on their specific needs. Specific object attributes and rules can be defined in a simple text based configuration template, and then the framework code generator automates the domain model and database generation based on the configuration templates.

To make sure the web service API methods can be used by EUDs, the toolkit generator can generate and extend the client development toolkit for EUDs. This simplifies the code required by end users to call the web service. EUDs need to only include the toolkit methods in their commonly used applications. They do not need to understand the web service calling mechanism. At PFR, a VBA toolkit generator is provided to generate VBA toolkits for Excel. This approach can also be employed to generate different types of language specific toolkit based on EUD's requirements.

In order to make the application development even simpler, an application generator can be developed to create prototype applications that EUDs can then customise. The generator approach can be applied to generate different types of applications based on user's requirements. At PFR we developed this for VBA-generating Excel-based applications. This significantly reduced the effort of client applications. It makes the framework approach available for EUDs to create their data management applications without coding experiences.

## Chapter 8 Framework User Trials

We implemented a framework to help end user developers (EUDs) create a database to manage different types of plant samples and associated assays at the New Zealand Institute for Plant & Food Research (NZPFR). As described in Chapter 7, the framework configuration tools were provided to help framework managers customise the domain model and implement the database and client application development toolkits. In addition, an application generator was provided to generate skeleton client applications to manage the data via the web service;

The primary guideline for the framework was that it should have a data model designed for a specific domain that allowed for modifications and extensions. In order to evaluate the framework approach, we carried out two sets of EUDs trials:

- **The framework manager trials** looked at how well the framework approach assisted framework managers to customise and extend the database.
- **The EUD trials** looked at how well the approach supported EUDs to create end user applications.

The trials involved several tasks for framework managers and EUDs to complete. After the user trials, we interviewed participants to get detailed feedback. We summarise the trial tasks, results and interview feedback in this chapter.

### 8.1 Framework Manager User Trials

Trial tasks were designed based on organisations' real world requirements. During the exploratory case study (discussed in Section 4.1), we found that end user developers often need to modify and extend their database tables. Therefore, two tasks were developed to test how well the framework supports framework managers to modify and extend a data model:

1. Extend an existing Apple sample type with additional data and validations. (Modify and extend an existing class/table)
2. Add a new type of sample for Oranges (including data and validations). (Extend database by adding a new class/table)

Eight participants acting as framework managers were involved in the trials. The participants, who have a variety of programming skills, came from four different research institutions. A framework manager (responsible for modifying or extending the database model) could be a professional developer or a knowledgeable end user developer. In most cases, the framework managers are likely to be knowledgeable end user developers. Therefore, we selected three professional developers (participant 1-3) and four non-professional developers (participant 4, 6, 7 and 8) as shown in Table 8-1.

In order to measure how the framework helped end users, we recorded the time spent on each task. Details are discussed in Section 8.1.3.

**Table 8-1 Participants in the framework manager trial**

Participant	Job role	Programming experience
1	Professional developer	Experienced PHP, Python
2	Professional developer	Seasoned ASP .NET
3	Professional developer	Seasoned C#
4	Genetic researcher	Occasional VBA
5	Genetic researcher	Frequent VBA
6	Genetic researcher	Novice PHP/SQL
7	Genetic researcher	Novice PHP/SQL
8	Chemistry PhD student	LaTeX scripts

Participants were provided with:

- A general set of instructions explaining how the schema could be modified (Appendix 2).
- An existing schema for an Apple sample.
- A framework configuration file for the Apple schema.
- A configuration tool (for updating the schema based on an altered configuration file).
- A toolkit generator to generate a VBA development toolkit for Excel development. EUDs are able to use the toolkit to manage the database via the web service API.
- An application generator to generate a skeleton application based on the updated schema.

- An installer to install the VBA development toolkit and application generator.  
Participants only need to double click on the installer and the installer then automatically installs the toolkit and generator.
- Some sample data to test data entry functionality of the generated application.

Participants were given a set of specific tasks (see Appendix 4) to carry out. These are described in the next sections.

### 8.1.1 Framework Manager Trial Task 1: Adapting Schema for Existing Samples

In this task, participants needed to extend an existing Apple sample. The existing Apple configuration file, Apple.csv (see Table 8-2) was provided for participants to customise.

**Table 8-2 Apple sample configuration template provided to participants**

class	Apple	PfrSample	
field	Measure	Double	
rule	Min	5	measure must be higher or equal to 5
rule	Max	20	measure must be lower or equal to 20
field	Colour	String	
rule	Pattern	red green	colour must be either red or green
field	population	String	
field	Size	Double	

The main steps in Task1 are described below (details are in Appendix 4). Participants were asked to:

1. Add two new fields to the configuration file: size (double) and quality (string). A validation rule for the field size needed to be defined with appropriate error messages.  
The additional information the participants need to add to the configuration file is shown in Table 8-3 .

**Table 8-3 Participant defined attributes and related validation rules**

field	Size	Double	
rule	Min	5	size must be higher or equal to 5
rule	Max	20	size must be lower or equal to 20
field	Quality	String	

2. Upload the configuration file and generate framework database tables using the configuration tool.
3. Install the client application toolkit and application generator using the installer.
4. Use the application generator to create a prototype Excel data entry application in order to test data loading functionality.

Use the sample data provided to test the application updates the new database correctly. (Data provided is passed into columns A to E in Figure 8-1, and response messages after uploading are generated in Column O).

	A	B	C	D	E	F	O
1	<b>name</b>	<b>description</b>	<b>entryDate</b>	<b>comment</b>	<b>contact</b>	<b>...</b>	<b>responseMessage</b>
2	Appledna001	Apple DNA	2010-09-02		Tom		The sample has been inserted
3	Appledna002	Apple DNA	2010-09-03		Tom		The sample has been inserted
4	...						

**Figure 8-1 Generated client applications**

### 8.1.2 Framework Manager Trial Task 2: Extending Schema for New Samples

In this task, participants needed to extend the framework to incorporate a new type of sample (Orange). Participants were required create a new configuration file for an Orange type with two fields and validation rules. The file the participants had to design and create is shown in Table 8-4. Participants then had to use the tools provided to update the database schema and generate the client applications (the same as Steps 2-4 in Task 1).

**Table 8-4 Participant defined configuration file for Task 2**

class	Orange	PfrSample	
field	material	String	
rule	Pattern	MaterialA MaterialB	Material must be either MaterialA or MaterialB
field	weight	Double	
rule	Min	5	Measure must be higher or equal to 5
rule	Max	20	Measure must be lower or equal to 20

### 8.1.3 Framework Manager Trail Results

Eight participants completed these two trial tasks. They all successfully defined the necessary configuration files and generated the framework, database and client application. The average time spent on Task 1 was 20 minutes and on Task 2 was 19 minutes (shown in Table 8-5).

**Table 8-5 The time spend on trial tasks**

Participant	Time spent on Framework Task 1 (minutes)	Time spent on Framework Task 2 (minutes)	
1	20	N/A	Professional developers
2	25	15	
3	23	20	
4	25	15	
5	N/A	23	Non-professional developers
6	12	10	
7	16	34	
8	22	13	
Average	20	19	

Notes:

- a. Participant 1 spent 20 minutes to finish Task 1 and did not do Task 2. However, the participant thought Task 2 was similar to Task 1 and was confident he would complete it successfully.
  - b. Participant 5 (a genetic scientist) was originally asked to carry out the end user development trials (described in the next section). However, he was interested in how to extend the framework so he also did framework trial Task 2 after completing the end user trials.
- Task 1 results:

All participants who attempted this task carried it out successfully. Only Participant 3 made a mistake in this task, forgetting to define an error message for the validation rules of the field *size*. As a result, an invalid Size value in the test data was inserted into the database and no error messages were generated. The participant recommended that if developers do not define error messages, the system should supply default error messages.



- Task 2 results:

All participants who attempted this task carried it out successfully. Only Participant 7 made a mistake in this task by putting a space in the class name in the configuration file. The development tool pointed out this error when the system was deploying and testing the web service. The participant was able to correct the error and the update database with minor assistance. This resolution of this type of problem would be to include a tool to check the configuration file before uploading it to the server.

In addition, two participants voluntarily defined new sample types based on their specific needs for Hieracium and Sheep data. Although the framework was originally designed for managing plant samples, it can also be easily adapted to manage many similar types of laboratory samples.

#### **8.1.4 Interview feedback summary**

We prepared questionnaires (see Appendix 6) to elicit users' opinions of their user trial experience. The questions covered framework flexibility, simplicity and learning curve. We interviewed participants to ask these questions. Participants' answers and feedback are summarised in the sections below:

- Flexibility

Participants told us that they are able to see the value of single database that could easily be extended to cater for a variety of requirements. For example, one of participants said, "*The framework can be applied to record different types of genetic research laboratory samples, such as DNA, RNA and protein samples*".

Participants told us they were immediately able to see how the framework could be used for their own data management problems. Two participants even wanted to immediately extend the database for their specific needs. For example, one participant (genetic scientist) defined new sample data type based on his real work data. After generating the database, he decided to include another new field. He spent only 10 minutes to redefine the configuration file and regenerate the database. This illustrates that the framework approach can help EUDs quickly update and extend databases.

One participant (a professional developer) was working on a project to provide Excel data entry templates for scientists to add data to a centralised database. He thought the framework

approach could be used to customise the database and develop different data entry applications as needed.

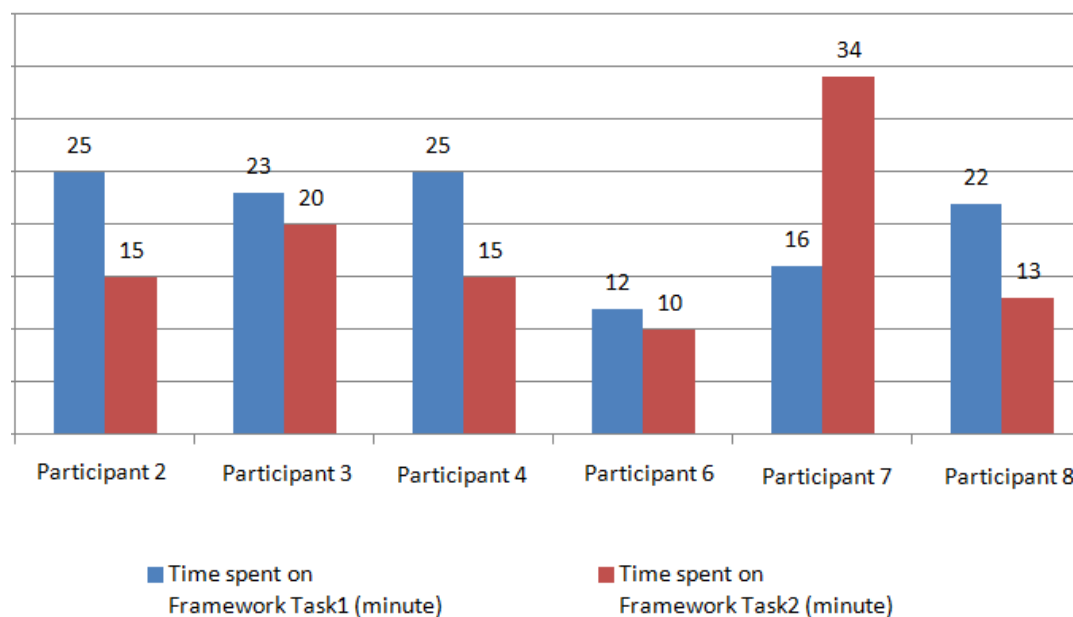
- Simplicity

Participants told us that they found it easy to modify and extend the database to meet specific requirements. They thought it was much simpler than setting up database system from scratch. One participant said, “*It is quite easy to get what you want (such as specific entities, attributes and validation rules)*”. All participants commented that they would like to use the system in their next data management project.

- Learning curve

All participants thought that the general instructions provided clearly explained how to define configuration files and use the tools to create the database and generate prototype client applications. They all felt confident they could follow the instructions to create their own databases and applications.

Figure 8-2 compares the time spent on Task 1 and Task 2 for the six participants who completed both tasks. All but one participant spent less time adding a new sample type in Task 2 than modifying an existing type in Task1. Once participants had figured out how to use the configuration file and tools, the time spent to customise the framework and database reduced.



**Figure 8-2 Time spend on trial tasks**

Note: Participant 7 (Table 8-5) spent 34 minutes on trial Task 2, because the participant made an error that required correcting and rebuilding. All other participants were quicker completing the second task.

- Suggestions

One participant suggested the instructions for validation rules could be improved. He suggested more examples be added to show users how to define different types of validation rules (e.g. defining a regular expression to ensure the Sample name starts with a specified prefix.)

## **8.2 EUD trials**

The key goal of the EUD trials was to evaluate how well the client development toolkits and application generator assisted EUDs to create and extend data management applications with minimal effort.

Participants were provided with:

- A general set of instructions explaining how to use the application toolkit and application generator to develop and customise applications (Appendix 3).
- A framework set up by a framework manager.
- A VBA toolkit and application generator installed on a windows PC. The generator is able to generate an Excel based data entry application to insert data into the database via the VBA toolkit.
- The application generator (to generate a skeleton application based on the database schema).
- Some sample data to test data entry functionality of the generated application.

We had two end user development trial tasks for participants (described in Appendix 5):

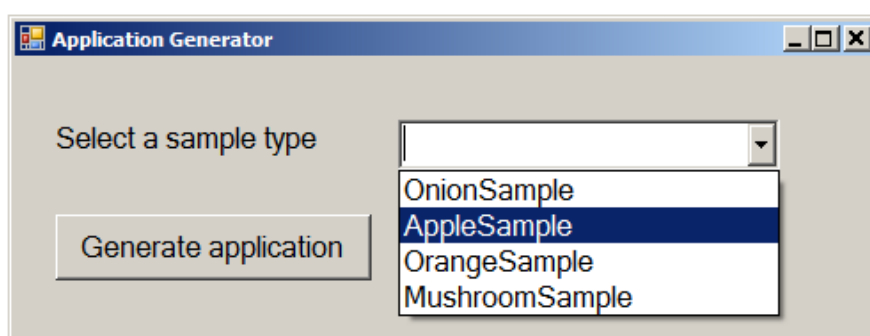
1. Create a new Excel based data entry application (for loading Orange sample data) using the tools provided.
2. Customise the Excel data entry application (generated in Task 1). Participants needed to alter the data entry layout of the Excel application and modify the VBA code to correctly load the data.

For this trial we wanted to test non-professional EUDs to see if they were able to modify the applications. Participants 4 to 8 (in Table 8 1) were asked them to carry out these EUD trials. Participants 4, 6, 7 and 8 had previously completed the framework trials while participant 5 did this trial first. The order of participating in the trials may have some influence on the final times as the end users doing this trial second would have seen how the data model was set up. There is no obvious difference in the actual times resulting from the order of the trials but this would need to be verified with a larger trial.

### 8.2.1 End User Development Trial Task1: Generating application

The trial Task 1 (details shown in Appendix 5) was designed to evaluate the usefulness of the application generator and the instructions to create client applications. The participants were asked to use the application generator (see Figure 8-3) to create an Excel-based data entry application. Participants need to carry out the following three steps:

1. Open the application generator
2. Select the Orange type from a drop down list of all sample types in the framework
3. Click generate application button in order to generate Orange data entry application



**Figure 8-3 Application generator**

The generator automates the generation of an Excel prototype application. It includes a Worksheet (see Figure 8-4) and a VBA macro (in this case “insertOrange”) for bulk inserting of data.

	A	B	C	D	E	F	G	H	O
1	sampleName	description	entryDate	comment	contact	labBookRef	location	...	responseMessage
2									
3									
4									
5									
6									

### Figure 8-4 Generated Excel data entry worksheet

We provided a separate worksheet that contained example data for some Orange samples. The participants were required to copy that data and paste it in the appropriate columns in the generated worksheet. After that, they needed to run the “insertOrange” macro to insert the data into the database.

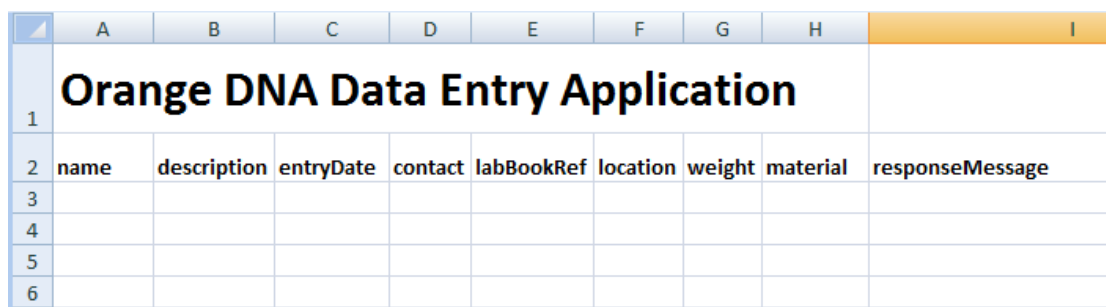
### 8.2.2 End User Development Trial Task 2: Customising the application

Task 2 evaluates the time and effort required to customise the generated applications.

Participants were asked to customise the generated Orange data entry application following main steps below:

#### 1. Customise Excel worksheet (UI)

The participants were asked to remove some columns and add a header for the data entry spreadsheet in Row 1. The generated data entry worksheet included 13 columns, the participants were asked to remove five of them. The final worksheet layout is shown in Figure 8-5, below.



	A	B	C	D	E	F	G	H	I
1	<b>Orange DNA Data Entry Application</b>								
2	name	description	entryDate	contact	labBookRef	location	weight	material	responseMessage
3									
4									
5									
6									

Figure 8-5 Customised date entry worksheet

#### 2. Modify VBA code for the customised worksheet

Participants needed to read instruction B for EUDs (in Appendix 3) in order to understand the code generated and use of the toolkit. The generated code copies the values from the worksheet columns and rows to a VBA Orange object and calls the toolkit method to save the data. In order to alter the code to match the customised user interface, participants completed these steps.

- Alter the row numbers from two to three to take account of the inserted header row.

- Change the code to copy the Orange data from appropriate columns from the customised worksheet. Five data entry fields had been removed from the original spreadsheet, so participants had to remove the code for copying these fields.
- Modify the code to display the response message in the database response Column I. Since five columns had been removed from the spreadsheet, the last column should be Column I.
- Add a button to the worksheet and assign the insertOrange macro to the button.
- Copy the sample data from the provided worksheet and paste the data to the appropriate columns in the customised worksheet. They then needed to insert the data using the new add data button.

The final customised code is represented in Table 8-6. The lines that are crossed out needed to be removed from the generated code and the lines with + needed to be added by the participant.

**Table 8-6 Customised code for End User Development Task 2**

	Code	Comment
-	Sub insertOrange()  Dim sample As New Orange  rownumber = 2	Altered the row number from 2 to 3
+	rownumber = 3  Do While Len(Range("A" & rownumber).Formula) > 0 sample.Name = Range("A" & rownumber) sample.Description = Range("B" & rownumber) sample.EntryDate = Range("C" & rownumber) sample.Comment = Range ("D" & rownumber) sample.Cntact = Range("D" & rownumber) sample.labBookRef = Range("E" & rownumber) sample.Location = Range("F" & rownumber) sample.concentration = Range("H" & rownumber) sample.species = Range("I" & rownumber) sample.Keywords = Range("J" & rownumber) sample.Method = Range("K" & rownumber) sample.Weight = Range("G" & rownumber) sample.material = Range("H" & rownumber) res = service.insertSample(sample)  Range("N" & rownumber) = res	Altered the columns to collect required values
+	Range("I" & rownumber) = res  rownumber = rownumber + 1  Loop End Sub	Altered the column to display database response messages

### 8.2.3 EUD Trials Results

All participants successfully completed both tasks in a short time. The times spent on trial tasks are summarised in Table 8-7. The average time was six minutes for Task1 and 18 minutes for Task 2.

**Table 8-7 Time spend on trial tasks**

Participant	Time spent on Task 1 (minute)	Time spent on Task 2 (minute)
4	7	25
5	9	15
6	5	15
7	6	15
8	5	20
Average	6	18

### 8.2.4 Interview feedback summary

We prepared questionnaires (see Appendix 7) designed to collect users' feedback about the EUD trials and the end user development approach. It included questions about the application development tools (toolkit and application generator), application flexibility and suggestions. We interviewed participants to ask these questions. Participants' answers and feedback are summarised in the sections below:

- **Users' experiences of the development tools**

All participants told us that it was easy to use the application toolkit methods to manage their data. Participants said, "*The toolkit methods are very much aimed at people who have basic programming experiences, e.g. VBA, and it does not require database modelling and SQL skills*". The application toolkit and generator allowed the EUDs to create applications from the framework with minimal effort.

Many participants commented that "*with the application toolkit and generator, it is much easier and quicker to develop database applications*". The participants thought the generator allowed them to create a basic data entry application without much coding. Participants thought that most of their colleagues would be able to use the tools and instructions to generate prototype applications.



- **Instructions**

All participants thought that both sets of instructions were very clear. They thought they would be able to develop their next data management application with the instructions and the development tools.

- **Suggestions**

One participant suggested some tips to include in the instructions for EUDs. For example, if some attributes were not needed, they do not have to be removed from the generated example application. The unnecessary columns can be hidden and the related code can be commented out. If these columns/fields were needed in the future, they can be unhidden and the code can be reactivated.

### **8.3 Summary**

The framework trials were designed to test how well the framework supported framework managers and EUDs to modify and extend their databases and applications. Eight participants acting as framework managers successfully completed the framework trials. The results show the framework and associated tools allow a database to be modified and extended to meet different requirements.

The EUD trials evaluated whether EUDs could use the toolkit and application to create and modify applications from an existing framework schema. We designed two end user development trial tasks and five participants successfully completed the trials. The results show that the toolkit and generator are able to help EUDs to generate and modify applications with minimum effort.

More importantly, participants were able to immediately see how the framework could be applied to their own problems. Two of the participants customised the framework based on their real data during the trial. The participants were also able to correct their initial implementation errors, such as missing the required attributes. The result illustrates that non-professional developers are able to confidently use our framework tools to generate and customise their applications.

## Chapter 9 Framework Evaluation

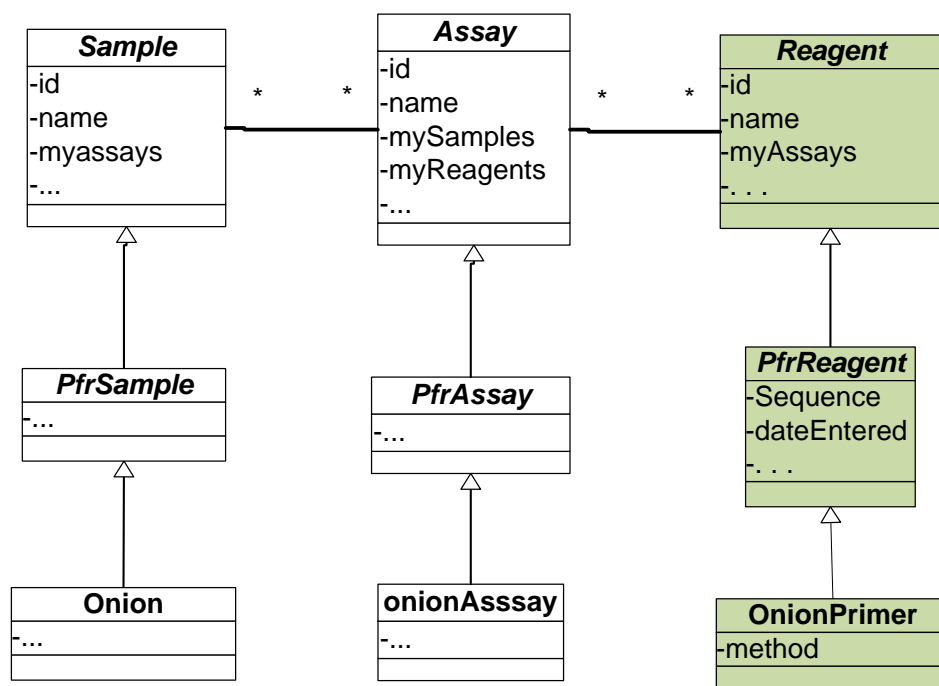
We have implemented a framework that supports end user developers (EUDs) in evolving data management applications. The framework approach incorporates a flexible data store, data management API (web service) and framework development tools (e.g. framework configuration tool, toolkit generator and end user application generator). The framework approach was evaluated by framework managers and in EUD trials. The results show that the framework approach helps end users to easily customise databases for their specific needs. The trials included customising individual level entities and developing client applications to manage the database. In this chapter, we will discuss aspects of framework extension (Section 9.1) and generalisation (Section 9.2) not covered in the user trials.

### 9.1 Framework extension

The guidelines (defined in Section 4.2.2) pointed out that the framework should allow for modifications and extensions without affecting the existing data and applications. If the requirements are changed over time new entities will necessitate the addition of new framework base classes.

The framework manager trials evaluated the ease of adding and modifying individual level classes. However, the trials did not test the ability to include new base level classes. For example, in addition to the base level classes Sample and Assay we might require a new class, such as Reagent.

In this section, we evaluate how easily the framework could be extended to include new base class requirements. At NZPFR, some user groups need to record information about the reagents needed for the experiments. We will use an example of adding a Reagent entity to the framework to illustrate the process involved. Reagents are chemical materials needed for the assays. The relationship between Assay and Reagent is many-to-many (see Figure 9-1).



**Figure 9-1 Framework base class extension**

To extend the framework for a new entity requires input from the framework provider to update the framework data model and web service. The general steps of how to add a new entity in the framework are shown in Table 9-1.

**Table 9-1 Effort needed to include a new data type in the framework**

Framework provider		Actions required
Add new classes	Base level	Define a base class definitions in a new ORM file
	Organisational level	Add Org. level class in the ORM file
Add new constraints	Base level	Define constraints for the base class in a new constraint file
	Organisational level	Add Org. level constraints in the constraint file
Extend the web service	Validator & DAO	No changes
	Web service	Add new methods to manage the new entity. Add a new method to allow EUDs to store assay and Reagent associations in the database

With the above additions completed, individual level classes can be customised and toolkits can be regenerated by framework managers (as described in Section 7.2)

Table 9-2 shows the tasks needed to include a new entity in the end user applications. Once the end user development toolkit is regenerated by framework managers, EUDs are able to use the toolkit and the existing application generator to generate and customise end user applications, as explained in Section 7.3. The revised framework will not require changes to be made to existing applications (except to make use of the new entities).

The tasks in Table 9-2 make use of the framework tool described in Section 7.2

**Table 9-2 Tasks needed to include a new entity in the end user applications**

Framework manager		Actions required
<b>Customise classes &amp; constraints</b>	Individual level	Use the framework development tool (described in Section 7.2) to customise individual level class in the ORM file  Use the framework tool to customise individual level constraints in the constant file
<b>Update Database</b>	Tables	Use the framework tool to generate the database
<b>Update toolkit</b>	Toolkit Methods	Use the framework tool automatically generate the toolkit

In the following sections we will describe the tasks described above in more detail.

### 9.1.1 Add base level and organisational level classes

This section describes how to extend the framework to include new classes (e.g. Reagent) in the example LIMS framework. Framework providers need to follow these steps:

- **Define base level Reagent in a new ORM file**

This can be easily done by creating a new ORM file. Section 6.2 explained how framework providers can define class details (field names and types), ORM details (table field names, types) and relationship mapping details in the ORM file. Section 6.2.5 discussed how framework providers can also define the constraints. The Reagent ORM.xml is shown in Figure 9-2: Line 1 defines the mapping of the class (Reagent) to the database table (REAGENT). Line 2 maps the reagent id as the primary key of the Reagent table. This also

defines the id attributes and type. Line 3 maps a class attribute (name) to a table column with an appropriate data type (string).

Association mapping details are shown in Lines 4-7. They map the associations between classes (assays and reagents). In the Reagent class the attribute myAssays represent a collection of assay objects associated with a reagent object. The attribute myAssays is mapped onto the intermediate Table REAGENT\_ASSAYS between REAGENT and ASSAY to store the combinations of sample and assay.

	<pre><u>&lt;-- Base class/table --&gt;</u> 1 &lt;class name="Reagent" table="REAGENT"&gt; 2   &lt;id name="id" type="int" column="REAGENT_ID"&gt;&lt;generator   class="native"/&gt;&lt;/id&gt; 3   &lt;property name="name" type="string" type="string"/&gt; 4   &lt;set name="myAssays" table="REAGENT_ASSAYS"&gt; 5     &lt;key column="REAGENT_ID"/&gt; 6     &lt;many-to-many class="Assay" column="ASSAY_ID"/&gt; 7   &lt;/set&gt;   . . .</pre>
--	--

**Figure 9-2 Example Reagent ORM file (base level)**

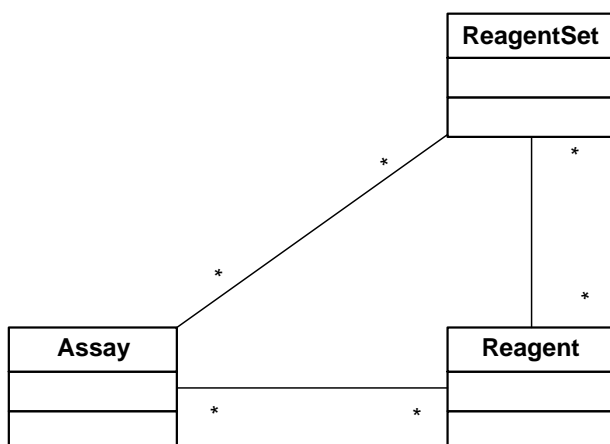
Similarly, associations will need to be defined in the Assay ORM file. For example, in the Assay class, an attribute myReagents represents a collection of reagent objects associated with an assay object. The new reagent functionality requires changes to the existing Assay Class and Table but the myReagents attribute is optional. If EUDs do not need to use reagents, they do not have to change their end user applications. In addition, cardinality constraints can be defined at the individual level to force users to supply reagent information (see Section 6.2.5). For example, mushroom assays could require reagents but onion assays do not require any reagent, so the onion application could remain unchanged.

- **Supporting Reagent Sets**

Section 4.3.2, we discussed how the Onion Group need to deal with Reagent sets (or primer sets). In order to support the concept of a ReagentSet, framework providers will need to include a new ReagentSet class that has a many-to-many relationship between both Assay and Reagent (see Figure 9-3). The many-to-many relationships can be defined in the same ORM file, as shown in Figure 9-2.

The model in Figure 9.3 allows an assay to have a number of reagents and/or reagent sets. The ReagentSet functionally should be optional so that if a research group does not need a ReagentSet, they do not have to update their applications. In addition, cardinality constraints (described in Section 6.2.5) can be defined to restrict user groups to using only ReagentSet

(cannot use Reagents directly). For example, the Onion Research group only needs a ReagentSet, so a cardinality constraint can be defined in the individual level onion Assay Class to disallow elements in the myReagents collection of the assay class.



**Figure 9-3 Adding a new ReagentSet class**

- **Add Organisational level reagent in the ORM file:**

The framework provider should work with framework managers to design the organisational level classes. They need to agree on issues such as which attributes are required for the entire organisation and define the organisation level class in an ORM file similar to the one shown in Figure 9-4 for organisational level class (PfrReagent). The “joined-subclass” in Line 14 is the syntax for implementing the one table per class mapping with inheritance between base, organisational and individual level classes, as described in Section 6.2.4 .The PfrReagent class with a sequence attribute (Line 16) is added in the ORM file to define Organisational level class.

14 15 16 	<pre> &lt;!-- Organizational level class/table --&gt;     &lt;joined-subclass name=" PfrReagent" table="PFRREAGENT"&gt;       &lt;key column="id"/&gt;       &lt;property name="sequence" type="string"/&gt;     . . . </pre>
--------------------	---

**Figure 9-4 Example Reagent ORM file (organisational level)**

### 9.1.2 Add new constraints

The validation logic for both base level and organisational level classes must be defined in the constraint configuration file. An example of ReagentConstraint.xml is shown in Figure 9-5.

- **Base level constraints**

Any base level constraints must be applicable for all reagents in the system. The rules defined in the base classes will be inherited by all the organisational individual level classes. Line 2 defines which attribute is going to be checked (name in the Reagent class in this case). Line 3

defines the validation constraints, e.g. NotNull in order to make sure the end user has supplied the required reagent name.

```
1 <!-- Base level constraints -->
2 <bean class="Reagent">
3   <field name="name">
4     <constraint annotation="javax.validation.constraints.NotNull"/>
5     <message>ERR: the reagent name is required</message>
6   </field> ...
```

**Figure 9-5 Constraint configuration file**

The constraint will be automatically inherited by the individual level classes (via the organisational class) ensuring sample names are always provided. Line 4 defines a message to inform end users the sample name is required.

- **Organisational level constraints**

The organisational constraints are also defined in the same constraint file. Validation rules at the organisational level should be applicable for all users within the particular organisation. For example, if the reagent sequence was always necessary at PFR a NotNull constraint could also be defined at the organisational level.

### 9.1.3 Extend the web service

Once the new classes, tables and constraints are defined in the configuration files, corresponding web service methods will be required to manage the new entity data. Framework providers need to create a set of web service methods in order to simplify the database access for EUDs.

Figure 9-6 shows an example of a web service method. Section 6.3 described how the DAO object and validator object are provided for developers to easily create the data management methods. Lines 2 to 7 show a simplified insertReagent method. It uses the validator to validate the reagent and use the DAO to save the data in the database. Other methods, such as update, delete and retrieval need to be provided in the same web service (see Section 6.3.5).

```
1 public class LimsService {
2   public insertReagent(Reagent reagent) {
3     //handle validation logic and catch errors
4     this.validator.validate(reagent);
5     //handle persistence logic
6     this.dao.save(reagent);
7   }
8   ...
9 }
```

### **Figure 9-6 Web service sample code**

In order to create associations between assay and reagent a new insertAssay method needs to be added to the web service. How to store associations between assays and other objects was discussed in Section 6.3.5. The method should take an assay object and a collection of ReagentIDs as input parameters in order to store the assay object and associated reagents in the database. The validator and DAO can be used to simply add this method within a few lines of code. In addition, the new method will only be used by those research groups with assays that require reagents and no code changes required for other groups' assays.

#### **9.1.4 Discussion of framework extension process**

Table 9-1 shows the entire process of extending the framework to include a new Reagent entity. Framework providers need to define the basic and organisation level classes in the configuration file and provide a web service for managing the newly added classes.

Framework managers can then use the configuration tool to customise the individual level classes and use the application generator to automate the data management application.

The framework extension does not affect any existing client applications (if users do not need the newly added functionality). Section 9.1.3 discussed how the framework can add new methods to the web service without altering existing web service methods. That means existing end user applications do not need to be updated if they do not require the new entities. Moreover, if EUDs need to update their applications to use the new functionality provided (e.g. include reagents in assays) they only need to add extra code to the existing application, such as dropdown list of reagents and a few lines of code to call the new methods.

More importantly, the framework was extended in a flexible manner. Section 9.1.1 introduced defining a many-to-many relationship between Assay and Reagent which allows some groups to require one reagent while others may require several reagents. In order to control the cardinality (described in Section 6.2.5), a constraint can be defined in the individual level class (e.g. Potato or Mushroom) to require the number of elements in the myReagents collection of the assay class.

Using the framework approach has four major benefits to handle changing requirements:

1. Adding a new framework base class (Reagent) does not interfere with the existing data management applications. For example, groups which do not require reagent information need not alter their Sample and Assay data entry applications even though new entities may be added to the framework.



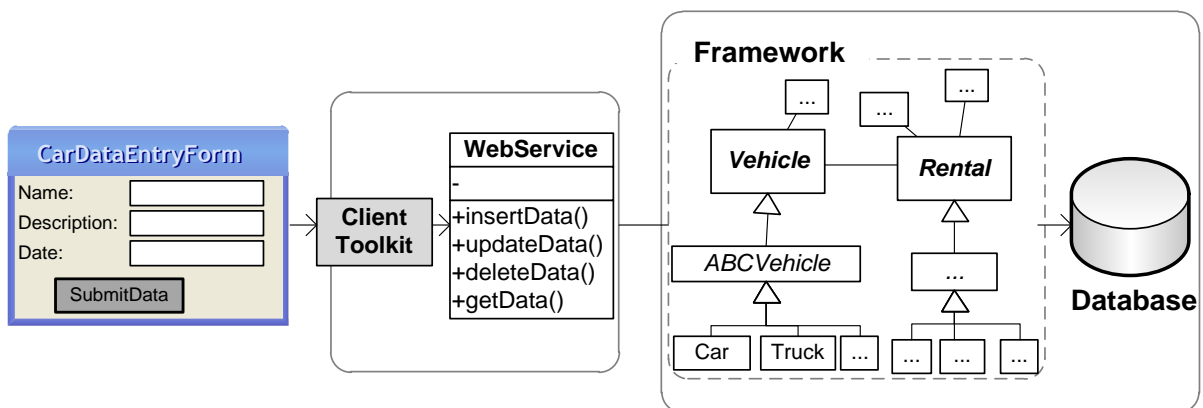
2. It is easier than manually redesigning and implementing the database and applications. The framework providers only need to define the configuration files and code the web service methods. The framework configuration tool generates the database and applications based on these definitions. In addition, it is easier to manage and maintain all similar data (laboratory information in this case) stored in the single database rather than scattered in different ones.
3. It is easier to extend the system than developing new database functionality from scratch. The details of data access and data validation are encapsulated in the web service objects. As a result, only a few lines of code need to be added to the web service to provide the additional data management functionality.
4. It is easy to deliver the new functionality for EUDs to create applications. Once the new base level classes are added to the framework, framework managers can customise individual level classes based on their specific requirements. In addition, the system will automatically generate the client development toolkit for EUDs to work with.

## 9.2 Framework generalisation

The framework approach can also be applied to a different application domain. In our study, the example framework was implemented for a genetic research laboratory domain for managing samples and experimental data. However, the same approach and system could be applied in different application domains. For example, a framework could be designed to support managing car or truck rental data. The steps described below are required to create a new framework for vehicle rental information management (Figure 9-7 illustrates the completed system).

1. **Define the middle way:** The middle way must be identified before creating the framework base classes. An appropriate middle way could be to concentrate on rentals for wheeled vehicles, i.e. allow users to record different types of information for cars, trucks, vans and buses. It could exclude other types of vehicles, such as trains, boats or aircraft which are not typically handled by small vehicle rental companies.
2. **Create the domain model:** The framework base level classes should be designed based on the middle way point. For example, a Vehicle base class associated with a Rental base class.

- a. Framework providers can create configuration files for vehicle rentals. After the configuration is created, the system will automatically generate the classes and tables based on the definitions in the configuration file. The framework providers also provide an organisational level and individual level class configuration template for framework managers.
3. **Provide web service:** The web service is used to help EUDs in Vehicle and Rental information management. Since all data access and validation code are encapsulated in the DAO and Validator (discussed in Section 6.3), the web service implementation becomes smaller and simpler. IT professionals need only write a few lines of code to complete the web service for inserting an object into the database.
  4. **Provide toolkit and application generator:** Framework providers also need to provide the client application development toolkit and application generators. This can be done using steps similar to those described in Sections 7.2.3 and 7.4.1. Appropriate documentation for the toolkit methods also needs to be provided for EUDs.



**Figure 9-7 The framework approach applied to a vehicle rental application domain**

Framework managers can use the existing framework development tools (described in Sections 7.2.1 and 7.2.2) to customise the individual level classes. EUDs will be able to use the application generator to create skeleton data management applications

### 9.3 Summary

It is important to support EUDs to evolve their applications as new data entities or functionality are required. The framework must be able to be extended to meet new requirements without disturbing existing applications. In this chapter we have demonstrated the extensibility of the LIMS framework without disrupting existing applications.

The framework provider needs to define base level classes and provide the web service for managing the newly added classes. Framework managers and EUDs are then able to use the configuration tool for customising their individual level classes and use the application generator to automate the data management application. More importantly, framework extension does not affect existing client applications, if they do not require the newly added functionality.

The example LIMS framework extension demonstrates the ability to support flexible end user development. It was also shown how the framework approach could be applied to different problem domains (such as vehicle rentals) to help other EUDs develop more flexible applications.

## Chapter 10 Future work

In the previous chapters we proposed and evaluated a framework approach to support end user developers (EUDs) to more easily create and evolve database systems. In order to demonstrate our framework flexibility, we implemented and evaluated an example LIMS framework to support EUDs to customise their databases. In this chapter, we discuss implementation challenges, such as performance, system reliability, transaction, security and further work required to provide a more robust and efficient system,

### 10.1 Database and web services performance

The framework we developed used the class per table mapping strategy to map the domain objects to database entities. That means that the queries must join tables from each of the three levels in the class hierarchy in order to insert or retrieve a complete logical record. These joins will obviously add some overhead to database access. During the framework trials, two professional developers expressed a concern that as the database accumulated more and more records, the performance may deteriorate.

We devised a test to examine the performance of inserting records as the database increases in size. The LIMS framework system was run on a virtual machine (CentOS Linux 4, Intel I5 2.4GHz, 1G memory) hosted on a Mac computer. We monitored the time to insert and retrieve a sequence of sample records using a Java client program, also running on the Mac computer.

The client program inserted 5,000 records into a database containing varying numbers of records (from 0 to 100000). The results (see Table Table 10-1) show that the average insertion time is around 1.3 milliseconds for one record. There were no significant changes between inserting records into an empty database or a database with 100,000 records.

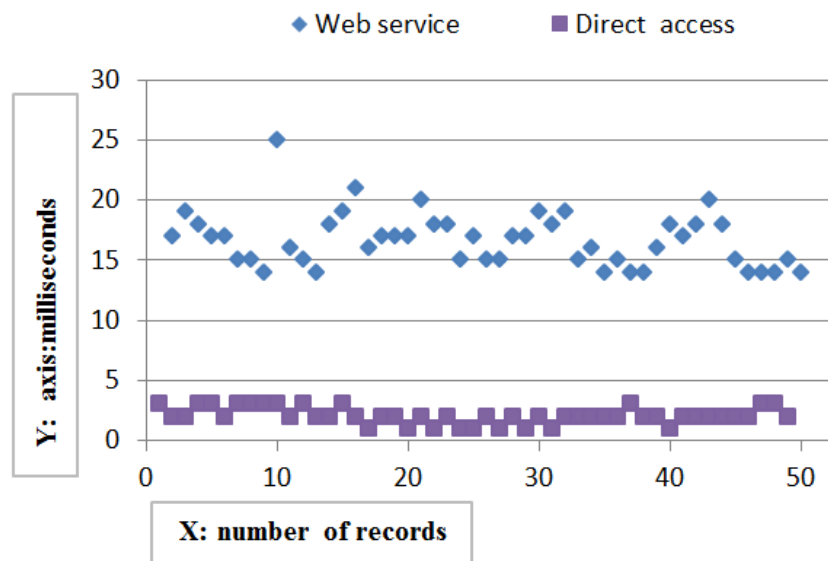
In addition, we tested retrieving all existing records from the database; the average time to retrieve one record was between 1.5 and 1.6 milliseconds. Again, the size of the database did not seem to affect performance.

**Table 10-1 Performance testing results**

Number of existing records in the database	0	20,000	40,000	60,000	80,000	100,000
Average Time to insert a record (ms)	1.323	1.330	1.332	1.342	1.345	1.341
Average Time to retrieve a record (ms)	N/A	1.583	1.563	1.597	1.564	1.589

## 10.2 Web service performance

Using a SOAP web service to access data is slower than direct access, because it imposes additional overheads over direct database access, such as encoding and decoding native data into XML formats. In order to test the efficiency of the web service implementation, we monitored the elapsed time of a client program inserting 50 sample records (without applying validation logic) in the same testing environment. The average web service elapsed time for each record is about 16.65 milliseconds. However, it takes an average of 2.04 milliseconds to insert 50 sequences of sample records using direct database access (see Figure 10-1).

**Figure 10-1 Elapsed time to insert 50 sequences records**

Note that the figure does not show the times to process the first record, which are 185ms for the web service and 47ms for direct access. We think subsequent access should reuse the cached database connections in the server side, thus making them considerably faster.

The web service performance is about eight times slower than direct access. SOAP web services use a generic protocol with large overheads and the performance difference is most likely caused by the processing overhead for SOAP encoding and decoding.

After the framework trials, we were able to improve performance by providing a bulk loading facility, i.e. the ability to submit multiple records for loading in one web service call. This reduced the number of request/response cycles and the amount of data transmitted via the network. The average web service bulk loading elapsed time was about 10.86 milliseconds, which is a decrease of about 35% compared to inserting records one at a time.

In future, we intend to explore ways to improve system performance. For example, by compressing XML messages for efficient transmission or using an alternative web service protocol, e.g. Representational State Transfer (REST) to reduce system overheads.

### **10.3 System reliability**

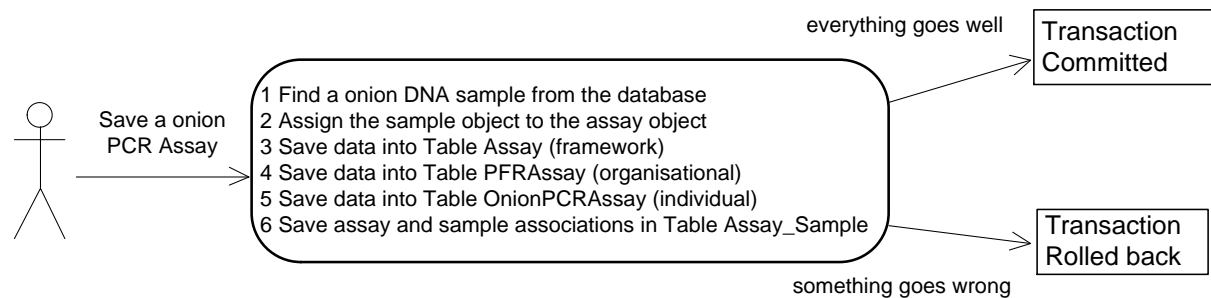
It is important that the system stores data reliably. In the framework trials, each user inserted around 30 records into the database without any problems. In the performance testing, we inserted more than 100,000 records via the web service (including bulk loading a set of 5,000 records) with no problems. However, there could be two issues in real production environments:

1. The system stores records in too many tables (base, organisational and individual level). If unexpected errors happen on the server (such as system crash), the system must ensure that the data is stored consistently, i.e. all saved to the corresponding tables or else rolled back.
2. In high concurrency situations (i.e. multiple users updating the same sets of data), the system must handle any update conflicts so that no data is lost.

#### **10.3.1 Ensuring consistent updates**

We have defined transaction boundaries for the database updates in the web service by using a begin transaction and commit transaction around all updated operations. Within a single transaction, all database updates must be either completed in entirety or aborted. This prevents data from becoming corrupted or inconsistent. For example, saving an Onion PCR assay object to the database requires inserting data into four different tables (see Figure 10-2): a framework level table, organisational level table, individual level table and the table for storing the associations. If everything goes well, the transaction is committed.

In contrast, if something goes wrong in updating any of the tables, the whole transaction will be automatically rolled back so that no records in any of the tables are changed. That guarantees that the system updates all related tables or leaves them all unchanged. In addition, the transaction is handled by the web service, so the EUDs do not need to worry about transaction details. The system will return an error message to the application if the update was not successful.



**Figure 10-2 Transaction saving an Assay record in the database**

### 10.3.2 Handle conflicting updates

The system should also provide concurrency control in order to avoid conflicts with data being updated by more than one user. For example, a user retrieves 20 sample records to update. After five minutes, the user submits the modified data and expects this will be saved. In this case, the user also believes that she/he is the only person updating those 20 records and no conflicting updates happen during the five minutes. If another user has updated those samples in the meantime, the system should reject conflicting updates from the first user, and return an error message to the client application. The system should help EUDs to make sure there are no conflicting updates.

At NZPFR, each user group has only one or two users updating the data which makes it a low concurrency environment. Therefore, we did not implement concurrency control which means the last committed transaction for a particular record “wins”. However, if the system is employed in a high concurrency environment, tools must be provided for framework managers to handle conflicting updates.

We used Hibernate to manage the persistence logic and it supports two concurrency strategies (similar strategies are available in other ORM libraries): Pessimistic and Optimistic Concurrency Control (PCC and OCC).

1. PCC locks table rows when users retrieve data for updating. The lock can be enabled to prevent conflicting updates when “SELECT ... FOR UPDATE” is detected.

2. OCC does not lock the data when the first user retrieves a record for updating. The system can use timestamps or version numbers to check whether another user updates data record after the first user retrieved it. If conflicting updates are detected, the system rolls back the transaction and sends an error message to the first user. The system is able to force the first user to retrieve the latest version for updating.

Concurrent modifications can be committed only if the modifications do not conflict.

In future, we will look at improving the framework configuration tools to make it easy to enable concurrency control. In order to prevent lost updates in a high concurrency environment, we will recommend using the OCC strategy rather than the PCC strategy. PCC simply locks records when updating is detected. That means it is not possible for multiple users to work with the same records at the same time. Instead of locking records, OCC can detect and prevent conflicting updates. It allows multiple users work on the same record as long as there is no conflict detected.

If a conflict is detected by the framework an error message must be returned to the client application so it can handle the problem. The client application can display an error message to inform the users or retrieve the latest version of data.

## 10.4 Retrieving data

The framework trial did not ask EUDs to write an application for retrieving or updating existing data. However, all participants wanted to know how this would be done so we provided participants with example code to do this. End user developers only need to call `findSamplesbyType` (described in Section 6.3) and pass the type of sample to retrieve a collection of Sample objects (see Figure 10-3).

```
Sub retrieveSamples()  
...  
    samples = service.findSamplesbyType("OnionDnaSample")  
    Range("a" & rownumber) = sample.ID  
    Range("b" & rownumber) = sample.Name  
...  
End Sub
```

**Figure 10-3 Example code of retrieving and update data**

In addition, EUDs may need to define specific criteria for selecting data. For example, users may want to retrieve Onion samples from a particular location. In our implementation, the

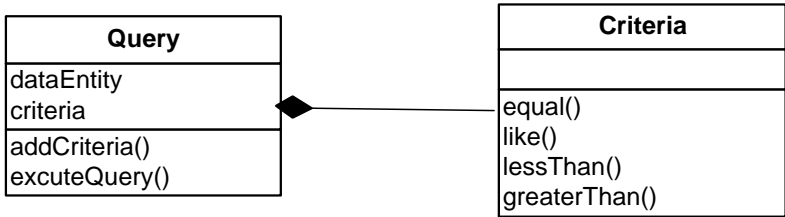


Hibernate Query Language (HQL) can be used to specify an SQL-like query using the findSamples web service function (see Figure 10-4).

```
samples = service.findSamples("OnionDnaSample sample where sample.location = 'RoomA'")
```

**Figure 10-4 Example code for retrieving subset of the data**

However, not all end user developers are able to write HQL scripts. More importantly, the system does not require the use of Hibernate. Other ORM libraries can be used to replace Hibernate. Therefore, we propose to generalise and simplify data querying by providing a query object which will enable EUDs to easily define query criteria. A Query object is used to hold a collection of criteria objects and return the query results (Fowler, 2003). There are two attributes in the Query object: Data Entity and Criteria (see Figure 10-5). The Attribute dataEntity allows EUDs to create a query for a specific data type and a collection of criteria. The Criteria class includes methods to create different types of criteria, such as a field equalling a particular value. The Query and Criteria methods would be available to EUDs via the client application toolkit.



**Figure 10-5 Proposed Queryobject and Criteria class**

Figure 10-6 illustrates an example code to demonstrate how to use these two classes to perform a custom query following these steps

1. Create a query object: Line 1 shows that EUDs need to create a query object for a specific data type, such as OnionSample.
2. Add criteria: EUDs can add several criteria in order to narrow down the query results. (see Lines 2 -3 adding different criteria).
3. Run the query: The executeQuery method retrieves (Line 4) a collection of samples which match the specified criteria for the specified entity.

```
//initialise a query object
1 QueryObject queryObj = new QueryObject(OnionSample) ;
//find all onion samples stored in RoomA
2 queryObj.add(Criteria.equal("location","RoomA"));
```

	// find all onion samples name started with "abc"
3	queryObj.add(Criteria.like("name","abc%"))
	// list all samples (in an array list) match above criteria
	...
4	samples = queryObj.excuteQuery ;

**Figure 10-6 Use a query object to retrieve records**

The criteria added to a query object are grouped by the AND operator by default. That means the results need to meet all of the criteria, e.g. samples stored in RoomA and sample names starting with "abc".

To support the OR operation we could introduce an additional class, CriteriaSet. This would be a collection of criteria. All the criteria in a particular set would be ANDed within the set. Results between CriteriaSets would be ORed.

## 10.5 System authentication and authorisation

It is important to make sure the data can only be retrieved, inserted, updated and deleted by authenticated users with the correct permission (rather than by any user who has access to the web service). This involves both authenticating users and determining which operations they are authorised to perform.

### 10.5.1 Authentication

Currently, the system only provides HTTP Basic Authentication to control access to the database. Basic authentication requires end users to provide a valid user name and password in order to access the web service. However, the user name and password are transmitted in a plaintext format. More secure protocols, such as SSL or TLS are normally not required in a trusted network environment (e.g. the NZPFR internal network). In order to provide more secure authentication, we will look at using Digest authentication (Spring, 2012), as it transmits credentials in a more secure manner (e.g. with encryption).

### 10.5.2 Authorisation

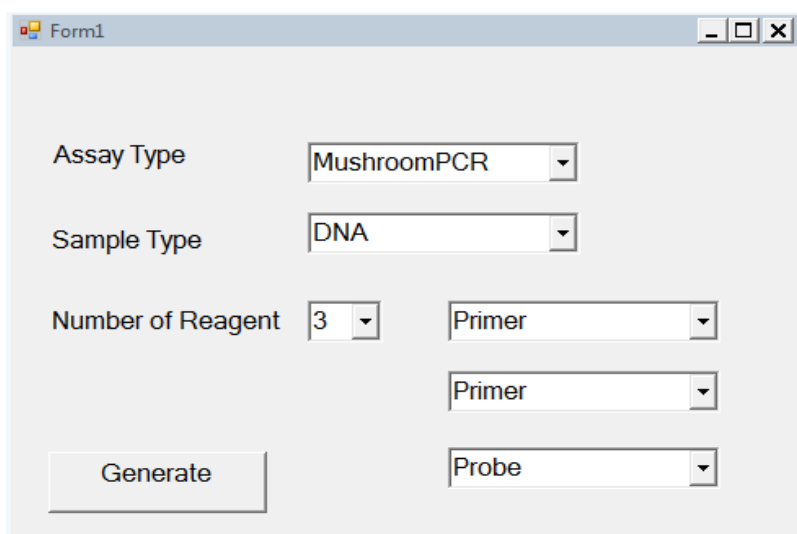
Currently, all authenticated users are able to access all methods to retrieve, insert, update and delete data. Using the framework approach, the data for the entire organisation would be stored in a shared database, thus authorisation functionality must be added to control and audit access. In future we plan to add this to the system recording each action against an authenticated user; for example, the actions could include:

- Retrieval of specific types of data (e.g. Onion or Apple)
- Entry of specific types of data
- Modification or removal of specific types of data

## 10.6 Application generator

Section 7.3 explained how to develop applications that can manage data of different types, such as Assay and Sample. Currently, we have not provided an application generator to create skeleton applications based on multiple entities, so end user developers need to manually create such applications. In future, we plan to simplify the data entry application development by providing a generator to handle the creation of more complicated skeleton applications.

Figure 10-7 shows an example generator designed for creating complicated Assay data management applications. EUDs only need to select the type of assay, sample and reagent and the number of reagents required. For example, Mushroom DNA is selected with different types of reagents, such as two Primers and one Probe (a subtype of Reagent). The generator would then create the skeleton application based on EUDs' requirements.



The screenshot shows a window titled 'Form1' with a light gray background. It contains several input fields and a button. The 'Assay Type' field is a dropdown menu with 'MushroomPCR' selected. The 'Sample Type' field is a dropdown menu with 'DNA' selected. The 'Number of Reagent' field is a spinner box with '3' selected. To the right of the spinner are three stacked dropdown menus for reagent types: the first two are 'Primer' and the third is 'Probe'. A 'Generate' button is located at the bottom left of the form area.

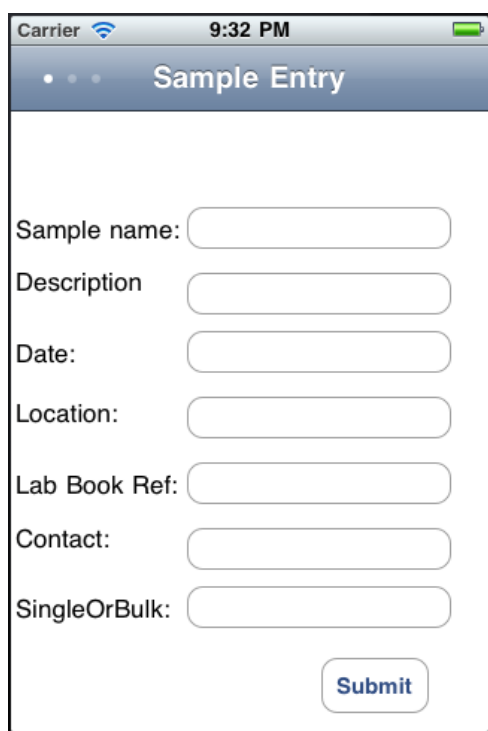
**Figure 10-7 Assay data management application generator**

## 10.7 Different client application platforms

Although the system implemented for NZPFR currently only generates an Excel skeleton application, new client toolkits and the application generators can be provided by the framework providers to create different types of client toolkits and applications, e.g. web or smart phone applications. Generators for other platforms will be similar to the one developed

for generating the Excel applications. There are two main steps involved in the process and we illustrate this for generating iPhone applications:

1. **Create toolkit generator:** Framework providers need to create a new toolkit generator for Objective C (the language for iPhone development) to map remote web service methods. There are libraries, e.g. wsdl2objc (Wsdl2objc, 2012), to support generating client application toolkits.
2. **Create application generator:** Framework providers need to create a new application generator to generate skeleton applications in Objective C. A data entry UI needs to be generated based on a specific data entity (see Figure 10-8). In addition to the UI, the application logic must be included to interact between the UI and toolkit methods.

The image shows a screenshot of an iPhone application interface. At the top, the status bar displays 'Carrier', a signal strength indicator, the time '9:32 PM', and a battery level icon. Below the status bar is a blue header with three white dots on the left and the title 'Sample Entry' in white. The main content area is white and contains a form with seven text input fields, each preceded by a label: 'Sample name:', 'Description', 'Date:', 'Location:', 'Lab Book Ref:', 'Contact:', and 'SingleOrBulk:'. At the bottom right of the form is a blue button with the word 'Submit' in white.

**Figure 10-8 UI prototype for iPhone**

The client application toolkit and application generator minimise the effort of developing end user applications. EUDs do not have to program from scratch but merely amend the generated application code (as discussed in the EUDs trials in Section 8.2.2).

## 10.8 More complex data models

The prototype framework was designed to manage plant samples, experiments and reagents. The association between related entities was a single many-to-many. The results show that the framework can handle many-many and hence 1 to many relationships well. In the future, we would need to look at how to apply the framework to data models with more complex

features, such as multiple relationships between two entities and self-relationships. Our prototype framework uses the Hibernate Object Relational Mapping (ORM) Library to manage ORM and persistence logic, which support both these types of relationship (Hibernate, 2007).

## **10.9 Summary**

The framework approach helps EUDs to easily develop flexible end user applications. However, the current framework implementation could be improved in areas such as performance, reliability and security. In this chapter we looked at some of the future work required to provide a more robust, efficient and productive system.

## Chapter 11 Conclusions

Software flexibility and reusability are critical issues for end user developed databases and applications. End user developers generally lack software development expertise and are often under time pressure to develop applications to meet their immediate and specific needs. It can be very difficult to adapt and reuse these highly specific EUDAs for other user groups who have similar requirements.

To create flexible systems for a variety of user groups with similar, but different, requirements, it is crucial that the database can be evolved to handle new, or changed, entities, attributes, relationships and constraints. This requires professional software development skills and considerable time, which EUDs may not have. Due to limited development skills and time, most end user developers find it is easier to create separate databases and applications for different user groups, even if each group's requirements are similar. This wastes considerable resources and causes long term data management and integration problems.

### 11.1 Guidelines and the Middle Way

In order to maximise EUDA flexibility and reusability, this study proposed that a data model should be designed for a specific domain but allow for modifications and extensions within that domain. We also proposed guidelines to support EUDs create flexible and reusable applications. It is especially important that any solution keeps interactions with the underlying database simple, insulating EUDs from complex domain, persistence and validation logic.

In order to meet the guidelines we proposed that IT professionals assist end user developers to find a "Middle Way" between very specific and very generic designs. An appropriate Middle Way point would include all domain objects/entities of likely interest but exclude more generic ones that are unlikely to be needed. Our solution was to design a framework for the appropriate Middle Way point, which can readily be adapted to support different specialised entities and requirements. We applied the framework approach to a specific case study involving a LIMS system to store sample and assay data for research groups at NZPFR.

The framework included a domain specific data model that can be applied to a range of end user applications. The initial data model only includes a set of base classes with minimal attributes, designed and implemented by "framework providers" (IT professionals).

Configuration tools allow “framework managers” (responsible for local databases) to customise the data model to meet the specific data management needs of the organisation and the various user groups within it.

Common data management methods are provided via a web service and client application toolkit to allow end user developers to create applications without being concerned about the details of the framework design. We also provided a client application generator to help end user developers quickly produce and customise data entry applications.

## 11.2 Framework evaluation

We evaluated how well the framework approach met our guidelines in two ways:

1. Software development trials involving framework managers and end user developers working with, and extending, the LIMS framework. The overall feedback was that the approach would satisfy end user development needs while not requiring large investments of time to implement.
2. We self-assessed the flexibility of the framework by proposing extensions to the existing LIMS data model and examined the impact this would have on new and existing applications.

Thus, the Middle Way and framework approach appears to be a powerful and useful way to support EUDs to create flexible applications. Below we summarise how this approach meets each of our guidelines.

- **Solutions should have a data model that allows for modifications and extensions within a specified domain**

The framework transforms EUDAs from very specific to more generic designs. This provides a flexible data model for creating similar applications to meet specific needs. The separation of the framework into different levels (base, organisation, individual) allows it to support different specialist applications at the same time and to cater for new requirements as they arise.

- **Allow end users to easily manage their data**

The framework approach provided a single organisational store for end users to manage a variety of data sets. The framework provides client application toolkits for EUDs to

create applications based on their familiar UIs, such as spreadsheets. More importantly, EUDs need only customise the generated applications to meet specific requirements.

- **Solutions should allow modifications and extensions without affecting existing data and applications**

Adding new entities and attributes, etc to the framework does not require changing existing applications. This also means that existing data can continue to be used without modification or reloading. The software trials illustrated this as different types of samples (e.g. Sheep or Hieracium) were added to the LIMS framework but did not affect existing data or applications.

- **Solutions should allow EUDs to easily create and extend applications for different requirements within the same domain**

The support tools provided simplify application development for both framework managers and EUDs. There is no need to write code to create database tables, web services and toolkits, etc. Framework managers only need to add their specific requirements to configuration files that are then read by the tools to customise the framework. This saves a huge amount of development (and redevelopment) effort.

More importantly, the client toolkit and application generator significantly minimise end user development effort. End user developers can concentrate on designing the user interface and use appropriate toolkit functions to implement database interactions. Because the data management logic is encapsulated in the framework and client toolkit, development is greatly simplified.

The client toolkits do not require a high level of programming expertise to use. They can also be provided for a variety of platforms so EUDs can choose to work in a language they are familiar with. In addition, EUDs can learn from, and adapt, the example applications created by the application generator.

The framework and client toolkit not only simplifies the EUD's job but also means that applications will have a very similar structure. For example, an application to manage Apple samples should be very easy to adapt to Orange samples. If more substantial changes are required (e.g. new attributes or entities), the framework manager or provider may need to undertake this. But once modifications are made, EUDs will be able to work with the updated toolkit to create or extend applications. All participants in the software trial commented that



the toolkit methods were very generic and easy to understand and they thought it would be easy to adapt an application to handle similar, but different, requirements.

A final benefit of the framework is that all the data for an organisation will be in one central database rather than in many separate applications. This simplifies the overall management.

### **11.3 Final remarks**

This study looked at the challenges of EUDA flexibility and reusability. A framework approach was proposed to help end user developers create adaptable applications with similar, but different, requirements. The approach involved professional developers (working in consultation with end users) designing a Middle Way domain specific data model in three levels to provide a reusable and flexible structure. The system also provided a number of tools to simplify the creation and evolution of the data model and to simplify the programming required of EUDs.

Other cooperative design approaches such as Meta design (Fischer, 2009) and SSWs (Costabile, 2003) are similar to this study in that professional developers and end users work together to design flexible and customisable applications. The difference in our approach is that end users and professional developers work together to design a flexible data model - not an application. With this approach, EUDs are able to modify the underlying data model and generate their own applications rather than just customising existing applications.

More importantly, our framework allows a basic data model to have several co-existing variations which satisfy the application requirements of different user groups in a common domain. The main contribution of this work is that EUDs are able to evolve their part of the data model for just their applications; any changes will not affect applications of other user groups. In addition, because the top level databases tables are the same across all the co-existing data models, it is possible to integrate data across an organisation.

The approach was demonstrated with a real world case study and evaluated by software trials and a self-assessment of the flexibility of the design. The feedback from the trials suggests the approach would be a very effective way to help end user developers create flexible data management applications. The self-evaluation showed that the approach would provide the required flexibility. The overall result is that the framework approach would allow different user groups to have their specialist requirements met within a single organisation-wide database.

This study has made an important contribution to the research in end user development. The core contributions can be summarised as the development of the guidelines, framework approach and supporting tools.

- **Guidelines:** These provide recommendations for how IT professionals can develop solutions to help end user developers to create flexible applications.
- **Framework:** The framework provides a flexible data model that allows modification and extension, oriented around an appropriate Middle Way point. This framework idea can be applied to a number of problem domains where EUDs create various applications to manage similar data but with different specific requirements.
- **Development tools:** The configuration tools developed allow framework managers to easily adapt and extend the framework to cater for specific data requirements. The client toolkits and application generators significantly reduce the effort of EUDs in creating a data management application.

This study also provides a better understanding of the partnership required between professionals and end user developers in order to support the efficient and effective creation of data management applications that can genuinely be reused. The significance of this work may be best captured by a quote from a software trial participant:

*“It is very important to provide a solution for non- professional developers to capture and maintain data. I would like to use the framework for my next data management project.”*

## References

- Ahrens, J. D., & Sankar, C. S. (1993). Tailoring Database Training for End Users. *MIS Quarterly*, 17(4), 419-439.
- Alex, B., & Schmidt, S. (2009). *Spring Roo - Reference Documentation*. Retrieved 10, April, 2010, from <http://static.springsource.org/spring-roo/reference/html/index.html>
- Alur, D., Crupi, J., & Malks, D. (2003). *Core J2EE patterns : best practices and design strategies* (2nd ed.). Upper Saddle River, NJ: Prentice Hall PTR.
- Anders, I. M., rch, Gunnar, S., Markus, W., Markus, K., Yvonne, D., et al. (2004). Component-based technologies for end-user development. *Commun. ACM*, 47(9), 59-62.
- Apache. (2011). *Apache Tomcat 6.0*. Retrieved 11 May, 2009, from <http://tomcat.apache.org/tomcat-6.0-doc/introduction.html>
- Apple. (2011). *Developing for iOS 4*. Retrieved 4 April, 2011, from <http://developer.apple.com/devcenter/ios/index.action>
- Bhattacharya, P., & Neamtiu, A. I. (2010). *Dynamic updates for web and cloud applications*. Paper presented at the 2010 Workshop on Analysis and Programming Languages for Web Applications and Cloud Applications, Toronto, Canada.
- Booch, G. (1994). *Object-oriented analysis and design with applications* (2nd ed.). Redwood City, Calif.: Benjamin/Cummings Pub. Co.
- Burnett, M., Rothermel, G., & Cook, C. (2006). An Integrated Software Engineering Approach for End-User Programmers. In *End User Development* (9 ed., pp. 87-113): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_1](http://dx.doi.org/10.1007/1-4020-5386-X_1). doi:10.1007/1-4020-5386-x\_1
- Cappiello, C., Daniel, F., Matera, M., Picozzi, M., & Weiss, M. (2011). Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits. In M. Costabile, Y. Dittrich, G. Fischer & A. Piccinno (Eds.), *End-User Development* (6654 ed., pp. 9-24): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_3](http://dx.doi.org/10.1007/978-3-642-21530-8_3). doi:10.1007/978-3-642-21530-8\_3
- Carlo Curino , H. J. M., Carlo Zaniolo. (2009). *Automating Database Schema Evolution in Information System Upgrades*. Paper presented at the Third Workshop on Hot Topics in Software Upgrades, Orlando, Florida, USA.
- Carter, E., & Lippert, E. (2009). *Visual Studio tools for Office 2007 : VSTO for Excel, Word, and Outlook*. Upper Saddle River, NJ: Addison-Wesley.

- Chappell, D. (2009). *Introducing Windows Communication Foundation in .NET Framework 4*. Retrieved 12 Sep, 2011, from <http://msdn.microsoft.com/library/ee958158.aspx>
- Christian, D., Volkmar, P., Moritz, W., & Volker, W. (2008). *End-user development: new challenges for service oriented architectures*. Paper presented at the Proceedings of the 4th international workshop on End-user software engineering.
- Churcher, C., McLennan, T., & McKinnon, A. (2001). *From conceptual model to end user implementation*. [Lincoln] N.Z.: Applied Computing, Mathematics and Statistics Group, Lincoln University.
- Clarke, B. J. (2010). *An introduction to object-oriented systems development with JADE* (5th ed.). Christchurch, N.Z.: Jade Software Corporation.
- Cleve, A. (2010, 12-18 Sept. 2010). *Program analysis and transformation for data-intensive system evolution*. Paper presented at the Software Maintenance (ICSM), 2010 IEEE International Conference on.
- Costabile, M. F., Fogli, D., Fresta, G., Mussio, P., & Piccinno, A. (2003). *Building environments for end-user development and tailoring*. Paper presented at the Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on.
- Costabile, M. F., Fogli, D., Mussio, P., Piccinno, A., Lieberman, H., Paternò, F., et al. (2006). *End-User Development: The Software Shaping Workshop Approach*. End User Development. In (9 ed., pp. 183-205): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_9](http://dx.doi.org/10.1007/1-4020-5386-X_9). doi:10.1007/1-4020-5386-x\_9
- Costabile, M. F., Mussio, P., Provenza, L. P., & Piccinno, A. (2008). *End users as unwitting software developers*. Paper presented at the Proceedings of the 4th international workshop on End-user software engineering.
- Cunha, c., Jo, Saraiva, o., & Visser, J. (2009). *From spreadsheets to relational databases and back*. Paper presented at the Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation.
- Curino, C. A., Moon, H. J., Tanca, L., & Zaniolo, C. (2008). *Schema Evolution Benchmark*. Paper presented at the 10th International Conference on Enterprise Information Systems, Barcelona, Spain.
- Cushing, J. B., Nadkarni, N., Finch, M., Fiala, A., & E. M.-H., Delcambre, L., et al. (2007). *Component-based end-user database design for ecologists*. *Springer Science, Business Media*(10.1007/s10844-006-0028-6), 18.

- Demeyer, S. (2008). Object-Oriented Reengineering. In T. Mens & S. Demeyer (Eds.), *Software Evolution* (Vol. XVIII., pp. 347): Springer.
- Deng, Y., Abell, W., Churcher, C., & McCallum, J. (2007). *Using Web Services for Customised Data Entry*. Paper presented at the The Thirteenth Australasian World Wide Web Conference 07, Coffs Harbour, Australia.
- Deng, Y., Churcher, C., Abell, W., & McCallum, J. (2011). Designing a Framework for End User Applications. In M. Costabile, Y. Dittrich, G. Fischer & A. Piccinno (Eds.), *End-User Development* (6654 ed., pp. 67-75): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_7](http://dx.doi.org/10.1007/978-3-642-21530-8_7). doi:10.1007/978-3-642-21530-8\_7
- Deng, Y., McCallum, J., Churcher, C., & Abell, W. (2009). *Excel Based PCR Workflow LIMS For Small Laboratories*. Paper presented at the Plant & Animal Genomes XVII Conference. from [http://www.intl-pag.org/17/abstracts/P08b\\_PAGXVII\\_822.html](http://www.intl-pag.org/17/abstracts/P08b_PAGXVII_822.html)
- Diestelkamp, W., & Lundberg, L. (2000). Performance Evaluation of a Generic Database System. *International Journal of Computers and Their Applications*, 7(3), 8.
- Dittrich, Y., Lindeberg, O., & Lundberg, L. (2006). End-User Development as Adaptive Maintenance. In H. Lieberman, F. Paternò & V. Wulf (Eds.), *End User Development* (9 ed., pp. 295-313): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_14](http://dx.doi.org/10.1007/1-4020-5386-X_14). doi:10.1007/1-4020-5386-x\_14
- Django. (2008). *Django documentation*. Retrieved 7 July, 2008, from <http://docs.djangoproject.com/en/dev/intro/overview/#intro-overview>
- Eclipse. (2011). *Eclipse Modeling Framework Project (EMF)*. Retrieved 10 Sep, 2011, from <http://www.eclipse.org/modeling/emf/?project=emf>
- Elliott, J., O'Brien, T., & Fowler, R. (2008). *Harnessing Hibernate*. Beijing ; Sebastopol, [Calif.]: O'Reilly.
- Eriksson, J. (2008). *Supporting the Cooperative Design Process of End-user Tailoring*. Department of Interaction and System Design, Blekinge Institute of Technology.
- EUD-Net. (2003). *End-User Development: Empowering people to flexibly employ advanced in-formation and communication technology*. (12,Jan). Retrieved from <http://giove.isti.cnr.it/EUD-NET/pdf/Research%20Agenda%20&%20Roadmap.pdf>
- Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*: Addison Wesley.
- Fayad, M., & Schmidt, D. C. (1997). Object-oriented application frameworks. *Commun. ACM*, 40(10), 32-38.

- Fenster, L. (2006). *Effective use of Microsoft Enterprise Library : building blocks for creating enterprise applications and services*. Upper Saddle River, NJ: Addison-Wesley.
- Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A. G., & Mehandjiev, N. (2004). Meta-design: a manifesto for end-user development. *Commun. ACM*, 47(9), 33-37.
- Fischer, G., Nakakoji, K., & Ye, Y. (2009). Metadesign: Guidelines for Supporting Domain Experts in Software Development. *IEEE Software*, 26(5), 37-44.
- Floch, J. (2011). A Framework for User-Tailored City Exploration. In M. Costabile, Y. Dittrich, G. Fischer & A. Piccinno (Eds.), *End-User Development* (6654 ed., pp. 239-244): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_20](http://dx.doi.org/10.1007/978-3-642-21530-8_20). doi:10.1007/978-3-642-21530-8\_20
- Fowler, M. (1997). *Analysis patterns : reusable object models*. Menlo Park, Calif.: Addison Wesley.
- Fowler, M. (2003). *Patterns of enterprise application architecture*. Boston: Addison-Wesley.
- Fowler, M., & Sadalage, P. (2004). *Evolutionary Database Design*. Retrieved 05 December, 2008, from <http://martinfowler.com/articles/evodb.html>
- Galante, R. d. M., dos Santos, C. S., Edelweiss, N., & Moreira, Á. F. (2005). Temporal and versioning model for schema evolution in object-oriented databases. *Data & Knowledge Engineering*, 53(2), 99-128.
- Gamma, E. (1995). *Design patterns : elements of reusable object-oriented software*. Reading, Mass.: Addison-Wesley.
- Ghiani, G., Paternò, F., & Spano, L. (2011). Creating Mashups by Direct Manipulation of Existing Web Applications. In M. Costabile, Y. Dittrich, G. Fischer & A. Piccinno (Eds.), *End-User Development* (6654 ed., pp. 42-52): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_5](http://dx.doi.org/10.1007/978-3-642-21530-8_5). doi:10.1007/978-3-642-21530-8\_5
- GMOD. (2008). *Chado - Getting Started*. Retrieved 12, Dec, 2008, from <http://gmod.org/wiki/Chado>
- Google. (2011a). *Google Projects for Android*. Retrieved 4, April, 2011, from <http://code.google.com/android/>
- Google. (2011b). *Tutorial: Creating Your First Spreadsheet Script*. Retrieved 4, April, 2011, from <http://code.google.com/googleapps/appsscript/articles/yourfirstscript.html>
- Google. (2012). *Google Translate API* Retrieved 1 Mar, 2012, from <http://code.google.com/apis/language/translate/overview.html>

- Hainaut, J.-L., Cleve, A., Henrard, J., & Hick, J.-M. (2008). Migration of Legacy Information Systems. In T. Mens & S. Demeyer (Eds.), *Software Evolution* (Vol. XVIII., pp. 347): Springer.
- Halbert, D. C. (1984). *Programming by Example*. University of California, Berkeley.
- Hathi, R., & Balani, N. (2008). *Design and implement POJO Web services using Spring and Apache CXF, Part 1: Introduction to Web services creation using CXF and Spring*. Retrieved March 6, 2009, from <http://www.ibm.com/developerworks/webservices/library/ws-pojo-springcxf/>
- Hay, D. C. (1996). *Data Model Patterns: Conventions of Thought*. New York: Dorset House Publishers, Inc.
- Heckel, R., Correia, R., Matos, C., El-Ramly, M., Koutsoukos, G., & Andrade, L. (2008). Architectural Transformations: From Legacy to Three-Tier and Services. In T. Mens & S. Demeyer (Eds.), *Software Evolution* (Vol. XVIII., pp. 347): Springer.
- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Q.*, 28(1), 75-105.
- Hibernate. (2007). *Hibernate Reference Documentation (Version 3.2.2)*. Retrieved from [http://www.hibernate.org/hib\\_docs/v3/reference/en/pdf/hibernate\\_reference.pdf](http://www.hibernate.org/hib_docs/v3/reference/en/pdf/hibernate_reference.pdf)
- Hick, J.-M., & Hainaut, J.-L. (2006). Database application evolution: A transformational approach. *Data & Knowledge Engineering*, 59(3), 534-558.
- IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 1.
- JADE. (2008). *JADE Object Management and Persistence in Java* Retrieved 12, Sep, 2011, from [http://www.jade.co.nz/downloads/jade/papers/JADE\\_WP\\_JavaPersistence.pdf](http://www.jade.co.nz/downloads/jade/papers/JADE_WP_JavaPersistence.pdf)
- Jayashree, B., Praveen T Reddy, Y Leeladevi, Jonathan H Crouch, V Mahalakshmi, & Hutokshi K Buhariwalla. (2006). Laboratory Information Management Software for genotyping workflows: applications in high throughput crop genotyping. *BMC Bioinformatics*, 7:383, 6.
- Johnson, R. E., & Foote, B. (1988). Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2).
- Klann, M., Paternò, F., & Wulf, V. (2006). Future Perspectives in End-User Development End User Development. In H. Lieberman (Ed.), (9 ed., pp. 475-486): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_21](http://dx.doi.org/10.1007/1-4020-5386-X_21). doi:10.1007/1-4020-5386-x\_21

- Koehne, B., & Redmiles, D. (2011). Extending the Meta-design Theory: Engaging Participants as Active Contributors in Virtual Worlds. In M. Costabile, Y. Dittrich, G. Fischer & A. Piccinno (Eds.), *End-User Development* (6654 ed., pp. 264-269): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_24](http://dx.doi.org/10.1007/978-3-642-21530-8_24). doi:10.1007/978-3-642-21530-8\_24
- Krasner, G. E., & Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object Oriented Program.*, 1(3), 26-49.
- Lee, J. M., Davenport, G. F., Marshall, D., Ellis, T. H. N., Ambrose, M. J., Jo Dicks, et al. (2005). GERMINATE. A Generic Database for Integrating Genotypic and Phenotypic Information for Plant Genetic Resource Collections1. *American Society of Plant Biologists*, 139, 619-631.
- Letondal, C. (2006). Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users. In *End User Development* (9 ed., pp. 207-242): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_10](http://dx.doi.org/10.1007/1-4020-5386-X_10). doi:10.1007/1-4020-5386-x\_10
- Lieberman, H. (2001). *Your wish is my command: programming by example*: Morgan Kaufmann Publishers.
- Lieberman, H., Paternò, F., Wulf, V., & Klann, M. (2006). End-User Development: An Emerging Paradigm. In *End User Development* (9 ed., pp. 1-8): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_1](http://dx.doi.org/10.1007/1-4020-5386-X_1). doi:10.1007/1-4020-5386-x\_1
- Lin, D.-Y., & Neamtiu, A. I. (2009). *Collateral evolution of applications and databases*. Paper presented at the Joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops.
- Liu, L., & Özsu, M. T. (2009). *Encyclopedia of Database Systems* (Vol. LXX3752): Springer.
- Mackay, W. E. (1990). *Users and Customizable Software: A Co-Adaptive Phenomenon*. Massachusetts Institute of Technology.
- Meyer, B. (1998). *Object-Oriented Software Construction*: Prentice Hall.
- Microsoft. (2011a). *ADO.NET Entity Framework*. Retrieved 19 Sep, 2011, from <http://msdn.microsoft.com/en-us/library/bb399572.aspx>
- Microsoft. (2011b). *Creating an XML Web Service Proxy*. Retrieved 5 April, 2011, from <http://msdn.microsoft.com/en-us/library/d2s8y7bs.aspx>
- Microsoft. (2011c). *Microsoft Enterprise Library 5.0*. Retrieved 6 Sep, 2011, from <http://msdn.microsoft.com/en-us/library/ff632023.aspx>



- Microsoft. (2011d). *The Validation Application Block*. Retrieved 14 Sep, 2011, from [http://msdn.microsoft.com/en-us/library/ff664378\(v=PandP.50\).aspx](http://msdn.microsoft.com/en-us/library/ff664378(v=PandP.50).aspx)
- Mungall, C. J., & Emmert, D. B. (2007). A Chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, 23, i337 - 346.
- Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge MA: MIT Press.
- NCICB. (2002). *Technical Overview- caLIMS(Cancer Laboratory Information Management System)*. Retrieved 2, Oct, 2007, from [http://calims.nci.nih.gov/caLIMS/calims\\_spec.doc](http://calims.nci.nih.gov/caLIMS/calims_spec.doc)
- NCICB. (2008). *Overview- caLIMS2(Cancer Laboratory Information Management System)*. Retrieved 2, May, 2009, from <https://wiki.nci.nih.gov/display/ICR/caLIMS2+Project+Overview>
- Nestler, T., Namoun, A., & Schill, A. (2011). *End-user development of service-based interactive web applications at the presentation layer*. Paper presented at the Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems.
- O'Reilly, T. (2005). *What Is Web 2.0 Design Patterns and Business Models for the Next Generation of Software*. Retrieved November, 2006, from <http://www.oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- Oracle. (2010). *Global Single Schema*. Retrieved 25 Oct, 2011, from <http://www.oracle.com/us/products/applications/master-data-management/047208.pdf>
- Oracle. (2011). *Python-UNO bridge*. Retrieved 4, April, 2011, from <http://contributing.openoffice.org/programming.html>
- Oscar, N., & Dennis, T. (Eds.). (1995). *Object-oriented software composition*: Prentice Hall International (UK) Ltd.
- Panko, R. R. (2008). *Spreadsheet Errors: What We Know. What We Think We Can Do*. *ArXiv e-prints*, 0802.3457, 9.
- Peck, G., & ebrary Inc. (2004). *Crystal reports 10 the complete reference*. New York: McGraw-Hill/Osborne. Retrieved from <http://ezproxy.lincoln.ac.nz/login?url=http://site.ebrary.com/lib/lincoln/Doc?id=10137249>
- Qing, L., & McLeod, D. (1994). Conceptual database evolution through learning in object databases. *Knowledge and Data Engineering, IEEE Transactions on*, 6(2), 205-224.

- Rashid, A. (2001, 2001). *A database evolution approach for object-oriented databases*. Paper presented at the Software Maintenance, 2001. Proceedings. IEEE International Conference on.
- RDevelopmentCoreTeam. (2011). *An Introduction to R* Retrieved 5 Sep, 2011, from <http://www.sap.com/solutions/sap-crystal-solutions/index.epx>
- RedHat, & Morling, G. (2011). *Hibernate Validator Reference Guide*. Retrieved 13 Sep, 2011, from <http://docs.jboss.org/hibernate/validator/4.2/reference/en-US/html/>
- Roddick, J. F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37, 383--393}.
- Rode, J., Howarth, J., & Pérez-Quinones, M. A. (2003). *An End-User Development Perspective on State-of-the-Art Web Development Tools*. Retrieved 25, Apr, 2009, from <http://eprints.cs.vt.edu/archive/00000708/01/webtoolsevaluation.pdf>
- Rode, J., Rosson, M. B., & Quiñones, M. A. P. (2006). End User Development of Web Applications. In H. Lieberman, F. Paternò & V. Wulf (Eds.), *End User Development* (9 ed., pp. 161-182): Springer Netherlands. Retrieved from [http://dx.doi.org/10.1007/1-4020-5386-X\\_8](http://dx.doi.org/10.1007/1-4020-5386-X_8) doi:10.1007/1-4020-5386-x\_8
- Rodriguez, A. (2008). *RESTful Web services: The basics*. Retrieved 12 Sep, 2011, from <https://www.ibm.com/developerworks/webservices/library/ws-restful/>
- RubyonRails. (2009). *Rails Guide*. Retrieved 3,Nov, 2008, from [http://guides.rubyonrails.org/getting\\_started.html](http://guides.rubyonrails.org/getting_started.html)
- Segal, J. (2009). Software Development Cultures and Cooperation Problems: A Field Study of the Early Stages of Development of Software for a Scientific Community. *Comput. Supported Coop. Work*, 18(5-6), 581-606.
- Sommerville, I. (2007). *Software engineering* (8th ed.). Harlow, England ; New York: Addison-Wesley.
- Sosnoski, D. (2010). *Java web services: Introducing CXF*. Retrieved June 6, 2010, from <http://www.ibm.com/developerworks/java/library/j-jws12.html>
- Spring. (2009). *The Spring Framework - Reference Documentation*. Retrieved 7, Jan, 2009, from <http://static.springsource.org/spring/docs/2.5.x/reference/index.html>
- Spring. (2012). *Digest Authentication*. Retrieved 3 Mar, 2012, from <http://static.springsource.org/spring-security/site/docs/2.0.x/reference/digest.html>
- Stevenson, S. (2010). *Cocoa and Objective-C: Up and Running*: O'Reilly Media.
- W3C. (2000). *Simple Object Access Protocol*. Retrieved from <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

- W3C. (2001). *Web Services Description Language (WSDL) 1.1*. Retrieved from <http://www.w3.org/TR/wsdl>
- Wajid, U., Namoun, A., & Mehandjiev, N. (2011). Alternative Representations for End User Composition of Service-Based Systems. In M. Costabile, Y. Dittrich, G. Fischer & A. Piccinno (Eds.), *End-User Development* (6654 ed., pp. 53-66): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_6](http://dx.doi.org/10.1007/978-3-642-21530-8_6). doi:10.1007/978-3-642-21530-8\_6
- Walls, C., & Breidenbach, R. (2008). *Spring in action* (2nd ed.). Greenwich: Manning.
- Wendl, M., Smith, S., Pohl, C., Dooling, D., Chinwalla, A., Crouse, K., et al. (2007). Design and implementation of a generalized laboratory data model. *BMC Bioinformatics*, 8(1), 362.
- Wsd2objc. (2012). *Generates Objective-C (Cocoa) code from a WSDL for calling SOAP services*. Retrieved from <http://code.google.com/p/wsd2objc/>
- Wulf, V., Pipek, V., & Won, M. (2008). Component-based tailorability: Enabling highly flexible software applications. *International Journal of Human-Computer Studies*, 66(1), 1-22.
- Yahoo. (2011). *Pipes Overview*. Retrieved 15 Sep, 2011, from <http://pipes.yahoo.com/pipes/docs?doc=overview>
- Zhu, L., Vaghi, I., & Barricelli, B. (2011). MikiWiki: A Meta Wiki Architecture and Prototype Based on the Hive-Mind Space Model. In *End-User Development* (6654 ed., pp. 343-348): Springer Berlin / Heidelberg. Retrieved from [http://dx.doi.org/10.1007/978-3-642-21530-8\\_37](http://dx.doi.org/10.1007/978-3-642-21530-8_37). doi:10.1007/978-3-642-21530-8\_37

# **Appendix 1: Interview questions for in-house development**

## **Questions for end user developers**

- 1. Do you have any analysis or database systems developed (or extended) by end users? E.g. excel or access based applications**
- 2. What is end users' software development background?**
- 3. When the software requirements change,**
  - a. How do you handle new data types?
  - b. How do you handle relationships among the new data?
  - c. How do you add new functionality to the existing systems?
  - d. Are there any other issues with requirement changes
  - e. Do you have professional IT support to extend/update your EUDAs?

## Questions for professional developers

### 1. General questions about in-house development:

- a. Do you use open source applications?
- b. Do you have in-house developed LIMS applications/databases?
- c. What is the common laboratory workflow in your institution?
- d. How many types of experiments/data do you have?
- e. Is all data stored in the same centralised LIMS database? If not, how do you get data from different sources?

### 2. If users change the requirements, does the system support to:

- a. Add new types of entities?
- b. Add new relationships between the new entities and existing entities?
- c. Alter the LIMS workflow for new types of entities? E.g. adding or removing some functionality?
- d. If not, what kind of tools/supports would you like to achieve this?
- e. Are there any flexibility issues in your LIMS?

### 3. Laboratory Instrument Management (for genetic research institutions only)

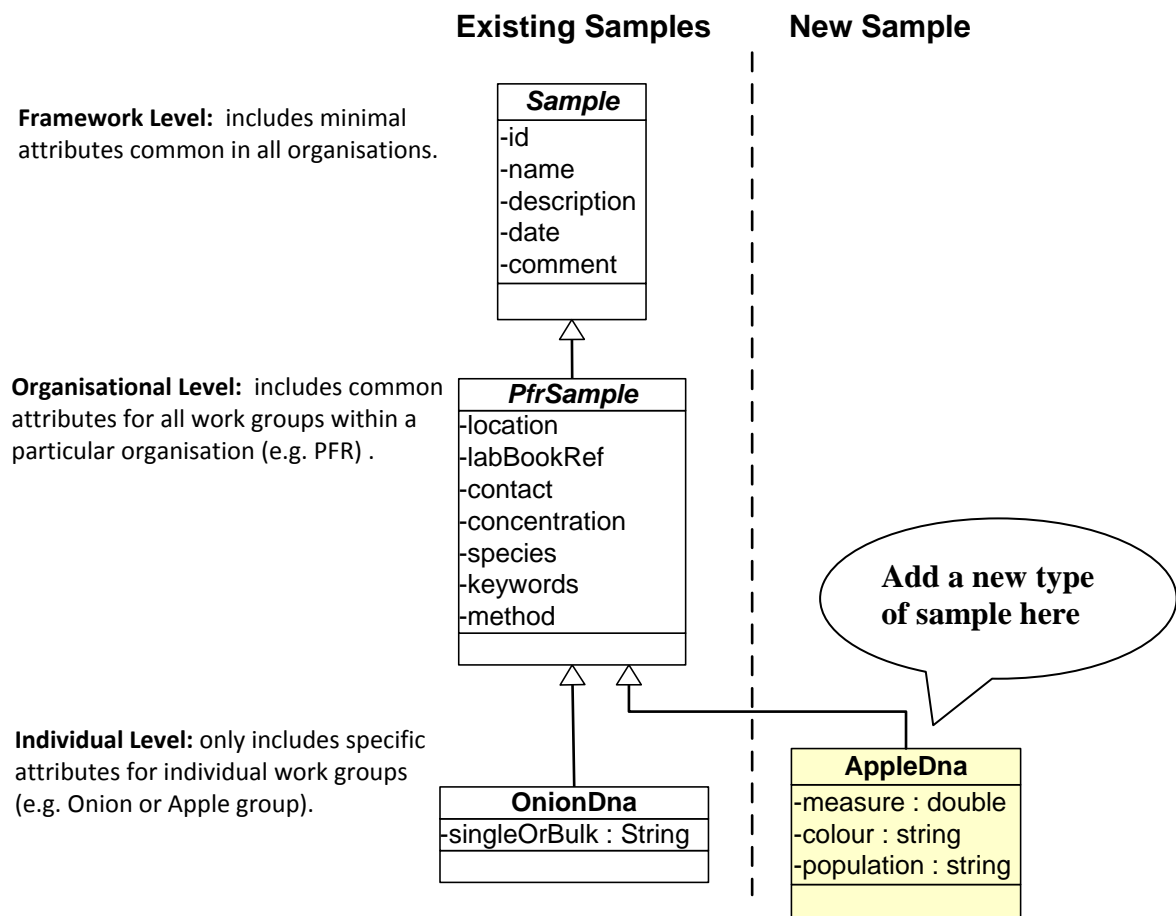
- a. How do you store the samples in the lab? E.g. in plates or tubes? Does the system allow users to manage samples in different containers?
- b. How do you manage primers or other reactions? e.g. Primers as sets?
- c. How many types of robots and sequencers in your organisation?
- d. Does the system generate robot instructions?
- e. Is the system adapted for different types of robots? If not, what kind of tools/supports do you need to achieve this?

## Appendix 2: Framework manager instructions

### Introduction

A database framework was implemented at Plant and Food Research (PFR) to manage DNA sample records (see Figure 1).

In order to manage your particular sample records, you only need to add new sample fields at the **individual level**. These instructions show how to add new samples (such as *AppleDna*) with three specific fields: *measure*, *colour* and *population*.



**There are three steps to add a new type of sample:**

1. Creating a new sample configuration file (Instruction A).
2. Generating the database (Instruction B).
3. Generating the client toolkit and example application (Instruction C).

## Instruction A: Creating a new configuration file and database

In order to add a new sample, you need to create a sample configuration file in Comma Separated Values (CSV) format to **define sample name, fields and validation rules**. You can easily do this in Excel. Below is an example configuration file (AppleDna.csv) in Excel.

Sample	1	class	AppleDna	PfrSample	<b>Note: Case is important</b> E.g. <u>M</u> ax is not the same as <u>m</u> ax
Field	2	field	measure	double	
Rules	3	rule	Min	5	measure must be higher or equal to 5
	4	rule	Max	20	measure must be lower or equal to 20
	5	field	colour	string	
	6	rule	Pattern	red green	colour must be either red or green
	7	field	population	string	

**class** Defines a new type of sample. Row 1 shows how the *AppleDna* sample is defined. You must also provide the Organisational Level name (see Figure 1), in this case *PfrSample*.

**field** Defines a field in the sample. Row 2 shows how to define a field *measure* and set its Type to *double* (i.e. a numeric value).

**rule** Defines a validation rule for a field. Rows 3 and 4 mean that user input for *measure* must be higher or equal to 5 and lower or equal to 20.

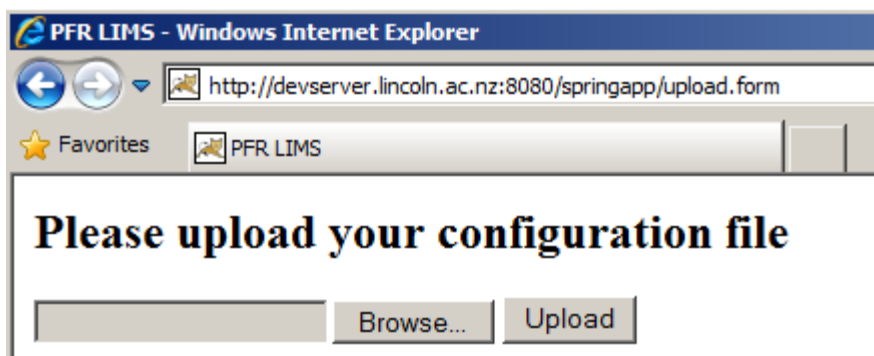
You can also define an *error message* to help users to correct invalid input. The error message text goes in the next column of the rule.

Validation rules are optional; you do not have to define rules for every field. E.g. there are no rules for *population*.

**Note:** You only have to define the additional fields required by your specific sample type. All the other fields from the Framework and Organisational levels will automatically be included.

A development tool (web page) is provided to help you to upload the configuration file and create the database.

1. **Double click** *Database Development Tool* on your desktop to open the web page shown below, for uploading your configuration file.



2. **Browse** to the configuration file on your PC (e.g. AppleDna.csv), and **click Upload**. You will see the message “file has been uploaded” in a new page.
3. **Click** *Create database and web services* which will list the configuration files previously uploaded.
4. **Select** your configuration file and **click Create** to create the database. It may take a minute or two to do this.



5. **Check** the displayed result. If the process was successful, the web page displays “Database created **successfully**”. Otherwise, the web page will display an error message “Database creation **unsuccessful**”.

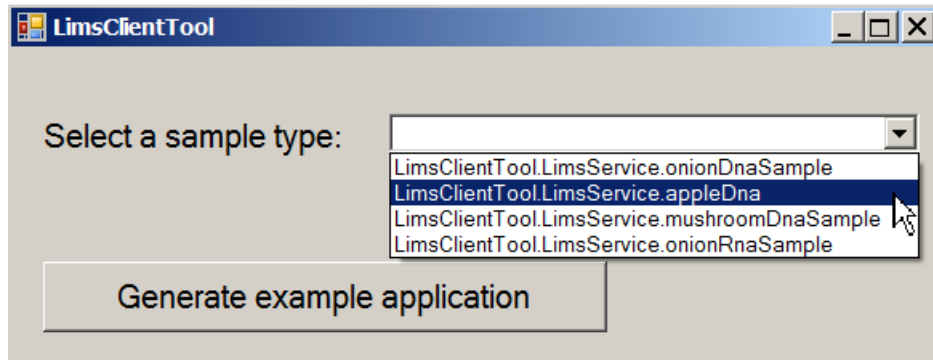
**Note:** *If you get the unsuccessful message, please **check** your configuration file, e.g. for spelling and correct case or **ask** for assistance.*



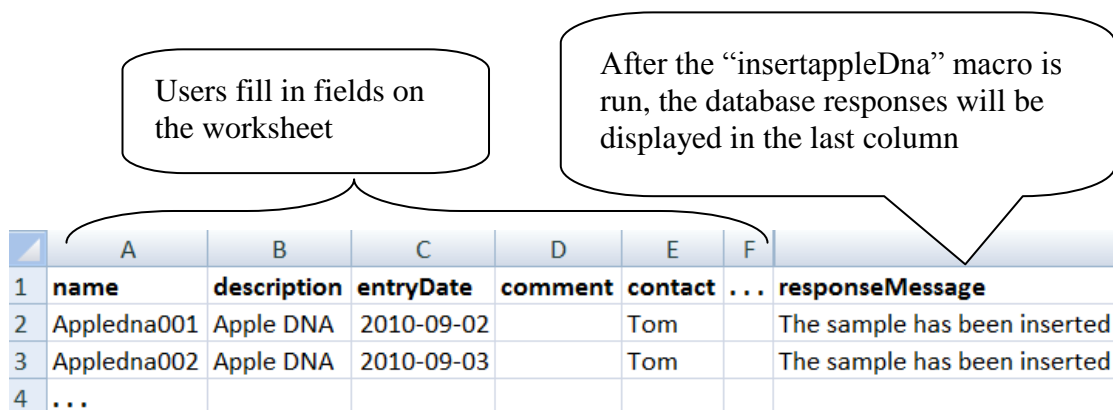
## Instruction B: Creating the toolkit and prototype applications

A tool to generate a VBA client toolkit to access the sample database is provided. To assist client application development, another tool also generates an example Excel application that uses the toolkit to access the database.

1. **Right click** *buildToolkit.bat* on your desktop and **select** *Run as Administrator*. (This reinstalls the updated client toolkit)
2. **Double click** *ClientDevelopmentTool* on your desktop to open the client development



3. **Select** a sample type (e.g. appleDna) and **click** *Generate example application*. This creates an Excel Worksheet and a VBA macro (in this case “insertappleDna”) for bulk inserting of data.



	A	B	C	D	E	F
1	<b>name</b>	<b>description</b>	<b>entryDate</b>	<b>comment</b>	<b>contact</b>	<b>... responseMessage</b>
2	Appledna001	Apple DNA	2010-09-02		Tom	The sample has been inserted
3	Appledna002	Apple DNA	2010-09-03		Tom	The sample has been inserted
4	...					

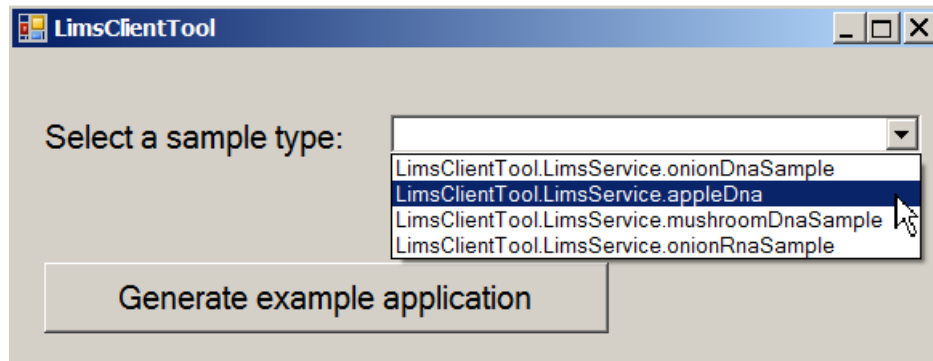
4. **Save it** as *AppleDna.xlsm* (Macro Enabled Workbook).

## Appendix 3: End user developer instructions

### Instruction A: Creating prototype VB applications for Excel

To assist client application development, we provide a tool to generate example Excel applications to insert sample data into the database.

1. **Double click** *ClientDevelopmentTool* on your desktop to open the tool.



2. **Select** a sample type (e.g. appleDna) and **click** *Generate example application*. This creates an Excel Worksheet and a VBA macro (in this case “insertappleDna”) for bulk inserting of data.

Users fill in fields on the worksheet

After the “insertappleDna” macro is run, the database responses will be displayed in the last column

	A	B	C	D	E	F	G
1	name	description	entryDate	comment	contact	...	responseMessage
2	Appledna001	Apple DNA	2010-09-02		Tom		The sample has been inserted
3	Appledna002	Apple DNA	2010-09-03		Tom		The sample has been inserted
4	...						

3. **Save it** as *AppleDna.xlsm* (Macro Enabled Workbook).

## Instruction B: Accessing the generated VBA code

The generated VBA macro code demonstrates how to insert sample data into the database.

1. Press **Alt +F8** and **select** the macro (e.g. insertappleDna). Click **Edit** to display the macro in the VBA code editor.



2. Read the comments below to understand the example code.

Example code	Comment
<pre>Sub insertappleDna()      Dim sample As New appleDna     rownumber = 2      Do While Len(Range("A" &amp; rownumber).Formula) &gt; 0         sample.Name = Range("A" &amp; rownumber)         sample.Description = Range("B" &amp; rownumber)         sample.EntryDate = Range("C" &amp; rownumber)         sample.Comment = Range("D" &amp; rownumber)         ...         res = service.insertSample(sample)          Range("N" &amp; rownumber) = res         rownumber = rownumber + 1      Loop End Sub</pre>	<p><i>create a new appleDna object</i></p> <p><i>start loading input data from row 2</i></p> <p><i>loop to read every row with a value in column A</i></p> <p><i>collect values from worksheet cells and assign them to the sample object</i></p> <p><i>call the insertSample function to insert a record into the database</i></p> <p><i>display the response message returned from the database</i></p> <p><i>process the next row</i></p>

## Appendix 4: Framework manager user trial tasks

### Task 1: updating the existing Apple sample

In this task, you will add two new fields to the existing AppleDna sample and create a data entry application to test your changes.

1. Use **Excel** to open the *AppleDna.csv* configuration file on your desktop. This file defines the AppleDna sample described in Instruction A.

Instruction A

2. Add two new fields (*size* and *quality*) and validation rules with appropriate error messages as specified below.

AppleDna
-measure : double
-colour : string
-population : string
-size : double
-quality : string

**Field:** *size* (Type *double*)

**Rule:** must be higher or equal to 5  
must be lower or equal to 20

**Field:** *quality* (Type *string*)

(No validation rules for this field)

3. Save the file after you have changed it.

Instruction B

4. Upload the configuration file and **create** the database.

Instruction C

5. Create the client toolkit and **generate the** example application.

6. Open *ExampleData.xlsx* on your desktop. This contains example data for AppleDNA. Copy the data from Worksheet *FrameworkTrial1* in this file and paste the data to the appropriate columns in the example application worksheet.
7. Press *Alt + F8* and run the *insertappleDna* macro to insert the data into the database. Check the response messages to confirm the data has been added.

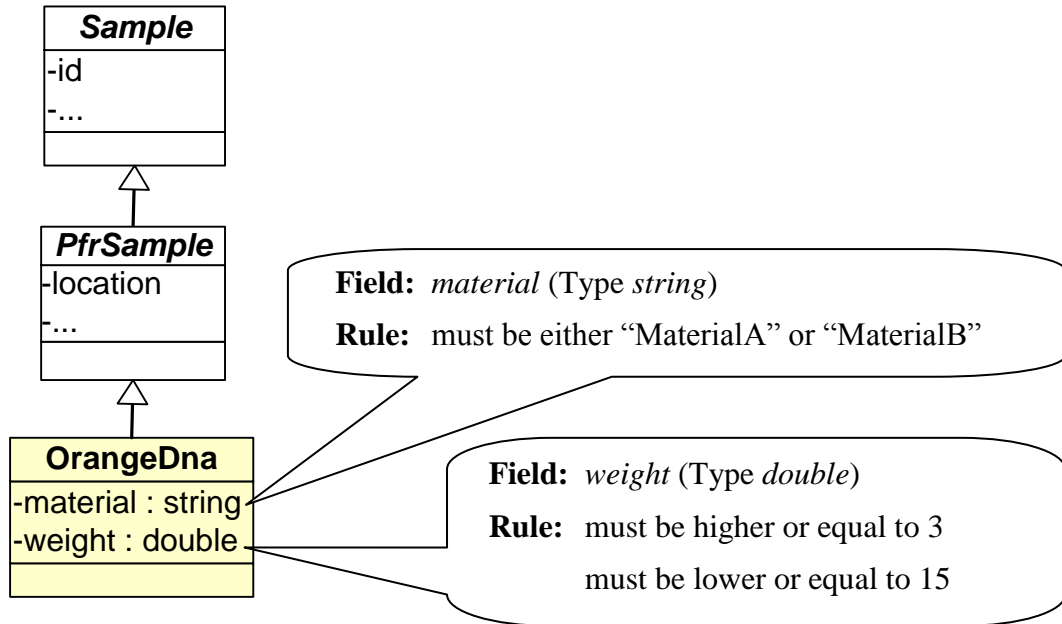
*You have finished the first task - well done! ☺*

## Task 2: Defining a new Orange sample

In this task, you will create a new Orange DNA sample with two fields.

### Instruction A

1. **Use** Excel to create a new configuration file for the Orange DNA sample and **define** two fields (*material* and *weight*) and validation rules with appropriate error messages as specified below.



2. **Save** the configuration file as *OrangeDna.csv* on the desktop.

### Instruction B

3. **Upload** the configuration file and **create** database.

### Instruction C

4. **Create** the client toolkit and **generate** the example application.
5. **Open** *ExampleData.xlsx* on your desktop. **Copy** the data from *Worksheet FrameworkTrial2* in this file and **paste** the data to the appropriate columns in the example application worksheet.
6. **Run** the *insertorangeDna* macro to insert the data into the database. Check the response messages to confirm the data has been added.

You have finished the second task – thank you! 😊

## Example Data (worksheet FrameworkTrial2)

<b>name</b>	<b>description</b>	<b>date</b>	<b>contact</b>	<b>labBook Ref</b>	<b>location</b>	<b>material</b>	<b>weight</b>
orangedna001	orange DNA	2010-09-20	Tom	Book001	Room001	materialA	0.50
orangedna002	orange DNA	2010-09-21	Tom	Book001	Room001	materialA	1.10
orangedna003	orange DNA	2010-09-22	Tom	Book001	Room001	materialB	2.00
orangedna004	orange DNA	2010-09-23	Tom	Book001	Room001	materialB	3.00
orangedna005	orange DNA	2010-09-24	Jerry	Book001	Room001	materialA	4.00
orangedna006	orange DNA	2010-09-25	Tom	Book001	Room001	materialA	5.00
orangedna007	orange DNA	2010-09-26	Tom	Book001	Room001	materialB	4.30
orangedna008	orange DNA	2010-09-27	Tom	Book001	Room001	materialB	5.87
orangedna009	orange DNA	2010-09-28	Tom	Book001	Room001	materialA	6.63
orangedna010	orange DNA	2010-09-29	Tom	Book001	Room001	materialA	7.39

## Appendix 5: End user developer trial tasks

### Task 1: Generating an Excel application

#### Instruction A

1. **Generate** an example application for Orange DNA samples.
2. **Save** the generated file as *OrangeTemplate.xlsm* on your desktop.
3. **Open** *ExampleData.xlsx* on your desktop. This contains example data for OrangeDNA. **Copy** the data from Worksheet *ExcelDevelopmentTrial1* in this file and **paste** the data to the appropriate columns in the example application worksheet.
4. **Press** *Alt +F8* and **run** the *insertorangeDna* macro to insert the data into the database. Check the response messages to confirm the data has been added.
5. **Quit** the application **without saving**.

*You have finished the first task – great work! ☺*

### Example Data (ExcelDevelopmentTrial1)

name	description	date	contact	labBook Ref	location	material	weight
orangedna011	orange DNA	2010-09-30	Jerry	Book001	Room001	materialB	8.14
orangedna012	orange DNA	2010-10-01	Tom	Book001	Room001	materialB	8.90
orangedna013	orange DNA	2010-10-02	Tom	Book001	Room001	materialA	9.66
orangedna014	orange DNA	2010-10-03	Tom	Book001	Room001	materialA	10.41
orangedna015	orange DNA	2010-10-04	Tom	Book001	Room001	materialB	1.17
orangedna016	orange DNA	2010-10-05	Tom	Book001	Room001	materialB	0.93
orangedna017	orange DNA	2010-10-06	Jerry	Book001	Room001	materialA	12.69
orangedna018	orange DNA	2010-10-07	Tom	Book001	Room001	materialA	3.44
orangedna019	orange DNA	2010-10-08	Tom	Book001	Room001	materialB	54.20
orangedna020	orange DNA	2010-10-09	Tom	Book001	Room001	materialB	14.96

## Task 2: Customising the Excel application

1. **Open** *OrangeTemplate.xlsm* (previously saved on your desktop) and **go to** *orangeDna\_Data\_Entry\_Sheet* .
2. **Delete** columns *comment*, *concentration*, *species*, *keywords* and *method* from the sheet and **add** a header “Orange DNA Data Entry Application” in row 1. The worksheet layout should look like the example below.

	A	B	C	D	E	F	G	H	I
1	Orange DNA Data Entry Application								
2	name	description	entryDate	contact	labBookRef	location	weight	material	responseMessage
3									
4									
5									
6									

### Instruction B

3. **Read and understand** the code for the *insertorangeDNA* macro.
4. **Modify** the macro code to only copy the required values from the appropriate worksheet columns and rows to the Orange DNA object.
5. **Add** a button to the worksheet (e.g. “Add Data”) and **assign** the *insertorangeDna* macro to the button. (Note the *Insert button* option is under the *Developer* tab on the ribbon)
6. **Open** *ExampleData.xlsx* on your desktop. This contains example data for OrangeDNA samples that matches the revised field list. **Copy** the data from Worksheet *ExcelDevelopmentTrial2* in this file **and paste** the data to the appropriate columns in the example application worksheet.
7. **Click** the “Add Data” button to insert the data into the database.  
Check the response messages to confirm the data has been added.

*You have finished the second task – thank you! ☺*



## Example Data (ExcelDevelopmentTrial2)

name	description	date	contact	labBook Ref	location	material	weight
orangedna021	orange DNA	2010-09-30	Jerry	Book001	Room001	materialB	1.14
orangedna022	orange DNA	2010-10-01	Tom	Book001	Room001	materialB	8.90
orangedna023	orange DNA	2010-10-02	Tom	Book001	Room001	materialA	2.66
orangedna024	orange DNA	2010-10-03	Tom	Book001	Room001	materialA	15.41
orangedna025	orange DNA	2010-10-04	Tom	Book001	Room001	materialB	11.17
orangedna026	orange DNA	2010-10-05	Tom	Book001	Room001	materialB	6.93
orangedna027	orange DNA	2010-10-06	Jerry	Book001	Room001	materialA	12.69
orangedna028	orange DNA	2010-10-07	Tom	Book001	Room001	materialA	13.44
orangedna029	orange DNA	2010-10-08	Tom	Book001	Room001	materialB	14.20
orangedna030	orange DNA	2010-10-09	Tom	Book001	Room001	materialB	14.96

## **Appendix 6: Interview questions for framework manager**

**Participant Name:**

**Job Title/Organisation :**

**Experience of using Programming:** New      Occasional      Often

**Summary of the trials** (count time & Problems)

1. Tell me about adding new samples? e.g. how many steps and what are they?
2. Would you be confident to do this?
3. Any comments about the instructions?
4. Tell me about your colleges and how do you think they would do this.
5. Tell me some situations where this type of framework for adapting & adding types of object (e.g. samples) would be useful?
6. If someone adds a new type of DNA samples, would that affect your orange DNA data entry applications?
7. How does it compare with setting up and adapting databases?
8. Would you like to use this type of framework in you next database project?
9. Questions and comments

## Appendix 7 Interview questions for end user developers

Participant Name:                      Job Title/Organisation :

Experience of using Macros/VBA: New              Occasional              Often

Summary of the trials (count time & Problems)

1. Tell me about creating a data entry application by using this e.g. how many steps and what are they?
2. Would you be confident to create a new sample object and collect values form worksheet cells and assign them to the sample object?
3. Would you be confident to use *insertSample* function to insert a record into the database and display the response message returned from the database?
4. Any comments about the instruction?
5. Tell me about your colleges and how do you think they would do this.
6. If Someone else add a new field, tell me how to adapt your existing application in order to use the new field
7. Compare with other Office/Data Management applications you have written, how does this approach compare?
8. Questions and comments