Assignment Four

Dominic Love

Submission For Unit 4

Advanced Diploma in IT Systems Analysis & Design

Department for Continuing Education

University of Oxford

CONTENTS

QUESTION 1

1.1 Preliminary class diagram

Perhaps it be odd to open this assignment with a structure model. While we are told that preliminary analysis has identified ten classes in the Orchard Dental Surgery (ODS) system, its functional scope is still unbeknown to us by use case (Cockburn, 2000, p. 45). As Fowler, suggests, Object Managment Group's Unified Modelling Language (UML) is an iterative prone open standard (2004, p. 21). So I started after analysing use cases and draw on those to feed my object design criteria.

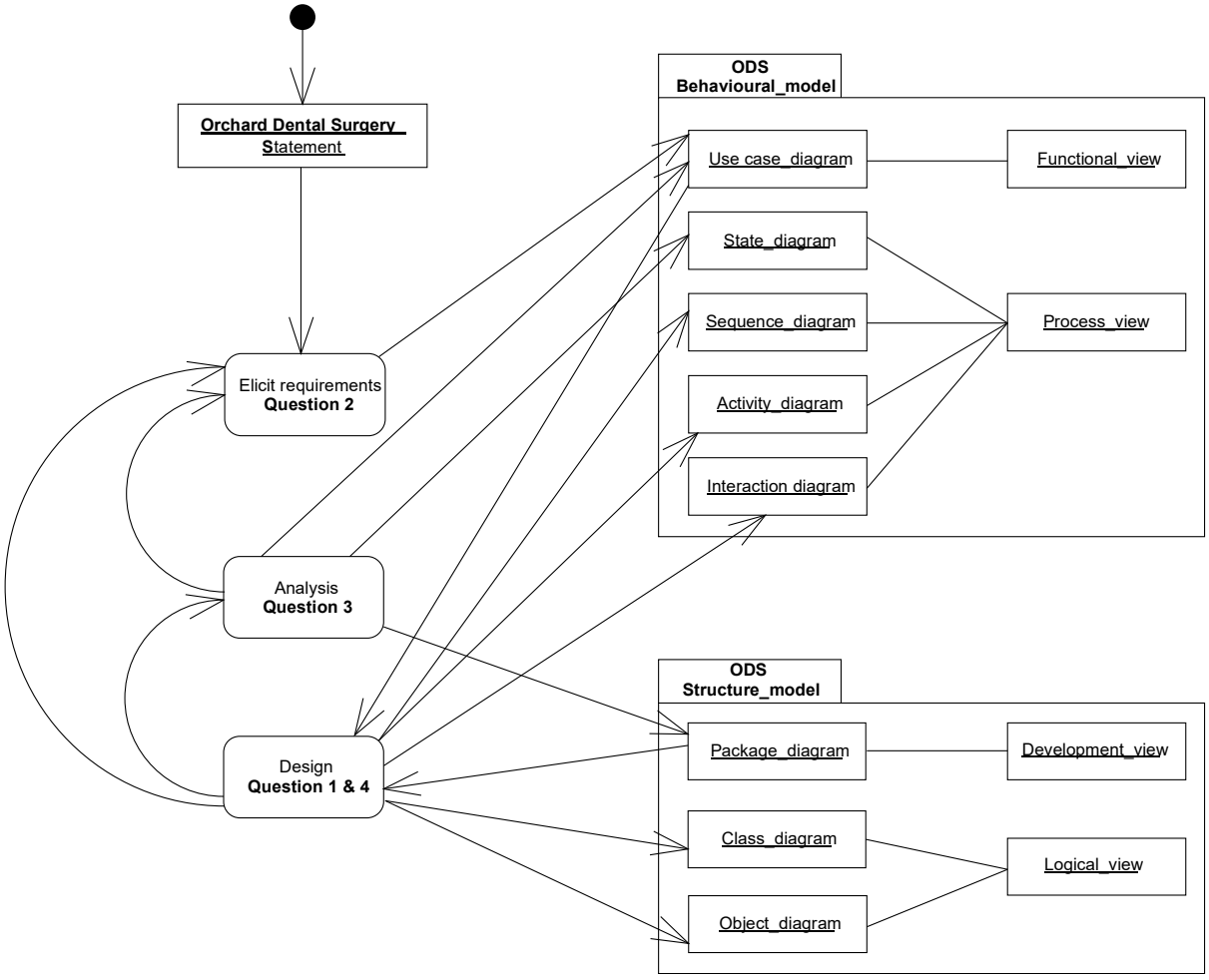
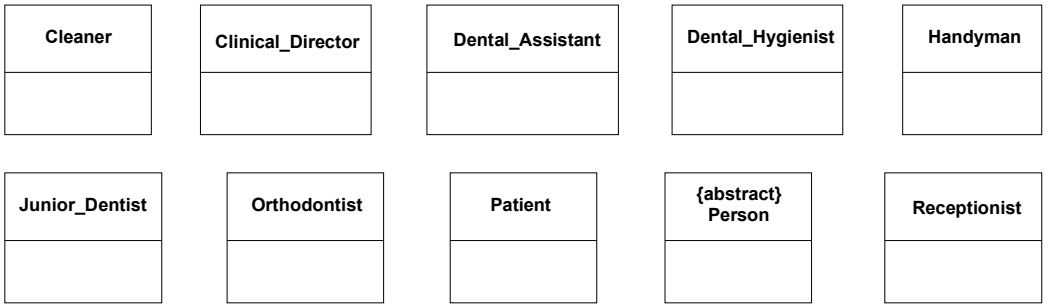


Figure 1.1 Assignment roadmap – Bruegge & Dutoit adaption (2010, p. 123).

2.1.1 Classes

Question 1 gives us ten classes to build a blueprint schema for a set of objects to manifest in the system that are purported to have been found from analysis. As a design specific, nine of those classes must inherit from an {Abtract} Person class.

Figure 1.2 Ten ODS classes



2.1.2 Specification

A list of five requirements can be obtained from the ODS statement to form a specification.

Figure 1.3 ODS Specification

SpID	Description	Requirement
1	Used by administrators to control the management of a dental surgery	Manage dental surgery
2	The surgery provides its patients with a number of treatments	Treat patient
3	The surgery accepts patients both privately and as NHS provision	Accept NHS provision
4	For special treatments such as tooth implant an external Orthodontist is engaged on a case to case basis	Engage orthodontist
5	A web page that patients can book their appointments for general treatments such as check-up and visit to the hygienist	Book appointment

Note that SpID 4 and 5 are use cases contained in Question 2 - Instruct orthodontist and Book appointment.

2.1.3 Objects and classes

While the specification guides what functions the system must have at a minimum, it does not advise how properties should be modelled. To do this, some model components may be

extracted from the natural language of the textual specification, aka NLP, per Abbott (1983). I use a parse tree generator (National Centre for Text Mining, 2012) for each requirement of the specification to aid the O-O design schema of the 'real-world' Bajwa (2009) with mindfulness to Bruegge & Dutoit limitations (2010, pp. 180-181)

.

Figure 1.4 Parse trees for ODS specification

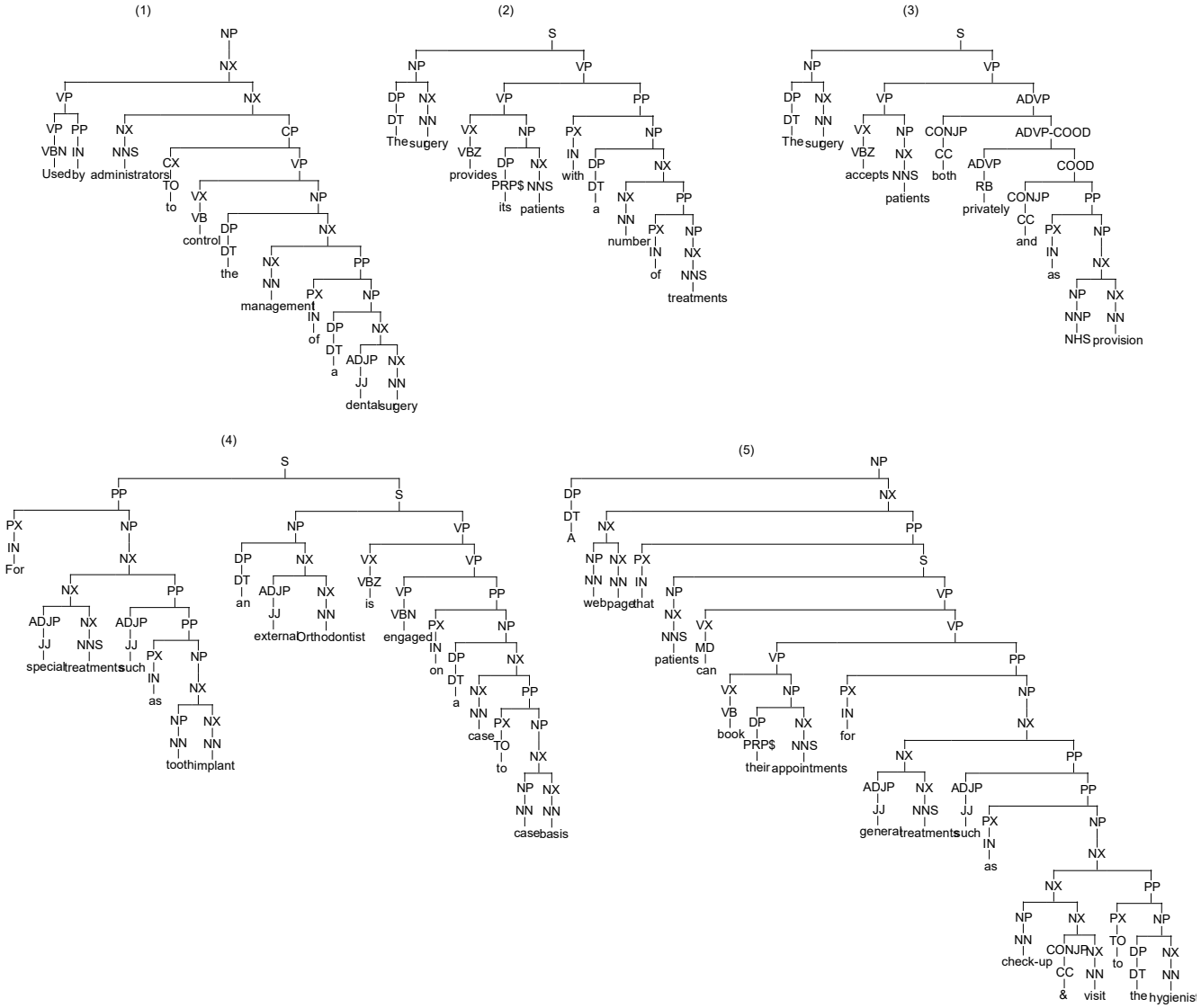


Figure 1.5 Model candidate component table

SpID	Classes (Noun NP)	Attributes (Adjective ADJP)	Association (Verb VP)	Inheritance (Is a kind of/ is one of either)
1	'Used' becomes 'User'	'dental': Redundant data.	'control'	'Used by administrators…': Discriminates non-administrators.
	'administrators' eliminate plural		'Management' NP becomes 'Manages' association	
	Surgery': Redundant data.			
2	'Surgery': Ditto		'Provides a number of': Becomes 'Provides' [*] association	
	'treatment'			
	'patient'			
3	'surgery': Ditto		'accepts'	'…both privately and as NHS provision….' Possible differentiation between class of patient but could be an attribute.
	'patient'		'Provision' NP becomes 'Provides' association	
		'NHS' NP becomes attribute		
4	'tooth' Redundant data.	'special'	'is engaged'	'For special treatments…': Distinguishes from general treatment
	'Orthodontist'	'external'		'…external Orthodontist….' As opposed to an internal object
	'case'	'Implant' VP becomes an attribute .		
	'basis': Suggests multiplicity [1..*]	'Treatment' NP ditto.		
5	'Web' 'Page': combine.	'general'	can	'…for general treatments….': A kind of treatment
	'hygienist'	'Treatments' like treatment NP ditto.	book	'…visit to the hygienist….' A kind of staff.
	'patient'	check-up NP becomes attribute		
	'appointment' i.e. 'Booking' Treatments.		NP visit becomes attribute.	

2.1.4 Preliminary class diagram

One such distinction of inheritance is 'Used by administrators…'. Drawing on the use case 1.3 of Question 3, staff like Dentist, is a kind of class General_administrator. This use case is also adamant that there is a Root_administrator with different functions from a General_administrator. It may be assumed that the remaining staff classes can also follow suit to fulfil inheritance from an intermediary General_administrator class.

Figure 1.6 (a) Non-administrators (b) Administrators

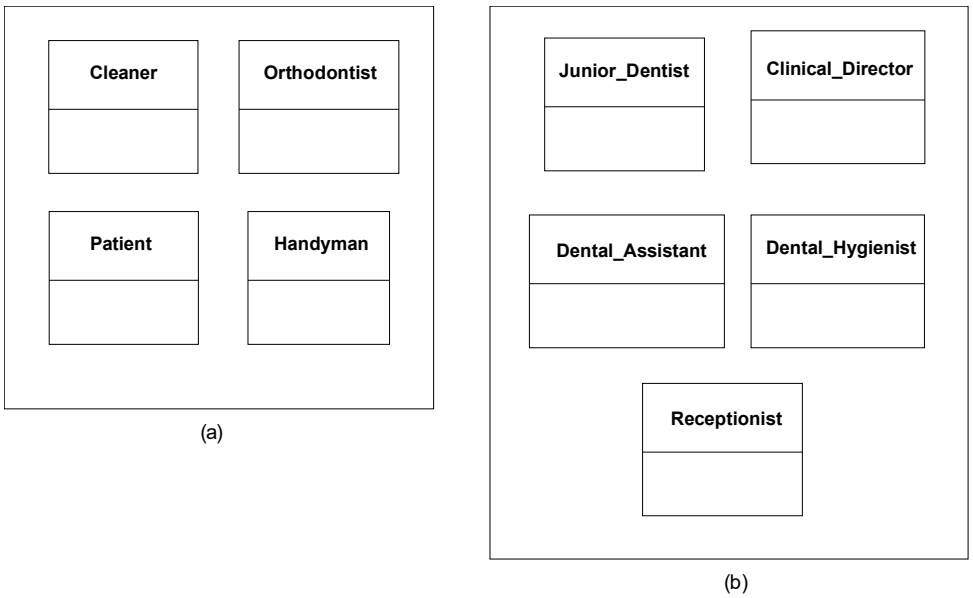


Figure 1.7 New classes

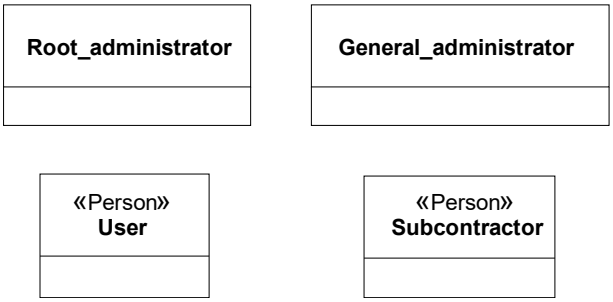
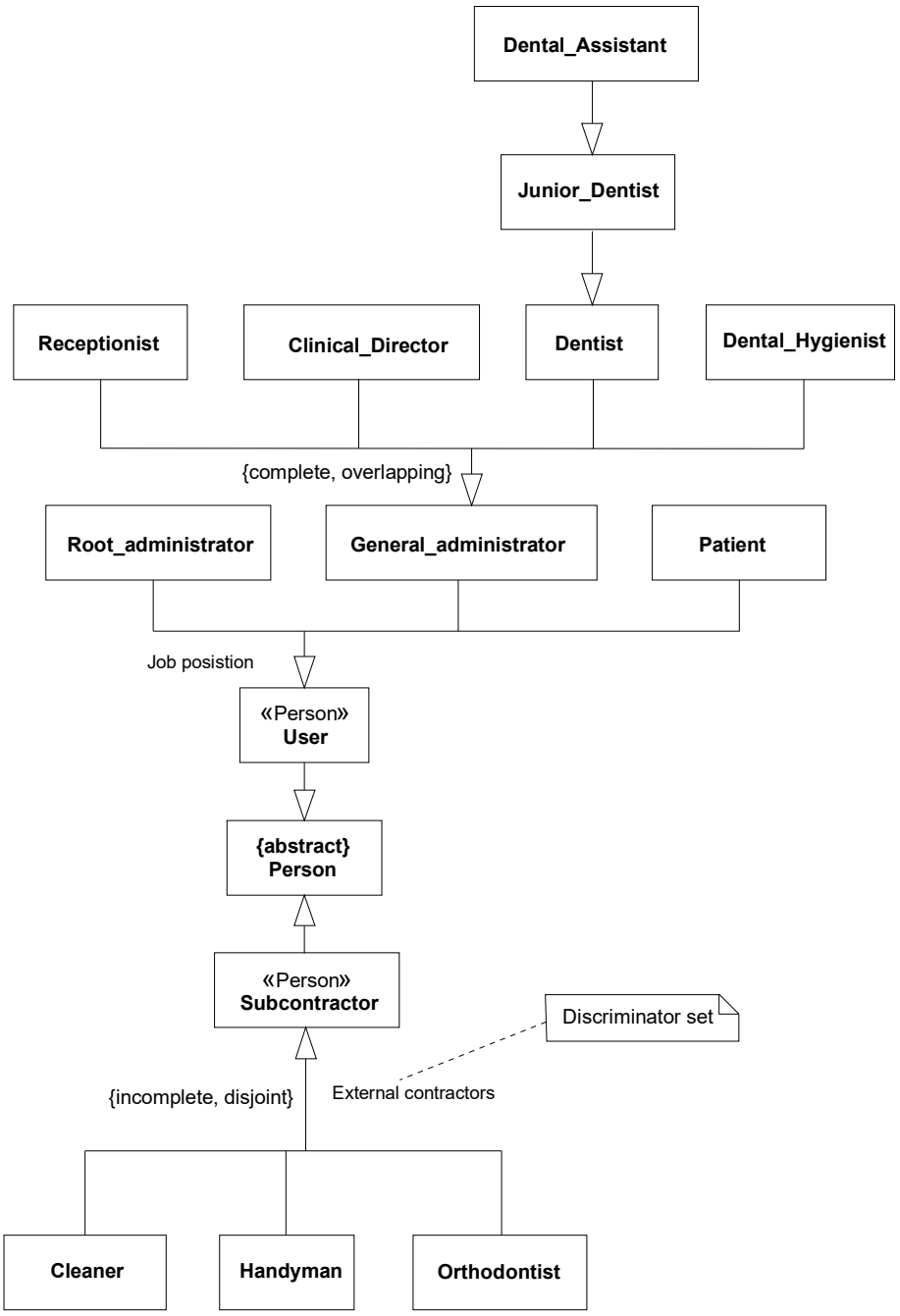
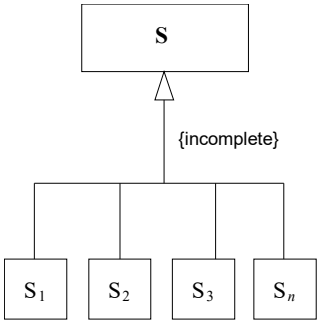


Figure 1.8 Preliminary class diagram



The Subcontractor class has an opposite discriminator set. A subclass of Subcontractor cannot be a combined instance of another subclass, but could be extended – that is, other types of Subcontractor 'S' may be stored in the database (Berardi, et al., 2005, p. 79)



$$\forall x.\, \mathrm{S}_i(x) \supset \mathrm{S}(x), \text{ for } i = 1, \ldots n$$

Figure 1.9 Formal inheritance - S

2.1.5 Extended class diagram

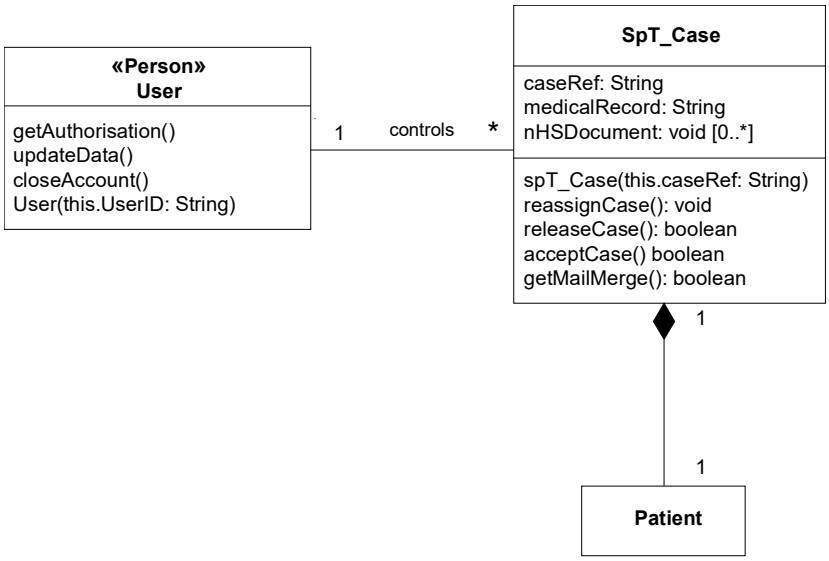


Figure 1.10 Class association User – spT_Case

Figure 1.11 (a) Association class (b) Qualifying association

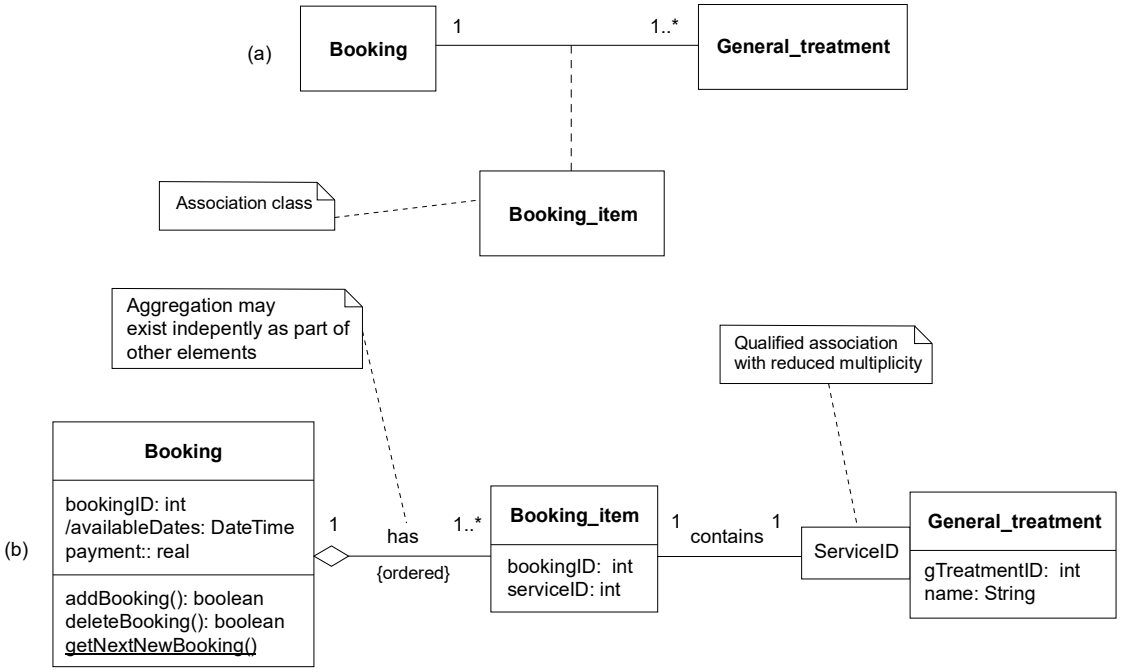
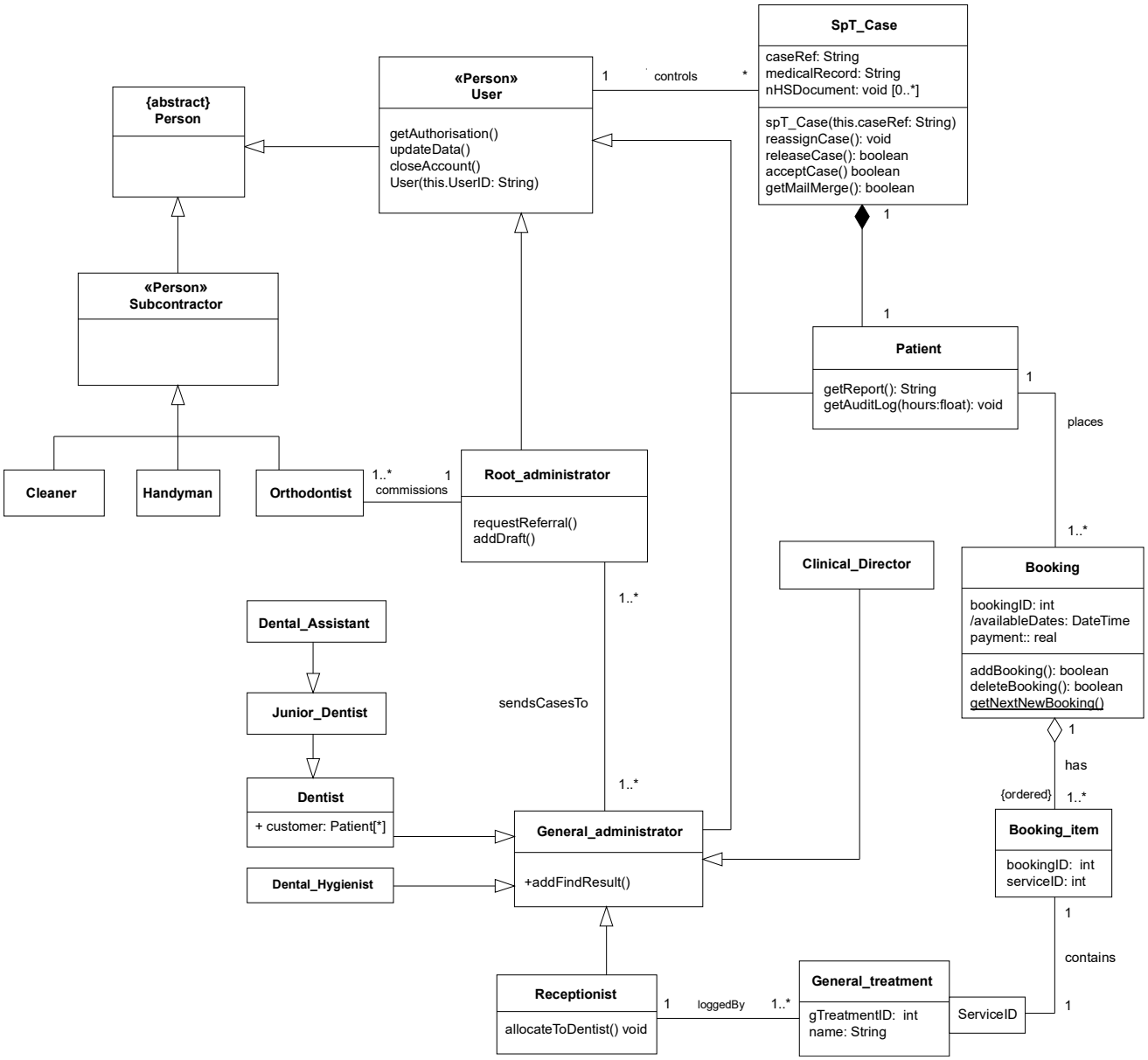


Figure 1.12 Extended preliminary class diagram



1.2 Five attributes that should be defined for the Person class

A kind of Person P_S is a specialization of {abstract} Person P where the extension of P_S is a subset of the said P (Henz, 1998, p. 18). Therefore, where an inheritance exists, and they have no private visibility, every subclass gets all class attributes of generalized {abstract} Person, save that the abstract class itself cannot be instantiated – Larman's "100%" (2002, p. 395).

An instance variable belonging to an object inherited from a Person class will be a public primitive attribute i.e. property. It is a location for the ODS system to store object data in a program which are defined by the object's concrete class like this:

```
firstName = "Bob" ; // which assigns Bob to firstname instantiated from a subclass of
                    {abstract} Person.
```

Child classes with attributes resembling a kind of Person are given (1) firstName and (2) secondName. Each user and non-user will have a (3) address, and (4) email address protected # from lesser parts of the class structure (Hamilton & Miles , 2006, p. 96). I disregard telephone number as an essential attribute since they are more liable to updates and name changes. It would also help us to identify this taxonomy by an enumeration data type (5) roleType, thereby discriminating the Person baseclass exclusively as User {*xor*} Subcontractor. This is possible in O-O systems which supports polymorphism so that a reference points to objects of different types (Jalote, 2008, p. 145)

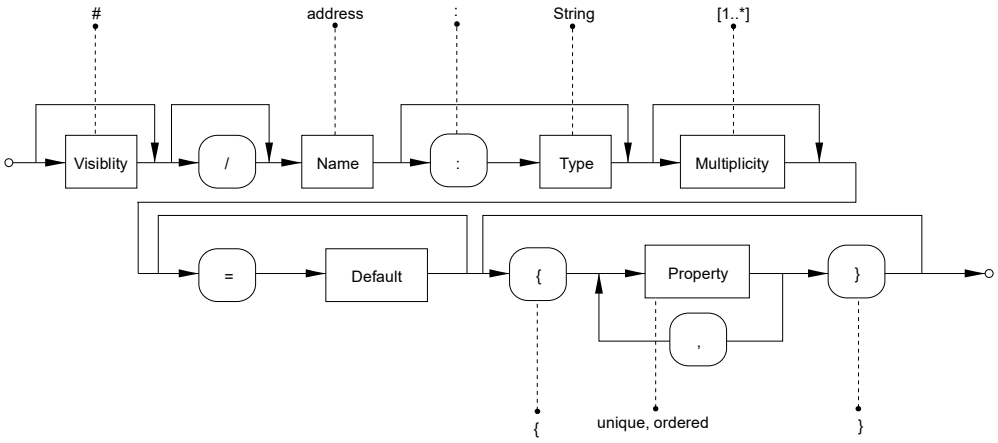
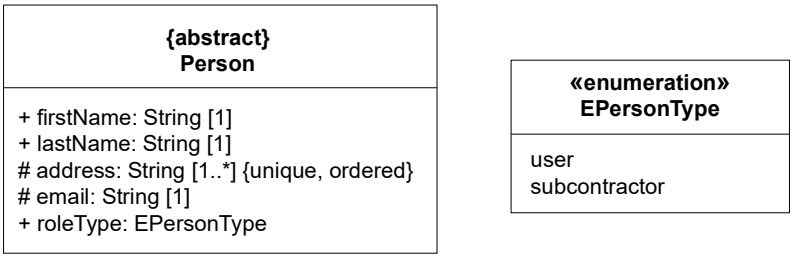


Figure 1.13 Railroad (Seidl, et al., 2012) – *adapted ODS attribute 'address'*




```
abstract class Person {
   public String firstName;
   public String lastName;
   protected String[] address;
   protected String email;
   public EPersonType roleType;
}

Enumeration EPersonType {
   user;
   subcontractor;
}
```

Figure 1.14 Five attributes and Java code

1.3 Five attributes that should not be defined for the Person class

While the transitivity of generalization passes {abstract} Person class features to multiple child classes, each can have their own additional attributes and operations encapsulated. Class User, for example, has already a unique set of methods, but needs additional attributes such that they 'partition the universe of objects' in class {abstract} Person for utility - Dathan & Ramnath (2015, p. 225). If a class User is found to be particular useful, and needs extra attributes, a new class can extend from it with the added attributes and methods (Edward , 2006, p. 247). To discover attributes that are undefinable for an {abstract} Person class is to find classes that are what Meyer refers to as open for extension so code can be added as an 'extension' of the baseclass (Meyer , 1988, p. 57).

```
class User extends Person // class hierarchy extension
{
// User attributes
}
```

Concrete subclasses have two parts. The first, a derived hidden part, 1.2 above, and an incremental extended part, forming a 'substitutable' subclass for its parent {abstract} Person – Liskov Substitution Principle (1988). A User then will have five attributes that cannot constitute a Person. The first, a (1) UserID, - something a Subcontractor cannot possess too – is added with an «invariant» constraint. The rationale for this is that every user will have a user identification number that is not null thereby forming an inheritance contract (Bruegge & Dutoit, 2010, p. 374). (2) username, and private login (3) password for a Person is necessary for users to access the system. Since users are employees of the ODS, it would be useful to reveal their National Insurance number (4) nINumber for Pay-As-You-Earn tax. Analysis of use case 1.3 indicates that ODS employees have variable functionality that extends so an enumeration of (5) authorisation, is included in the User class.

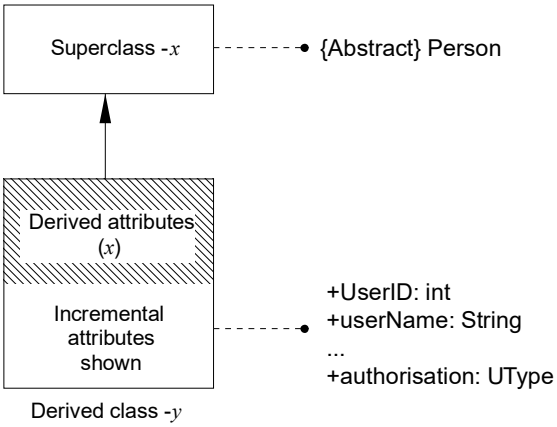
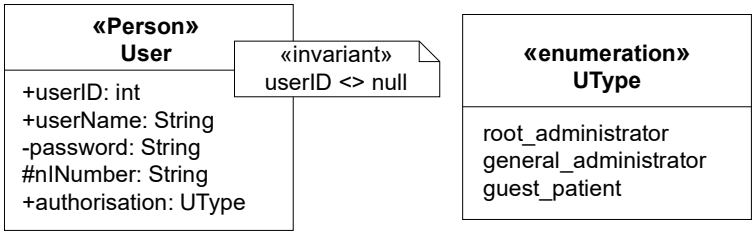


Figure 1.15 Derived attribute (Jalote, 2008) - *adaption for ODS*




```
class User extends Person {
   public int userID;
   public String userName;
   private String password;
   protected String nINumber;
   public UType authorisation;
}

Enumeration UType {
   root_administrator;
   general_administrator;
   guest_patient;
}
```

Figure 1.16 Five attributes and Java code

QUESTION 2

2.1 Associations Clinical_Director - Patient and Orthodontist – Dental Assistant

Where there is a persistent non-hierarchal relation between at least two classifiers, and their instantiated links have common semantics and structures, an 'association' is formed. This means that an association is both conceptually and physically disposed, for if class C_1 has a binary association A with class C_2, a given object of the former will be related to the latter (Berardi, et al., 2005, p. 76) .

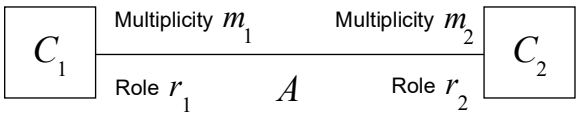


Figure 2.1 Class binary association

2.1.1 Clinical_Director – Patient

Use cases in question 3 do not narrate what scenarios might affect a Clinical_Director object nor give account of a possible Patient communication partner. However, note that operation getReport() is defined in the Patient class preliminary diagram in Figure 1.12. This refers to a named method that vests object behaviour bound by a 'contract or obligation' of the Patient classifier to disclose records (Larman, 2002, p. 216). In other words, a Patient object has responsibility to give information about itself at the call of knowing objects, so class methods and attribute visibilities are set to public. This is 'need to know' management information for a Clinical_Director object related to an arity *m* of one:many Patient objects.

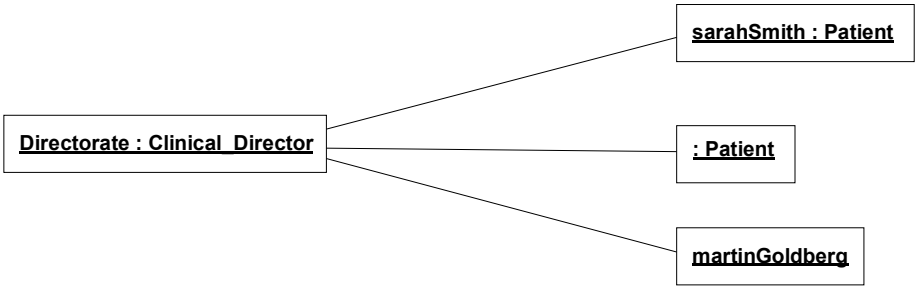


Figure 2.2 Object diagram – links in varying notation

A collection class should be used to gather the instances of Patient objects with each having a reference kept in a vector implementing Java library `java.util` (Arlow & Neustadt, 2002, pp. 284-285).

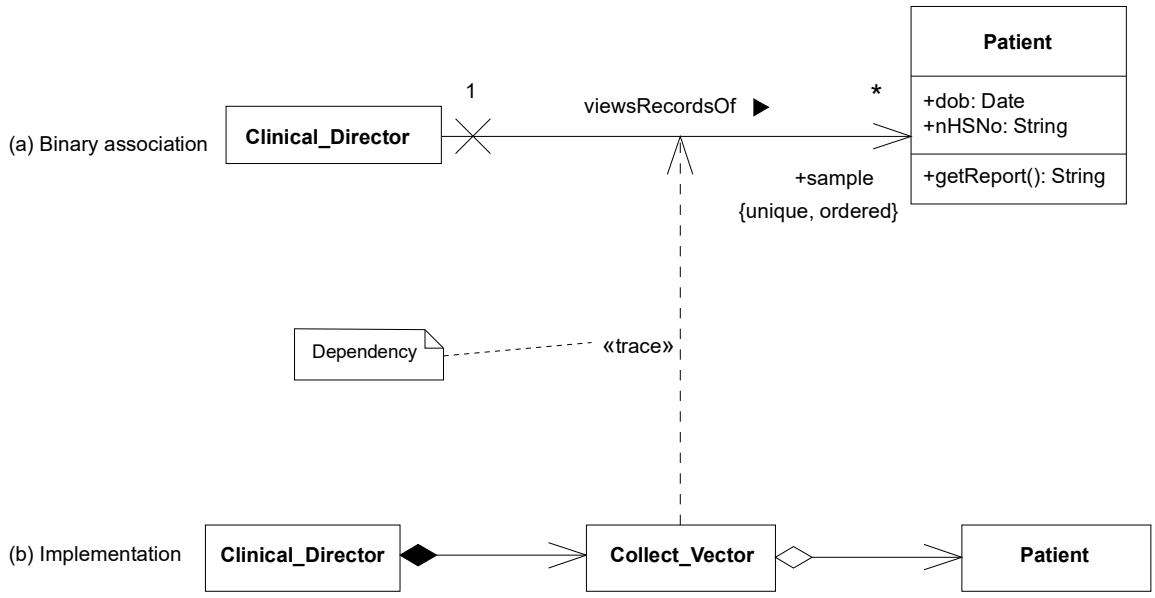


Figure 2.3 (a) Plausible association Clinical_Director – Patient (b) Vector collection

2.1.2 Orthodontist – Dental assistant

Looking at use case 1.3 below, the precondition 'A Dentist must write their instructions in the Patient referral case.' sets us on course to finding an association between Orthodontist and Dental_Assistant. Moreover, Preliminary class diagram Figure 1.8 shows us that a Dental_Assistant is a traceable specialisation of Dentist. It can be surmised from both sources, that a dentist may devolve this precondition to dental assistants who would locate the appropriate Orthodontist to write an instruction email. An Orthodontist object, from a collection, returns details of the chosen practitioner to outsource. Physically, this association is 'bi-directional' with multiplicity [1..*] at either end (Dathan & Ramnath, 2015, p. 51),

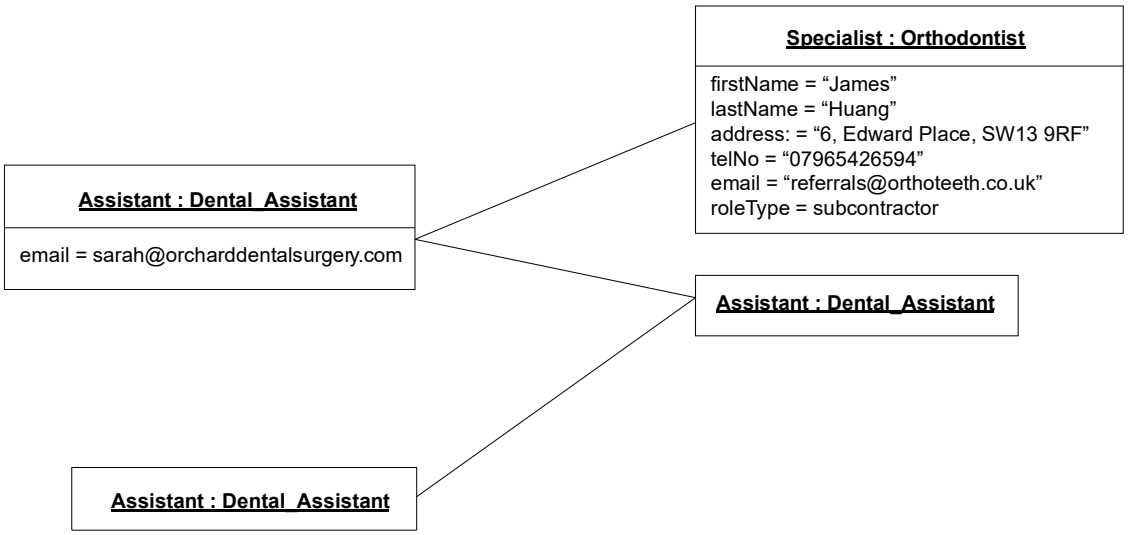


Figure 2.4 object diagram – Bi-directional links in varying notation

.

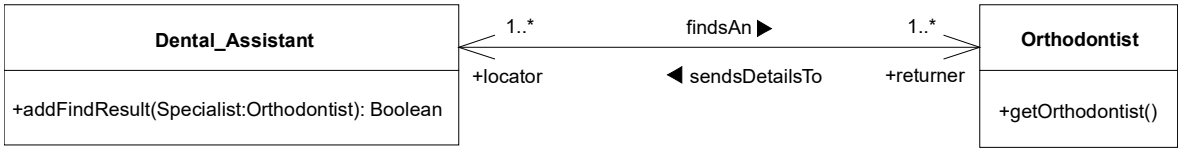


Figure 2.5 Plausible association Dental_Assistant – Orthodontist

The association diagram in Figure 2.5 is missing intermediary elements. This is because use case 1.3 states that it is a Root_administrator object that sends, but not writes, a commissioning email to an Orthodontist. It also confirms that this communication is associated with a 'referral queue hub' to which cases are sent. The sequence between each class can be summarised in an *n*-array association (Seidl, et al., 2012, p. 64):

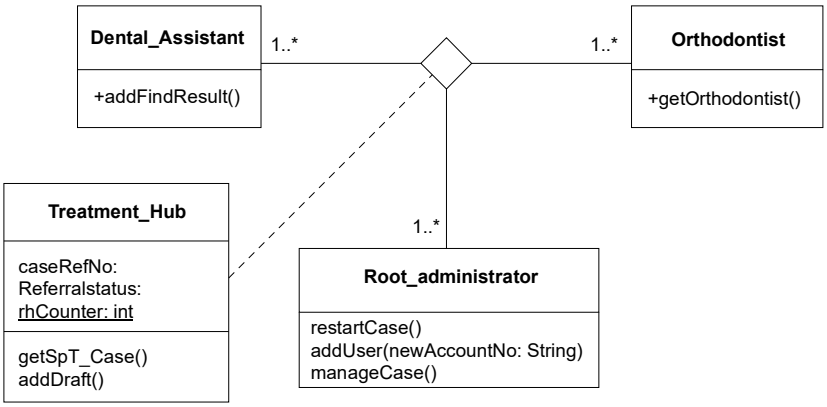
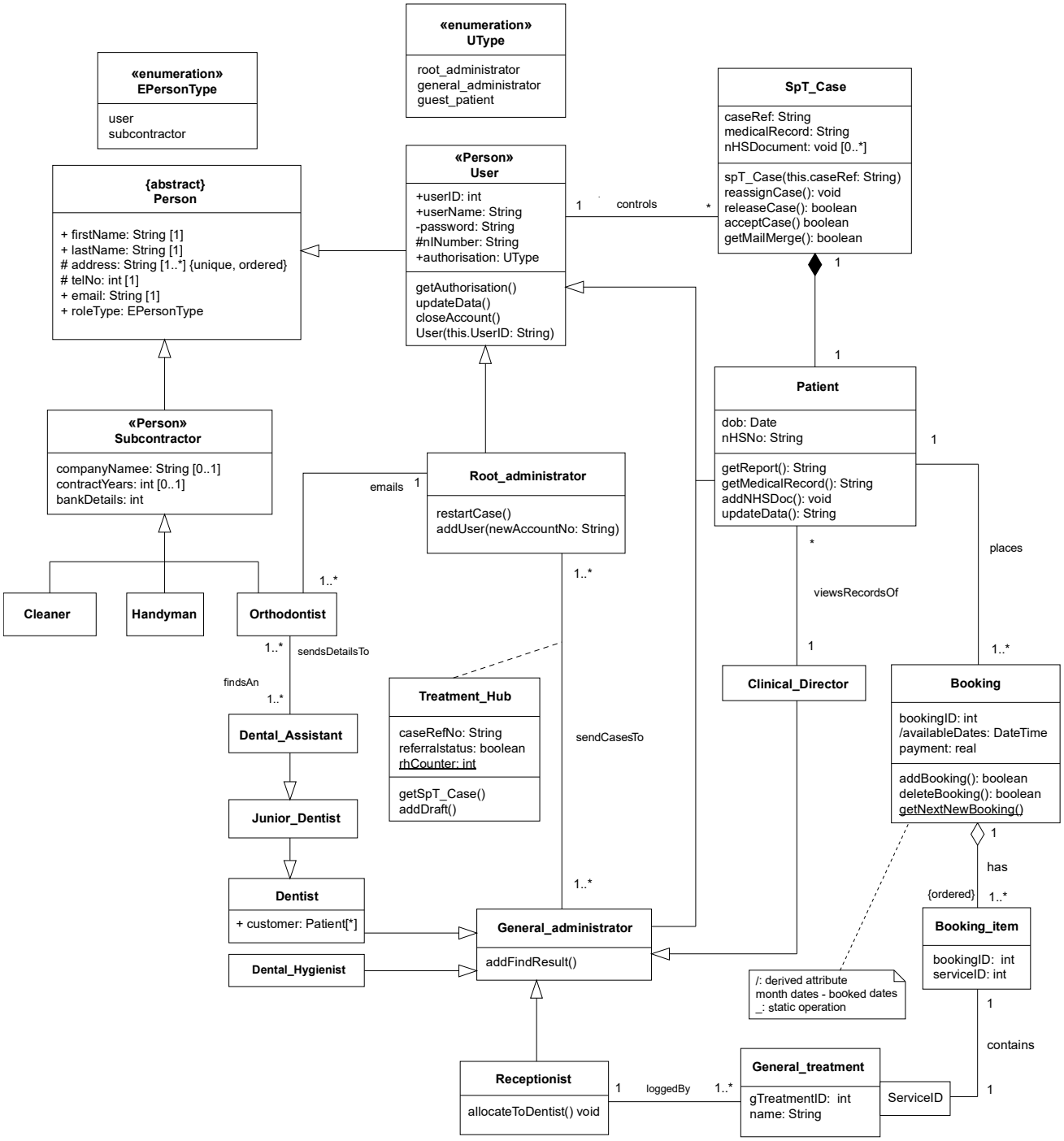


Figure 2.6 Plausible n-array association

Figure 2.7 Updated preliminary class diagram



2.2 Three use cases associated with the system

2.2.1 Identifying use cases

We can goals by assessing the triggers to which the system should respond by decomposing elicited actions as 'use cases' in an event table (Muhairat & Rafa E. , 2009). Spotting triggers as events may be got by asking 'What business events occur that will require the system to respond?' (Satzinger, et al., 2012, p. 71). But as you will see next, the answer is not straightforward.

Figure 2.8 Event table – 'Used by administrators to control the management of a dental surgery'

ID	Event	Source	Action	Object	«Include»	Destination
1.0	Admin controls management	Root_admin	{abstract} Manage surgery data	{Abstract} Surgery data		Root_admin

Here, for instance, the textual specification about administrators controlling management is an event that cannot account for a single action since we are not told how it will respond. The role of managing surgery data is, therefore, an 'abstraction', for there are no use case specifics. (Seidl, et al., 2012, pp. 29-32).

Figure 2.9 Generalised event table - Admin controls management

ID	Event	Source	Action	Object	«Include»	Destination
1.1	Administrator receives request for orthodontist from patient	Root_admin	Log orthodontist request	Server		Root_admin
1.2	Administrator receives patient booking	Root_admin	Allocate patient to dentist	Booking		Dentist
1.3	Dentist refers patient to orthodontist	Root_admin	Instruct orthodontist	Patient case	Own case	Orthodontist
1.4	Administrator views NHS support documents	Root_admin	View NHS files	Binary large		Root_admin
1.5	Administrator produces management information	Root_admin	Download dataset	Server		Root_admin
1.6	Administrator	Root_admin	Invoice patient	Invoice		Root_admin

	issues patient bill					
1.7	Administrator opens a referral case	Root_admin	Own case	Patient case		Root_admin

The table can continue to document further events from other sources like so:

Figure 2.10 ODS non-exhaustive event table continued

2.1	Time to issue appointment notification to patient	Server	Email patient	Booking		Patient
2.2	Time to permanently delete patient account	Server	Delete patient	Account		Root_admin
2.3	Patient confirms account	Server	Authorise user	Patient account	Log-in	Patient
2.4	Patient enters data	Server	Save entries	Patient account	Update account	Patient
2.5	Patient confirms account	Server	Authorise user	Patient account	Log-in	Patient
2.6	Patient enters data	Server	Save entries	Patient account	Update account	Patient

3.1	Patient registers	Patient (new)	Create account	Account		Patient
3.2	Patient requests an orthodontist	Patient	Request orthodontist	Server		Patient
3.3	Patient makes an appointment	Patient	Book appointment	Webpage	Find time and date	Patient
3.4	Patient searches availabilities	Patient	Find time and date	Calendar		Patient
3.5	Patient pays for treatment	Patient	Purchase treatment	Merchant ID		Consumer bank account
3.6	Patient cancels appointment	Patient	Delete appointment	Booking		Patient
3.7	Patient uploads NHS treatment documentation	Patient	Save NHS treatment file	Account		Patient
3.8	Patient updates account	Patient	Update account	Account		Patient
3.9	Patient closes account	Patient	Close account	Account	Delete patient Pre-condition: 30d inactivity	Patient

4.1	Dentist has to cancel the booking	Dentist	Cancel booking	Booking	Email patient	Patient
4.2	Dentist writes up treatment	Dentist	Record report	Treatment file	Update patient record	Dentist
4.3	Dentist prints write up	Dentist	Print report	Treatment file		Dentist
4.4	Dentist views patient record	Dentist	View patient record	Patient account		Dentist
4.5	Dentist edits patient record	Dentist	Update patient record	Patient account	View patient record	Dentist

Notice that I have separated the events table into four parts according to the type of source Root_administrator, Server, Patient, Dentist. I pick one row from each at random, except *'Server'*, (its notation will be prohibited in this assignment per Lockhart & Chakaveh (2016, p. 49), I then use the 'action' verb as the basis to name three use cases.

Figure 2.11 Three use cases

ID	Use Case	Actor	«Include»
3.3	Book appointment	Patient	Find time and date
4.2	Record report	Dentist	Update patient record
1.3	Instruct orthodontist	Root_administrator	Own referral case

2.3 Informal account of what the use cases achieve

Using the event table above, a basic informal context diagram can be developed to identify the main actors of the system so readers all view the information flow in the same way (Bittner & Spence, 2003 , p. 132).

Figure 2.12 ODS context diagram

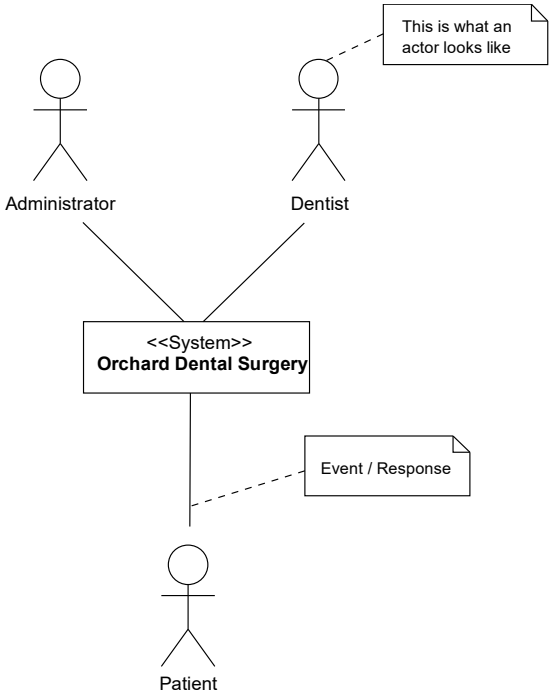
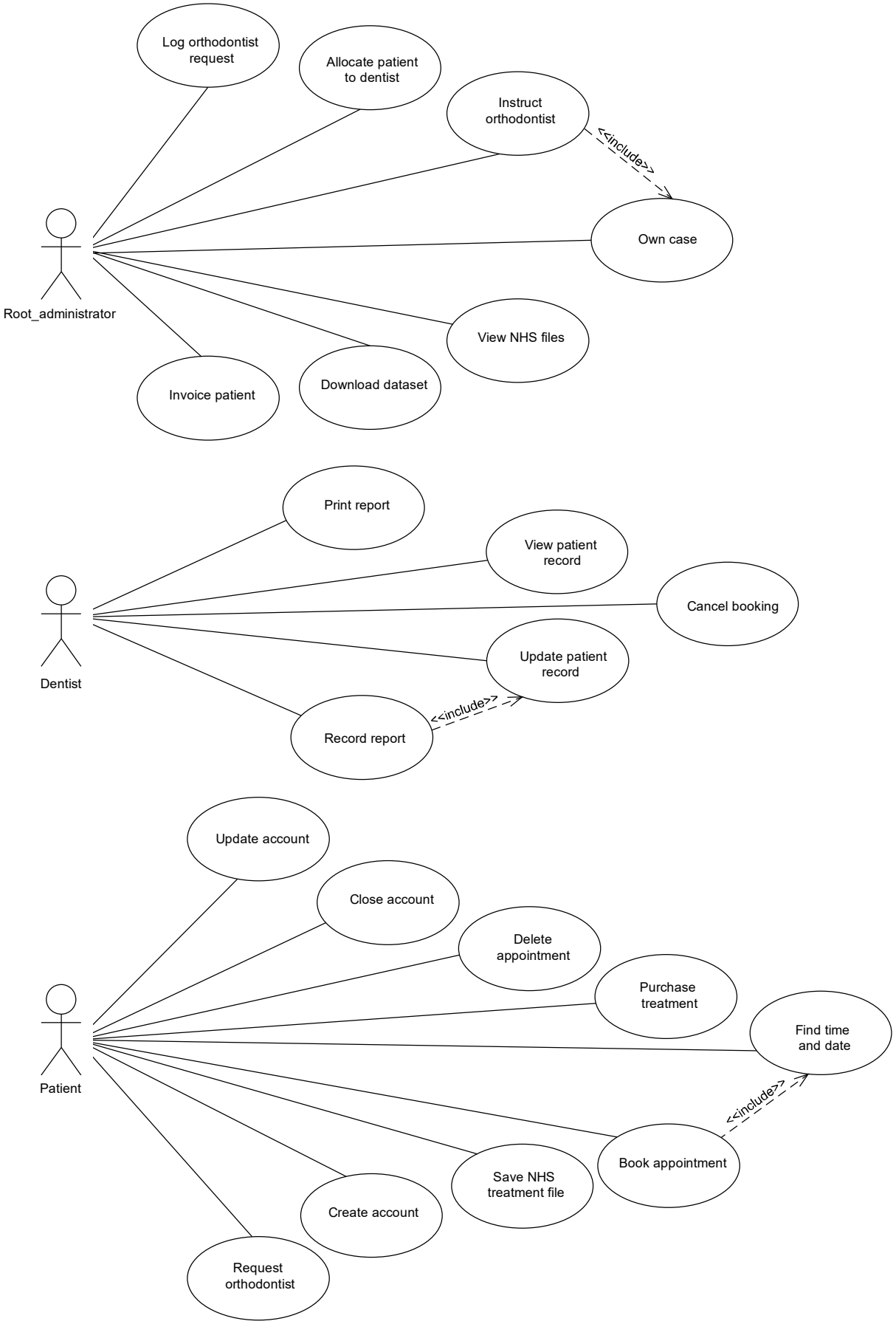


Figure 2.13 Use case diagram – ODS management system



A deliberate omission of the system boundary box has been made (an example is in the Appendix) - this is use case notation which may not be useable as per Lockhart & Chakaveh, (2016, p. 49).

2.3.1 Use case accounts

ID	3.3
Name	Book appointment
Goal:	To manage the functionality of general treatment booking appointments for check-ups and hygiene
Short description	After choosing their treatment, a Patient searches for available booking slots. The Patient books an appointment of their choice from the results to receive dental treatment on a specific date and time.
Actor	Patient (Primary)

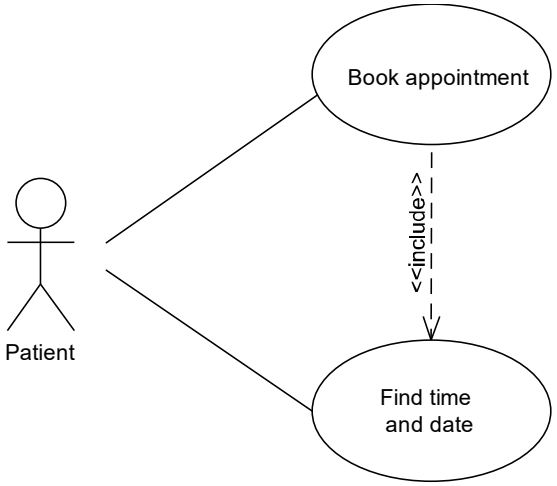


Figure 2.14 ID 3.3

ID	4.2
Name:	Record report
Goal	To maintain customer health records that can be requested under the Data Protection Act 1998 per NHS (2016); a likely ‘offstage actor’ (Larman, 2002, p. 70).
Short description	A Dentist checks a Patient and writes a report of the treatment that they gave in their dental record. The treatment report has to be saved in order to record it.
Actor	Dentist (Supporting)

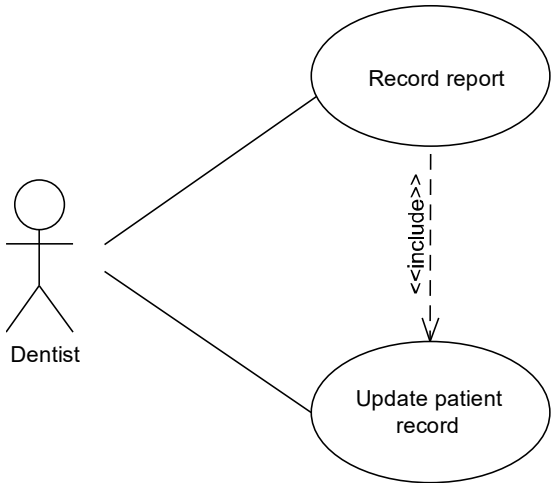


Figure 2.15 ID 4.2

ID	1.3
Name	Instruct orthodontist
Goal	Engage an external orthodontist to treat a patient on referral at the ODS.
Short description	A Root_administrator receives a patient referral case in their work stream from a dentist to accept and sends the case in a prewritten commissioning email to an external contracted orthodontist.
Actor	Root_administrator (Primary)

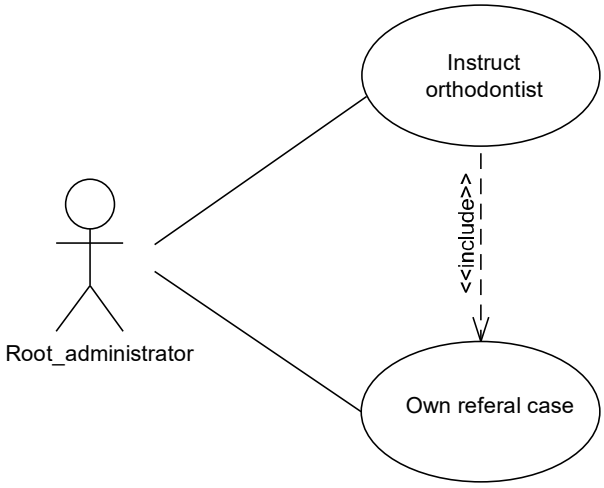


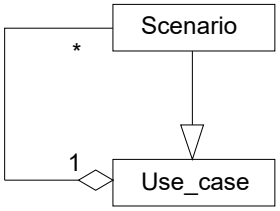
Figure 2.16 ID 1.3

QUESTION 3

3.1 Plausible scenarios associated with one use case

The ODS focuses its specification on '…a system that is to be used by administrators'. On this basis, 1.3 `Instruct orthodontist`, is a use case class that I will invoke to instantiate a set of scenarios (Larman, 2002, p. 47). In effect, as Cockburn (1997 , p. 7) tells us, a scenario can be a use case, vice versa. So, I would say, this means one use case will contain many scenarios as a 'shared aggregation' each of which 'inherits' the characteristics of a use case like this:

Figure 3.1 Recursive scenario – use case



That said, I can think of further scenarios which may be imperfect use case 'objects', i.e. 'variations on a theme' of Instruct orthodontist listed below (Bittner & Spence, 2003 , p. 32).

Figure 3.2 Plausible scenarios for use case 1.3 'Instruct orthodontist'

Scenario	Root_administrator	Use Case
1	Accepts an orthodontist referral case and sends a commissioning email to an external contracted orthodontist.	Instruct orthodontist
2	Accepts an orthodontist referral case but reassigns the case back to the dentist for a specific reason.	Reject patient referral
3	Accepts an orthodontist referral case but then closes the case.	Close patient referral
4	Attempts to send a commissioning email but cannot as key field entries are missing.	Incorrect entries
5	Attempts to send a commissioning email but cannot as they have not accepted the case.	Stop commissioning email
6	Accepts an orthodontist referral case but assigns it to another administrator	Reassign patient referral
7	Attempts to send a commissioning email but the orthodontist's contact details has not been added to the system.	Unavailable orthodontist
8	Cannot accept a referral case under view as it is taken by another administrator.	Refresh server connection
9	Attempts to accept a referral case but is found not to have sufficient privileges.	Decline user

3.2 Main success scenario

The sets above comprise nine exceptions and one instance that commits the goal of the use case Instruct orthodontist such that they are primary and secondary scenarios respectfully (Jalloul, 2004, pp. 19-22). Therefore the success of this use case depends on scenario #1: 'An Administrator attempts to accept a referral case but is found not to have referral management privileges.'

3.2.1 Flow of events

The success scenario can be described in terms of its event flows. By describing the flow of events, the interaction between actor Root_administrator, and system Server can be realised against the boundary to learn what 'responsibilities' belong exactly to whom. (Dathan & Ramnath, 2015, p. 42).

Figure 3.3 Main success scenario flow of events

Step	Root_administrator	Step	Server
1.	Logs into the system with their password.	2.	Validates password
3.	Enters the orthodontist referral que hub.	4.	Returns a table of referred patient cases in real time
5.	Views list of patient referral cases.		
6.	Opens a new patient referral case.	7.	Presents option to accept or view case
8.	Accepts the new patient referral case.	9.	Presents case details
10.	Opens a commissioning email.	11.	Opens up a template commissioning email
12.	Chooses an Orthodontist and response date.		
13.	Previews commissioning email.	14.	Shows the composed email
15.	Sends the commissioning email		
		16.	Attaches email to the case

3.2.2 Class, Responsibility, Collaboration (CRC)

A selection of classes can be discerned from the main success scenario and represented as CRC cards (Beck & Cunningham, 2012):

Figure 3.4 CRC's

Treatment_Hub «concrete»	
Responsibilities	Collaborators
1. Gets special treatment cases for administrators to manage 2. Acts as an association class where cases can be parked for collection	1. Root_administrator 2. General_administrator – likely Dental_Assistant

spT_Case «concrete»	
Responsibilities	Collaborators
1. To allow administrators to manage a special treatment case by invoking a range of methods 2. To atomically save outgoing and incoming emails	1. User – mainly all types of administrators 2. Patient 3. External email senders

Orthodontist «controller»	
Responsibilities	Collaborators
1. To proffer a collection of details about orthodontist from a query call from a User	1. User 2. Root_administors may interact exclusively to update details

Other use cases can be understood to be associated with the success scenario and added to the event table beneath:

Figure 3.5 Extended Root-administrator event table

ID	Event	Source	Action	Object	«Include»	Destination
1.8.	Administrator receives a request for access	Root_ administrator	Grant privileges	General_ administrator		Person
1.9	Administrator decides to give a case to an administrator	Administrator	Assign case	Patient case		Administrator
1.10	Administrator disowns a case	Administrator	Release case to surgery hub	Patient case		
1.11	Patient quits the surgery	Root_ administrator	Close case	Patient case		Patient
1.12	Orthodontist does not reply to commission by response date	Administrator	Follow up commissioning email	Mail merge		Orthodontist
1.13	Patient re-registers with surgery	Administrator	Reopen case	Patient case	Release case to surgery hub	Administrator
1.14	Orthodontist's email address is invalid	Root_ administrator	Update orthodontist	Orthodontist		
1.15	Administrator reads a case	Administrator	View patient case	Patient case		Administrator

Next, these event ID's will be added to a full use description as related use cases.

3.3 Use case description

A use case is a description may be written in varying formats and formalities. The approach I am asked to take in this question follows Lockhart & Chakaveh (2016, pp. 51-58). It is purposely detailed, black-boxed and 'fully dressed' in a table i.e. regarding what not how. In doing so, responsibilities of the use case are defined viz. the functional requirements of the O-O system (Larman, 2002, p. 49).

Figure 3.6 Detailed use case description

Use case ID 1.3	Commission orthodontist
Goal	Commission an external orthodontist to treat a patient on referral using the Orchard Dental Surgery online Medical Practice Management system.
Description	When a Dentist assigns a patient case to the referral queue hub, a Root_administrator accepts the case to manage and sends a commissioning email to an orthodontist.
Actors	Root_administrator - superuser with functions: • Create a new patient case. • Close/Restart) a patient case. • Update a patient case contact details. • Read a patient case medical record. • Manage a patient case. • Use mail merge email • Reassign a patient case.
Constraints	• Recorded information about a Patient in their case must be saved • Patient records must be accessible and kept securely for a maximum 30 years on retention • Mail merge commissioning emails must be automatically attached to the patient case in less than or equal to one minute. • Screen dialog must always show the Patient unique identification number. • Cases and orthodontist data cannot be deleted
Pre-conditions	• Root_administrator. must be a registered superuser that has privileges to manage the commissioning process of a referral case. • Root_administrator must be logged into the system. • Root_administrator must accept a Patient referral case (Own referral case). • A Dentist must write their instructions in the Patient referral case. • A Dentist must submit a request for a patient referral to an orthodontist to the referral queue hub. • The instructions from the Dentist must transfer automatically into the commissioning email. • A commissioning email, unless new, has not already been sent to an orthodontist • The orthodontist email address must be valid.
Main success scenario	• Root_administrator accepts an orthodontist referral case from a Dentist and sends a commissioning email to an orthodontist with instructions from a Dentist. (1) Logs into the system with their password. (2) Enters the orthodontist referral que hub. (3) Views list of patient referral cases. (4) Opens a new patient referral case. (5) Accepts the new patient referral case. (6) Opens a commissioning email. (7) Chooses an Orthodontist and response date. (8) Previews the commissioning email. (9) Sends the commissioning email.
Post-conditions for main success scenario	• The commissioning email must attach to the Patient case. • The Server must update the Patient case referral status from 'new' to 'open'. • An audit log is recorded of the Root_administrator. • Root_administrator releases ownership the Patient case back into the referral queue hub.
Other scenarios(named in brackets)	• Accepts an orthodontist referral case but rejects the case back to the dentist for a specific reason (Reject patient referral). • Accepts an orthodontist referral case but then closes the case (Close

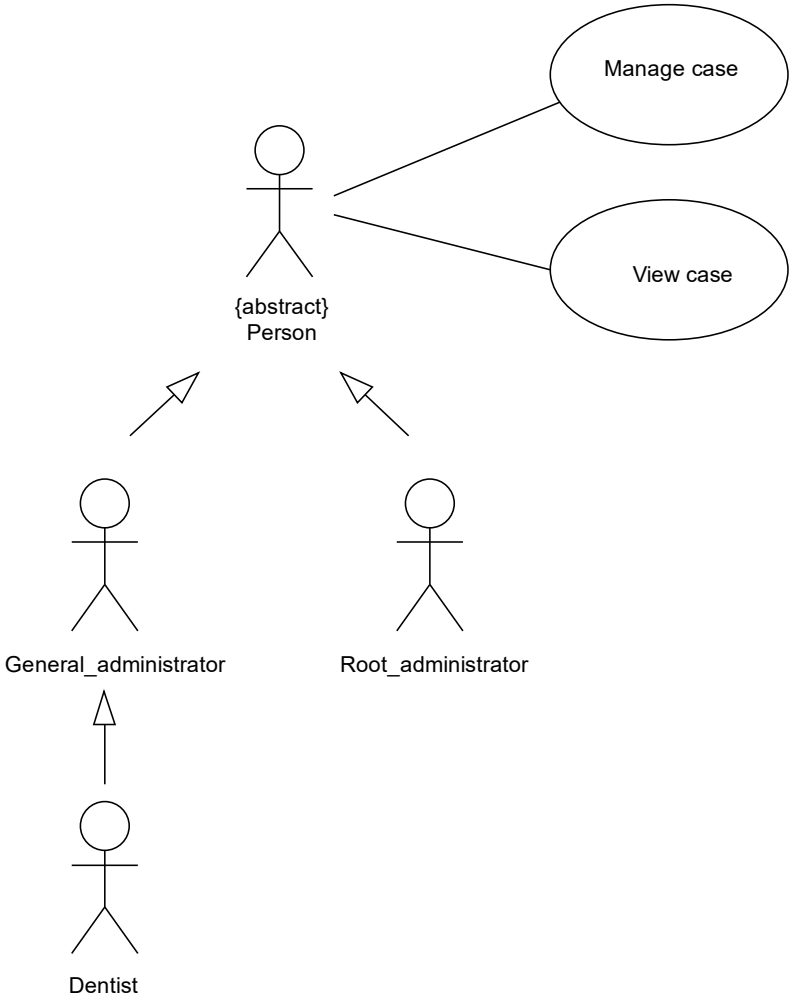
	patient referral). • Attempts to send a commissioning email but cannot as key field date or orthodontist is not selected or mis-entered (Incorrect entries). • Attempts to send a commissioning email but cannot as they have not accepted the case. (Stop commissioning email). • Accepts an orthodontist referral case but assigns it to another Administrator (Reassign patient referral). • Attempts to send a commissioning email but the instructed orthodontist contact details have not been added to the system (Unavailable orthodontist). • Cannot accept a referral case under view after it is accepted by another administrator (Refresh server connection). • Attempts to accept a referral case but is found to have insufficient privileges (Decline user).
Related use cases	Involving Root_administrator • ID 1.8 - Grant privileges • ID 1.9 - Assign case • ID 1.10 - Release case to surgery hub • ID 1.11 - Close case • ID 1.12 – Follow up commissioning email • ID 1.13 - Reopen case • ID 1.14 - Update orthodontist • ID 1.15 – View patient case
Frequency of occurrence	Special treatment to see an orthodontists are low relative to general dental treatment but very continuous and in demand
Test generation	• All scenarios constituting this use case should be subject to alpha testing. • Most crucial aspect is to test the commissioning email is sent, received, and attached to the case in time.
Notes	• Administrators should undergo training for this use case. • It is possible for exactly one Root_administrator to perform this use case. • System errors should be reported to the maintenance team. • Alternative orthodontist may need to be used where an email bounces back • Emails will be sent from a mailbox with an alias email address • Responses will be received in the mailbox

This use case aims to be self-explanatory but I am forced to augment a few areas needing some attention.

3.3.1 Actors

The use case description uncovers that there is a class hierarchy discriminator given that an administrator can be declined if one possess insufficient user privileges to commission an orthodontist. This means that there are two 'kinds of' administrator which I will different as 'root' and 'general'. The main difference is that a Root_administrator can issue commissioning emails to an orthodontist whereas a General_administrator needs to have its usability 'extended' to do so. It may well be that a Dentist could be a General_administrator:

Figure 3.7 Person inheritance



3.3.2 Constraints

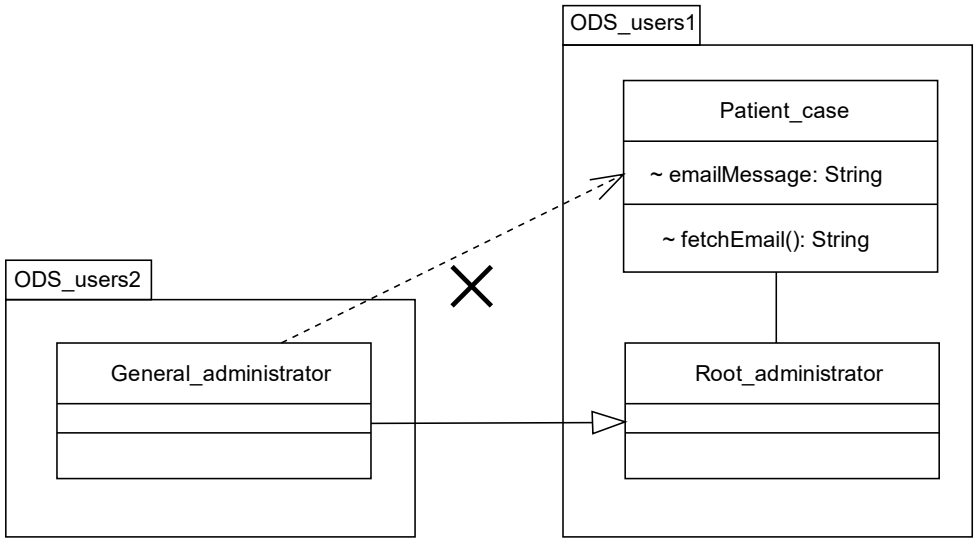
Constraints in the use case are derived from non-functional requirements vis-à-vis common dental operations. In particular, recorded information about a Patient must be saved. This rule is elicited from Standards for the Dental Team 4.1 (General Dental Council, 2014). The second constraint, that Patient records must be accessible and kept securely for a maximum 30 years on retention, is partly legislative. The legal parts of this constraint would be retrievable storage of commissioning emails saved to the server via, say, a JavaMail API. The UML standard does

allows for Object Constraint Language to supplement models (Object Management Group, 2006). Thus ‘a commissioning email can be stored 30 years minimum can be added as a note that:

```
Context Email inv: self.years->size()>=30
```

A Patient class could have a modifier that allows visibility for classes only contained within its package to call the fetchEmail method which reduces the extendibility needed for General_administrator (Hamilton & Miles , 2006, pp. 94-98).

Figure 3.8 Package diagram showing a restricted message



However, time is not statutory since Schedule 1 Data Protection Act (DPA (1998)) gives no definite scale, leaving the NHS to provide the said years (Information Governance Alliance , 2016). On the other hand, security of information and access is from the DPA, which constrains a nested email to the binary ‘state’ of being Saved to the server (Seidl, et al., 2012, pp. 96-97).

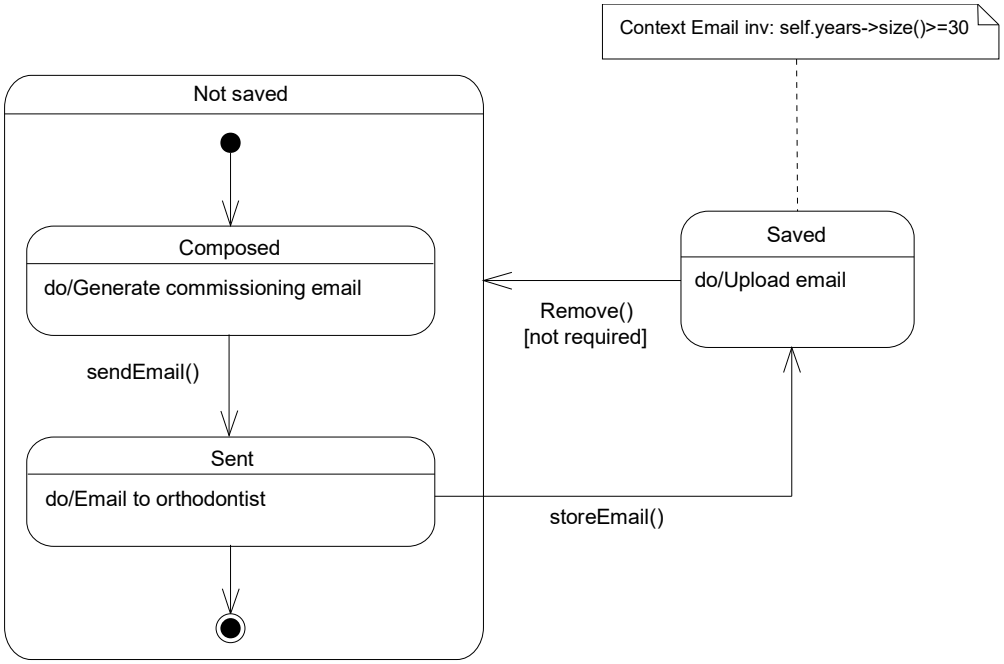
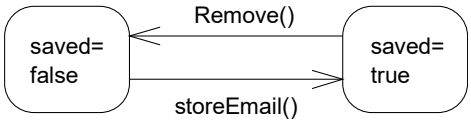


Figure 3.9 Composite state diagram storing an email




```
class Email {
 private boolean saved;
 private void storeEmail() {
 saved=true;
 }
 Private void Remove() {
 saved=false;
 }
}
```

Figure 3.10 State pseudocode for candidate email class

QUESTION 4

4.1 Activity and Sequence diagrams and how to distinguish them

4.1.1 Activity Diagram

As you will see in 4.2, when we use a directed graph with nodes that denotes components and control elements of use case, the execution semantics suffices to represent the stepwise actions of its activity diagrammatically. This feature is inherent in an Activity Diagram, but the means of controlling flow is what adds to the uniqueness of its UML in my view. This is owing to the ability to specify actions based on Petri token principles (Reisig, 2012, p. 3). In other words, actions can be conceptualised as activated events *e* in places *p* whenever it receives a passing dot (Seidl, et al., 2012, p. 146).

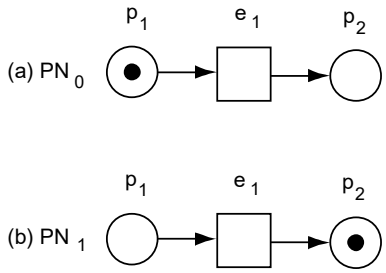


Figure 4.1 Petri Net (PN) (a) before event (b) after event

4.1.2 Sequence Diagram

Inter-object behaviour is the essence of a Sequence Diagram. If we wanted to specify how partnering objects interact in a use case, this exchange can be plotted as a distribution of sequential messages over time. Accordingly, objects are bound by state transitions with a lifeline connected by message calls that fulfils an executive task. On the flip side, static semantics parallel class organisation used to identify 'missing objects' (Bruegge & Dutoit, 2010, p. 185) . So, sequence diagrams are different, for we can cross-reference against the UML's taxonomy in respect to the structure and behaviour of an objects life. Syntactically, its construct is hierarchical, conforming to an ordered branch of roles *r,* classes, *c* and methods *m* (Li, et al., 2004, pp. 5-8).

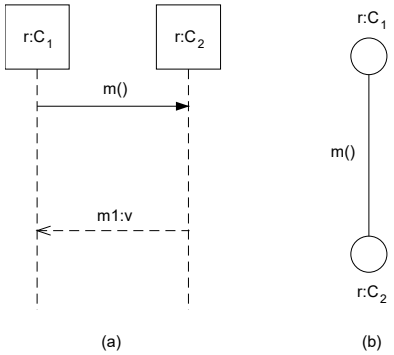


Figure 4.2 (a) Sequence (b) hierarchy

4.2 Activity diagram - success scenario

The textual flow of events, as ascertained in use case 3.4, Instruct Orthodontist allows us to decipher the action nodes of the main success scenario as the activity to be diagrammed.

Figure 4.3 Action construct from use case 1.3

Step	Success scenario	Action	
1	Logs into the system with their password.	Input login	
2	Enters the orthodontist referral que hub.	Select Treatment_Hub	
3	Views list of patient referral cases.		
4	Opens a new patient referral case.		
5	Accepts the new patient referral case	Manage new patient referral case	
6	Opens a commissioning email.	Create commissioning email	
7	Chooses an Orthodontist and response date.	Select orthodontist	Search orthodontist against name
			Select orthodontist
			Enter reply date
8	Previews commissioning email.	Preview commissioning email	
9	Sends the commissioning email	Send orthodontist email	

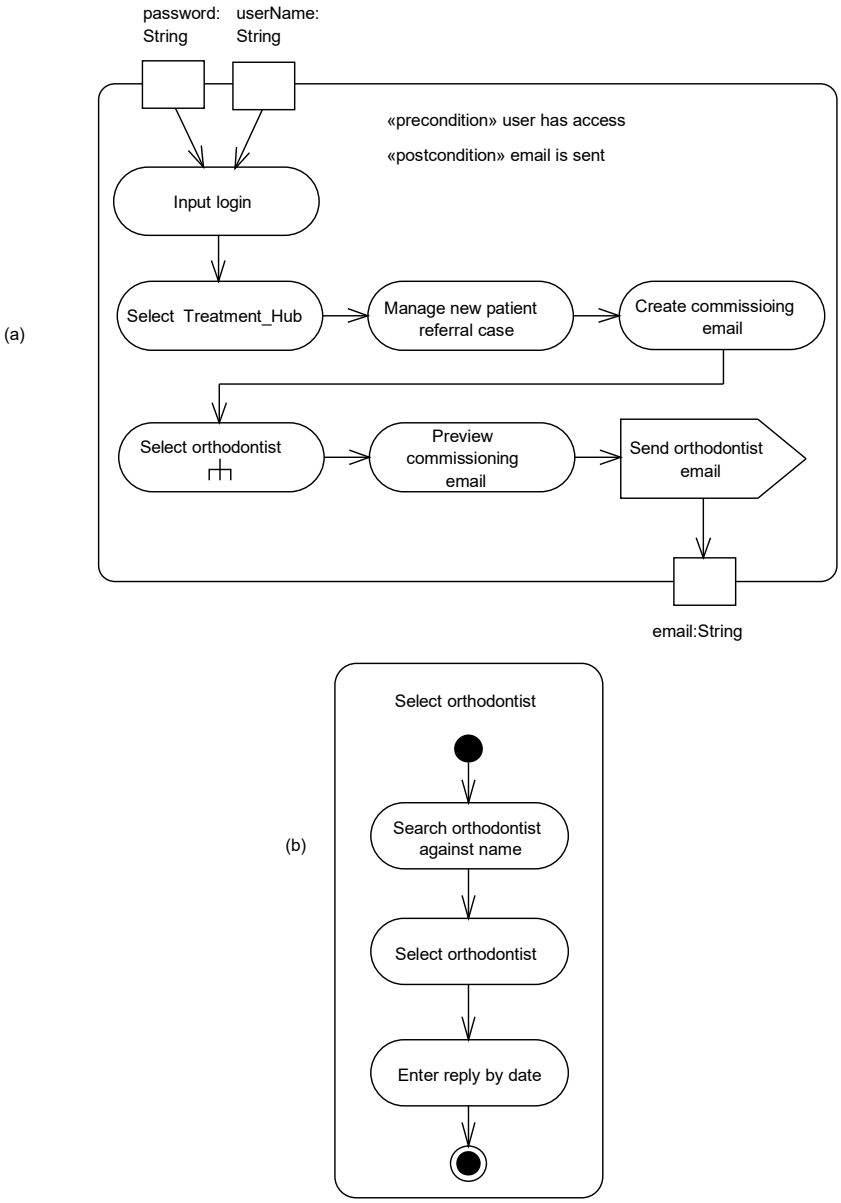
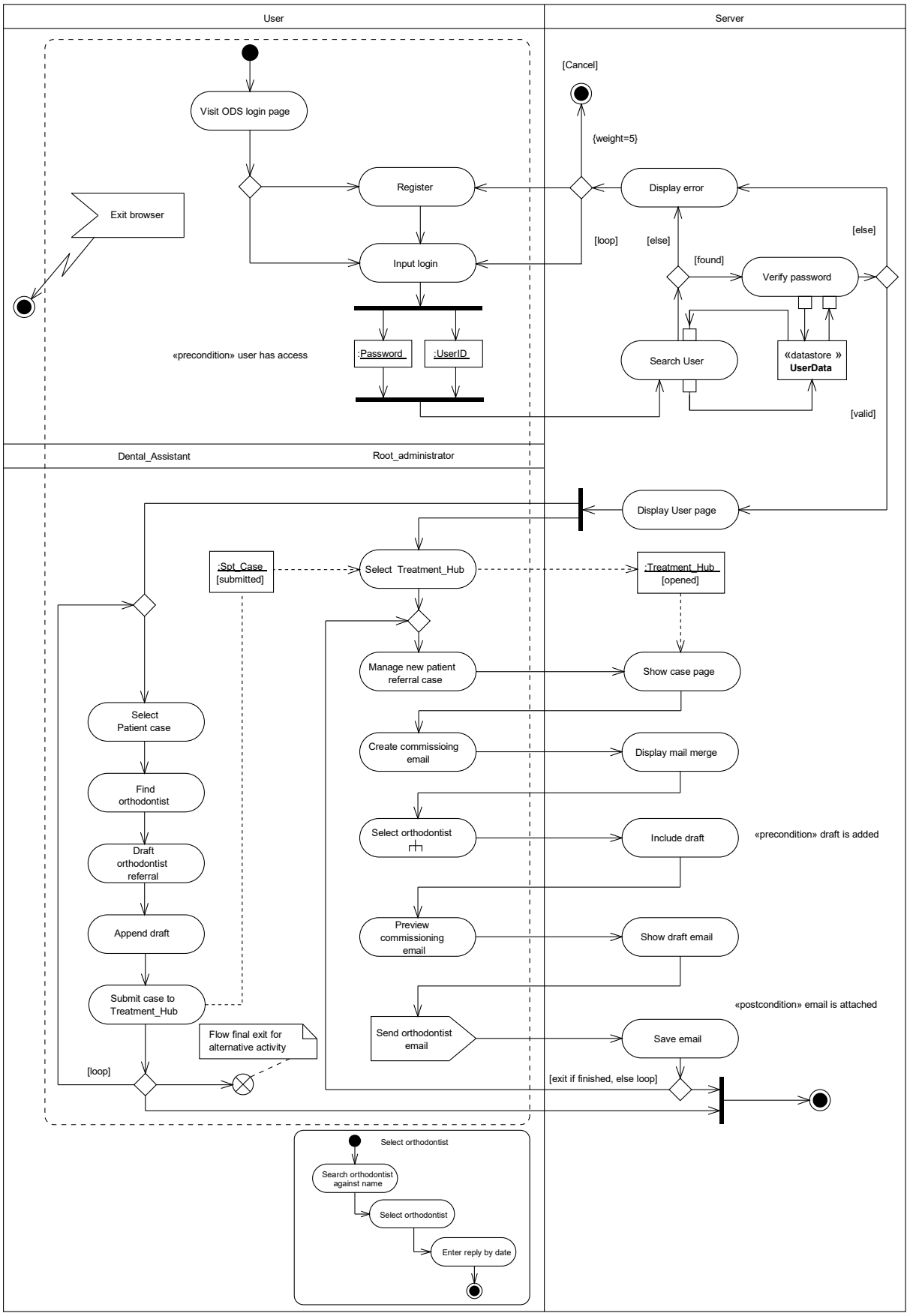


Figure 4.4 Root_Administrator (a) Activity diagram (b) call behaviour

In the above diagram, I have also added conditions pulled from the use case. But there are further dimensions to these terms as to the sort of actors involved. For instance, while use regards a Root_administrator, a Dental_Assistant is also involved.

Figure 4.5 Activity diagram – Instruct Orthodontist



4.3 Sequence diagram - use case

The modules constituting the ODS class diagram can be checked to find what objects will be partners in connection to a sequence of messages. Plus, as Fowler alludes, CRC cards can be drawn on to illicit what objects interact among themselves in order to transpose their behaviours into a sequence diagram (Fowler, 2004, pp. 62-63).

Figure 4.6 CRC cards

Mailmerge «concrete»	
Responsibilities	Collaborators
1. Allows administrators to compose generic emails 2. To add a pre-drafted letter to an orthodontist 3. To send emails to multiple subcontractors	1. Root_administrator

UserData «controller»	
Responsibilities	Collaborators
1. To process calls for access to the ODS system 2. To find user data and user objects to which cases may be assigned	1. User 2. ODS_System

ODS_System «interface»	
Responsibilities	Collaborators
1. To act as a boundary between human and the ODS system 2. To provide a GUI to a user	1. User

Server «actor»	
Responsibilities	Collaborators
1. To exchange emails via IMAP/POP	1. Root_administrator

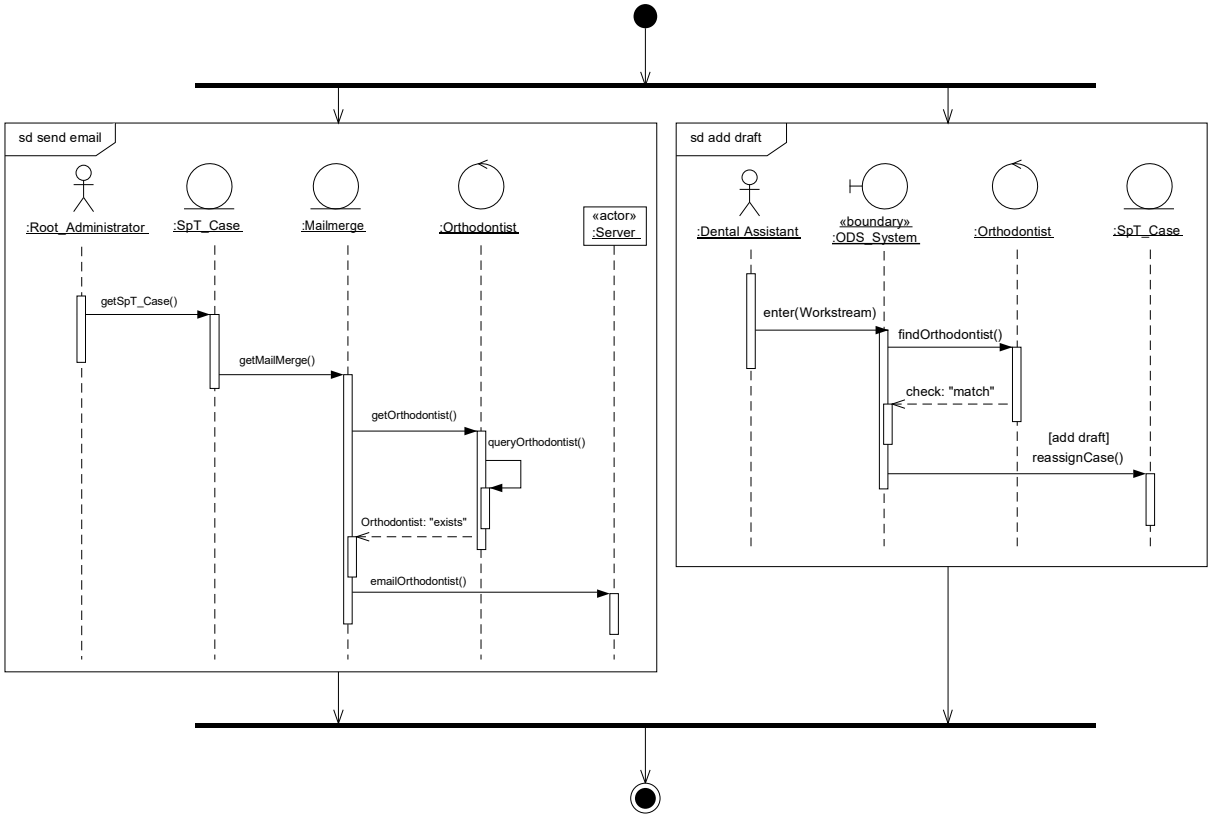


Figure 4.7 Interaction diagram

Since the y-axis can be utilised to show a time-consuming message. In the next sequence diagram to follow, you will see this apropos of the email response from an external orthodontist. This diagram also shows combined fragments to delineate operands for alternative scenarios, and loop, break, optional, and critical operators.

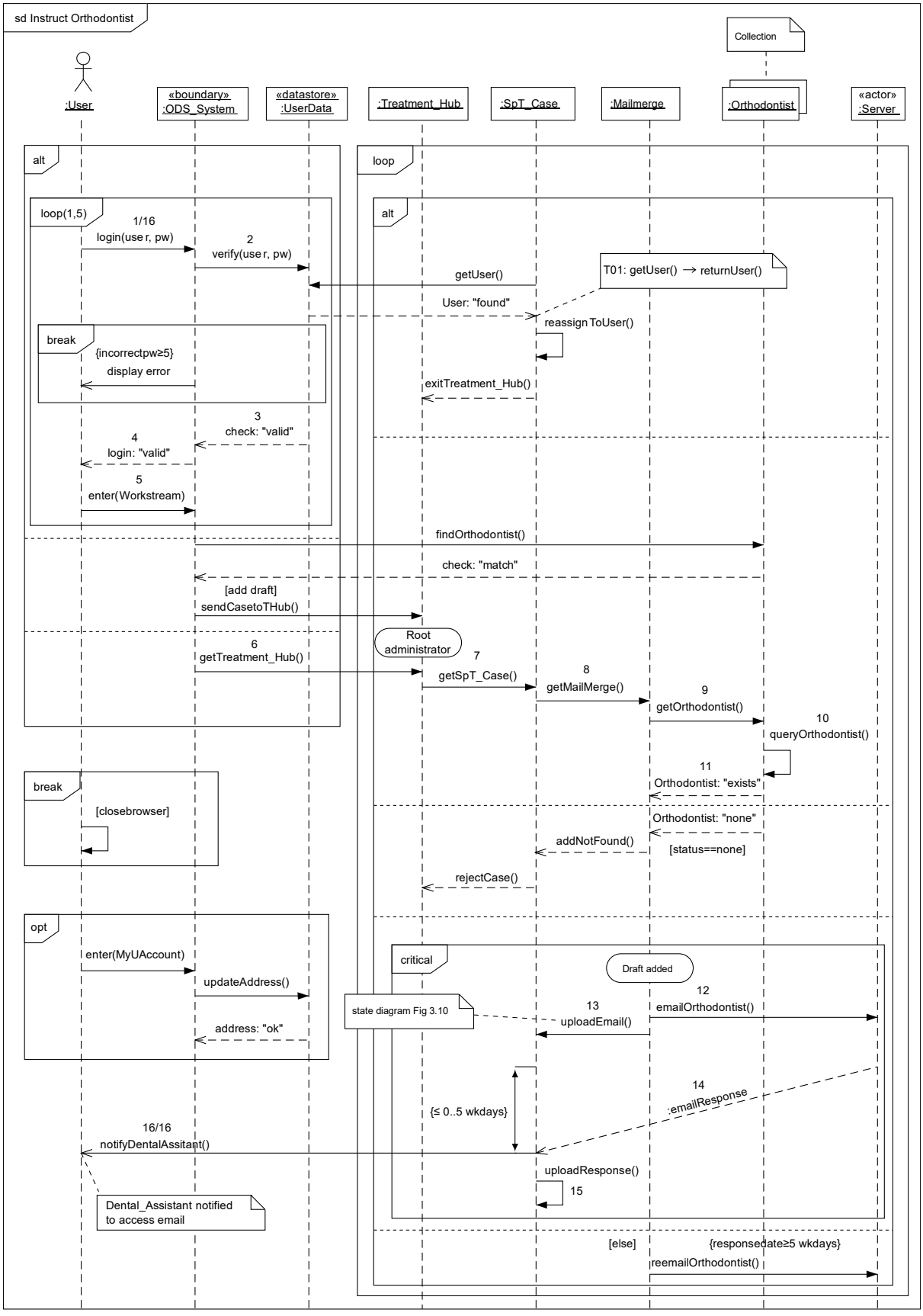


Figure 4.8 Sequence diagram – Instruct Orthodontist

QUESTION 5

5.1 Well-formed XML document

5.1.1 Forming the XML

When an XML document complies with the rules of its syntactical structure it is said to be ‘well-formed’, according to the WS3 (Bray , et al., 2006). The XML document I form comes with a specification to implement. I create database schema in SQL as it appears in the Access tables, Porterhouse_Library_Database.accdb, (Appendix), extract fourteen relevant columns that cover these requirements as a single book table, and insert three tuples of random data to be queried:

Figure 5.1 SQL schema and inserts

```
#
# TABLE STRUCTURE FOR: Book
#

DROP TABLE IF EXISTS `Book`;

CREATE TABLE `Book` (
  `ISBN` varchar(17) COLLATE utf8_unicode_ci NOT NULL,
  `Title` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `AvgLoan_4y` int(11) NOT NULL,
  `AvgRequest_4y` int(11) NOT NULL,
  `AuthorFName` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `AuthorSName` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `Publisher_Name` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `Publishing_Date` date NOT NULL,
  `Edition` int(11) NOT NULL,
  `Volumes` int(11) NOT NULL,
  `PAddressline1` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `PAddressline2` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `PAddresscode` varchar(10) COLLATE utf8_unicode_ci NOT NULL,
  `ContactTelephone` varchar(12) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`ISBN`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO `Book` (`ISBN`, `Title`, `AvgLoan_4y`, `AvgRequest_4y`, `AuthorFName`,
`AuthorSName`, `Publisher_Name`, `Publishing_Date`, `Edition`, `Volumes`,
`PAddressline1`, `PAddressline2`, `PAddresscode`, `ContactTelephone`) VALUES
('978-2-8691-4951-9', 'Deleniti', 20, 12, 'Reyna', 'Bailey', 'Larson-Schultz',
'1985-08-08', 3, 9, '29348 Walsh View Suite 364', 'Koelpinbury', '54944-8088',
'09999508834');
INSERT INTO `Book` (`ISBN`, `Title`, `AvgLoan_4y`, `AvgRequest_4y`, `AuthorFName`,
`AuthorSName`, `Publisher_Name`, `Publishing_Date`, `Edition`, `Volumes`,
`PAddressline1`, `PAddressline2`, `PAddresscode`, `ContactTelephone`) VALUES
('978-2-5076-0558-2', 'Est', 18, 10, 'Ramona', 'Williamson', 'Jast-Simonis',
'1984-04-03', 2, 4, '092 Keebler Freeway', 'Alanischester', '32519-2138', '649-
867-3233');
INSERT INTO `Book` (`ISBN`, `Title`, `AvgLoan_4y`, `AvgRequest_4y`, `AuthorFName`,
`AuthorSName`, `Publisher_Name`, `Publishing_Date`, `Edition`, `Volumes`,
```

```
`PAddressline1`, `PAddressline2`, `PAddresscode`, `ContactTelephone`) VALUES
('978-1-5357-4263-4', 'Velit', 6, 2, 'Pablo', 'Leuschke', 'Sipes and Sons', '1979-
10-02', 4, 6, '899 Daugherty Brooks', 'Stellastad', '27145', '(604)992-042');
```

I feed this data to an online XML converter where I can rename the output root and child elements of the document to `PorterhouseLibrary` and `Book` respectively. The foregoing code shows a well-formed XML document with comments against the five conforming areas: A root element, closing tags, elements properly nested, case matched, and values quoted (Goldberg , 2009, p. 5).

Well-formed XML document – Porterhouse Library

Well-formed at: https://codebeautify.org/xmlviewer/cb169037

```
<?xml version="1.0" encoding="UTF-8"?> <!-- 5 value "" quoted -->
<PorterhouseLibrary>                                    <!-- 1 Root element-->
  <Book>                                                <!-- 3 nested-->
    <ISBN>978-2-8691-4951-9</ISBN>
    <Title>Deleniti</Title>
    <AvgLoan_4y>20</AvgLoan_4y>
    <AvgRequest_4y>12</AvgRequest_4y>
    <AuthorFName>Reyna</AuthorFName>
    <AuthorSName>Bailey</AuthorSName>
    <Publisher_Name>Larson-Schultz</Publisher_Name>
    <Publishing_Date>1985-08-08</Publishing_Date>
    <Edition>3</Edition>
    <Volumes>9</Volumes>
    <PAddressline1>29348 Walsh View Suite 364</PAddressline1>
    <PAddressline2>Koelpinbury</PAddressline2>
    <PAddresscode>54944-8088</PAddresscode>
    <ContactTelephone>09999508834</ContactTelephone>
  </Book>                                                       <!-- 2 closing tag-->
  <Book>
    <ISBN>978-2-5076-0558-2</ISBN>
    <Title>Est</Title>
    <AvgLoan_4y>18</AvgLoan_4y>
    <AvgRequest_4y>10</AvgRequest_4y>
    <AuthorFName>Ramona</AuthorFName>
    <AuthorSName>Williamson</AuthorSName>
    <Publisher_Name>Jast-Simonis</Publisher_Name>
    <Publishing_Date>1984-04-03</Publishing_Date>
    <Edition>2</Edition>
    <Volumes>4</Volumes>
    <PAddressline1>092 Keebler Freeway</PAddressline1>
    <PAddressline2>Alanischester</PAddressline2>
    <PAddresscode>32519-2138</PAddresscode>
    <ContactTelephone>649-867-3233</ContactTelephone>
  </Book>                       <!-- 4 Case sensitive, always B for Book -->
  <Book>
    <ISBN>978-1-5357-4263-4</ISBN>
    <Title>Velit</Title>
    <AvgLoan_4y>6</AvgLoan_4y>
    <AvgRequest_4y>2</AvgRequest_4y>
    <AuthorFName>Pablo</AuthorFName>
    <AuthorSName>Leuschke</AuthorSName>
    <Publisher_Name>Sipes and Sons</Publisher_Name>
    <Publishing_Date>1979-10-02</Publishing_Date>
    <Edition>4</Edition>
    <Volumes>6</Volumes>
    <PAddressline1>899 Daugherty Brooks</PAddressline1>
    <PAddressline2>Stellastad</PAddressline2>
    <PAddresscode>27145</PAddresscode>
    <ContactTelephone>(604)992-042</ContactTelephone>
  </Book>
</PorterhouseLibrary>
```

5.2 Validating an XML document

5.2.1 Format XML schema

Validating an XML document is crucial for Porterhouse Library's database being defined according to XML schema, wherein constraints on information storage and data types apply to avoid anomalous runtime (Ray , 2003, p. 109). To make my XML schema valid, the crux of it should have elements defined as either a simple type (text only) or complex type (attributes and/or child elements) per Goldberg (2009, pp. 113-114).

```
<!-this is simple type-->
<element name="ISBN" type="string”/>


<!-and this is complex type-->
<element name="Bookdata">
   <complexType>
     <sequence>
...
     </sequence>
  </complexType>
</element>
</html>
```

Figure 5.2

At the top of the document, we declare `<?xml version="1.0"?>` and include `<xs:schema` which gives definition to the Porterhouse Library root element**.** You may have seen that there is no `xs:` prefix in Figure 5.2, so we follow through with `xmlns:xs="http://www.3.org/2001/XMLSchema` which announces the document namespace, datatypes, and elements. Root element tags should be closed before the rules of the schema can be added, which may be ended with `</xs:schema>`. Once the XML Schema is complete, we should, associate this XML schema with the Porterhouse Library XML document from the definition of its root element like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<PorterhouseLibrary xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
"PLdata.xsd">
```

Figure 5.3 Referencing XML Schema in the XML document

We can continue with this approach or convert the XML document to yield an XSD document which can be parsed by a validator as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Bookdata">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="Book">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ISBN" type="xs:string" />
              <xs:element name="Title" type="xs:string" />
              <xs:element name="AvgLoan_4y" type="xs:unsignedByte" />
              <xs:element name="AvgRequest_4y" type="xs:unsignedByte" />
              <xs:element name="AuthorFName" type="xs:string" />
              <xs:element name="AuthorSName" type="xs:string" />
              <xs:element name="Publisher_Name" type="xs:string" />
              <xs:element name="Publishing_Date" type="xs:date" />
              <xs:element name="Edition" type="xs:unsignedByte" />
              <xs:element name="Volumes" type="xs:unsignedByte" />
              <xs:element name="PAddressline1" type="xs:string" />
              <xs:element name="PAddressline2" type="xs:string" />
              <xs:element name="PAddresscode" type="xs:string" />
              <xs:element name="ContactTelephone" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 5.4 Valid XSD document – Porterhouse Library

Validated at: https://codebeautify.org/xmlvalidator/cbad2ff7

5.3 Steps to display XML on W3

"...XML does not do anything."

(Lee & Foo, 2008)

5.3.1 Parse

Much like a function, a transformer needs an input, and for XML to be displayed, two documents need to be parsed. The first is an XSLT document - an XML application, forming part of the Extensible Style Sheet Language (XSL) that is responsible for processing sources to render new source trees at run time (Kirjavainen, 2000). So we need XSLT to display the source of an XML document, our second input. This necessitates a template that declares control over element namespaces (prefix `xsl`). This offers output data instructions that relate the `PorterhouseLibrary` source tree with attributes. In so doing, the template matches class nodes from the source document, and specifies the result of the tree. We start the beginning of the XSLT like this:

Figure 5.5 Part 1/2 - XSLT style sheet initiation

```
<?xml version="1.0"> <!--the XSLT style sheet is an XML document-->

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <!—specifies namespace and
prefix xsl-->

… <!—root template goes here-->

</xsl:stylesheet> <!—completes the style sheet-->
```

A comment is mentioned with reference to a 'root template' in Figure 5.5. This is a description of the process that outputs root node content. It is needed when an XSLT document is parsed, as the processer will first identify the rules applying to the root node of the Porterhouse XML document (Ray , 2003, p. 235). We start this sub-template with `<xsl:template match="/">` to tell the transformer to match the root node `PorterhouseLibrary` from the XML source document, and end with `</xsl:template>`. In between the two, rules and parameters are defined that applies to the content of the `PorterhouseLibrary` root and all

its children such as, tables, colour, and alignments. Below, I choose to fill the void with code such that the XSLT will display a table called `Book Information.` It aims to show the `Title` and `ISBN Number` of three book entry tuples using `<xsl:value-of>` element to output node content by declaring string values.

Figure 5.6 Part 2/2 - XSLT document with root template

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="html" />
   <xsl:template match="/">
      <html>
         <body>
            <h2>Book Information</h2>
            <table border="1">
               <tr bgcolor="#dcdcdc">
                  <th style="text-align:left">Title</th>
                  <th style="text-align:left">ISBN Number</th>
               </tr>
               <xsl:for-each select="PorterhouseLibrary/Book">
                  <tr>
                     <td>
                        <xsl:value-of select="Title" />
                     </td>
                     <td>
                        <xsl:value-of select="ISBN" />
                     </td>
                  </tr>
               </xsl:for-each>
            </table>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

5.3.2 Transform

The XLST cannot do what it has been asked without being processed together with the well-formed XML document in 5.1 to yield HTML. It is the reason why I inserted `<xsl:output method="html" />` into the header information in Figure 5.6, which sets the processors output method accordingly. The transformation process boils down the Porterhouse XML document into a new one, matching templates as it processes it source tree node. (Ray , 2003, p. 236).

5.3.4 Display

Something new is created whenever the XLST transform is run so one can always reorganise elements and build on the data we have – in my view, following familiar lines with Mayer's (1988) extension ideology noted in 1.3. This is unlike CSS, which is also limited by a strict ordering of text that follows the sequence of input data, but which may still be applied post output (Ladd & O'Donnell, 2001, p. 394). The rendered result of the Porterhouse XML document is known as serialization, and, as such transformation is not just for want of HTML, but for variable [1..*] data stream outputs (Amiano, et al., 2006, p. 80).

```
<html>
   <body>
      <h2>Book Information</h2>
      <table border="1">
         <tr bgcolor="#dcdcdc">
            <th style="text-align:left">Title</th>
            <th style="text-align:left">ISBN Number</th>
         </tr>
         <tr>
            <td>Deleniti</td>
            <td>978-2-8691-4951-9</td>
         </tr>
         <tr>
            <td>Est</td>
            <td>978-2-5076-0558-2</td>
         </tr>
         <tr>
            <td>Velit</td>
            <td>978-1-5357-4263-4</td>
         </tr>
      </table>
   </body>
</html>
```

Figure 5.7 Transformed XML – Porterhouse Library

Book Information

Title	ISBN Number
Deleniti	978-2-8691-4951-9
Est	978-2-5076-0558-2
Velit	978-1-5357-4263-4

Figure 5.8 Rendered XML in HTML

Validated at https://codebeautify.org/htmlviewer/cb74ee46

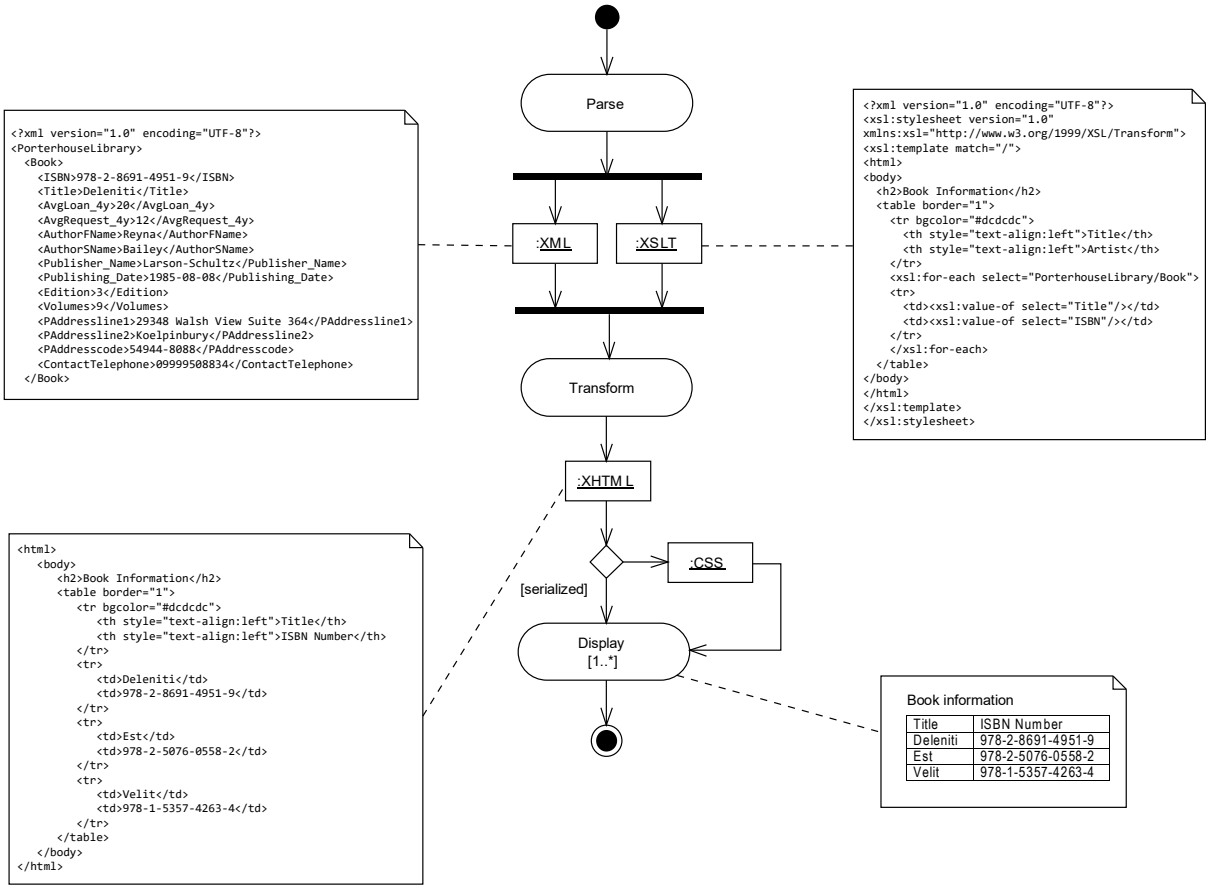


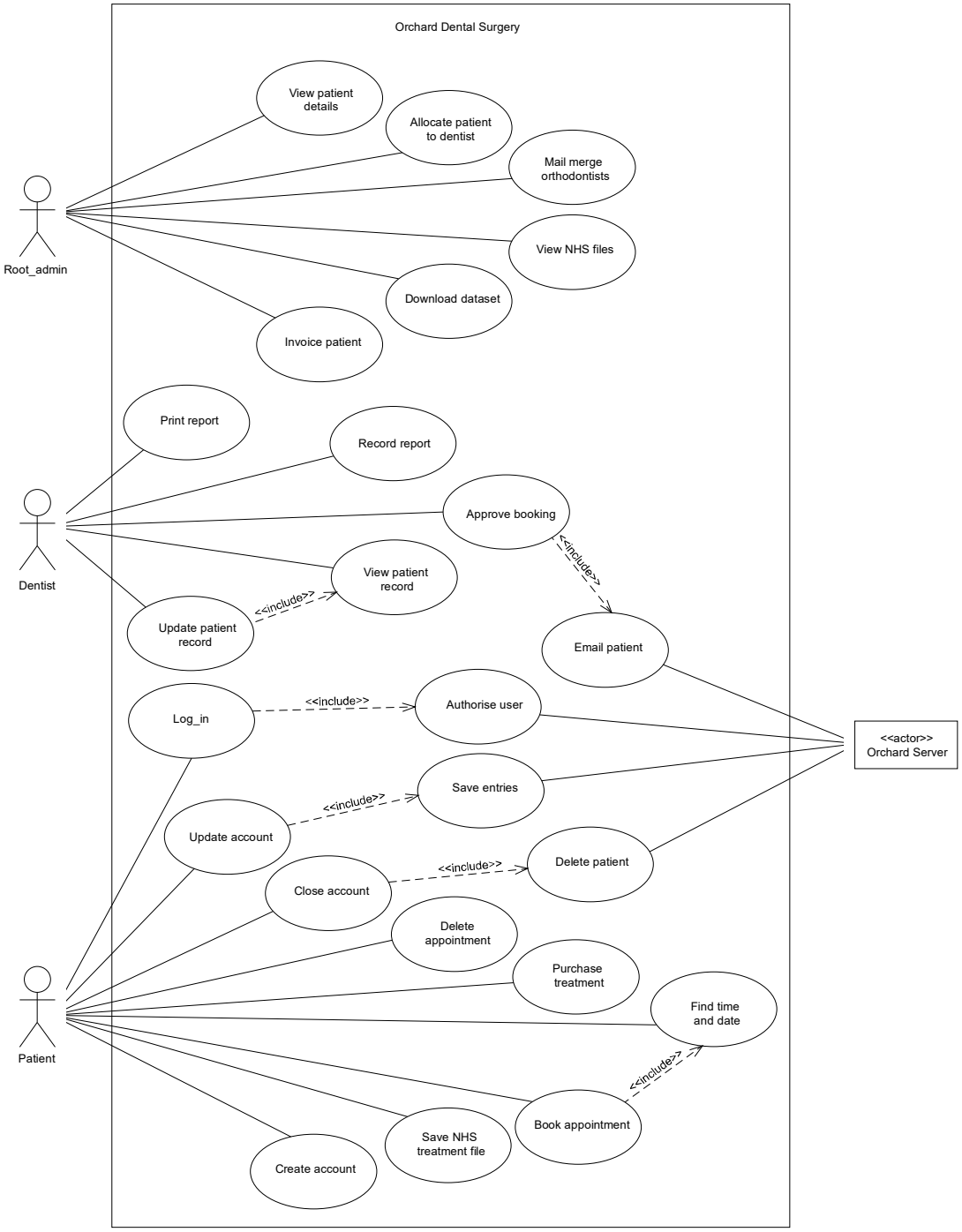
Figure 5.9 Activity diagram – XSLT Transform

APPENDIX

Orchard Dental Surgery –Statement

You are asked to produce some preliminary design documents for a system that is to be used by administrators to control the management of a dental surgery. The surgery provides its patience with, a number of treatments. The surgery accepts patients both privately & as NHS provision. For special treatments such as tooth implant an external Orthodontist is engaged on a case to case basis. In addition the Dental Surgery has recently published a web page that patients can book their appointments for general treatments such as check-up & visit to the hygienist also on-line. This will be an object oriented system and, since this is a preliminary design, should not be too complicated so try to give simple answers addressing the questions asked on the next pages. This will be an object oriented system and, since this is a preliminary design, should not be too complicated so try to give simple answers addressing the questions asked on the next pages.

Draft use case diagram with boundary and server notation



```
CREATE TABLE Author
(
Author_ID                 INTEGER            NOT NULL,
First_Name                VARCHAR(50)        NOT NULL,
Second_Name               VARCHAR(50)        NOT NULL,
PRIMARY KEY(Author_ID)
);

CREATE TABLE Publisher
(
Publisher_ID              INTEGER            NOT NULL,
Publisher_Name            VARCHAR(100)       NOT NULL,
Addressline1              VARCHAR(100)       NOT NULL,
Addressline2              VARCHAR(100)       NOT NULL,
Addresscode               VARCHAR(10)        NOT NULL,
ContactTelephone          VARCHAR(12)        NOT NULL,
PRIMARY KEY(Publisher_ID)
);

CREATE TABLE Book
(
ISBN               VARCHAR(17)        NOT NULL,
Title              VARCHAR(100)       NOT NULL,
Author_ID          INTEGER            NOT NULL,
Publisher_ID       INTEGER            NOT NULL,
Edition            INTEGER            NOT NULL,
PRIMARY KEY(ISBN),
FOREIGN KEY(Author_ID) REFERENCES Author(Author_ID),
FOREIGN KEY(Publisher_ID) REFERENCES Publisher(Publisher_ID)
);

CREATE TABLE Volume
(
Volume_ID          INTEGER            NOT NULL,
ISBN               VARCHAR(17)        NOT NULL,
Quality            VARCHAR(9)         NOT NULL, /* Quality was swapped
for  'Condition' in the access table which is an SQL reserved word  */

When_acquired      DATE               NOT NULL, -- no. times the volume taken
out
Notes              VARCHAR(100)       NULL,
CHECK(Quality IN ('Poor','Good','Excellent')),
PRIMARY KEY(Volume_ID),
FOREIGN KEY(ISBN) REFERENCES Book(ISBN)
);


CREATE TABLE Customer
(
Customer_ID               INTEGER            NOT NULL,
First_Name                VARCHAR(50)        NOT NULL,
Second_Name               VARCHAR(50)        NOT NULL,
ContactTelephone          VARCHAR(12)        NOT NULL,
AddressLine1              VARCHAR(50)        NOT NULL,
AddressLine2              VARCHAR(50)        NOT NULL,
Addresscode               VARCHAR(10)        NOT NULL,
AccountValue              INTEGER            NOT NULL,
```

```
EmailAddress              VARCHAR(50)        NOT NULL,
PRIMARY KEY(Customer_ID)
);

CREATE TABLE Loan
(
Loan_ID                   INTEGER            NOT NULL,
Volume_ID                 INTEGER            NOT NULL,
Date_issued               DATE               NOT NULL,
Customer_ID               INTEGER            NOT NULL,
Date_due                  DATE               NOT NULL,
Date_returned             DATE               NOT NULL,
PRIMARY KEY(Loan_ID),
FOREIGN KEY(Volume_ID) REFERENCES Volume(Volume_ID),
FOREIGN KEY(Customer_ID) REFERENCES Customer(Customer_ID)
);

CREATE TABLE Reservation
(
Reservation_ID            INTEGER            NOT NULL,
When_reserved             DATE               NOT NULL,
ISBN                      VARCHAR(17)        NOT NULL,
Customer_ID               INTEGER            NOT NULL,
PRIMARY KEY(Reservation_ID),
FOREIGN KEY(ISBN) REFERENCES Book(ISBN),
FOREIGN KEY(Customer_ID) REFERENCES Customer(Customer_ID)
);

CREATE TABLE
(
UserName                  VARCHAR(50)        NOT NULL,
Password                  VARCHAR(10)        NOT NULL,
Capabilites               VARCHAR(100)       NULL,
PRIMARY KEY(UserName)
);
```

REFERENCE LIST

Abbott, R. J., 1983. Program design by informal English descriptions. *Communication of the ACM,* 26(11), pp. 882-894.

Amiano, M., D'Cruz, C., Ethier, K. & Thomas, M. D., 2006. *XML: Problem - Design - Solution.* Firt ed. Indianna, US: John Wiley & Sons.

Arlow , J. & Neustadt, I., 2002. *UML and the Unified Process.* First ed. London : Pearson Education .

Bajwa, I. S., Ali, S. & Mumtaz, S., 2009. Object Oriented Software Modeling Using NLP Based Knowledge Extraction. *European Journal of Scientific Research,* 35(1), pp. 22-33.

Barbara , L., 1988. Keynote address - data abstraction and hierarchy. *ACM SIGPLAN Notices - Special issue: 'OOPSLA '87: Addendum to the proceeding,* 5(23), p. 17–34.

Beck, K. & Cunningham, W., 2012. A laboratory for teaching object oriented thinking. *ACM SIGPLAN Notices,* 24(10), p. 6.

Berardi, D., Calvanese, D. & De Giacomo, G., 2005. Reasoning on UML class diagrams. *Artificial Intelligence,* 168(1-2), pp. 70-118.

Bittner, K. & Spence, I., 2003 . *Use Case Modeling.* First ed. London: Addison-Wesley Professional.

Bray , T. et al., 2006. *Extensible Markup Language (XML) 1.1.* Second Edition ed. s.l.:World Wide Web Consortium (W3C).

Bruegge, B. & Dutoit, A. H., 2010. *Object-Oriented Software Engineering Using UML, Patterns, and Java™.* Third ed. s.l.:Pearson Education.

Cockburn, A., 1997 . *Structuring Use Cases with Goals,* s.l.: Journal of Object-Oriented Programming.

Cockburn, A., 2000. *Writing Effective Use Cases.* First ed. s.l.:Addison-Wesley Professional.

Dathan, B. & Ramnath, S., 2015. *Object-Oriented Analysis, Design and Implementation - An Integrated Approach.* Second ed. London: Springer .

Edward , C., 2006. *Fundamentals of Programming Using Java.* First ed. London: Thomson.

Fowler, M., 2004. *UML Distilled.* Third ed. New Jersey: Pearson Education.

General Dental Council, 2014. *Focus on standards.* [Online]
Available at: https://standards.gdc-uk.org/Assets/pdf/Standards%20for%20the%20Dental%20Team.pdf
[Accessed 12 04 2018].

Goldberg , K. H., 2009. *XML.* Second ed. CA: Peachpit Press.

Hamilton, K. & Miles , R., 2006. *Learning UML 2.0.* First ed. CA: O'Reilly Media, Inc..

Henz, M., 1998. *Objects for Concurrent Constraint Programming.* First ed. London : Kluwer Academic publishers .

Information Governance Alliance , 2016. *Records Management Code of Practice for Health and Social Care.* [Online]
Available at: https://digital.nhs.uk/records-management-code-of-practice-for-health-and-social-care-2016
[Accessed 13 04 2018].

Jalloul, G., 2004. *Uml by Example (SIGS: Advances in Object Technology).* First ed. Cambrdige: Cambridge University Press.

Jalote, P., 2008. *A Concise Introduction to Software Engineering.* 1 ed. London: Springer-Verlag .

Kirjavainen, M., 2000. *Browsers How XML Is Displayed in a Browser.,* s.l.: Conference Proceedings.

Ladd, E. & O'Donnell, J., 2001. *Platinum Edition Using XHTML, XML and Java 2.* First ed. s.l.:Que Publishing.

Larman, C., 2002. *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process.* Second ed. Upper Sadle River : Prentice Hall Professional.

Lee, W.-M. & Foo, S. M., 2008. *XML Programming Using the Microsoft XML Parser.* First ed. s.l.:Aress .

legislation.gov.uk, 1998. *Data Protection Act.* [Online]
[Accessed 13 04 2018].

Li, X., Liu, Z. & Jifeng, H., 2004. *A formal semantics of UML sequence diagram.* Macau, 2004 Australian Software Engineering Conference. Proceedings.

Lockhart, R. & Chakaveh, 2016. *Software Construction using Objects (Object-oriented analysis and design notations & architectures).* Sixth ed. Oxford : Department for Continuing Education, University of Oxford.

Meyer , B., 1988. *Object-oriented Software Construction.* First ed. Santa Barbara, CA: Prentice-Hall.

Muhairat, M. I. & Rafa E. , A.-Q., 2009. An Approach to Derive the Use Case Diagrams from an Event Table. *8th WSEAS Int Conference on Software Engineering, Parallel and Distributed Systems.*, 21 - 23 February , pp. 33-38.

National Centre for Text Mining, 2012. *Enju 2.4 online demo.* [Online]
Available at: http://www.nactem.ac.uk/enju/demo.html
[Accessed 22 04 2018].

NHS, 2016. *NHS Choices.* [Online]
Available at:
https://www.nhs.uk/NHSEngland/thenhs/records/healthrecords/Pages/overview.aspx
[Accessed 02 April 2018].

Object Management Group, 2006. *Object Constraint Language - Available Specification Version 2.0.* s.l.:Object Management Group.

Ray , E., 2003. *Learning XML.* Second ed. CA, USA: O'Reilly.

Reisig, W., 2012. *Petri Nets - An introduction.* First ed. Berlin : Springer-Verlag.

Satzinger, J. W., Jackson, R. B. & D. Burd, S., 2012. *Systems Analysis and Design.* First ed. Boston MA: Course Technology, Cengage Learning.

Seidl, M., Scholz, M., Huemer , C. & Kappel , G., 2012. *UML@Classroom - An Introduction to Object-Oriented Modeling.* First ed. Heidelberg, Germany : Springer International Publishing.