SOFTWARE ENGINEERING

Week 7

Software Architecture and High Level Design

Agenda

1. Architectural Design of Software
2. Unified Modeling Language
3. Model Driven Engineering

1. Architectural Design of Software
2. Unified Modeling Language
3. Model Driven Engineering

Architectural Design of Software

7.1

Software architecture

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design.
- The output of this design process is a description of the software architecture.

Architectural design

- An early stage of the system design process.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

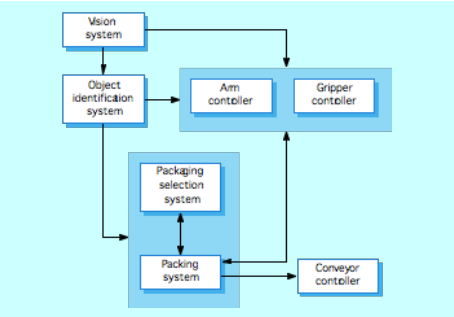
Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.

Architecture and system characteristics

- Performance
  - Localise critical operations and minimise communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localise safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Packing robot control system



Box and line diagrams

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

Architectural styles

- The architectural model of a system may conform to a generic architectural model or style.
- An awareness of these styles can simplify the problem of defining system architectures.
- However, most large systems are heterogeneous and do not follow a single architectural style.

Architectural models

- Used to document an architectural design.
- Static structural model that shows the major system components.
- Dynamic process model that shows the process structure of the system.
- Interface model that defines sub-system interfaces.
- Relationships model such as a data-flow model that shows sub-system relationships.
- Distribution model that shows how sub-systems are distributed across computers.

Architectural models

Architectural models can be examined in two levels

- Organizational Decomposition Models
- Modular (Sub-System) Decomposition Models

1. Architectural Design of Software
2. Unified Modeling Language
3. Model Driven Engineering

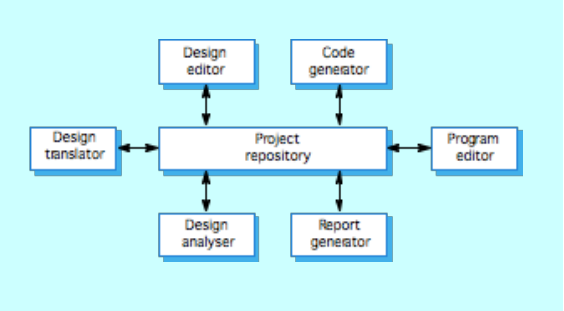
Organizational Decomposition Models

7.1.1

The repository model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems: ***the repository model***;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

CASE toolset architecture



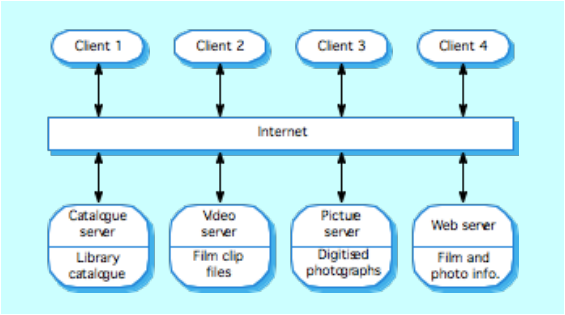
Repository model characteristics

- Advantages
  - Efficient way to share large amounts of data;
  - Sub-systems need not be concerned with how data is produced;
  - Centralised management e.g. backup, security, etc.
  - Sharing model is published as the repository schema: ***easy integration***;
- Disadvantages
  - Sub-systems must agree on a repository data model. Inevitably a compromise;
  - Data evolution is difficult and expensive: ***e.g., changing the data model is expensive or even impossible***;
  - No scope for specific management policies: ***sub-systems may have different requirements for security, backup, etc. ...***
  - Difficult to distribute efficiently.

Client-server model

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone ***servers*** which provide specific services such as printing, data management, etc.
- Set of ***clients*** which call on these services.
- ***Network*** which allows clients to access servers.

Film and picture library



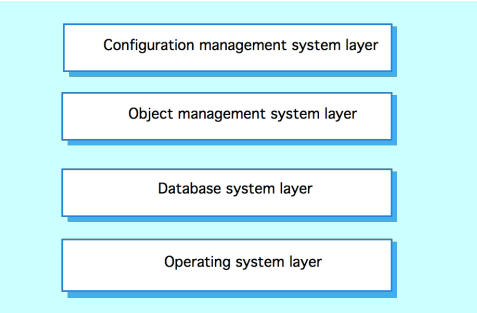
Client-server characteristics

- Advantages
  - Distribution of data is straightforward;
  - Makes effective use of networked systems. May require cheaper hardware;
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
  - Redundant management in each server;
  - No central register of names and services - it may be hard to find out what servers and services are available.

Layered model

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

Version management system



1. Architectural Design of Software
2. Unified Modeling Language
3. Model Driven Engineering

Modular Decomposition Models

7.1.2

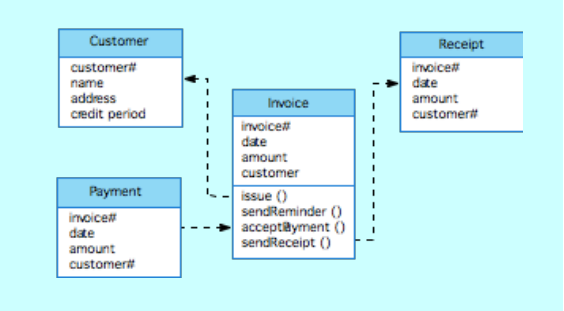
Modular decomposition

- A ***sub-system*** is a system in its own right whose operation is independent of the services provided by other sub-systems.
- A ***module*** is a system component that provides services to other components but would not normally be considered as a separate system.

Object models

- Structure the system into a set of loosely coupled objects with well-defined interfaces.
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- When implemented, objects are created from these classes and some control model used to coordinate object operations.

Invoice processing system



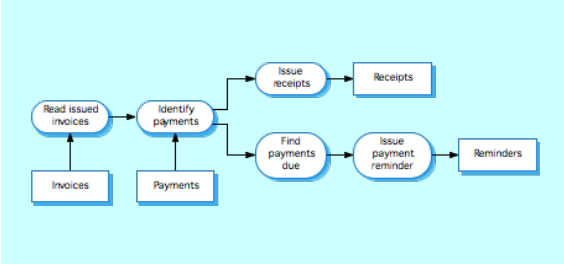
Object model advantages

- Objects are loosely coupled so their implementation can be modified without affecting other objects.
- The objects may reflect real-world entities.
- OO implementation languages are widely used.
- However, object interface changes may cause problems and complex entities may be hard to represent as objects.

Function-oriented pipelining

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in UNIX shell).
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- Not really suitable for interactive systems.

Invoice processing system



Pipeline model advantages

- Supports transformation reuse.
- Intuitive organisation for stakeholder communication.
- Easy to add new transformations.
- Relatively simple to implement as either a concurrent or sequential system.
- However, requires a common format for data transfer along the pipeline and difficult to support event-based interaction.

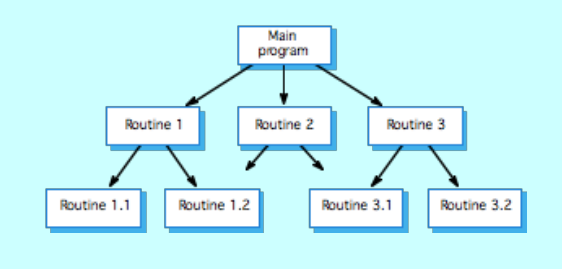
Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.
- Centralised control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems.
- Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

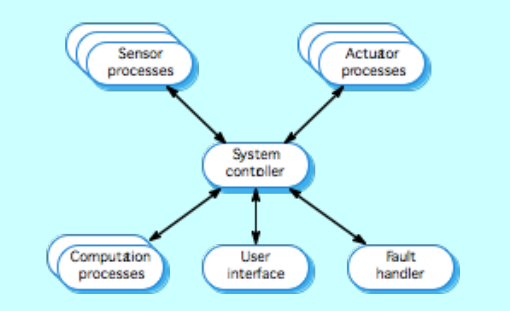
Centralised control

- A control sub-system takes responsibility for managing the execution of other sub-systems.
- Call-return model
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems.
- Manager model
  - Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can also be implemented in sequential systems as a case statement:
    - a management routine calls particular the subsystem depending on the value of some state variables

Call-return model



Real-time system control



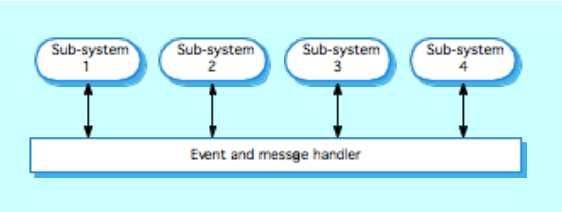
Event-driven systems

- Driven by externally generated events where the timing of the event is outwith the control of the sub-systems which process the event.
- Two principal event-driven models
  - Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so;
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.
- Other event driven models include spreadsheets and production systems.

Broadcast model

- Effective in integrating sub-systems on different computers in a network.
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event.
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them.
- However, sub-systems don't know if or when an event will be handled.

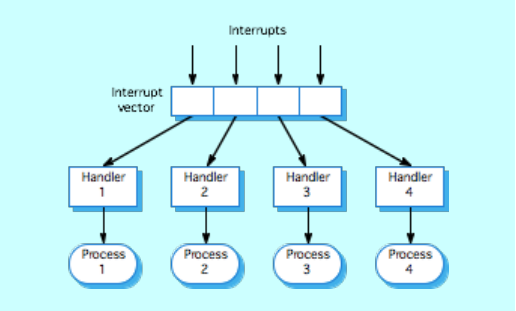
Selective broadcasting



Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential.
- There are known interrupt types with a handler defined for each type.
- Each type is associated with a memory location and a hardware switch causes transfer to its handler.
- Allows fast response but complex to program and difficult to validate.

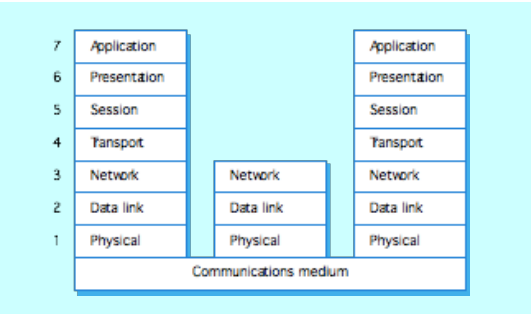
Interrupt-driven control



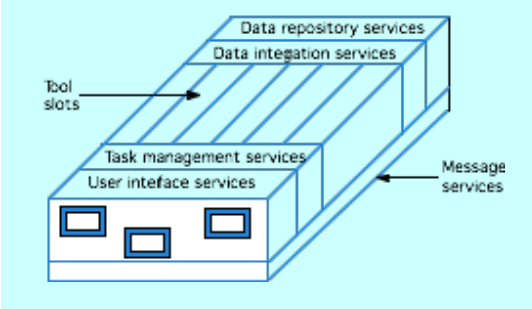
Reference architectures

- Reference models are derived from a study of the application domain rather than from existing systems.
- May be used as a basis for system implementation or to compare different systems. It acts as a standard against which systems can be evaluated.
- OSI model is a layered model for communication systems.

OSI reference model



The ECMA reference model



1. Architectural Design of Software
2. Unified Modeling Language
3. Model Driven Engineering

Unified Modeling Language

7.2

Analysis

What is UML?

- Unified Modeling Language (UML) is the standard tool for visualizing, specifying, constructing, and documenting the artifacts of an object-oriented software.
- UML is not a programming language, but only a visual design notation.
- Can be used with all software development process models.
- Independent of implementation language.
- Many CASE tools uses UML for automatic code generation. Examples: IBM Rational, ArgoUML, etc.
- You may be familiar with some UML concepts introduced in Object Oriented Programming course.

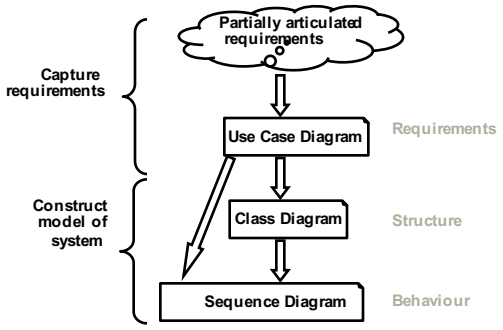
Background

- UML is the result of an effort to simplify and consolidate the large number of **Object Oriented** development methods and notations.
- Developed by the Object Management Group based on work from:
  - Grady Booch [91]
  - James Rumbaugh [91]
  - Ivar Jacobson [92]
- The latest version is UML 2.0 (See http://www.omg.org or http://www.uml.org)

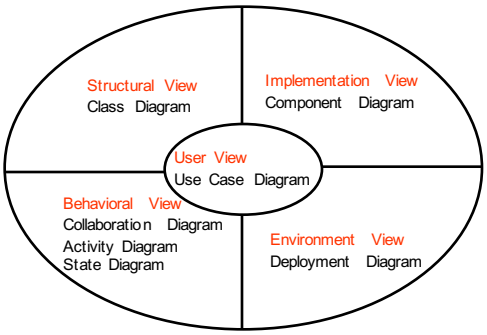
Types of UML Diagrams

- **Use Case Diagrams (*)**
- **Class Diagrams (*)**
- Interaction Diagrams
  - **Sequence Diagrams (*)**
  - Collaboration Diagrams
- State Transition Diagrams
- Activity Diagrams
- Implementation Diagrams
  - Component Diagrams
  - Deployment Diagrams

Minimal UML Process



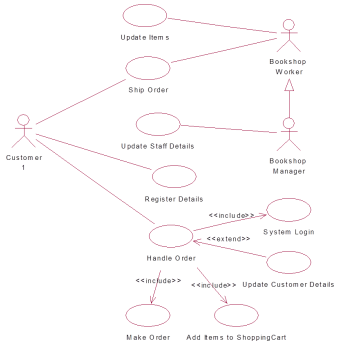
Views of UML Diagrams



Use Cases

- Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.
- Provide a way for developers, domain experts and end-users to Communicate.
- Serve as basis for testing.
- Main authors: *Booch, Rumbaugh, and Jacobson*
- The Object Management Group (OMG) is responsible for standardization. (www.omg.org)
- Current version is UML 2.0

Example : Use Case Diagram



Example : Use Case (Money Withdraw) - I

- **Use Case:** Withdraw Money
- **Author:** ZB
- **Date:** 1-OCT-2004
- **Purpose:** To withdraw some cash from user's bank account
- **Overview:** The use case starts when the customer inserts his credit card into the system. The system requests the user PIN. The system validates the PIN. If the validation succeeded, the customer can choose the withdraw operation else alternative 1 – validation failure is executed. The customer enters the amount of cash to withdraw. The system checks the amount of cash in the user account, its credit limit. If the withdraw amount in the range between the current amount + credit limit the system dispense the cash and prints a withdraw receipt, else alternative 2 – amount exceeded is executed.
- **Cross References:** R1.1, R1.2, R7

Example : Use Case (Money Withdraw) - II

- **Actors:** Customer
- **Pre Condition:**
  - The ATM must be in a state ready to accept transactions
  - The ATM must have at least some cash on hand that it can dispense
  - The ATM must have enough paper to print a receipt for at least one transaction
- **Post Condition:**
  - The current amount of cash in the user account is the amount before the withdraw minus the withdraw amount
  - A receipt was printed on the withdraw amount
  - The withdraw transaction was audit in the System log file

Example : Use Case (Money Withdraw) - III

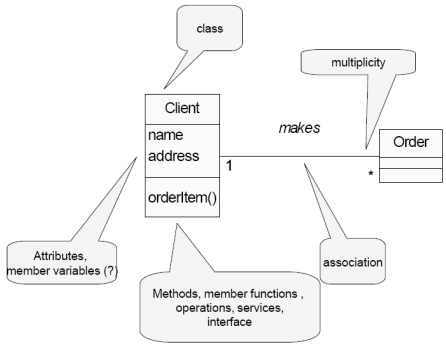
Typical Course of events:

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deduces the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

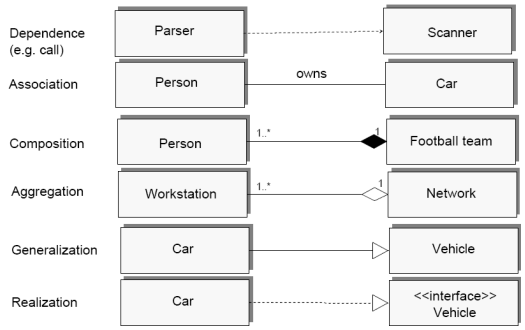
Example : Use Case (Money Withdraw) - IV

- **Alternative flow of events:**
  - Step 3: Customer authorization failed. Display an error message, cancel the transaction and eject the card.
  - Step 8: Customer has insufficient funds in its account. Display an error message, and go to step 6.
  - Step 8: Customer exceeds its legal amount. Display an error message, and go to step 6.
- **Exceptional flow of events:**
  - Power failure in the process of the transaction before step 9, cancel the transaction and eject the card

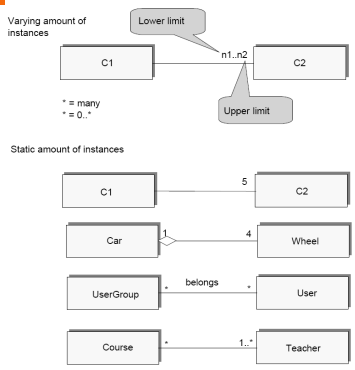
Class Diagrams



Class Relations

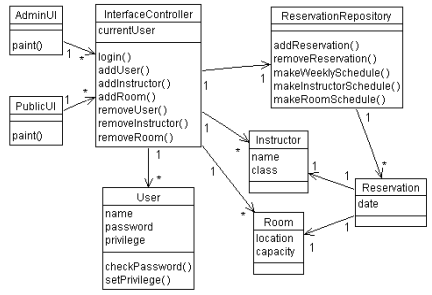


Cardinality

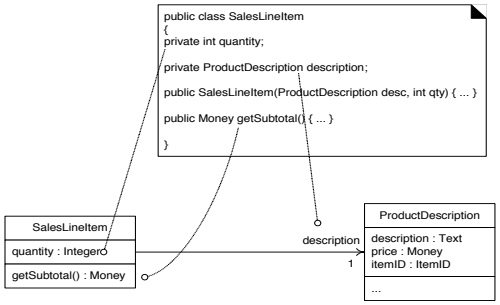


Example : Class Diagram

A classroom scheduling system: specification perspective.



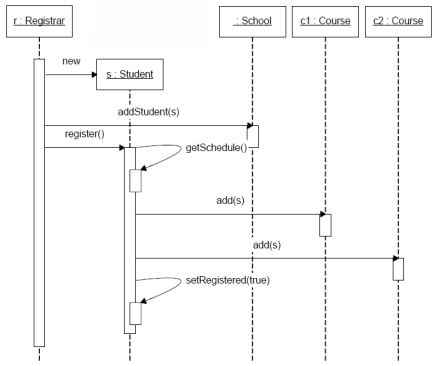
Example : From Class Diagram to Code



Interaction Diagrams

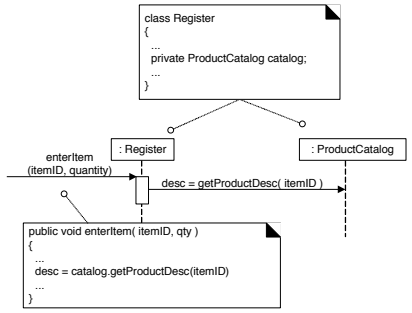
- UML presents two types of interaction diagrams.
  1. Sequence Diagrams
  2. Collaboration Diagrams

Sequence Digrams

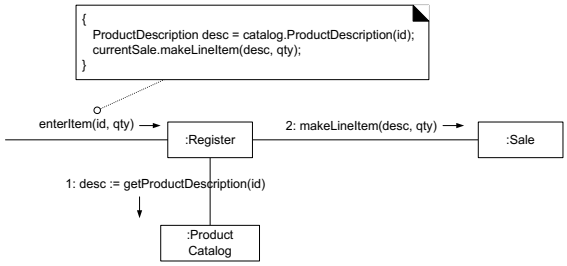


Example : Mapping Diagram to Code

- Mapping sequence diagram to Java code

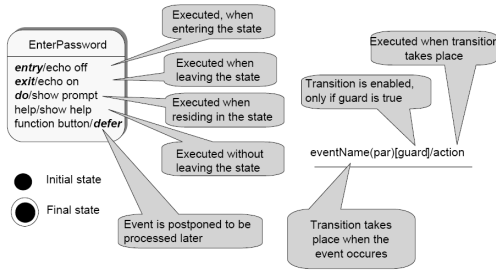


Collaboration Diagrams

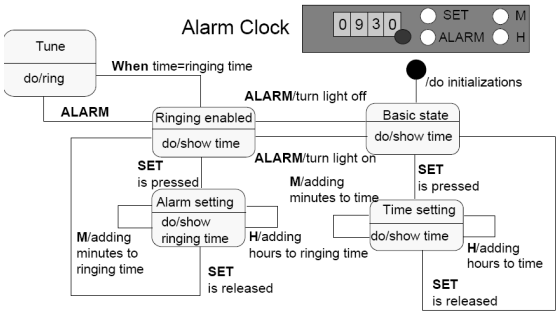


State Chart Notation

- State transition diagram shows
  - The events that cause a transition from one state to another
  - The actions that result from a state change

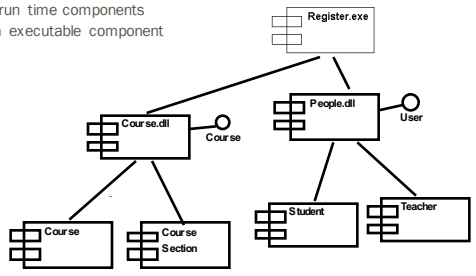


Example : State Chart Diagram



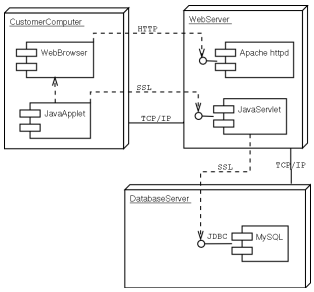
Package Diagrams

- Component diagrams illustrate the organizations and dependencies among software components in physical world.
- A component may be
  - A source code component
  - A run time components
  - An executable component



Deployment Diagrams

- Captures the distinct number of computers involved
- Shows the communication modes employed
- Component diagrams can be embedded into deployment diagrams effectively



1. Architectural Design of Software
2. Unified Modeling Language
3. Model Driven Engineering

Model Driven Engineering

7.3

Model Driven Engineering

- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

Usage of Model-Driven Engineering

- Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.
- Pros
  - Allows systems to be considered at higher levels of abstraction
  - Generating code automatically means that it is cheaper to adapt systems to new platforms.
- Cons
  - Models for abstraction and not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.

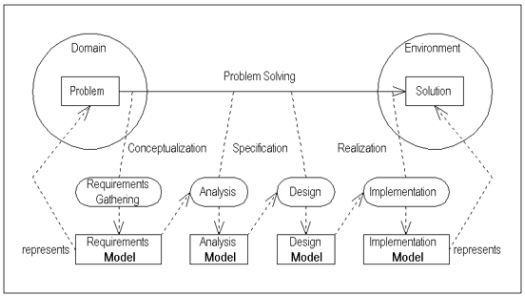
Model Driven Architecture

- Model-driven architecture (MDA) was the precursor of more general model-driven engineering
- MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.
- Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

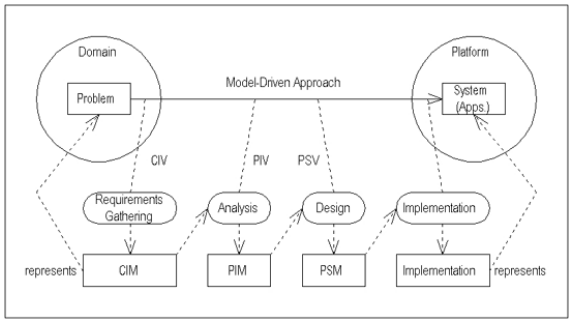
Types of Model

- A computation independent model (CIM)
  - These model the important domain abstractions used in a system. CIMs are sometimes called domain models.
- A platform independent model (PIM)
  - These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- *Platform specific models (PSM)*
  - These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

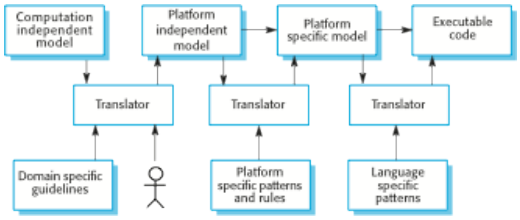
Systems Development Lifecycle Process



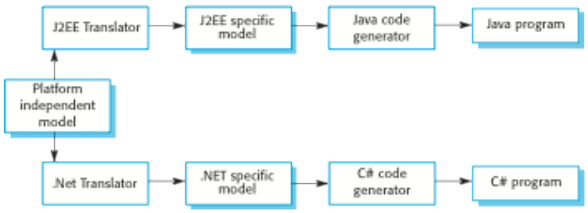
Foundational Concepts of the MDA



MDA Transformations



Multiple Platform-Specific Models



Executable UML

- The fundamental notion behind model-driven engineering is that completely automated transformation of models to code should be possible.
- This is possible using a subset of UML 2, called Executable UML or xUML.

Features of executable UML

- To create an executable subset of UML, the number of model types has therefore been dramatically reduced to these 3 key types:
  - Domain models that identify the principal concerns in a system. They are defined using UML class diagrams and include objects, attributes and associations.
  - Class models in which classes are defined, along with their attributes and operations.
  - State models in which a state diagram is associated with each class and is used to describe the life cycle of the class.
- The dynamic behaviour of the system may be specified declaratively using the object constraint language (OCL), or may be expressed using UML's action language.

Wrap-up

This week we present

- Architecrual Models
  - Different architectural models may be produced during the design process
  - Each model presents different perspectives on the architecture
- UML and MDE
  - UML is a notation, not a methodology. It can be used in conjunction with any methodology
  - Every UML diagram consists of a small required part plus any number of options
  - Not every feature of UML is applicable to every information system
  - To perform iteration and incrementation, features have to be added stepwise to diagrams

Next Week

- We will be covering *Software Design Engineering*!!!