**OOAD AND UML (I M.SC COMPUTER SCIENCE)**

**UNIT IV**

Object oriented design – Design axioms – Designing classes – Layering the Software design: - data access layer, User interface layer, Control/business Logic layer.

**UNIT - V**

**U**ML - Examples on: Behavioral models – Structural models – Architectural Models from real world problems.

**Text Books:**

1. Bahrami Ali, Object oriented systems development, Irwin McGraw Hill, 2005 (First 4 units covered here).
2. Booch Grady, Rumbaugh James, Jacobson Ivar, The Unified modeling language – User Guide, Pearson education, 2006 (ISBN 81-7758-372-7) (UNIT -5 covered here).

 **UNIT-IV**

1.**DBMS** – Database Management System is a set of programs that enables the creation and maintenance of collection of related data. The DBMS and associated programs access, manipulate, protect and manage the data.

**.**

 **2.Define Axioms?**

An axiom is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception.

 3**.Define Corollary?**

 A corollary is a proposition that follows from an axiom or another proposition that has been proven

 4. **What are the Advantages of OOD?**

* + - Easier maintenance.
		- Objects may be understood as stand-alone entities Objects are appropriate reusable components For some systems, there may be an obvious mapping from real world entities to system objects

**5 MARKS**

**1.Explain Access Layer Design?**

The need of access layer is to design/ create a set of classes that have rights and the way to

communicate with the database or data storage place. It isolates following information from the

business layer hence it gives service to the business layer.

1. Where data resides?

Local, Local server, remote server etc.

1. How data resides?

In a database, in a file, DBMS, RDBMS, ORDBMS, Internet etc.

1. How to access the stored data?

SQL, stream, File stream, ORB (for DCOM/ EJB) etc.

Access layer provides 2 important service to business layer

Translate Request The business layer is not aware of the protocol for accessing data as the

internal details are known only to the access layer classes. So any request from the business layer for

data cannot be transformed to storage as such. Such request are translated in to a form that storage

managers can understand and then transformed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Business | Request |  | Translated |  |
| Layer | Access | Database/ |
| as |  |
| Object/ | Layer |  | Data Storage |
| messages | Request as |
| Classes | Classes |  |
|  | query etc |  |
|  |  |  |  |
|  |  |  |  |



Translate Result The business layer objects/ classes cannot understand the data send as such from the database/ storage. But the access layer classes can understand the format of result data from the storage as well as the format the business layer can understand. Hence the access layer classes translate the result data in to a form so that business layer can understand.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Business | Translated |  | Unformatted |  |
| Layer | Access | Database/ |
| result | result from |
| Object/ | Layer | Data Storage |
|  | storage |
| Classes |  | Classes |  |
|  |  |  |
|  |  |  |  |  |



Persistent Data Persistent data is one which exists between executions. These data is to be stored permanently for future executions.

E.g. In a student class the name, no, address etc are persistent data.

Transient Data Transient data is one that may not exist between executions. These data are need not to be stored in database for future execution.

E.g. In a student class the variables meant for temporary purpose like temp\_tot etc are transient data

**Access Layer Design Sub Process:**

1. Design access layer
	1. Create mirror class for all classes identified in business layer which contains persistent data.
	2. Identify access layer class relationship
	3. Simplify access layer classes and class relationship
		1. Remove redundant classes

  2.Singe method classes can be removed and added in another class.

1. Iterate and refine.

id Component Model

EA 4.00 Unregistered Trial Version

Access Layer

EA 4.00 Unregistered Trial Version

ClassDB1

EA 4.00 Unregistered Trial Version EA 4.00 Unregistered Trial Version

ClassDB3

EA 4.00 Unregistered Trial Version

**2.Explain Database Models:?**

database model is a collection of logical constructs used to represent the data structure and data relationships with in the database.

The conceptual model represents the logical nature of organization of data where a implementation model represents the real implementation details.

1. Hierarchical Model

This model represents the data as a single rooted tree structure. Each node represents the data object and connection between various nodes represents the parent – child relationship. This relationship resembles the generalization relationship among objects. A parent node can have any number of child node where each child node shouldn’t have more than one parent node.

Motor Vehicle

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Bus |  | Truck |  | Car |
|  |  |  |  |  |  |  |  |

1. Network Model

A network database model is similar to hierarchical model. Here in this model each parent can have any number of child nodes and each child node can have any number of parent nodes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Motor Vehicle |  | Transport |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Bus | Truck | Car |

1. Relational Model

This model defines 4 basic concepts. Table, Primary Key, Foreign Key and relation between tables.

Table – It’s a collection of records form the table. The Table is composed of various rows (tuples) and columns (attributes).

A primary key is a combination of one or more attributes which is used to identify any tuple unambiguously. Primary never gets duplicated in a table.

Foreign key is an attribute of a table that is a primary key of another table.

Relation between tables – The primary key of one table is the foreign key of another table.

Also data can be searched with the combination of more then one table.

Because of these reasons the relational model is the most widely used model.

STUDENT\_UNIV STUDENT\_COLLEGE



Univ % of

Reg\_no Name mark

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | College |  | Bus |  |
|  | id\_no | Name | root | Area |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |



|  |  |  |
| --- | --- | --- |
| Name | Bus | time |
|  | Root |  |
|  |  |  |

STUDENT\_TRANSPORT

Database Interface:

The interface of a database includes Data Definition Language (DDL), Data Manipulation Language (DML) and a query.

There are two ways to establish relation with the database

1. By embedding SQL in a program that needs an interface. Since SQL (Structured Query Language) is one of widely accepted language usage of SQL in a program makes programmers feel easy.
2. Extending the programming language to manage data. Here the programmers have to know about the data models and implementation details.

DDL – Data Definition Language is the language used to describe the structure of

Objects (data) stored in a database and relation between them. This structure of information is called Database Schema. DDL is used to create tables in a database.

E.g

CREATE SCHEMA COLLEGE

CREATE DATABASE COLLEGE\_DB

CREATE TABLE STUDENT\_TRANSPORT (Name char (10) NOT NULL, Busroot number (2) NOT NULL, time TIME)

DML and Queries:

Data Manipulation Language is used for creating, changing and destroying data inside a table. SQL (Structured Query Language) is the standard language for making queries.

A query usually specifies

* The domains of the discourse over which to ask the query.
* The elements of general interest.
* The conditions are constraints that apply.
* The ordering, sorting, or grouping of elements and the constraints that apply to the ordering or grouping.

Traditional DML specifies what are the data desired and specifies how to retrieve the data. Object Oriented DML just specifies what data is desired and not how. While developing applications that uses SQL the mostly used way is to embed the SQL statements inside the program.

Transaction:

A transaction is a unit of change in which many individual modifications are aggregated into a single modification that occurs entirely or not at all. Thus all the changes inside the transactions are done fully or none at all.

A transaction is said to be commit if all the transactions made are successfully updated to the database and said to abort if all the changes made cannot be added to database.

Concurrent Transaction:

A transaction is said to be concurrent if it uses a database which is used by another transactions. Hence a database lock is used to avoid problems like “last updated”. When a transactions starts using a database it is locked and is not available to other transactions.

Distributed Database is one in which a portion of database lies of one node and other on another node.

**3.Explain Client Server Computing.?**

Client – Node that request for a service

Server – Node that services the request.

Client Server computing is the logical extension of modular programming. The fundamental concept behind the modular programming is decomposing the larger software in to smaller modules for easier development and maintainability.

Client Server computing is developed by extending this concept i.e, modules are allowed to execute in different nodes with different memory spaces. The module that needs and request the service is called a client and the module that gives the service is called a server.

The network operating system is the back bones of this client sever computing. It provides services such as routing, distribution, messages, filing and printing and network management. This Network Operating System (NOS) is called middleware.

Client Program:

* It sends a message to the server requesting a service (task done by server).
* Manages User Interface portion of the application.
* Performs validation of data input by the user.
* Performs business logic execution (in case of 2 tier).
* Manages local resources.
* Mostly client programs are GUI.

Server Program:

* Fulfills the task requested by the client.
* Executes database retrieval and updation as requested by the client.
* Manages data integrity and dispatches results to the client.
* Some cases a server performs file sharing as well as application services.
* Uses power full processors and huge storage devices.

File Server – Manages sharing of files or file records. Client sends a message to the file

server requesting a file or file record. The File Server checks the integrity and availability of file/record.

Data Base Servers – Client pass the SQL query in the form of messages to the server in turn server performs the query and dispatches the result.

Transaction Servers – Client sends message to the server for a transaction (set of SQL statements) where the transaction succeeds or fails entirely.

Application Servers – Application servers need not to be database centric. They may Serve any of user needs such as sending mails, regulating download.

Characteristics of Client Server Computing:

1. A combination of client/ front end process that interacts with the user and server/ backend process that interacts with the shared resources.
2. The front end and back end task have different computing resource requirements.
3. The hardware platform and operating system need not be the same.
4. Client and Server communicate through standard well defined Application Program Interface (API).
5. They are scalable.

Distributed and cooperative processing

In Distributed Computing the applications and business logic are distributed across multiple processing platforms. It implies that the processing should be carried out in different process for the transaction to be completed. These processes may not run at same time. Proper synchronization mechanism is provided if needed.

Cooperative processing is a type of distributed computing where more then one process is to be completed for completing the entire transaction. These processes are executed concurrently on different machines and good synchronization and inter process mechanism is provided.

Distributes Object Computing offers more flexible way of distributed computing where mobile software components (objects) travel around the network and get executed in different platforms. They communicate with application wrappers and manage the resources they control. In this computing the entire system is made up of users ,objects and methods.

Various DOC standards are OMG’s CORBA, OpenDoc, Microsoft ActiveX/ DCOM.

**4.Explain Object Relation Mapping.?**

In a relational database system the data are stored in the form of tables where each table contains a set of attributes/fields and tuple/rows. In an object oriented environment the counterpart of class is a table.

In the mapping the classes are mapped to table such a way that the persistent data members of classes will become the attributes. Each row in the table corresponds to an object. The following mapping types are used in object oriented environment

* 1. Table – Class Mapping
	2. Table – Multiple Class Mapping
	3. Table – Inherited Class Mapping
	4. Tables – Inherited Class Mapping
1. Table Class Mapping

It’s a simple one – to – one mapping of a class to a table and properties of class are become the fields. Each row in the table represents an object and column

|  |  |
| --- | --- |
| represents a property of objects. | CAR |
|  |  |  |  |  | Cost |
| CAR TABLE | Color |
| Cost | Color | Make | Model |  | Make |
|  |  |  |  |  | Model |



1. Table – Multiple Classes Mapping

Here a single table is mapped to more than one non inherited classes. So all the persistent properties of mapped classes represents the columns of the table. The column value that is not common for the mapped classes can be used to identify the instance.

In the below example the Employee Class and Customer Class are mapped to person table. Instances of employee class can be identified from the rows whose custID value is NULL. Also instances of Customer class can be identified from the rows whose empID is NULL.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Employee |  |
|  |  |  |  |  | Name |  |
| Name | Address | CustID | empID |  | Address |  |
|  |  |  |  |  | empID |  |
|  |  |  |  |  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Mr. X | A1, a2 | C123 | NULL |  |  |
| Mr.Y | A2, A3 | NULL | E123 |  | Employee |
|  |  |  |  |  | Name |
|  |  |  |  |  |
|  |  |  |  |  | Address |
|  |  |  |  |  |
|  |  |  |  |  | empID |
|  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |



1. Table Inherited Classes Mapping

In this case a single table is mapped to more than one class which has a common super class.

This mapping allows user to represent the instances of super class and subclasses in a single table.

In the given example the instances of Employee class can be identified from the rows whose wage and salary are NULL. The instances of Hourly Employee can be identified from the rows whose salary is NULL. The instances of Salaries Employee can be identified from the rows with Wage as NULL.

1. Multiple Tables – Inherited Classes Mapping.

This kind of mapping allows is a to be established among tables. In a relational database this is possible by using primary key and foreign key.

In the below example Employee and Customer inherits the properties of Person class. The Person table is mapped to Person class, Employee table is mapped to Employee class and Customer table is mapped to Customer Class. There exist is a relation between Employee, person and customer, person.



|  |  |  |  |
| --- | --- | --- | --- |
| cd Data Model |  |  |  |
| nregistered Trial Version |  | EA 4.00 Unr |
|  |  | Person |  |  |
|  | - | ssn: |  | EA 4.00 Unr |
| nregistered Trial Version |  |
|  | - | name: |  |  |
|  | - | address: |  |  |
| nregistered Trial Version |  | EA 4.00 Unr |
| nregistered Trial Version |  | EACustomer4.00 Unr |
|  | Employee |  |  |  |
| - | dept: |  | - | name: |
| - | slary: |  | - | address: |
|  |  | EA 4.00 Unr |
| nregistered Trial Version |  |
|  |  |  | - | company: |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Address | SSN |  | Name | Dept | SSN | Salary |
|  |  |  |  |  |  |  |  |

Name Address Company

Study of Multi Database System and Open Database Connectivity



DBMS RDBMS OODBMS DBMS

LOCAL DATABASES

MDBS

MDBS

APPLICATION

Multi database system is a heterogeneous data base system facilitate the integration of heterogeneous database systems and other information sources. Federated multi database systems are one that are unstructured or semi unstructured.

This multi database system provides single database illusion to the users. The user initiate a single transaction that in turn uses many heterogeneous databases. Hence the user performs updation and queries only to a single schema. This schema is called the global schema and it integrates schemata of local databases. Neutralization solves the schemata conflicts.

The query and updates given to global schema by the user is decomposed and dispatched to local databases. The local databases are managed by gateways as one gate way for each local database.

Open Database Connectivity (ODBC) is an API (Application Program Interface) that provides database access to application programs. The application programs can communicate with the database through function calls (message passing) regardless of the type and location of the database.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | ODBC Driver Manager | Datab |
|  | O |  | ase A |
|  |  |  |  |  |
|  | D |  |  |  |  |  |
|  |  |  | Driver for Database A |  |  |
|  | B |  |  |  |  |
| Application |  |
| C |  |  |  |  |  |
|  |  |  |  |  |
| Program |  |  |  |  |  |
| A |  |  | Driver for Database B |  | Datab |
|  |  |  |  |
|  | P |  |  |  |  | ase B |
|  |  |  | Driver for Database C |  |
|  | I |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |



Datab

ase C

The above diagram shows the logical view of Virtual Database using ODBC. The application program uses ODBC API to communicate with the database. Application programs passes same messages to interface irrespective of the type and location of database. ODBC maintains a set of drivers necessary for communicating with the database. This reduces the database related complexities for a programmer.

The driver manager loads and unloads drivers, performs status checking, manages multiple connection and heterogeneous databases.

**10 Marks**

1.Explain the Design of Business Layer?

Business layer involves lot of logic that is to be implemented in order to achieve the customer needs. Analysis is carried out for business layer objects. Hence the design for business layer has got a strong dependency with the model produced in the analysis phase.

The activities involved in Business Layer design are

1.Business Layer Class Design – apply design axioms for designing classes for business layer. Designing classes includes designing their attributes, methods and relationships.

I. Design/ Refine UML Class diagram developed in previous phase/ iteration.

* + 1. Design/ Refine attributes (Use OCL)
			1. Add left out attributes
			2. Specify visibility, data type and initial value if any for attributes
		2. Design/ Refine Methods (Use OCL and UML Activity diagram)
			1. Add left out methods
			2. Specify visibility of the protocol (+, - ,#)
			3. Specify the argument list and return type
			4. Design the method body using UML Activity diagram and OCL.
		3. Refine association
		4. Refine Generalization and aggregation.
		5. Iterate and refine.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Refine/ Design |  | Refine/ Design |  | Refine class |
| Attributes identified |  | Methods identified |  | association, |
| in previous iteration/ |  | in previous iteration/ |  | inheritance and |
| Phase |  | Phase |  | aggregation. |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |



Refining Attributes:

Attributes represents the information maintained by each object. Complete list of attributes should be identified in order to maintain a complete set of information. Detailed information of attributes is not specified in analysis phase but in design phase.

OCL is used to represent the attribute details inside a class diagram/ notation.

Various types of attributes are

1. Single valued attributes – an attribute represents one value at a time.
2. Multi valued attributes – an attribute can store more than one value
3. Reference attributes – an attribute refers (alias) another instance.

OCL format for representing attributes:

The OCL specification for specifying attributes is

Visibility attribute name : type

OR

Visibility attribute name : type = initial value

E.g.

* Name : String
	+ represents a public attribute Name of type String

# Name : String = “Hello”

* + represents a protected attribute Name of type String with initial value Hello

Designing/ Refining Methods:

Designing methods involves design of Method protocol and Method body. A method protocol defines the rule for message passing to this particular object where the method body gives the implementation details. The types of methods provided by class are

1. Constructor – Method that is responsible for creating objects/ Method invoked during instantiating.
2. Destructor – The method that destroys instances/ Method invoked when an object is freed from memory.
3. Conversion Method – Methods responsible for converting one form of date to other form.
4. Copy Method – Methods responsible for copying information.
5. Attribute Set – Method responsible for setting values in attributes
6. Attribute Ger – Method responsible for getting the values from an attribute
7. I/O Methods – Method responsible for getting and sending data from a device

 Domain Specific – Those methods responsible for some functionality in a particular domain.

Designing Protocol:

Protocol gives the rule for message passing between objects. Protocol is the interface provided by the object. Based on the visibility of the protocol it can be classified into

1. Public Protocol
2. Private Protocol
3. Protected Protocol.

Private protocols specify messages that can be send only by the methods inside the class.

They are visible only inside the class.

Protected protocols specify messages that can be send only by the methods inside the class.

But they can be inherited by the subclass.

Public protocols specify messages that can be send by the methods with in the class as well as objects outside the class.

Protocol and Encapsulation leakage – If protocols aren’t well designed unnecessary messages are made available outside the class results in encapsulation leakage.

Internal layer contains the private and protected protocols where an External layer contains public protocols.

OCL Specification for Protocol Design:

The specification is

Visibility protocol name (argument list) : return type

Where argument list is arg1: type, arg2: type, arg3: type … argn: type E.g.

+ getName () : String

It’s a public protocol named getName with no parameters and it returns a value of type String.

- setData (name : String, no : Integer) : Boolean

It is a private protocol that accepts 2 arguments one of type String and other of type Integer. It returns a value of type Boolean.

Designing Method body:

UML Activity diagram along with OCL specification can be used for representing the

body of the method. Activity diagram representing the method body says how the

work should be done.

[Note: Use an activity diagram from case study]

Business layer involves lot of logic that is to be implemented in order to achieve the customer needs. Analysis is carried out for business layer objects. Hence the design for business layer has got a strong dependency with the model produced in the analysis phase.

The activities involved in Business Layer design are

1. **Business Layer Class Design** – apply design axioms for designing classes for business layer. Designing classes includes designing their attributes, methods and relationships.

I. Design/ Refine UML Class diagram developed in previous phase/ iteration.

* + - Design/ Refine attributes (Use OCL)
			* Add left out attributes
			* Specify visibility, data type and initial value if any for attributes
		- Design/ Refine Methods (Use OCL and UML Activity diagram)
			* Add left out methods
			* Specify visibility of the protocol (+, - ,#)
			* Specify the argument list and return type
			* Design the method body using UML Activity diagram and OCL.
		- Refine association
		- Refine Generalization and aggregation.
		- Iterate and refine.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Refine/ Design |  | Refine/ Design |  | Refine class |
| Attributes identified |  | Methods identified |  | association, |
| in previous iteration/ |  | in previous iteration/ |  | inheritance and |
| Phase |  | Phase |  | aggregation. |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |



Refining Attributes:

78

Attributes represents the information maintained by each object. Complete list of attributes should be identified in order to maintain a complete set of information. Detailed information of attributes is not specified in analysis phase but in design phase.

OCL is used to represent the attribute details inside a class diagram/ notation.

Various types of attributes are

1. Single valued attributes – an attribute represents one value at a time.
2. Multi valued attributes – an attribute can store more than one value
3. Reference attributes – an attribute refers (alias) another instance.

OCL format for representing attributes:

The OCL specification for specifying attributes is

Visibility attribute name : type

OR

Visibility attribute name : type = initial value

E.g.

* Name : String
	+ represents a public attribute Name of type String

# Name : String = “Hello”

* + represents a protected attribute Name of type String with initial value Hello

Designing/ Refining Methods:

Designing methods involves design of Method protocol and Method body. A method protocol defines the rule for message passing to this particular object where the method body gives the implementation details. The types of methods provided by class are

1. Constructor – Method that is responsible for creating objects/ Method invoked during instantiating.
2. Destructor – The method that destroys instances/ Method invoked when an object is freed from memory.
3. Conversion Method – Methods responsible for converting one form of date to other form.
4. Copy Method – Methods responsible for copying information.
5. Attribute Set – Method responsible for setting values in attributes
6. Attribute Ger – Method responsible for getting the values from an attribute
7. I/O Methods – Method responsible for getting and sending data from a device
8. Domain Specific – Those methods responsible for some functionality in a particular domain.

Designing Protocol:

Protocol gives the rule for message passing between objects. Protocol is the interface provided by the object. Based on the visibility of the protocol it can be classified into

1. Public Protocol
2. Private Protocol
3. Protected Protocol.

Private protocols specify messages that can be send only by the methods inside the class.

They are visible only inside the class.

Protected protocols specify messages that can be send only by the methods inside the class.

But they can be inherited by the subclass.

Public protocols specify messages that can be send by the methods with in the class as well as objects outside the class.

Protocol and Encapsulation leakage – If protocols aren’t well designed unnecessary messages are made available outside the class results in encapsulation leakage.

Internal layer contains the private and protected protocols where an External layer contains public protocols.

OCL Specification for Protocol Design:

The specification is

Visibility protocol name (argument list) : return type

Where argument list is arg1: type, arg2: type, arg3: type … argn: type E.g.

+ getName () : String

It’s a public protocol named getName with no parameters and it returns a value of type String.

- setData (name : String, no : Integer) : Boolean

It is a private protocol that accepts 2 arguments one of type String and other of type Integer. It returns a value of type Boolean.

Designing Method body:

UML Activity diagram along with OCL specification can be used for representing the

body of the method. Activity diagram representing the method body says how the

work should be done.

**2.Explain Object Oriented Design process?**

Software Design represents the logic of the software system providing more dependency to the computer domain than physical/ user domain. Design actually deals with “LOGIC TO IMPLEMENT IN PROGRAM TO ACHIEVE THE SYSTEM GOAL”

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Analysis |  | SOFTWARE | Design and Design Document |
| Model |  |  |
|  | DESIGN PROCESS |  |
|  |  |  |
|  |  |  |  |
|  |  |  |  |



1. **SOFTWARE DESIGN PROCESS:**

Software Design Process is the set of activities involved in developing a good and quality design.

This is a sub process of Software Engineering Process.

|  |  |  |
| --- | --- | --- |
| DESIGN AXIOMS AND COROLLARIES |  | Design Patterns |
|  |  |  |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Design/ Refine |  | Design/ Refine |  | Design/ |  | Verify the |
| the classes for |  | Classes for |  | Refine User |  | Design/ |
| Business Layer |  | Access Layer |  | Interface |  | Usability test |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |



Iterate

The above diagram shows the different sub phases in software design process.

Unified Approach suggests 3-tired architecture. Since design has strongly dependency with implementation, the design should carried out for these layers separately.

Business Layer Class Design – apply design axioms for designing classes for business layer. Designing classes includes designing their attributes, methods and relationships.

I. Design/ Refine UML Class diagram developed in previous phase/ iteration.

 Design/ Refine attributes (Use OCL)

* + 1. Add left out attributes
		2. Specify visibility, data type and initial value if any for attributes
	1. Design/ Refine Methods (Use OCL and UML Activity diagram)
		1. Add left out methods
		2. Specify visibility of the protocol (+, - ,#)
		3. Specify the argument list and return type
		4. Design the method body using UML Activity diagram and OCL.
	2. Refine association
	3. Refine Generalization and aggregation.
	4. Iterate and refine.

II. Design access layer

* 1. Create mirror class for all classes identified in business layer.
	2. Identify access layer class relationship
	3. Simplify access layer classes and class relationship
		1. Remove redundant classes
		2. Singe method classes can be removed and added in another class.
	4. Iterate and refine.
	5. Design View Layer
		1. Design the macro level user interface – identify view layer objects
		2. Design micro level user interface
			1. Design view layer objects by applying design axioms and corollaries.
			2. Build a prototype of view layer interface
		3. Verify Usability and User Satisfaction
		4. Iterate and refine
1. Iterate and refine the above steps in necessary.
2. **DESIGN AXIOMS AND COROLLARIES:**

Axiom is a fundamental truth that has no exception or counter proof.

A corollary is one derived from axiom or another proves theorem.

These can be used in the software design for the following reasons

1. Making the design more informative and uniform.
2. Avoid unnecessary relationships and information.
3. Increase the quality.
4. Avoid unnecessary effort.

**Axiom1**  The independence axiom Maintain independence of components/ classes/ activities

 **Axiom2** The information axiom Minimize the information content of the design

 **Axiom1 The independence axiom**

It says that when we implement one requirement of an user it should not affect the other requirement or its implementation. I.e., each component should satisfy its requirements without affecting other requirements.

Consider the component C1 responsible for sending Multimedia files.

Choice 1: C1 reads the files and send the file header first and then the content in a byte stream. Here the component satisfies the first requirement where it fails to satisfy the second requirement.

Choice 2: C1 reads the files and compress the content and file header contains the file and compression information. Since the file size transferred is reduced this choice satisfies both requirements.

**Axiom2 The information axiom**

It deals with the simplicity and less information content. The fact is less number of information makes a simple design, hence less complex. Minimizing complexity makes the design more enhanced. The best way

to reduce the information content is usage of inheritance in design. Hence more information can be reused from existing classes/ components.

**E.g.**

|  |  |  |  |
| --- | --- | --- | --- |
| Chioce1: (with out inheritance) |  |  |  |
|  |  |  |  |  |  |
|  | Vehicle |  |  | Car |  |
|  |  |  |  |  |  |
|  | Name |  |  | Name |  |
|  | Brand |  |  | Brand |  |
|  | Owner |  |  | Owner |  |
|  |  |  |  | Color |  |
|  | Stop() |  |
|  |  |  | Engine No |
|  |  |  |  |
|  | Start() |  |  |  |  |
|  |  |  | Stop() |  |
|  |  |  |  |  |
|  |  |  |  |
|  |  |  |  |  |  |

Class car maintains more information even though they are already maintained in vehicle class. Since car class maintains more information the design that contains the car class makes the design look more complex.

Vehicle 3 properties and 2 methods.

Car 5 properties and 3 methods.

Chioce2 (with inheritance)



Car

Vehicle

Color

Name Engine No

Brand

Owner Cngecolor()

Stop()

Start()

In this case the class car inherits some reusable methods and properties from vehicle class and hence it has to maintain 2 attributes and 1 method. Hence the class car looks simple.

**Corollaries:**

Corollaries are derived from design axioms(rules). These corollaries are suggestions to the designer to create a quality design.

They are

1. Corollary 1 Uncoupled Design with less information content (from Axiom1 and 2)
2. Corollary 2 Single purpose classes (from Axiom1 and 2)
3. Corollary 3 large number of simple classes (from Axiom1 and 2)
4. Corollary 4 Strong Maping (from Axiom 1)
5. Corollary 5 Standardization (from Axiom 2)
6. Corollary 6 Design with inheritance (from Axiom 2)

**Corollary 1 Uncoupled design with less information content.**

This corollary explains the concept of dependency by cohesion and coupling.

Cohesion is the dependency among the classes inside a component

Coupling the measure of dependency between 2 components.

Designers prefer design with

1. Low coupling between components
2. High cohesion among the classes inside a component.

Hence by reducing the strength of coupling between classes/ components reduces the complexity of the design.

Coupling It’s the measure of association established between 2 objects/ components. Designers prefer weak coupling among components because effect of change in one component should have less impact on the other component. The degree (strength) of coupling is the function of

1. How complicated the connection is?
2. Whether the connection refers to object itself or something inside the referred object
3. What message/ data is being send and received.

Interaction coupling exist between 2 objects if there is a message passing between those 2 objects.

The strength of interaction coupling depends on the complexity of messages passed between them.

Inheritance coupling exist between super and sub class. Inheritance coupling is desirable. Types of coupling

1. Content Coupling (Very High)
2. Common Coupling (High)
3. Control Coupling (Medium)
4. Stamp Coupling (Low)
5. Data Coupling (Very low)

Cohesion is the strength of dependency between classes with in a component. More cohesion reflects single purpose of the class. Designers prefer strong cohesion among contents of the component.

**Corollary 2 Single purpose**

Each class should have a single well defined purpose and the aim of the class is to full fill that responsibility. If the class aims at implementing multi purpose subdivide the class in to smaller classes.

**Corollary 3 Large number of simpler classes for reusability**

Complex classes are difficult to understand and hence for reuse needs more effort. Many times unnecessary members are reused as the super class was a complex one. The guideline says that “The

smaller are your classes, better your chances of reusing them in other projects. Large and complex classes are too specialized to be reused.”

**Corollary 4 Strong Mapping**

The designer, analyst and programmer should maintain strong dependency among the products obtained during the different phases of SDLC. Hence a class identified during the analysis is designed in

design phase and coded in implementation phase.

Designer should consider the programming language while creating design using technologies.

**Corollary 5 Standardization**

The Designer/ Programmer should be aware of the existing classes/ components available in the standard library. This knowledge will help the designer to reuse the existing classes and design the newly needed classes/ components. A class/ pattern repository is maintained to store all the reusable classes and

components. Even in most of the cases the repository is shared. Repository maintains the reusable components, description, commercial/ non commercial and usage.

**Corollary 6 Design with Inheritance**

[Note: Include inheritance , importance and need of inheritance, types with one example]

**3.Explain OCL – Object Constraint Language:?**

It’s a specification language used for representing properties of objects in an UML diagram.

It is English like language. The rules and semantics of the UML diagrams can be represented using

OCL. OCL specifications in UML diagrams make UML diagrams more clear and informative.

Sets, arithmetic expressions, Boolean expressions can be represented using OCL.

OCL Specifications:

1. Item. Selector

Selector Used to get the value of the attribute.

Item Entity to which the attribute belongs to.

E.g.

Stundent1. No = 30

Student1 is the Item and no is the selector.

2. Item. Selector [qualifier value]

Selector Used to identify a set of similar values.

Item Entity to which the attribute belongs to.

Qualifier specifies the particular value among the group.

E.g.

Student1. Mark [3]

Student1 is the Item, mark is the selector and 3 (qualifier) that represents 3rd mark

3. Boolean Expression

(Item1. Selector Boolean operation Item2.Selector)

E.g. S1. mark > S2. mark represents a Boolean value of true or false.

4. Set operation

Set select (Boolean expression) is used to select a group of objects that satisfies the Boolean expression.

Student select (mark >40) selects a list of students who has mark greater than 40.

5. Attribute specification

The OCL specification for specifying attributes is

Visibility attribute name : type

OR

Visibility attribute name : type = initial value

E.g.

* + Name : String
		- represents a public attribute Name of type String

# Name : String = “Hello”

* + - represents a protected attribute Name of type String with initial value Hello
1. Protocol Design Specification

The specification is

Visibility protocol name (argument list) : return type

Where argument list is arg1 : type, arg2 : type, arg3 : type … argn : type

E.g.

+ getName () : String

It’s a public protocol named getName with no parameters and it returns a value of type String.

- setData (name : String, no : Integer) : Boolean

It is a private protocol that accepts 2 arguments one of type String and other of type Integer. It returns a value of type Boolean.

7. OCL in representing function call



|  |  |
| --- | --- |
| Name = | # Student:: + getName() : String |
| getName() |
|  |

The activity diagram does not specify any details of getName where the OCL specification near the function call represents the clear idea about the method getName.

**4.Explain Designing View Layer Classes?**

View layer objects are more responsible for user interaction and these view layer objects have more relation with the user where business layer objects have less interaction with users. Another feature of view layer objects are they deal less with the logic. They help the users to complete their task in an easy manner.

The Major responsibilities of view layer objects are

1. Input – View Layer objects have to respond for user interaction. The

user interface is designed to translate an action by the user (Eg. Clicking the button) in to a corresponding message.

2. Output - Displaying or printing information after processing.

View Layer Design Process:

1. Macro Level UI Design Process
	1. Identify classes that interact with human actors
	2. A sequence/ collaboration diagram can be used to represent a clear picture of actor system interaction.
	3. For every class identified determine if the class interacts with the human actor. If so
		1. Identify the view layer object for that class.
		2. Define the relationship among view layer objects.
2. Micro Level UI Design Process
	1. Design of view layer objects by applying Design Axioms and Corollaries.
	2. Create prototype of the view layer interface.
3. Testing the usability and user satisfaction testing.
4. Iterate and refine the above steps.

**User Interface Design Rules:**

**UI Design Rule 1: Making the interface simple**

For complex application if the user interface is simple it is easy for the users to learn new applications. Each User Interface class should have a well define single purpose. If a user cannot sit before a screen and find out what to do next without asking multiple questions, then it says your interface is not simple.

**UI Design Rule 2: Making the Interface Transparent and Natural.**

The user interface should be natural that users can anticipate what to do next by applying previous knowledge of doing things with out a computer. This rule says there should be a strong mapping and users view of doing things.

**UI Design Rule 3: Allowing users to be in control of the Software.**

The UI should make the users feel they are in control of the software and not the software controls the user. The user should play an active role and not a reactive role in the sense user should initiate the action and not the software.

Some ways to make put users in control are

1. Make the interface forgiving.
2. Make the interface visual.
3. Provide immediate feedback.
4. Avoid Modes.

5.Make the interface consistent.

**UNIT-5**

**1.UML:**

The UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML gives you a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components.

**2.Model**

A model is a simplification of reality. A model provides the blueprints of a system. A model may be structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system.

**3.Why do we model**

We build models so that we can better understand the system we are developing.

Through modeling, we achieve four aims.

1. Models help us to visualize a system as it is or as we want it to be.
2. Models permit us to specify the structure or behavior of a system.
3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made.

We build models of complex systems because we cannot comprehend such a system in its entirety.

**4.Principles of Modeling**

There are four basic principles of model

1. The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.
2. Every model may be expressed at different levels of precision.
3. The best models are connected to reality.
4. No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.

**5 MARKS**

**1.EXPLAIN An Overview of UML?**

* + The Unified Modeling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system.
	+ The UML is appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems. It is a very expressive language, addressing all the views needed to develop and then deploy such systems.

The UML is a language for

* + - Visualizing
		- Specifying
		- Constructing
		- Documenting
* **Visualizing** The UML is more than just a bunch of graphical symbols. Rather,behind each symbol in the UML notation is a well-defined semantics. In this manner, one developer can write a model in the UML, and another developer, or even another tool, can interpret that model unambiguously
* **Specifying** means building models that are precise, unambiguous, andcomplete.
* **Constructing** the UML is not a visual programming language, but its modelscan be directly connected to a variety of programming languages
* **Documenting** a healthy software organization produces all sorts of artifacts inaddition to raw executable code. These artifacts include
1. Requirements o Architecture o Design

o Source code o Project plans o Tests

o Prototypes o Releases

To understand the UML, you need to form a **conceptual model of the language**, and this requires learning three major elements:

1. Things
2. Relationships

**2.Explain Things in the UML?**

There are four kinds of things in the UML: Structural things Behavioral things Grouping things Annotational things

**Structural things** are the nouns of UML models. These are the mostly staticparts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

1. Classes
2. Interfaces
3. Collaborations
4. Use cases
5. Active classes
6. Components
7. Nodes

**Class** is a description of a set of objects that share the same attributes,operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.



**Interface**

Interface is a collection of operations that specify a service of a class or component.

An interface therefore describes the externally visible behavior of that element.

An interface might represent the complete behavior of a class or component or only a part of that behavior.

An interface is rendered as a circle together with its name. An interface rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface



**Collaboration** defines an interaction and is a society of roles and other elementsthat work together to provide some cooperative behavior that's bigger than the sum of all the elements. Therefore, collaborations have structural, as well as behavioral, dimensions. A given class might participate in several collaborations.

Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name



**Usecase**

* Use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor
* Use case is used to structure the behavioral things in a model.
* A use case is realized by a collaboration. Graphically, a use case is rendered as an ellipse with solid lines, usually including only its name



**Active class** is just like a class except that its objects represent elements whosebehavior is concurrent with other elements. Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attributes, and operations



**Component** is a physical and replaceable part of a system that conforms to andprovides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs



**Node** is a physical element that existscomputational resource, generally having at processing capability. Graphically, a node including only its name



**Behavioral Things** are the dynamic parts of UML models. These are the verbs ofa model, representing behavior over time and space. In all, there are two primary kinds of behavioral things

Interaction

state machine

**Interaction**

Interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose

An interaction involves a number of other elements, including messages, action sequences and links

Graphically a message is rendered as a directed line, almost always including the name of its operation



**State Machine**

State machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events

State machine involves a number of other elements, including states, transitions, events and activities

Graphically, a state is rendered as a rounded rectangle, usually including its name and its substates



**Grouping Things:-**

1. are the organizational parts of UML models. These are the boxes into which a model can be decomposed
2. There is one primary kind of grouping thing, namely, packages.

**Package:-**

* A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package
* Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents

**Annotational things** are the explanatory parts of UML models. These are thecomments you may apply to describe about any element in a model.

**A note** is simply a symbol for rendering constraints and comments attachedto an element or a collection of elements.

Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment



**3.Write Relationships in the UML**: There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

**Dependency:-**

Dependency is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing

Graphically a dependency is rendered as a dashed line, possibly directed, and occasionally including a label



**Association** is a structural relationship that describes a set of links, a link beinga connection among objects.

Graphically an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names



**Generalization** is a special kind of association, representing a structuralrelationship between a whole and its parts. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent



**Realization** is a semantic relationship between classifiers, wherein one classifierspecifies a contract that another classifier guarantees to carry out. Graphically a realization relationship is rendered as a cross between a generalization and a dependency relationship



**4.Explain the Rules of the UML?**

The UML has semantic rules for

|  |  |  |
| --- | --- | --- |
| 1. | Names | What you can call things, relationships, and diagrams |
| 2. | Scope | The context that gives specific meaning to a name |
| 3. | Visibility | How those names can be seen and used by others |
| 4. | Integrity | How things properly and consistently relate to one |
|  | another |  |
| 5. | Execution | What it means to run or simulate a dynamic model |

Models built during the development of a software-intensive system tend to evolve and may be viewed by many stakeholders in different ways and at different times. For this reason, it is common for the development team to not only build models that are well-formed, but also to build models that are

1. Elided Certain elements are hidden to simplify the view
2. Incomplete Certain elements may be missing
3. Inconsistent The integrity of the model is not guaranteed

**5.Explain Common Mechanisms in the UML**

UML is made simpler by the presence of four common mechanisms that apply consistently throughout the language.

1. Specifications
2. Adornments
3. Common divisions
4. Extensibility mechanisms

**Specification** that provides a textual statement of the syntax and semantics ofthat building block. The UML's specifications provide a semantic backplane that

contains all the parts of all the models of a system, each part related to one another in a consistent fashion

**Adornments** Most elements in the UML have a unique and direct graphicalnotation that provides a visual representation of the most important aspects of the element. A class's specification may include other details, such as whether it is abstract or the visibility of its attributes and operations. Many of these details can be rendered as graphical or textual adornments to the class's basic rectangular notation.

**Extensibility Mechanisms**

The UML's extensibility mechanisms include

1. Stereotypes
2. Tagged values
3. Constraints

**Stereotype**

* Stereotype extends the vocabulary of the UML, allowing you to create new kinds of building blocks that are derived from existing ones but that are specific to your problem
* A tagged value extends the properties of a UML building block, allowing you to create new information in that element's specification

A constraint extends the semantics of a UML building block, allowing you to add new rules or modify existing ones

**10 marks:**

**1.Explain Class in UML?**

* A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.
* A class implements one or more interfaces.
* The UML provides a graphical representation of class



**Graphical Representation of Class in UML Terms and Concepts**

**Names**

Every class must have a name that distinguishes it from other classes. A name is a textual string that name alone is known as a simple name; a path name is the class name prefixed by the name of the package in

which that class lives.



**Simple Name** **Path Name**

**Attributes**

* An attribute is a named property of a class that describes a range of values that instances of the property may hold.
* A class may have any number of attributes or no attributes at all.
* An attribute represents some property of thing you are modeling that is shared by all objects of that class
* You can further specify an attribute by stating its class and possibly a default initial value

* A class may have any number of operations or no operations at all
* Graphically, operations are listed in a compartment just below the class attributes
* You can specify an operation by stating its signature, covering the name, type, and default value of all parameters and a return type

**esponsibilities**

* A Responsibility is a contract or an obligation of a class
* When you model classes, a good starting point is to specify the responsibilities of the things in your vocabulary.
* A class may have any number of responsibilities, although, in practice, every well-structured class has at least one responsibility and at most just a handful.
* Graphically, responsibilities can be drawn in a separate compartment at the bottom of the class icon



**Common Modeling Techniques**

**Modeling the Vocabulary of a System**

* You'll use classes most commonly to model abstractions that are drawn from the problem you are trying to solve or from the technology you are using to implement a solution to that problem.
* They represent the things that are important to users and to implementers

* To model the vocabulary of a system
1. Identify those things that users or implementers use to describe the problem or solution.
2. Use CRC cards and use case-based analysis to help find these abstractions.
3. For each abstraction, identify a set of responsibilities.
4. Provide the attributes and operations that are needed to carry out these responsibilities for each

class.

**Modeling the Distribution of Responsibilities in a System**

* Once you start modeling more than just a handful of classes, you will want to be sure that your abstractions provide a balanced set of responsibilities.
* To model the distribution of responsibilities in a system
1. Identify a set of classes that work together closely to carry out some

behavior.

1. Identify a set of responsibilities for each of these classes.
2. Look at this set of classes as a whole, split classes that have too many

responsibilities into

smaller abstractions, collapse tiny classes that have trivial responsibilities into larger ones, and

reallocate responsibilities so that each abstraction reasonably stands on its own.

1. Consider the ways in which those classes collaborate with one another, and redistribute their

responsibilities accordingly so that no class within a collaboration does

too much or too little.

**Modeling Nonsoftware Things**

* Sometimes, the things you model may never have an analog in software
* Your application might not have any software that represents them
* To model nonsoftware things
1. Model the thing you are abstracting as a class.
2. If you want to distinguish these things from the UML's defined building blocks, create a new

building block by using stereotypes to specify these new semantics and to give a distinctive

visual cue.

1. If the thing you are modeling is some kind of hardware that itself contains software, consider

modeling it as a kind of node, as well, so that you can further expand on its structure.

**Modeling Primitive Types**

.

Typically, these abstractions involve primitive types, such as integers, characters, strings, and even enumeration types

To model primitive types

Model the thing you are abstracting as a type or an enumeration, which is rendered using class

notation with the appropriate stereotype.

If you need to specify the range of values associated with this type, use constraints.

**Relationships**

In the UML, the ways that things can connect to one another, either logically or physically, are modeled as relationships.

Graphically, a relationship is rendered as a path, with different kinds of lines used to distinguish the kinds of relationships

In object-oriented modeling, there are three kinds of relationships that are most important:

Dependencies

Generalizations

Associations

**Dependency**

A dependency is a using relationship that states that a change in

specification of one thing may affect another thing that uses it but not necessarily the reverse.

Graphically dependency is rendered as a dashed directed line, directed to the thing being depended on.

Most often, you will use dependencies in the context of classes to show that one class uses another class as an argument in the signature of an operation

**Generalization**

1. A generalization is a relationship between a general thing (called the super

class or parent)and a more specific kind of that thing (called the subclass or child).

1. generalization means that the child is substitutable for the parent. A child inherits the properties of its parents, especially their attributes and operations



**Generalization**

**Association**

1. An association is a structural relationship that specifies that objects of one thing are connected to objects of another
2. An association that connects exactly two classes is called a binary association o An associations that connect more than two classes; these are called n-ary

associations.

o Graphically, an association is rendered as a solid line connecting the same or different classes.

o Beyond this basic form, there are four adornments that apply to association

**Name**

1. An association can have a name, and you use that name to describe the nature of the relationship



**Association Names**

**Role**

1. When a class participates in an association, it has a specific role that it plays in that relationship;
2. The same class can play the same or different roles in other associations. o An instance of an association is called a link



**Role Names**

**Multiplicity**

1. In many modeling situations, it's important for you to state how many objects

may be connected across an instance of an association

1. This "how many" is called the multiplicity of an association's role
2. You can show a multiplicity of exactly one (1), zero or one (0..1), many (0..\*), or one or more (1..\*). You can even state an exact number (for example, 3).



**Multiplicity**

**Aggregation**

1. Sometimes, you will want to model a "whole/part" relationship, in which one class represents a larger thing (the "whole"), which consists of smaller things (the "parts").
2. This kind of relationship is called aggregation, which represents a "has-a" relationship, meaning that an object of the whole has objects of the part
3. Aggregation is really just a special kind of association and is specified by adorning a plain association with an open diamond at the whole end



**Aggregation**

**Common Modeling Techniques**

**Modeling Simple Dependencies**

To model this using relationship

Create a dependency pointing from the class with the operation to the class used as a parameter

in the operation.

The following figure shows a set of classes drawn from a system that manages the assignment of students and instructors to courses in a university.

This figure shows a dependency from CourseSchedule to Course, because Course is used in both the add and remove operations of CourseSchedule.

The dependency from Iterator shows that the Iterator uses the CourseSchedule; the CourseSchedule knows nothing about the Iterator. The dependency is marked with a stereotype, which specifies that this is not a plain dependency, but, rather, it represents a friend, as in C++.