

|   |   |
|---|---|
| Two UML examples.....   | 2 |
| Ex1: CS4 administration (see Chapter 15 of text book) .....                   | 2 |
| 0. Introduction .....   | 2 |
| Current Situation [objective is to improve this via software] .....           | 2 |
| Glossary (or incipient data dictionary) .....                                 | 2 |
| Narrative of Current Situation ( <i>not that simple to write well!</i> )..... | 3 |
| 1 The Case Study.....   | 4 |
| Proposed automated system to be investigated.....                             | 4 |
| Use Case Model.....   | 5 |
| 1.1 Class Model .....   | 6 |
| 1.2 Dynamics.....   | 6 |
| 1.3 State diagrams .....  | 6 |
| 1.4 Activity diagrams .....   | 6 |
| 2 Discussion .....  | 7 |
| Brief source code notes.....  | 7 |
| 3 Brief note on Persistent Objects .....                                      | 8 |
| Points from Panel 3.2 of recommended text.....                                | 8 |
| Method 1 (simple files).....  | 8 |
| Method 2 (database) .....   | 8 |
| Points from book by Bruegge and Dutoit (very brief outline).....              | 9 |
| Brief indication of tools:.....   | 9 |

## Two UML examples

### ***Ex1: CS4 administration (see Chapter 15 of text book)***

#### **0. Introduction**

- Following notes summarise Chapter 15 of text.
  
- For more material, follow see web-site for text book. In particular, as noted earlier for practical work, Java source code is provided. However, it would require too much of a diversion, for this module, to examine this Java code in detail.
  
- Chapter 15 is not (intended to be) a comprehensive treatment.

#### **Current Situation [objective is to improve this via software]**

#### ***Glossary (or incipient data dictionary)***

|                           |   |
|---------------------------|---|
| CS                        | Computer Science (department)   |
| CS4 student               | Any student taking any 4 <sup>th</sup> year CS module, whether or not taking a CS degree        |
| Director of Studies (DoS) | Each student is advised, throughout their time in the university, by a member of staff (a DoS). |
| Lecturer                  | Persons who present lectures on modules   |
| UTO                       | Relevant person in Undergraduate Teaching Office  |

**Narrative of Current Situation** (*not that simple to write well!*)

*Towards end of each academic year, CS Syllabus Committee determines which modules will be available to CS4 students in following year. Also, around this time, CS head of department allocates duties to CS staff; in particular, one person is assigned to lecture each of next year's modules.*

*Lecturers update the course handbook in respect of their modules. CS4 coordinator updates other parts of each handbook and checks the module entries made by the lecturers. Module entries are in the LATEX formatting language and the CS4 coordinator produces the HTML versions by running the conversion application latex2html. The UTO produces the paper version of each course handbook.*

*The CS3 coordinator is supposed to give a list of the students entering CS4 from CS3 to the CS4 coordinator and to the UTO. In addition, the CS4 coordinator tells the UTO about any students entering CS4 via some other route. The UTO keeps a master list of students taking CS4 modules, which is known by email address cs4class.*

*Students provisionally register for modules by filling in paper forms and handing them in at the Undergraduate Teaching Office. The UTO checks that each student who registers is a CS4 student and that every CS4 student registers for a reasonable set of modules (consulting students' DoSs as necessary). The UTO then produces lists for lecturers of the students taking their modules (late!).*

## 1 The Case Study

Following the above presentation of the problem, the text discusses the important issue of clarifying terminology whether at a detailed or higher level.

[For example,

*What are the course handbooks and how many are there?*

*What is the CS4 mailing list, and how is it updated?].*

The last figure (15.5) in the Chapter is an "activity diagram for course handbook preparation" to show the overall workflow into which the proposed software is to fit.

### Proposed automated system to be investigated

(I) Is it possible, through software, to

- decrease the burden of routine work on all staff, esp. CS4 coordinator?
- allow students to register for modules on-line?
- make it easy to obtain up to date, reliable information from the UTO?
- make information such as course handbooks and lists of students taking modules available sooner by automating their production?

(II) CS4 administration system should also be able to provide reports on

Individual students: graduating or non-graduating? taking which modules? who is DoS? etc

Modules: Lecturer? Which students are taking? Part of which degree course?

### Use Case Model

**NB:** Text states that "querying use cases" can be sensibly provided with by "an off-the-shelf database in conjunction with standard techniques for making objects ***persistent*** [i.e. that outlive a single execution of the system]. We will return to this below under paragraph "**3.4 Persistent Objects**".

- With the 'querying use cases' removed there are 3 remaining use cases as depicted in the use case diagram Figure 15.1. The text offers the following description of one of the use cases; descriptions for the other 2 would also be needed, of course:

|                         |
|-------------------------|
| Produce course handbook |
|-------------------------|

|   |
|---|
| This use case can only be used after the syllabus committee has determined the set of modules which will be available and the head of department has allocated duties to lecturers. [ <i>precondition</i> ] |
|---|

|  |
|--|
| The CS4 course organizer updates the core (module-independent) sections of each course by getting the current text from the system, modifying it and returning the modified version to the system. |
|--|

|   |
|---|
| The lecturer of each module, similarly, updates the description of the module by getting the current text from the system, modifying it and returning the modified version to the system. |
|---|

|   |
|---|
| These updates can be in any order. The system keeps track of which updates have been done. Once all updates for the handbook have been done, the system sends the complete text of the handbook by email to the UTO who prints it and updates <sup>1</sup> the web pages from it. |
|---|

---

<sup>1</sup> Should this not be the "CS4 organizer"?

## 1.1 Class Model

- See Figure 15.2 for one possible class model.

- See Figure 15.3 for another possible class model. This includes another class UI (*user interface*).

## 1.2 Dynamics

See Figure 15.4 for CRC cards that authors came up with for the classes involved in use case Produce course handbook.

Point is made that more responsibilities will be added to these classes as the other use cases are analyzed but should not anticipate what these will be.

Q1. Use these CRC cards to help identify the necessary operations of the classes involved in the use case.

Q2. Develop the CRC cards for the other use cases and identify the operations associated with these classes.

## 1.3 State diagrams

No classes with interesting state changes.

## 1.4 Activity diagrams

See Figure 15.5 (already referred to above).

## 2 Discussion

The authors make 2 points:

- (i) This case study is "data heavy" and there isn't much behaviour. The case study is *atypical* of OO systems in this respect.
- (ii) Concludes that a single user interface (see figure 15.3) is not really suitable. Really need different interfaces to reflect that different actors need access to different parts of the system.

### **Brief source code notes (CAN OMIT in 2005-2006)**

- See text book web-site + opening practical sessions of this module
  
- A few miscellaneous notes on Java (a reminder, to help in looking at source):
  - (1) The java.lang package provides classes that are fundamental to the design of the Java™ programming language. The java.util.\* packages provide support for the event model, *collections* framework, date and time facilities, and contains various utility classes. The java.util.zip and java.util.jar packages provide support for ZIP file format and Java Archive (JAR) file formats respectively.
  - (2)  *Casting* is used a good deal in the given source code so would need to check up on that. Associated to this would need to find out about (if not already known) about *collections* in Java including “HashSet” in particular.
  - (3) An iterator is a software design pattern that abstracts the process of scanning through a *collection* of elements one at a time. An iterator consists of a sequence S, a current position in S, and a way of stepping to the next position in S and making it the current position. Java provides an iterator through its java.util.Iterator interface. This interface uses the name "next" for the next object and supports an additional method to remove the previously returned element from the collection.

### 3 Brief note on Persistent Objects

#### Points from Panel 3.2 of recommended text

Objects which must last longer than the lifetime of a running instance of a program are called *persistent* and the issue is "*how to store such objects?*".

#### Method 1 (simple files)

An object can write itself (i.e. the current values of its attributes) into a file, and an object can be reconstructed by reading its attribute values from the file. This method is inflexible and inefficient in dealing with links between objects; however, for a small number of objects that do not need to be accessed concurrently by multiple users it may be suitable (see Java object *serialization*).

#### Method 2 (database)

Databases (DBs) provides support for transactions as well as providing persistence. *Relational DBs* store data as tables and there is need for translation between the tabular form and object forms - this is non-trivial but there is an increasing level of tool support. Situation is usually simpler and more flexible if an *object oriented DB* (OODB) is used; however, relational DBs are more common.

**Points from book by Bruegge and Dutoit (very brief outline)**

For relational DBs and flat files [unlike OODBs] we need to map the object model to a storage schema and to provide an infrastructure for converting from and to persistent storage.

A *schema* is a description of the data. In UML, class diagrams are used to describe the set of valid instances that can be created by the source code. Similarly, in relational DBs, the DB schema describes the valid set of data records that can be stored in the DB. Relational DBs store persistent data in the form of tables [also called relations], and each column represents an *attribute*.

Mapping classes and attributes: Each class is mapped to a table of the same name and each attribute of the class is mapped to a column of the table. Each data record in the table corresponds to an instance of the class i.e. an object.

Mapping associations: In brief, "one-to-many" associations can be implemented through adding a "foreign key" to a table. For "many-to-many" associations are implemented using a separate 2-column table.

**Brief indication of tools:**

DB-independent APIs are available to store persistent objects in a relational DB. In particular, for Java, check out **JDBC** on internet (esp. <http://java.sun.com/products/jdbc/index.jsp>, checked April 10<sup>th</sup>, 2006).