Assignment A

UML (Use Case and Class diagram)

16-7213 Object Oriented Methods

Prepared by Group 1: **Chan Lai San (Student ID: 12034569)**

Fida Al-Obaisi (Student ID: 17032975)

Rebah Daw Sarreb (Student ID: 16033719)

Written for: **Dr. Alan Goude** **Date: 3-4-2008**

Table of contents:

1.0 INTRODUCTION TO UML ..... - 3 -
1.1 INTRODUCTION TO UML DIAGRAMS ..... - 3 -
2.0 USE CASE DIAGRAM ..... - 6 -
2.1 RELATIONSHIPS ..... - 7 -
3.0 CLASS DIAGRAM ..... - 9 -
3.1 ATTRIBUTES ..... - 10 -
3.2 MULTIPLICITIES ..... - 11 -
3.3 OPERATIONS ..... - 11 -
4.0 RELATIONSHIP BETWEEN USE CASE AND CLASS DIAGRAM ..... - 13 -
5.0 CONCLUSION ..... - 14 -
REFERENCES: ..... - 16 -

1.0 Introduction to UML (282 words by Lai San):

UML or Unified Modelling Language is a multipurpose modelling language that aims to provide a standard for modelling a system. UML consists of various diagrams used to model a system from initial idea to an implement able project. Each model carries the specifications and requirement of that same system from different point of view. For example, the users of a particular system only want to know what the system can do while the designer will design what and how many functions or tasks in that system. But a programmer or engineer needs to know how to perform a particular function or how each task affect each other. All this can be realise using UML diagrams. UML provides standard methods and notation to create these models as well as guideline to transform one model to another model while preserving the consistency between models.

UML is a collaboration of several traditional modelling concepts and notation. It is first created by Grady Booch, James Rumbaugh and Ivar Jacobson during 1994. UML is a non proprietary modelling language but its ownership and evolution responsibility is governed by Object Management Group (OMG). The application of UML is very wide. It is not only used in object oriented systems analysis and design. It is also used widely in all phases of complex software development life cycles, development of many systems engineering, as well as in modelling of many business processes. UML is not dependent on any programming languages and strongly highlight the concept of reuse, layering, partitioning and modularity. In general, UML is design to be flexible, extendable and open to many specific applications or industries. UML provides guideline on how to extend a system using stereotypes method.

1.1 Introduction to UML Diagrams (570 words by Lai San)

According to the new OMG's information [1], there are a total of thirteen types of diagrams define in the latest UML 2.0, which is divided into three categories namely static structure, behaviour and interaction. Static structure models include the Class diagram, Object diagram, Component diagram, Composite Structure diagram, Package diagram, and Deployment diagram. Behaviour models include the Use Case diagram, Activity diagram, and State diagram. And lastly interaction models include the Sequence diagram, Communication diagram, Timing diagram, and Interaction Overview diagram. Each of the diagrams serves its own purpose and is strongly related to each other. The details of Use Case diagram and Class diagram will be discussed in later sections.

Structural or static modelling consists of diagrams that is used to shows elements or functions and its relationships which a system have. In other words, it is used to shows different view of 'what' the system have or do, as well as the relationships but not what happen from the interaction. Class diagram, being one of the static model, uses set of classes to group objects with common properties together and shows the relationships between each class. Package diagram is a simplify version of complex class diagram, where related classes can be group into individual package. A dotted line is used to indicate there is a dependency among packages. Object diagram shows instances or objects generated from particular classes. It is useful in explaining complex recursive relationships between classes in class diagram. Figure 1 shows some simple example of the diagrams.

Class diagram, Package diagram and Object diagram are usually used during analysis and design stage. On other hand, Component, Composite Structure and Deployment diagrams are used in implementation stage or when the system is complete. Component diagram gives a view of what components (or pieces of parts) and its relationships that is inside the completed system. While Deployment diagram shows how to assemble these components together to form the systems or where these components belong to. Lastly, the OMG UML superstructure V2.1.2 [12], define a Composite Structure diagram as to depict the internal structure of a classifier, as well as the use of collaboration in a collaboration use. Figure 2 shows simple example of Component, Deployment and Composite Structure diagrams.

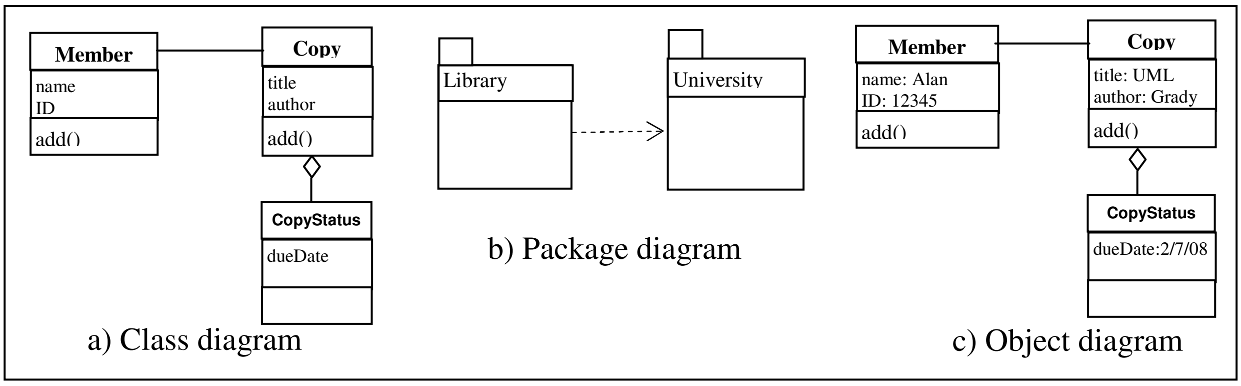


Figure 1: Example of Class, Package and Object diagrams

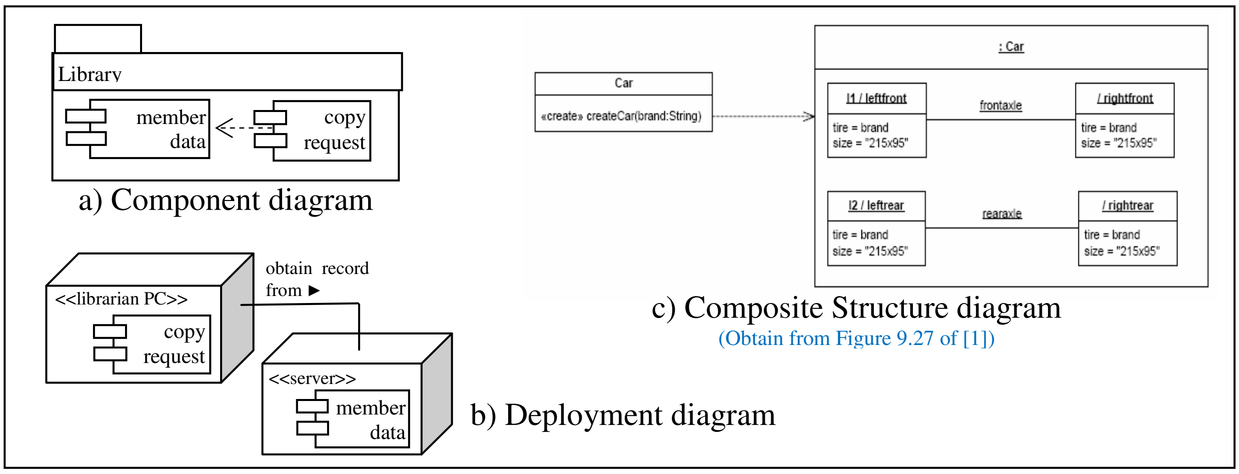


Figure 2: Example of Component, Deployment and Composite Structure diagrams

In addition, behaviour modelling show how each elements or functions in a system will behave or interact with each other. Use case diagram depict the functionality of a system with its users interaction. Statechart (also call State Machine) diagram shows the possible states of an object in a system and the transition that cause the change of state[4]. Activity diagram models how the control flows from one activity to another within a single process or function of a system.

Next, the interaction modelling category shows more details behaviour of things derived from the general behaviour models mentioned earlier. Sequence diagram shows step by step operations flow and the messages passes between lifelines or objects. Sequence diagram usually involve timing concept. Communication (also known as Collaboration) diagram also shows the messages passes between objects but it focus on the objects role rather than the timing concept. Timing diagram focus on events or conditions changes within objects and it time of occurrence. Lastly, an Interaction Overview diagram shows the overall flows control within the whole system which is a simplify version of all Activity diagrams of the system. Figure 3 shows simple example of behaviour and interaction type diagrams.

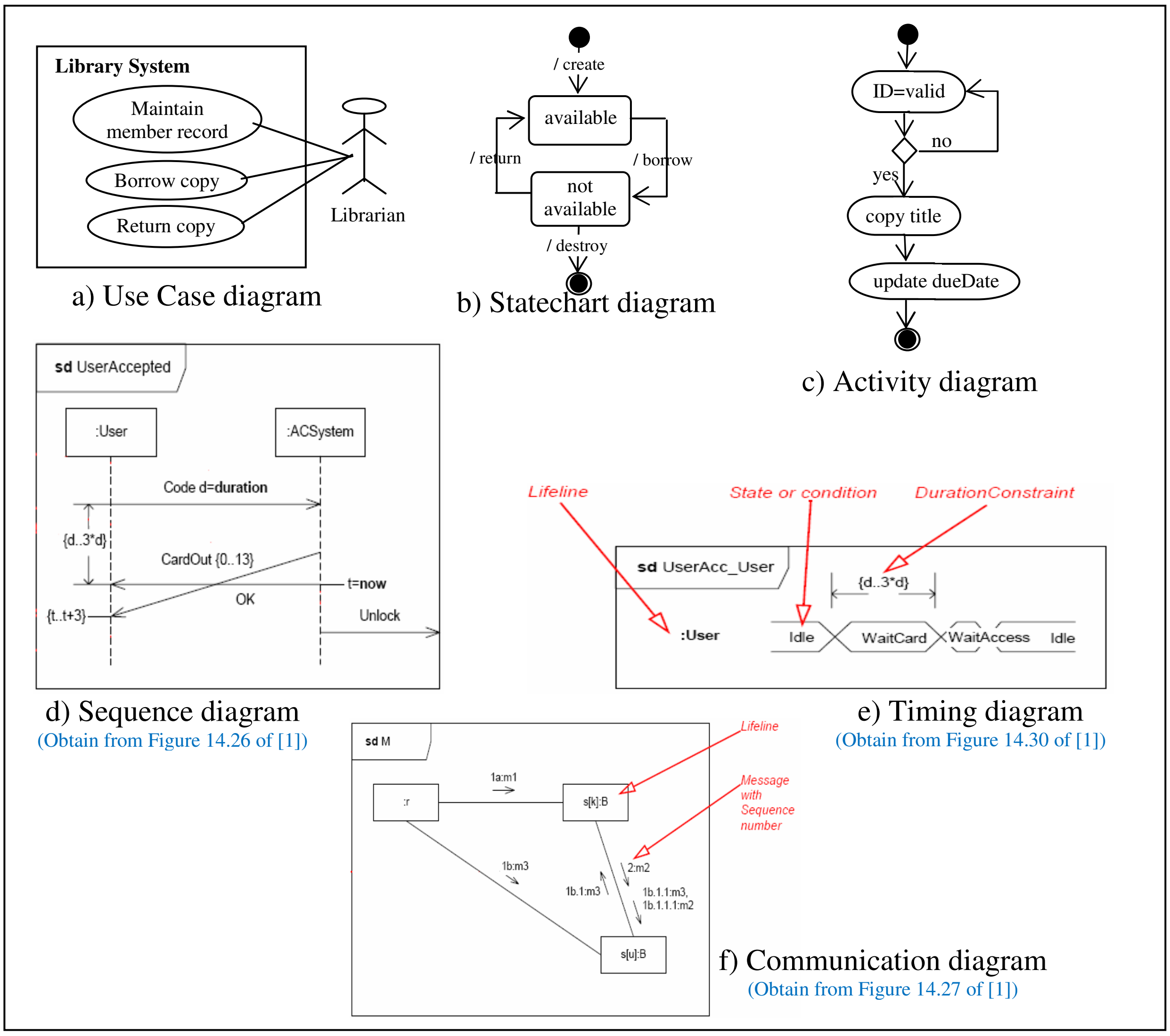


Figure 3: Example of behaviour and interaction models

<u>2.0 Use Case Diagram</u> (1004 words by Rebah)

It is graphical overview the functionality and requirement of the system and the interface with outside the system, as well as it shows the actors and the relationship between the actors and the use cases. Use case diagram shows the design features. Moreover, the use case diagram is the first point when designing new system by using UML and when explaining the requirement for the system in analysis, implementation and documentations stage. Furthermore, the use case diagram used to understand the system and what system is.

The use case diagram has four components: [15][16]

1- Actor
2- Use cases
3- System boundary
4- Relationship

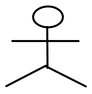
Actor	A role played by a person, other system external system	
Use case	A start-to-finish feature of the system	⬭
Association	The communication among an actor and a use cases	——
Extend	The relationship between use cases,when use case completely consists the behavior of another use case	<<extend>> --------->
Generalization	The relationship between use cases ,when the parent use case identify behaviour that its children can inherit	——▶
Include	The relationship between use cases.when one use case has explicitly contains the behavior of another use case	<<include>> ------->
Boundary	The boundary of the system contain the use cases and the relationships between use cases	▭

Figure 4: Use case general notations

Actor:

"It represents role that can play with regard to a system" [16], or it is external entity that interact with the system. Furthermore, it is the input and the output of the use case, and simple actor may achieve many use cases. The actor includes Person, organization, hardware, software or any external system that interact with the system. Furthermore, the only relationship between the actors is the generalization this is helpful to classifying overlap roles between actors and the relation between the actor and the use cases are association. Moreover, the Actor is represented by stick man figure with suitable name that portray the function as the user of the system and it is write below the figure and the relation represent by line. [15][17]

Use cases:
The use cases are sequence of actions that the user takes on a system to get particular target and it describe the interaction between the actor and the system, as well as is method for capture the task requirement of a system. Moreover, use cases explain what the system wants to do without identifying how the system will execute. The use case is represented by ellipse with name that includes an active verb and usually a noun phrase and it is help to understand the functional requirement of a system. [14][17]

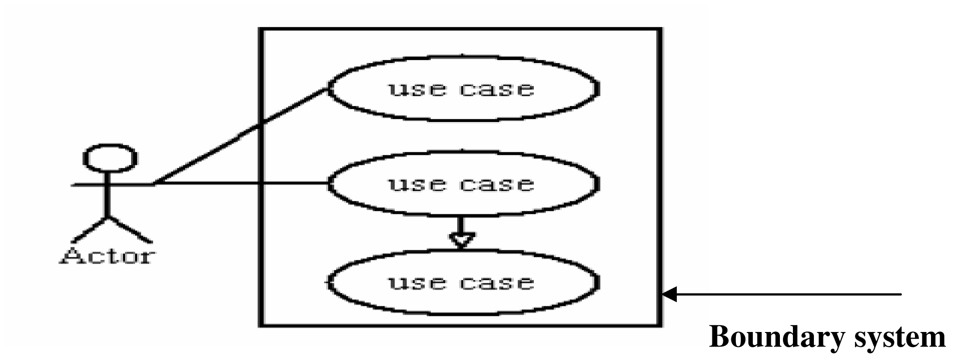


Figure 5: Use case diagram [14]

System boundary:
"System is a piece or multiple pieces of software that perform some sort of function for its users" [15].system boundary boxes is separate between the actors and the use cases and it represent by rectangle around the use cases of the system which is the part of the system and the actors are out side the rectangle which is external to the system, actors are joined with the use cases by lines. Moreover, the name of the system boundary is written inside the box Furthermore, system boundary boxes are rarely used. [17]

2.1 Relationships

The relationship is the association between the actors as well as it illustrate the actors that are participating in a use case. Moreover, the use cases and the path relationships are attach the diagram together. The purpose of the relationship is to explain that an actor is basically involved in a use case, not involve an information exchange in any direction. Relationships are represented by a line connecting between Actors and use cases.
The relationships between use case diagrams in UML have different type:
1- Includes relationship.
2- Extends relationship.
3- Generalizations relationship.[16][15]

Include relationship:-
The include relationship is used to indicate that one use case has explicitly contains the behavior of another use case to execute its function. Moreover, include relationship has two types of use cases that illustrate this situation, firstly including use case which required the functionality

from another use case, and the second kind included use case which is included in the first, including use case. [15][17]

The includes relationship is represented by dashed line with open arrow head which connected the include use case including use case started at including use case and ending at the include use case, as well as the ward <<include>> enveloping with guillemot is written Along the line . When develop the system into an application the included use case is facilitated to identify where can reuse functionality, which is major, advantage of design and development.

Extend relationship:
The extended relationship is used to indicate that use case completely consists of the behavior of another use case at one or specific point, which mean inheritance between the class in C++. The notation of extended relationship is similar that to include relationship. Furthermore, the use case of the include relationship might totally use again with another use case behavior but in the extended relationship the reuse was optional depended on system achievement decision ,as well as the Extends relationship is a conditional include . Extends relationship is represented by dashed line with an open arrow head and along the line is written “<<extend>>", enclosed in guillemets. An extend relationship points the line started at the use case which is conditionally include the other use case. [16][15]

Generalizations relationship:
The generalization relationship is used to represent that the parent use case identify behaviour that its children can inherit, and the child can add or override to that behaviour, as well as Generalization can be related to both actors and use cases to represent that the child inherit functional from parent .Furthermore, Generalization can separated more than two child use case as well as generalization can be hierarchal. Moreover, the use case inheritance is helpful to show that one use case is special kind of another use case. The generalization relationship is represented by a solid line with a hollow triangular arrow. The arrow is drawn indicating the direction of the Generalization.[16][15][17]

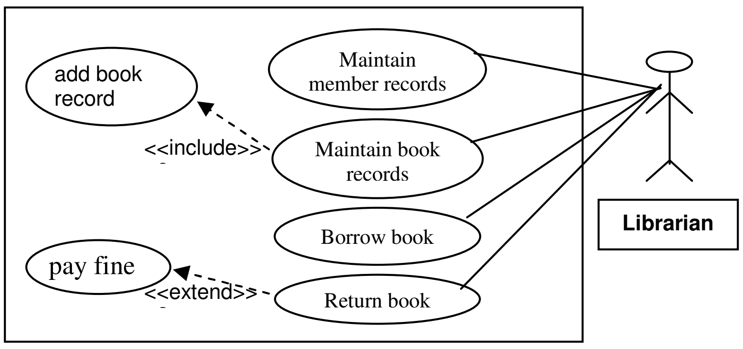


Figure 6: Library system use case

<u>3.0 Class Diagram</u> (1537 words by Fida)

The system static structure can be documented by class diagram. All system life time can be shown on a links (associations) between classes with their operations, attributes, and names. Class diagram define as a rectangular boxes .This boxes initially contain sections. First section gives for class name (or object name), second one contain the class attributes, and the bottom section for methods (operations) associated with the class as shown below [5].

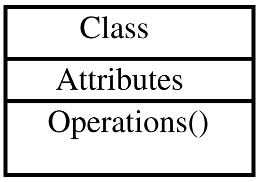


Figure 7: Class icon

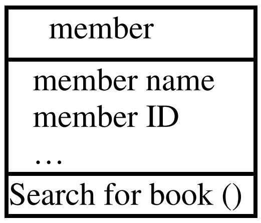


Figure 8: Member class

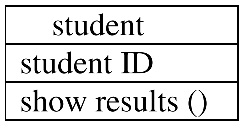
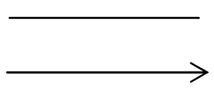
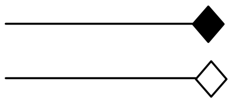
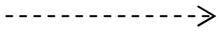
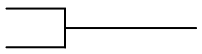
Name	Symbol	Description
Class		whole class rectangular contents : -class name (student). -class attributes (such as student ID). -class operations (such as show results).
Links		defaulted association between classes bi-directional directional link
		composition aggregation
		interfaces
		qualified links(association)
Multiplicities	1 1..* 0 0..1	constraints the numbers of existence links between classes.

Figure 9: Class diagram terminologies

Each object has a class in class diagram .This object may be a person, thing, place, or event for our system. In attribute section (…) indicate that there are other attributes related to that class. Before build a class, you should make a good class model that help to understand the system behavior. The class must meet all the system criteria, and realize the use case diagram, otherwise that the class model can not work well, whatever the techniques are used to build the class.

The class building should:

- Design it in cheap and quick ways, this mean each point required in system behavior must provide in sensible way, by the object of the class.
- Design a system to easy maintain and flexible to add future requirements. If object needs to update ,this should be easily done and should the designer keep a track to that object for any new changes ,such as when the super market change the item cost ,the updated values carry on directly to the system data base without need to make a new design or add other class for that! . A good class represents permanent classes of domain objects, which do not depend on particular function, this mean a name of class to call it is need to be clearer .example in the member class for library system gives the indication of the class contents rather than call the class moon, people in library or any name not related to class attributes!
- The design of classes should be chooses carefully and makes sure each class has a link with other class. Do not build a class without have work to do it.
- The model design must contain all the description of what should the system do.
- The class name should be a noun. The attributes are name phrases, the methods have a verb sentences, and the links (association) between that classes contain the exact operations between them such as: is a , part of, contain, has a …etc.[5],[8],and[9].

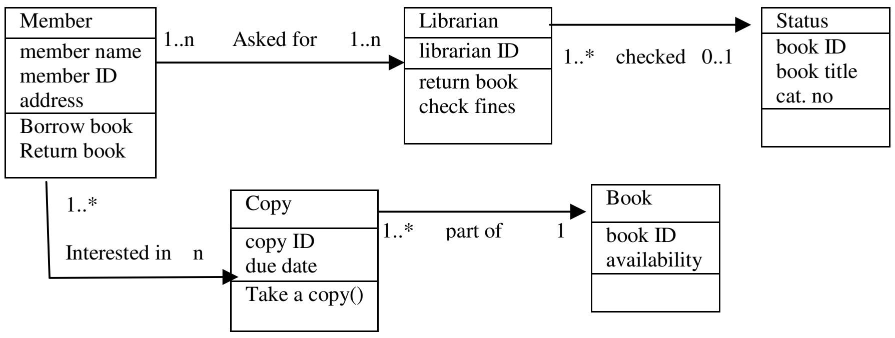


Figure 10: Class diagram for library system

3.1 Attributes

Any class cannot make it process without addressing the states and behavior of that object. It should have the attributes middle section of object (or class) box with lower case letter for each name. Each object have own attributes and they gives the all information that object have. In library system the member object attributes are: member ID, member name, address, phone no. , e- mail. All those have unique information for each member in library. Some attributes values could be changes during system life time and this should included in the first steps of building a system class diagram as we mentioned in good class points. In library, sometimes the member changes the phone no. or address ,but in address changes ,the good actor(designer) for class design need to add new class name address .Because the addresses have complex lines and many information, also the address object could be used from other class not for member class only .It may use for librarian class. This method reduces the interactions loads on system behavior. Some type of class diagram add the attributes information within class diagram like member class, also can but the programming languages arguments such as title: string, name: char ….etc as show in copy class below.[5],[9].

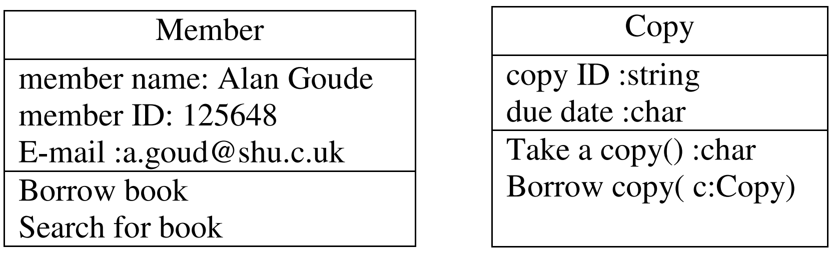


Figure 11: Class attributes declarations [9].

3.2 Multiplicities

The multiplicity of associations represents exactly the numbers between objects (classes) it gives more details about the amounts of relation .For example in library system as shown before ,1..* between the copy class and book class means each copy in library may have one or more copy of that edition …and so on. Each symbol has own meaning:

0..1	zero or one
0..*	zero or more
1..*	one or more
0..n	zero to n (where n>1)
1..n	one to n (where n>1). [7]

3.3 Operations

Operations are located in third (bottom) class section. They must define in each class box, because the object (class) did not know what should do and which other class should interact with. The operation tells object what is the message passed to the receiver object, and the last one invoked that message to perform operation. The familiar format for operation as shown in fig 4 gives (visibility name (parameters): type), Take a copy() :char There is a tight relationship between object state and operation .the object attributes can not change or update it values by it self. The object needs to have a service that wraps the assists modularity in a system life time. This service can provide it only at object's interface .It contain a signature for each operation to avoid the conflicting during sending or receiving messages between system objects. Normally the object just respond to a simple quires .There are objects need to send a message that have a valid call on an operation.

The operations write in verb sentences lower case letters to give the indicate about what the class (object)need to do and which other class(receiver class)operations or attributes are related to that operations . [7] [9], and [5]. In class diagrams the generalization operation gives important concepts. In library system we can call a library member is generalization of librarian, because each librarian is already a member of library. This go to other class diagrams relations such as, aggregation, composition, association class, interface, and others. They classified in table below.

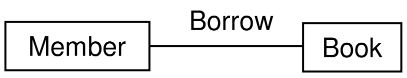
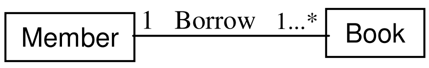
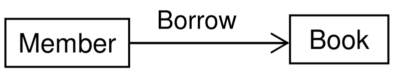
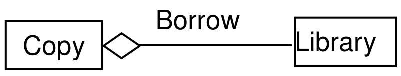
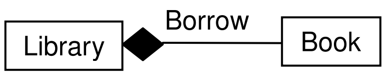
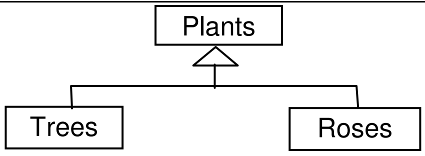
Relation	Symbol	Description
1. Association		The association established when two classes are connected to each other. <Member can borrow a book>
1.1 Multiplicity		Relation details by give one to one, one to many, many to many.<one or more books can borrowed by a member>
1.2Direct association		The arrowed association gives the one direct relation between classes, because the association by default comes with bi-directional links.
2. Aggregation		Hallow diamond means. If the copy class was damage, the library class still exists.
3. Composition		Solid diamond means, if the library class damaged, the book classes also damaged.
4. Generalization /inheritance		Tree and roses are part of plants, and they make photosynthesis shared from plants properties.

Figure 12: Examples of Class Operations

Generalization: in example below, the engineer, and doctor are goes to one people (human) class .they generalize, or inherit a people. We can call doctors, engineers, teachers are people, but never called the people are doctor! Generalization goes in one way from children class to parent's class.

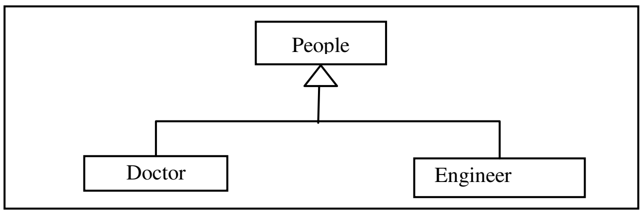


Figure 13: Generalization

Aggregation: Is represented by empty diamond. The seat is an aggregate of car .This mean if the seat class damaged, the car class still exist.

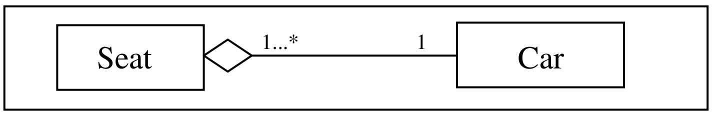


Figure 14: Aggregation

Composition: It represented by solid diamond. The engine is composed of car. Mean if the engine class damaged, the car also will damaged as well [10].

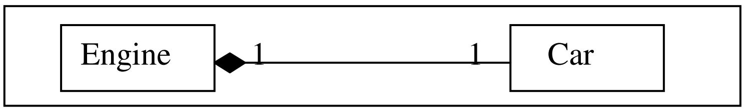


Figure 15: Composition

Associations: the UML versions implement different types of association operations. The basic concepts that labeled the association between classes called (role names) .it give multiplicity constraints [6]. Some versions noted two associates between two objects for cost effective of system, and label each one if the class have references from many classes such as company and client class as shown.

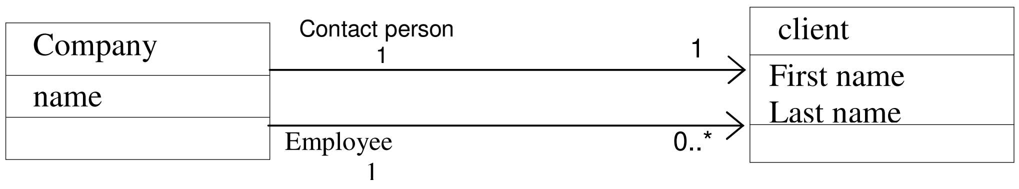


Figure 16: Associations [10]

For first association ,the one to one cardinality means for each company there is only one contact person for each company, in second association means there is zero or many employees in one company. This example shows how add two attribute that referenced to other class [10]. Operations not end in above examples .They are the general operations used ,but there are many of them used for advanced steps depends on how the system life time work and system needs for other operations such as, association with multiplicities and navigability(look like associations but with one link between classes), interfaces, pattern, and qualified association. The good system works perfectly if it build as simply as possible to understand it from others (clients), and easily to upgrade, updates, and maintenance.

4.0 Relationship between Use Case and Class Diagram (261 words by Lai San)

The use case diagram is usually deployed at the very beginning stage of any project. Use case diagram only shows what is the function and requirement of a system. Hence, the next stage of the project implementation will be transform the use case diagram to others diagram (usually another type of UML diagram) closer to its project implementation. This process is called realisation. One of the most popular diagrams used will be Class diagram. Following will be some guideline for use case realisation. Almost everything in UML is optional, hence it is not a necessary step to follow.

First, identify possible set of classes that can derived from the use case diagram. Then understand how those classes might relate to deliver the functionality of the use case. All details that are not directly related to the collaboration will be suppressed. At this stage, the class diagram might not really tell the whole story of the system. But as the design continues, more information will be added. The diagram will eventually form the picture as the model transform again. It is very important that the consistency of all diagrams used is maintained especially during the realisation process. At the end of realisation process, both the use case and class diagram should have structural and notational similarities to the collaboration. There should be a class for each nodes and each classes have associations that link them. Some class might not have any relationship yet but will be added in later stage. The end result must have all class correspond with a relationship. [5]

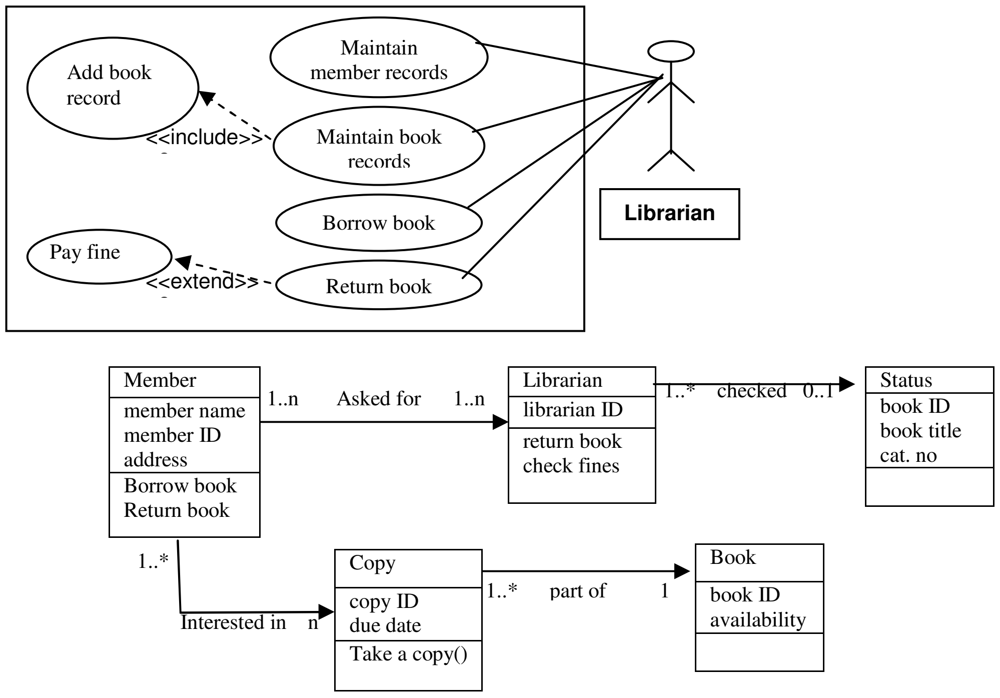


Figure 17: Example of Use Case and Class diagram for library system after realisation

<u>5.0 Conclusion</u> (320 words by Lai San and Fida)

In summary, UML is a simple to use guideline for creating model of a system. Because of its simplicity and flexibility, it is acceptable and applicable in various fields as long as there is a need of building model. This makes UML a common language of modelling for all level. Each of the thirteen UML diagrams holds some information of other diagram and can be used to realise the other diagram. Use Case, Class, Object, and Package, Statechart, and Activity diagrams are usually used in early project stage such as project design and analysis. While other diagrams like Sequence, Timing, or Communication diagram are used during project implementation. There are many methodologies like Object Oriented that provide guidelines to transform one diagram to another diagram while ensuring the consistency is still preserved.

However, there are no specific regulations or procedures that must be follows in UML. Most of UML diagrams is optional and can be ignore, as long as it suit the purpose of the project. Use Case diagram is a very useful model that shows the initial requirement of a system. It is normally use in gathering information and specification for the system. While Class diagram is used to classify the objects and relationships with their separate classes. The Class diagram help in defining further properties of objects and its functions for projects implementation. It shows the whole system behaviours from beginning to end. Both diagrams are very useful in a project design stage. Use Case diagram help to ensure the rest of project design are consistent with requirement and limitation. Statechart, Sequence and Activity diagrams will be realised from both Use Case and Class diagrams. Statechart, Sequence and Activity diagrams help the designer to determine how the system will react to each event and what type of database required. Other UML diagrams will also be realise from these two diagrams during the whole project design and implementation stage.

References:

[1] Introduction to OMG Unified Modelling Language™ (UML®) by OMG, updated on 11/09/2007. Web link: http://www.omg.org/gettingstarted/what_is_uml.htm

[2] Holt J., UML for system engineering, The Institution of Electrical Engineering, (2001).

[3] Boggs M. and Boggs M., Mastering UML with Rational Rose, Sybex Inc., (2002).

[4] Practical UML: A Hands-On Introduction for Developers by Randy Miller on 1/12/2003. On:http://dn.codegear.com/article/31863#use-case-diagram

[5] Bennett S., Mcrobb S. and Farmer R., Object Oriented System Analysis and Design using UML, McGraw Hill, (2006).

[6] Graham I., Object Oriented Methods Principles & Practice, third edition, (2001).

[7] Ambler S., The Object Primer: Agile Model-Driven Development with UML 2.0 (2004).

[8] Sommerville I., Software Engineering, seventh edition (2004).

[9] Stevens P., and Pooley R., Using UML Software Engineering with Objects and Components, (2000).

[10] Visual Case Tool – UML Tutorial, Artiso Visual Case, visited at 3-3-2008. On: http://www.visualcase.com/tutorials/class-diagram.htm

[11] Relationship table,visit at 7-3-2008. On: http://www.developer.com/design/article.php/10925_2206791_1

[12] OMG Unified Modelling Language (OMG UML), Superstructure, V2.1.2, November 2007. On: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF

[13] B. Henderson Seller 1999-2003, Module M5 UML case diagrams (Accessed: 04/03/2008)

[14] Jeremy T. Lanman, 25 February 2002, Using UML Use Cases and Activity Diagrams to Describe Software Requirements (Accessed: 04/03/2008)

[15] Jason T. Roff, 2003, The McGraw-Hill/Osborne, UML A Beginner's Guide.

[16] Kendall Scott, 2001 by Addison-Wesley, UML explained

[17] Russ Miles & Kim Hamilton, O' Reilly April 2006, UML 2.0