Conceptual Model of the UML

1. **The UML’s basic building blocks**
2. **The rules that dictate how the building blocks may be put together**
3. **Some common mechanisms**

Building Blocks of UML- **Things, Relationship and Diagrams**

1. **Things – Structural Things, Behavioral Things, Grouping Things and Annotational Things**
   1. **Structural Things / Classifiers**

**Nouns of the UML models.**

**Static part of model, representing elements that are either conceptual or physical.**

1. **Class**

**Set of objects sharing same attributes, operations, relationship and semantics**

1. **Interface**

**Collection of operations that specify a service of a class or component**

IPaint

Window

Window

IWindow

Window

1. **Collaboration**
2. **Use Case**
3. **Active Class**

|  |  |  |
| --- | --- | --- |
|  | **Event Manager** |  |
|  |
| **Operation1()**  **Operation2()** |

1. **Component**
2. **Artifacts**

|  |
| --- |
| **<<artifact>>**  **Window.dll** |

1. **Nodes**

Server

* 1. **Behavioral Things**

**Verbs of a model**

* + 1. **Interaction**

display

* + 1. **State machine**

Waiting

* + 1. **Activity**

Waiting

* 1. **Grouping Things**
     1. **Package**

Business Rule Package

* 1. **Annotational Things**

Return copy to Self

1. **Relationship**
2. **Dependency**
3. **Association**

\*

0..1

**Employer Employee**

1. **Generalization**
2. **Realization**
3. **Diagrams**

**Rules of UML**

* **Names**
* **Scope**
* **Visibility**
* **Integrity**
* **Execution**

Common Mechanisms in the UML

1. **Specifications**
2. **Adornments**
3. **Common Division**
4. **Extensibility Mechanisms**
5. Stereotypes
6. Tagged Values
7. Constraints

|  |
| --- |
| <<authored>>  EventQueue |
|  |
| Add()  Remove()  Flush() |

|  |
| --- |
| <<authored>>  Version=3.2  Author =cmp |

**Classifier – classes, associations, interfaces, datatypes, signals, components, nodes, use cases and subsystems.**

**Visibility**

public (+), private(-), protected(#), package(~)

**Scope** – Instance, Static

**Attributes**

[*visibility*]name[':' type]['['multiplicity] ']']['=' initial-value][property-string {',', property string}]

+origin: Point Name

name: String[0...1] Name, Type and Multiplicity

origin: Point={0, 0} Name, Type and Initial Value

id: Integer {readonly} Name and Property

**Operations**

[visibility]name['(' parameter-list ')'] [':' return-type] [property-string{',' property-string}]

Display Name

+display Visibility and Name

set(n: Name, s: String) Name and Parameters

getID(): Integer Name and Return Type

restart() {gaurded} Name and Property

query, sequential, gaurded, concurrent, static

Generalization

1. complete

2. incomplete

3. disjoint

4. overlapping

Dependancies

* 1. bind – Specifies source instantiate the target template using the given actual parameter
  2. derive – Specifies that the source may be computed from the target
  3. permit – Specifies that the source is given special visibility into the target,

**Practical No. 2**

Event – State Diagram

* Event properties and types of events
* States – A state is an abstraction of the attribute values and links of an object. Set of values are grouped together into a state according to properties that affect the gross behavior of the object.
  1. It can also specify response of the object to the input events.
  2. **A state corresponds to the interval between two events of an objects.** Events represents points in time; states represents intervals of time.
  3. A state may depends on past event also
  4. A state has a duration, it occupies an interval of time.
  5. A state is often associated with a continuous activity
  6. A state – an activity that takes time to complete
  7. A state is often associated with the value of an object satisfying some condition.

Scenarios and Event Traces

**Scenario for phone call**

Caller lifts receiver

Dial tone begins

Caller dials digits (5)

Dial tone ends

….

Ringing tone appears in calling phone

…

Caller hang up

Identify sender and receiver objects of each events

**Event Trace Diagram**

Caller Phone Line Callee

Caller lifts receiver

Dial tone begins

**State Diagrams**

Draw a event state diagram for landline telephone

**Conditions**

A condition is a Boolean function of object values.

“the temperature is below freezing”

A condition is valid over an interval of time.

e.g. the temperature was below freezing from Nov 15 to Nov 16.

Conditions can be used as gaurds on transition

Nesting State Diagrams

An activity in a state can be expanded as a lower level state diagram, each state representing one step of the activity.

Nested state diagram are one-shot state diagram with input and ouput transitions, similar to subroutines.

Concurrency

Interaction Diagram

* Sequence Diagram

DEWEY decimal numbering systems

Communication Diagrams

C: Client

1: create()

2: setActions (a, d, 0)

3: destroy()

Proxy(global)

P1: ODBDProxy

: Transaction

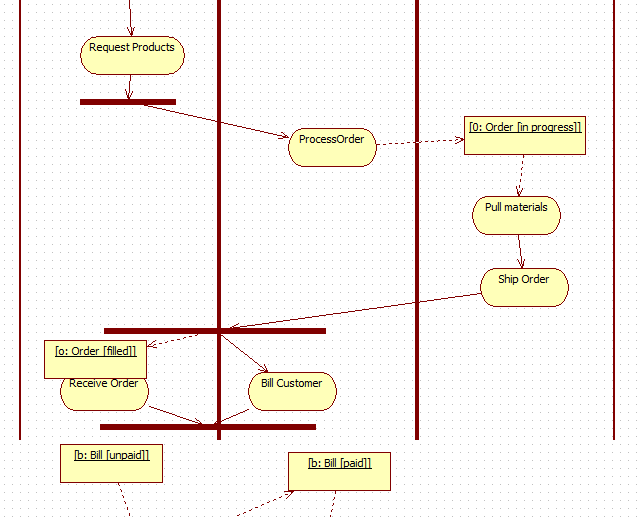
2.1: setValue (d, 3, 4)

2.2: setValue(a, “CO”)

Activity Diagrams:

Content:

1. Actions
2. Activity Nodes
3. Flows – Transition, Branching (Decision Flow), Forking and Joining, Swimlanes
4. Object Values (Object Flows, Expansion Regions)



Get Item

Compute Cost

Ship Order

Send Bill

:Order

Receive Order

:Product

Collection Input

:Money

:Shipment

:Bill

Collection Type

Collection Output

:LineItem

:Scalar Type

Modeling a Workflow:

1. Establish a focus for the workflow. For nontrivial systems, it’s impossible to show all interesting workflows in one diagram.
2. Select the **business objects** that have the high level responsibilities for parts of the overall workflow. These may be real things from the vocabulary of the system or they may be abstract. In either case, create a swimlane for each important business object or organization.
3. Identify the preconditions of the workflow’s initial state and the post conditions of the workflow final state. This is important in helping you model the boundaries of the workflow.
4. Beginning at the workflows’ initial state, specify the actions that take place over time and render them in activity diagram.
5. For complicated actions or for set of actions that appear multiple times, collapse these into calls to separate activity diagram.
6. Render the flows that connect these actions and activity nodes. Start with the sequential flows in the workflow first, next consider branching, and only then consider forking and joining.
7. If there are important object values that are involved in the workflow, render them in activity diagram as well. Show their changing values and state as necessary to communicate intent of the object flow.

**Component Diagram**

1. Interface – is a collection of operations that specify a service that is provided or requested from a class or component.
2. Port – is a specific window into an encapsulated component accepting messages to and from the component conforming to specified interfaces
3. Internal Structure – is the implementation of a component by means of a set of parts that are connected together in a specific way.
4. A part is the specification of a role that composes part of the implementation of a component. In an instance of the component, there is an instance corresponding to the part.
5. A connector is a communication relationship between two parts or ports within the context of the component.

Required Interface – The interface that a component uses is called required interface.

Provided Interface – Interface that component realizes

**Motion**

|  |
| --- |
| <<interface>>  imageObserver |
|  |
| imageUpdate(): Boolean |

Replaceability

**Lex: Lexical Analyzer**



**Parse:Parser**



**Gen: Code Genera..**



**Opt:Optimize[1..3]**

Compiler

Compile

`



Ticket sales

Booking



Normal sales

Component declaration

Ticket Seller

ports

Load Attractions

Ticket sales

Priority sales

Credit Cards

PORTS



Catalog Sales



:Inventory



:Order Taking



:OrderHandling

:Fulfillment

Charging: Credit

:OrderEntry

Direct connector

Connector by interface

Delegation connector

Package Diagram

Enclosing package name: sensors

Package name: vision

Qualified name: Sensors::Vision

A package forms a namespace, which means elements of the same kind must be named uniquely within the context of the enclosing package. Vision1::OrderForms; Vision2::OrderForms



+TrackingForms

+OrderForms

-Order

-Order

+TrackingForms

+OrderForms

Package may own other packages.

Sensors::Vision::Camera

Importing and Exporting Packages



exports

<<import>>

-LoggingService

-Database

-Order

+OrderForms

+TrackingForms



<<import>>

+Form

#EventHandler

+Window

-GUI::Window

+OrderRules

To model groups of elements,

1. Scan the modeling elements in a particular architectural view and look for clumps defined by elements that are conceptually or semantically close to one another.
2. Surround each of these clumps in a package.
3. For each package, distinguish which elements should be accessible outside the package. Mark them public, and all other as protected or private. When in doubt, hide the element.
4. Explicitly connect packages that build on others via import dependencies
5. In the case of families of packages, connect specialized packages to their more general part via generalizations.

**Deployment Diagram**

1. Names

Simple Name / Qualified Names



{remoteAdministrator Only}

Deploys

Pos.exe

Contacts.exe

Nodes and Artifacts



<<manifest>>

<<manifest>>

<<artifact>>

Contacts.exe

<<artifact>>

Pos.exe



Contract

Sale

1. Modeling and Embedded Systems
2. Identify the devices and nodes that are unique to your system
3. Provide visual cues, for unusual devices, by using the UML’s extensibility mechanisms to define specific stereotypes with appropriate icons.
4. Model the relationship among these processors and devices in a deployment diagram. Similarly, Specify the relationship between the artifacts in your system’s implementation view and the nodes in your system’s deployment view.
5. As necessary, expand on the intelligent devices by modeling their structure with a more detailed deployment diagram.

Timer



Left position encoder

Serial I/O port

<<RS232>>

1. Client Server Systems
2. Distributed Systems