Human-Centred Systems Design Group Project

1. Overview

This group project is designed to integrate everything you have learned over the semester. You will work in teams of four to develop an information system for a customer. You will develop a number of UML diagrams to capture the requirements and design of the system. You will develop a relational database in MySQL to store the necessary data. You will design suitable user-interfaces for the different end-users of the system.

- Postgraduates (PG): will also develop more detailed formal design models in UML
- Undergraduates (UG): will also develop a prototype Java implementation

1.1 Submission

The project period is six weeks long; at the end you will submit one hardcopy report per team to the DCS post-box; and then a backup copy of this and any software (zipped), to MOLE. The deadline is 3pm on the due date - please refer to MOLE.

1.2 Team Building

Project teams will be formed in the following way. Use the sign-up form on Dr Simons' module page to register yourself and a preferred friend, by the end of week 3. Teams of four will be randomly formed from these pairs, every few days. Team lists (separate UG, PG lists) can be viewed from the module page - refresh your browser regularly. We will deal with odd-men-out at the end.

1.3 UML Tools

Many free "UML tools" do not comply with the UML standard! They add non-UML, or draw diagrams incorrectly (missing features, wrong box style, wrong arrowhead). You could investigate the better free tools by Papyrus, Astah, Modelio, or UMLet. In the end, any good drawing package will do. Make sure your solutions are UML2.x compliant (the latest version is 2.5). Our UML lecture slides show the standard to follow.

1.4 Database Tools

The department offers MySQL as a database server. Your properly-constituted team may email Dr Simons for a MySQL group account (accounts are created by DCS support). There are many free MySQL clients for use in Linux, and in Windows. Some run inside IDEs like Eclipse. Postgraduates will only develop a MySQL client-based solution; undergraduates will also develop a Java Swing front-end.

1.5 Java Tools

Undergraduates will build a prototype in Java. We assume you have access to the Eclipse IDE and Java development tools. Other tools like NetBeans and IntelliJ are also OK. You may have access to a Java Swing interface designer tool - if so, you will need to understand how to adapt the generated

Java code (we assume prior knowledge of Java Swing). You will need to download and install the Connector/J package to obtain the Java database driver for MySQL (see lecture).

1.6 Team Working Rules

You may collaborate within your team to develop your solution; but you *may not collaborate across teams* - this will result in unfair means investigations and mark deduction. Students will describe their individual contribution to team reports at the end.

1.7 Project Report

You will create a team report that whose main content is no longer than 20 sides. A detailed description of what is expected will be given. It will include: a number of UML models (one per page); some screenshots of your UIs; some screenshots showing evidence of your system working. It will have an introduction and a conclusion, and will present two tables describing individual contributions in two different ways. This must be signed off by the team.

1.8 System Testing

You will individually be asked to test another team's work, and submit an online web-form with your evaluations. This will count towards your own individual mark for the project. You will be automatically assigned to test a particular team. Testing will be made available for five days, starting on the code hand-in day.

2. Scenario

The following scenario describes your customer, and some information that you have gained from them, over a series of interviews. This information should be used as the source content for your UML models (in requirements and design), your HCI design and your team's database system implementation. The first interview has established the business context. The second interview has established more detailed business information. The third interview has established how the stakeholders expect to interact with the system.

2.1 Business Context

A private dental practice, *Sheffield Dental Care*, has commissioned you to design an information system to manage client registration, appointments and fee calculation. The practice consists of a dentist and a hygienist, who are the partners who own the practice, and a secretary. The clients of the practice are known as patients. A patient may visit the hygienist to have their teeth cleaned (de-scaling to remove plaque), or visit the dentist for a regular check-up (an inspection only), or visit the dentist for some remedial treatment for tooth decay (such as a tooth filling).

A patient must first register at the practice, a process managed by the secretary. The patient may choose to subscribe to a healthcare plan, to cover the cost of regular check-ups and hygiene visits, in which case the secretary records that the patient is on the plan. As part of this she asks the patient to sign a bank mandate. Separately, patient's bank will then debit a monthly sum from the patient's bank account, paid to the practice, to cover the cost of regular visits.

Initially the secretary will book appointments for a patient to visit both the dentist and the hygienist (in any order) on the same visit. Thereafter, she books the next appointment(s) at the end of the patient's previous visit. Check-ups are typically every six months, but hygiene visits may be more frequent. If the dentist identifies any dental problems after a check-up, the secretary will book one or more treatment visits (a course of treatment) at the same time for the patient. All appointments are made in the same way on a calendar. Treatment visits take longer than check-up/hygiene visits. A patient can ring up in advance, to cancel an inconvenient visit, in which case the secretary cancels that appointment and books a later appointment.

The secretary and both partners can view the appointments calendar. The dentist and hygienist do this to find out which patients they will see each day. After each patient visit, they record that they have seen the patient, logging how much the check-up or treatment should cost. Treatment may cost more than the amount covered by any healthcare plan. The secretary checks out each patient after every visit, to see whether they owe anything, and to arrange their next appointment. If the patient is undergoing a course of treatment, payment is only required at the end. The system doesn't handle payments with the bank; checking out simply prints a receipt for the amount owed and paid by the patient. The patient pays by separate debit card transaction (e.g. Visa).

2.2 Business Information

The dental practice's information system handles the following information. Every patient record has a title (Mr, Ms, Dr, etc.), forename (given name, e.g. "John"), surname (family name, e.g. "Smith"), date of birth (in dd-mm-yyyy format), and a contact phone number. Names and birth-date are not sufficient to identify a patient uniquely. Groups of patients often come from the same family living at the same address. An address records the house number, the street name, the district name, the city name, and the post code. The house number and post code are sufficient to identify an address uniquely. At least one patient lives at each address.

Booking a single appointment involves finding a free time-slot on the calendar for the dentist or hygienist, between 09:00 and 17.00, Monday-Friday, excluding public holidays. A check-up or a hygienist appointment lasts for 20 minutes. A treatment appointment lasts for one hour. The calendar can be considered as a collection of appointments. Every appointment has a date, a start-time and an end-time. Each appointment is for one partner to see one patient. Each partner and each patient can have many (non-overlapping) appointments. An appointment is uniquely identified by date, start-time and partner. The secretary can also book empty appointments (with a default "blank" patient), to indicate that the partner is not available for a given time period (e.g. because they are on holiday).

After an appointment, a partner may record one or more treatments given during that appointment to the patient. These are chosen from a fixed list of possible treatments. Each treatment has a name (e.g. "hygiene", "check-up", "amalgam filling") and a cost. Sample costs include £45 for a check-up or hygiene visit, £90 for a silver amalgam filling, £150 for a white composite resin filling, or £500 for fitting a gold crown. After a visit, the system will print a paper receipt for the total cost of the appointment, itemising the costs of treatment and giving the total due, based on the sum of treatments. If a patient has subscribed to a healthcare plan, some treatments will be offered for free, considered pre-paid out of the plan.

A patient optionally has a healthcare plan; and may have none. Each plan has a unique name, a monthly payment and the level of service covered for check-ups, hygiene visits, and repair work (cosmetic work cannot be paid out of a plan). There are only a few different kinds of plan, and many patients may subscribe to the same plan. Examples include: the *NHS free plan* (only for children under 18), no monthly charge, 2 check-ups, 2 hygiene visits and 6 repairs per year; the *maintenance plan*, costing £15 per month to cover 2 check-ups and 2 hygiene visits only; the *oral health plan*, costing £21 per month, covering 4 hygiene visits and otherwise like the maintenance plan; and the *dental repair plan*, costing £36 per month, covering up to 2 repairs and otherwise like the maintenance plan. The system records how much of their prepaid treatments have been used each year, by each patient, and resets this information each year.

2.3 Stakeholder Interaction

The secretary works in a reception area and has access to a regular desktop computer. She must be able to perform the following actions easily (and be able to see the results):

- View all the appointments for a given week (week-to-view), showing the calendars for both partners (the dentist and hygienist) on separate tabbed panes.
- Book an appointment, or cancel an appointment, or find an appointment for a given patient to see one of the partners. The result should appear on the calendar (week-to-view).
- Book an empty appointment (i.e. with no patient) for either partner, to indicate days when the dentist or the hygienist are not available (e.g. on holiday).
- Subscribe a patient to a particular healthcare plan, or unsubscribe them later.

- Review the treatments given to a patient after one or more appointments and display these treatments, their individual costs, and their total cost (see below).
- Be able to see whether some of these treatments (see above) have been pre-paid, according to the patient’s healthcare plan, and therefore the total owed is less than the full amount.
- Record when a patient has paid their outstanding bill, such that their latest treatments have been paid for, and the total cost owed is now zero.

The dental partners work in a surgery and only have access to a simple touch-screen device. Each of the partners (the dentist and the hygienist) must be able to perform the following actions easily at the end of a consultation (and be able to see the results):

- View all their appointments for a given day, seeing clearly which patient is up for the next appointment on a given day.
- Record one or more treatments given to the patient in the last consultation, indicating the kind of treatment and its cost.
- Record that they have now finished seeing this patient, committing the above changes to the shared database.

3. UML Diagrams

You will develop a number of UML diagrams as part of your solution, capturing relevant information from the scenario. Marks are awarded for capturing information succinctly and accurately. Do not invent extra features; and try to create the simplest possible solution that captures the relevant information. *Some additional diagrams will be required for postgraduate teams*.

3.1 Use Case Diagram

[ALL STUDENTS]: Create a UML use case diagram to capture the information from the *business context* (see 2.1 above). The diagram should capture all the actors involved, all the tasks carried out, including those in the wider business context. Identify tasks which will be automated in the system by placing these inside the system boundary. Identify and label three subsystems dealing with registration, appointments and payments. You do not need to write full use case descriptions.

You may find it helpful to approach the problem in the following way:

- Identify the actors involved in the scenario and any commonality between actors. Hint: do some of them behave the same way, for some tasks?
- Identify the use cases in the scenario, choosing verb-style names or phrases if possible. Keep to the smallest set of high-level use cases. Hint: if the behaviour is identical for different actors, merge the use cases.
- Link the actors to the use cases in which they participate. In rough, this may look tangled, but you can rearrange the layout in your final version. Hint: can you simplify actor participations?
- Draw one or more boundaries to identify the system and different subsystems. In rough, the use cases may be badly placed for grouping inside a boundary, but you can rearrange them in your final version.
- Do any of the tasks have partial sub-tasks? Do any tasks have variant extensions? Not necessarily; but if so, use the appropriate dependency.

3.2 Activity Diagram

[POSTGRADUATES ONLY]: Create three activity diagrams to capture the information from the *business context* (see 2.1 above). Create one diagram to represent a regular check-up and hygienist visit, one diagram to represent a course of treatment visits, and one master diagram to represent registration, the visits, rebooking and payment. Ignore cancellation and re-booking of appointments.

You may find it helpful to approach the problem in the following way:

- Some activities are repeated in a cycle. Hint: separate out the cyclic activities and sketch the loop in rough, identifying choice and merge points for the loop.
- Some activities can happen in any order, or in either order. Hint: treat such activities as happening in parallel, identifying concurrent fork and join points.
- Some activities happen optionally. Hint: identify the choice point, and also any later merge point where the flows join again.
- Some activities happen in strict sequence, but others also require an elapsed time. Hint: use a timer trigger *in parallel with* the main control flow, meeting at a concurrent join.
- Make sure your diagrams have single entry points and, where appropriate, exit points. Hint: assume the master diagram continues forever.

- Using the UML syntax for guards, label each branching choice with a short phrase with a true/false interpretation. Hint: you may need state variables to regulate the flow.

3.3 Information Model (Class Diagram)

[ALL STUDENTS]: Create a UML class diagram to capture the information model from the *business information* (see 2.2 above), showing the classes, attributes and associations. Mark the ends of associations with suitable multiplicities and role-names. This stage requires an *Information Model*; not a normalised *Data Model*.

You may find it helpful to approach the problem in the following way:

- Identify and name the information entities in the description. Hint: entities are structured objects - do not create entities for unstructured concepts.
- Identify the attributes contained by each entity. Hint: attributes are atomic properties that are not deconstructed in the description; only include attributes mentioned explicitly in the information.
- Identify any associations between entities, marking off the multiplicities in each direction. Hint: for each source entity, how many destination entities?
- Sketch the diagram, using the information gathered above, making sure you distinguish entities and attributes. Hint: do not include, as an attribute, any entity that you have linked by an association.
- An information model should seek to capture exactly the information described - do not introduce extra key fields or extra linker tables that are part of a normalised data model (this comes later).
- Check whether there are any association classes required - are there any relationships for which we need to record attributes?
- [POSTGRADUATES ONLY]: check whether any classes should declare operation signatures, to support your OCL specifications (see below).

3.4 OCL Specifications

[POSTGRADUATES ONLY]: Using all the interview information given above, write suitable expressions in OCL to assert the following semantic properties of your design. Your answers may only refer to attributes, role-names and methods specified in your class diagram. Your answers must be in correctly-formed OCL syntax, specifying the relevant context:

- Write an OCL invariant stating that if the patient's age is under eighteen, then they have a NHS Free Plan, which has no monthly charge.
- Write an OCL invariant stating that the duration of an appointment is equivalent to its end time minus its start time. Hint: this is defining the result of a method.
- Write an OCL invariant stating that the total cost for an appointment is the sum of the costs of each treatment that the patient received.
- Write an OCL precondition stating that appointments can only be booked between 09:00 and 17:00, inclusively. Hint: constrain the construction arguments for an appointment.
- Write an OCL invariant stating that for all appointments, no appointment overlaps with any other appointment.
- Write an OCL postcondition for when an appointment is checked-out, stating how the remaining annual credit on the patient's healthcare plan is adjusted.

The last two are significantly harder to specify! See the OCL guides on Dr Simons' module page for further general OCL advice.

3.5 Data Model (Class Diagram)

[ALL STUDENTS]: Create a normalised *data model* in the UML class diagram database profile (not in other ERM tools please), by normalising your *information model* (see 3.3 above). Show any required linker tables, and all primary and foreign keys. Only add surrogate keys if the domain does not have a natural key or composite key, or if the composite key would need more than two attribute-parts. Add the direction of navigation to all associations.

You may find it helpful to approach the problem in the following way:

- Identify natural primary keys and composite keys in each class from your information model; if they have none, create a surrogate key for the corresponding table.
- Normalise any many-to-many associations between classes in your information model: these will need linker-tables in the normalised model;
- Identify association classes (if any) in your information model: these will be promoted to linker-tables in the normalised model;
- Write the revised, normalised multiplicities between the above linker-tables and the tables that they link;
- Identify the foreign keys used by linker-tables and ensure they correspond to the primary keys of the linked tables; these also become the composite key of the linker-table.
- Deal similarly with any one-to-many relationship, creating suitable foreign keys; and then add navigation directions to every association.

3.6 State Machine Diagram

[POSTGRADUATES ONLY]: Create a UML protocol state machine, describing the interaction modes of the touch-screen interface used by the partners. This machine should consist of a number of states in which certain actions are possible only in that state.

You may find it helpful to approach the problem in the following way:

- The default view will allow selection of either the dentist's or the hygienist's calendar for the day (different for each partner);
- The calendar will display a list of appointments for that partner; it is possible to exit to the default view, or select an appointment;
- When viewing an appointment, it is possible to select treatments, or exit to the calendar view (without commit), or commit the appointment (logging any treatments);
- When viewing treatments, it is possible to select a treatment to add it to the current appointment; selecting will also exit to the appointment view (displaying the added treatment); it is possible to exit (without selecting) to the appointment view.

4. Software System

You will develop a software system that implements your design for the *Sheffield Dental Care* information system. All teams will build a suitable MySQL database implementation to store the relevant information, and demonstrate that certain SQL queries run effectively and retrieve the intended results.

4.1 System Behaviour (with Evidence)

Your software systems will be tested to ensure that they can perform specific queries, which are detailed below. Your reports should contain captured screen-shots to demonstrate that your systems behave correctly (with before/after state of the database) in response to the queries.

- Postgraduates (PG): will give screenshots of SQL queries running on any suitable MySQL client, showing the queries, and before/after database snapshots;
- Undergraduates (UG): may give screenshots of their Java UI, showing how the state of the system is updated after each interaction (if displayed in Java); otherwise use database snapshots as above;

4.2 Query Processing

Marks will be awarded proportionately for being able to run each of the following queries, and for obtaining the correct responses and system states:

- Registering a new patient and then showing that the new patient exists in the DB.
- Subscribing a patient to a healthcare plan and then showing that the patient is linked with the relevant plan in the DB, through a new record of relevant treatment-credits for the plan, for the current year;
- Creating an appointment for a patient to see the dentist and showing that this appointment appears in the receptionist's week-to-view calendar for the dentist, and also appears in the dentist's appointments for that day;
- Attempting to create two appointments for a patient that are refused, either because the patient or the partner already have appointments at this time;
- Booking two days holiday for the hygienist and then showing that blank appointments fill the relevant two days on the hygienist's week-to-view calendar;
- Recording two treatments given by the dentist to a patient and then showing that these have been added to the bill for the appointment;
- Displaying the total cost of an appointment for a patient who is on a healthcare plan, showing the total cost of treatments and the amount owed by the patient;
- Processing a payment by a patient who is on a healthcare plan, showing how their treatment-credits for that year are adjusted, and that they now owe nothing.

4.3 User Interfaces

Marks will be awarded for the design of your user interfaces for the different stakeholders. These should clearly support the ergonomic working style of that kind of user.

- Postgraduates (PG): will develop full mock-ups of their user interfaces
- Undergraduates (UG): will develop a Java Swing implementation

You should give suitable screen-shots (UG) or mock-ups (PG) to demonstrate your layouts, and show how they support the working style.

4.4 Code Submission

You should submit an electronic zipped file with all your code to MOLE, by the due deadline.

5. The Report

Your group report should have the correct DCS coversheet (for the whole team, with four barcodes). It should identify which team you are (team numbers will be allocated during team-building).

5.1 Report Contents

To standardise the format and length of reports, please keep to the following:

- 1 page for an introduction, reflecting on how you organised the work.
- 1 page for the UML use case diagram [all]
- 1 page for the UML activity diagrams [PG]
- 1 page for the UML information model [all]
- 1 page for the OCL specifications [PG]
- 1 page for the UML normalised data model [all]
- 1 page for the UML state machine model [PG]
- 1 page for secretary UI screenshots [UG] or mock-ups [PG]
- 1 page for partner UI screenshots [UG] or mock-ups [PG]
- 6 pages (max) for evidence of query processing
- 1 page for effort declaration, reflecting on how effectively you worked.

5.2 Signed Effort Declaration

Finally, the report should contain a short reflection on how effectively the team worked; whether you made changes to work-allocation; whether there were serious problems; etc.

You should also give two tables, indicating:

- the actual work that each team member did, and also

- an estimate of the effort contributed by each member.

If every team member contributed equally, then indicate 100% effort per person. If the work was unequally shared, then indicate e.g. 120% and 80%, according to the perceived contribution. (The total effort is always 100% * team-size).

The whole team **must sign off** this declaration of effort, as proof that you all agreed to it.