Outline

- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study

Bibliography

- "UML distilled – Applying the standard object modeling language", Martin Fowler with Kendall Scott, Addison Wesley Longman Inc. 1997
Outline

- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study

UML meaning

**UNIFIED MODELING LANGUAGE**

**UML**: What is it?

The UML is a graphical language for specifying, visualizing, constructing and documenting models of software systems, as well as for business modeling and other non-software systems.
UML: What is it?

- UML is a modeling language, NOT a method.
- Most methods consist of a modeling language and a process.
- The modeling language is the notation that methods use to express design.
- The process is their advice on what steps to take in doing a design.

Outline

- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study

UML benefits overview

- It lowers costs of training and retooling when changing between projects and organizations.
- It provides new integration opportunity between tools, processes and domains.
- It enables developers to focus on delivering business value and provides them a paradigm to accomplish this.
UML benefits in Embedded Systems Design

- It is technology independent.
- It is Hardware/Software partitioning independent.
- It allows specification of both functional and non-functional requirements and constraints.
- It allows highly modular systems design.

Outline

- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study

Diagram Overview

- Structural diagrams
  - Used to visualize, specify, construct, document static system aspects

- Behavioural diagrams
  - Used to visualize, specify, construct, document dynamic system aspects
Diagram Overview

- Structural diagrams
  - Used to visualize, specify, construct, document static system aspects
- Behavioural diagrams
  - Used to visualize, specify, construct, document dynamic system aspects

Structural Diagram

- Class diagrams
- Package diagrams
- Object diagrams
- Component diagrams
- Deployment diagrams

Class: definition

- A **class** is a description of a set of objects that share the same attributes, operations, relationships and semantics.
- An **attribute** is a named property of a class that describes a range of values that instances of the property may hold.
- An **operation** is a service that can be requested from an object to affect behaviour.
### Class: notation

- **Name**
- **Attributes**
- **Operations**

### Relations

- Connections between classes:
  - Dependency
  - Generalization
  - Association

### Dependency

- A dependency exists between two elements if changes to the definition of one element may cause changes to the other.

**Example:** One class mentions another as a parameter to an operation.

```
Window
handleEvent(Event)
```
Generalization

A generalization is a relationship between a general thing (superclass or parent) and a more specific thing (subclass or child).

Shape

- Circle
- Rectangle

Association

An association is a relationship between instances of classes or between objects.

Example: A person works for a company
(An association between objects is called a link.)

Person  Company

Association Adornments

- Name
- Role
- Multiplicity
- Aggregation
- Composition
### Name, Role, Multiplicity

<table>
<thead>
<tr>
<th>Person</th>
<th>works for</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>employee</td>
<td>Company</td>
</tr>
<tr>
<td>Person</td>
<td>1..*</td>
<td>Company</td>
</tr>
</tbody>
</table>

---

### Aggregation

Aggregation is a relationship where one class represents a larger thing that consists of smaller things.

```
Company
  \--- Department
```

---

### Composition

Composition is a special form of aggregation where the parts are inseparable from the whole.

```
Window
  \--- Frame
```
Interface

An interface is a named collection of operations used to specify a service of a class without dictating its implementation. An interface may participate in generalization, association and dependency relationships.

Observer

```
update()
```

Realization

A realization is a relationship between an interface and the class that provides the interface’s services. A class may realize many interfaces.

```
"interface" Observer
```

TargetTracker

update()

Adornments and Extensibility

- **Notes**
  - A note is a graphical symbol containing text and/or graphics that offer(s) some comment or detail about an element within a model.
- **Stereotypes**
  - A stereotype is an extension of the vocabulary of the UML that allows you to create a new kind of “building block” that’s specific to the problem you’re trying to solve.
- **Tagged Values**
  - A tagged value is an extension of the properties of a model element that allows you to create new information within the specification of that element.
- **Constraints**
  - A constraint is an extension of the semantics of one or more model elements which specifies a condition that must be true.
A package is a general-purpose mechanism for organizing elements of a model such as classes or diagrams, into groups. Every element within a model is uniquely owned by one package. Also, that element’s name must be unique within that package.

Package Notation

Object Diagrams

An object diagram shows a set of objects, and their relationships, at a specific point in time.
Components

Components represents physical modules of code. 

Examples:
- Dynamic Link Library (DLL)
- COM component
- Executable (EXE)

Component notation

Deployment

A deployment diagram shows the physical relationships among Software and Hardware components in the delivered system.
Diagram Overview

- Structural diagrams
  - Used to visualize, specify, construct, document *static* system aspects

- Behavioural diagrams
  - Used to visualize, specify, construct, document *dynamic* system aspects

Behavioural Diagrams

- Use Case diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- Activity diagrams

Use Case

- The use case is a typical interaction between a user and a computer system.
- The use case captures some user-visible function.
- You capture a use case by talking to your typical users and discussing the various thing they might do with the system.
Actor

- An actor is a role that a user plays with respect to the system.
- A single actor may perform many use cases.
- A use case may have several actors performing it.
- An actor could be also a non-human player, e.g:
  - a software or hardware component
  - even a component of the system you have to describe (sensors, motors, etc.)

Use Case Diagram Notation

- Sequence Diagram
- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
A collaboration diagram is an interaction diagram that emphasizes the organization of the objects that participate in the interaction.

**Path:** link between objects.

**Sequence numbers:** indicate the time ordering of messages.

---

**State, Event, Signal**

- A **state** is a condition in which an object can reside during its lifetime while it satisfies some condition, performs an activity or waits for an event.
- An **event** is a significant occurrence that has a location in time and space.
- A **signal** is an asynchronous communication from one object to another.

---

**State Notation**

- **Lamp**
  - Light off
  - Light on
  - switch on
  - switch off
State Machine

- A state machine is a behaviour that specifies the sequences of states that an object goes through in its lifetime, in response to events, and in its responses to those events.

Initial and Final States

- The initial state is the default starting place for a state machine.
- The final state indicates the completion of the state machine’s execution.

Activity Diagrams

- An activity diagram, which is similar to a flowchart, is useful for modeling workflows and the details of operations.

Activity vs Interaction Diagrams:
- The interaction diagram looks at the objects that pass messages.
- The activity diagram looks at the operations that are passed among objects.
Activity Diagram Notation

Counter  Display  Manipulation

increment counter

no

stop

[check buttons]

start

stop ?

display

Alternate paths:

- Branch
- Merge

Parallel flows:

- Fork
- Join

Branch

A branch has one incoming transition and two or more outgoing transitions:

- checking

[check ok]  [check not ok]

plane ready  solve problem
Merge

A merge has two or more incoming transitions and one outgoing transition:

- plane ready
- passengers boarded

Join

A join represents the synchronization of two or more flows of control into one sequential flow of control:

- receive product
- bill customer

Fork

A fork represents the splitting of a single flow of control into two or more concurrent flows of control:

- receive event
- log event
- process event
Outline

- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study

Advantages by using UML

- Help in acquiring an overall view of the system
- Help in visualizing, specifying, constructing and documenting a hardware/software system
- Clarity: reference to its semantics
- Consistency Check:
  - Implementation vs design
  - Design vs specifications
- Easy to add new features

Disadvantages by using UML

- Formalization hard to deal with for non-expert users
- Too detailed formalization could create confusion between project stakeholders
Future perspectives

- UML System-level specification
  - HW/SW Codesign
  - VHDL code generation

Links

- Object Management Group:
  > http://www.omg.org
- UML V2.0:
  > http://www.u2-partners.org
- A UML Tutorial
  > http://odi-skopje.etf.ukim.edu.mk/uml-help

Outline

- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study
Object Oriented Process Streamlined

- Processes
- Analysis
- Design
- Implementation
- Testing

Processes: The spiral model

- All workflows are executed repeatedly in sequence.
- High risk parts are implemented first.
- New requirements may be built in at any time.
- The system grows incrementally by iterative prototypes.
Processes: Phases / Milestones

- Lifecycle decomposed over time into four sequential phases.
- Each concluded by a major milestone.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifecycle objectives milestone</td>
<td>Lifecycle architecture milestone</td>
<td>Initial operational capability milestone</td>
<td>Product release milestone</td>
</tr>
</tbody>
</table>

Processes: Vertical Prototyping

The lifecycle for the Rational Unified Process (RUP).

Structure: horizontally
Implementation: vertically

Analysis workflow

Requirements Analysis
Mechanical Design
Architectural Design
Design
Implementation
Testing
Party!
Notation
- Use Case Diagram
- Sequence Diagram
- Collaboration Diagram
- Statechart Diagram
- Activity Diagram

Steps
- Dynamic Analysis (Behavioural)
  - Actor
  - Use Case
  - Relation between actor and Use Case
  - Scenarios
  - Detailed Analysis of scenarios
- Static Analysis (Structural)
  - Objects / Classes
  - Values (attributes)
  - Behaviour (methods)
  - Responsibilities (activities)

Steps
- Detailed Analysis
  - Class Hierarchy
  - Refinement of Class Diagrams
  - Refinement of Statechart Diagrams
  - Refinement of Activity Diagram
  - Refinement of Scenarios
Dynamic Analysis

Use Case

Scenarios

Class Diagram

Static Analysis

Input

Output

Internal State

Responsibilities

Detailed Analysis

Refinement of Class Diagram
Analysis: Results

Use Case

Activity

State Chart

Scenarios

Class Diagram

Design workflow

Requirements Analysis

Object Analysis

Mechanistic Design

Architectural Design

Detailed Design

Mechanistic Design

Analysis

Testing

Validation Testing

Integration Testing

Unit Testing

Implementation

Notation

Class Diagram

Package Diagram

Component Diagram

Deployment Diagram
Steps

- Architecture
  - Defining packages
  - Interfaces between packages
  - Models, Frameworks, Libraries
- Error Handling
  - Different error handling models
  - Consequent error handling
- Initialization
  - Creation of system objects (factory)
  - Initialization of objects
  - Error handling during initialization

Components

- Separation into code units applying certain criterias
- Components contain a certain number of classes
- Executables
- Libraries
- DLL’s

Deployment

- Physical system image depending on certain criteria
- Partitioning Components – Software, Hardware

Definition of Packages

Class Diagram

Package Diagram
Initialization / Instantiation

Initialization states are very important! (i.e. Factory)

Components

Deployment

Master of Engineering in Embedded Systems Design
Master of Engineering in Embedded Systems Design

**Design: Results**

System

Deployment

Components

**Implementation workflow**

Requirements Analysis

Architectural Design

Mechanistic Design

Object Analysis

System Analysis

Analysis

Testing

Integration Testing

Validation Testing

Prototype Testing

Testing Party!

Coding

Unit Testing

Detailed Design

Mechanistic Design

Architectural Design

Object Analysis

Systems Analysis

Analysis

Implementation

Specifications

Integration

Testing

Validation

Prototype Testing

Testing Party!

Coding

Unit Testing

Detailed Design

Mechanistic Design

Architectural Design

Object Analysis

Systems Analysis

Analysis

Steps

- Specification of unit test cases
- Generation of code according to the model (translative or elaborative)
- Supplementary implementation
- Synchronize model with code and vice versa (translative or elaborative)
Master of Engineering in Embedded Systems Design

Steps
- Generation of components
- Verify the implemented code (metrics)
- Installation concepts
- Creation of technical documentation
- Unit test cases execution

Implementation: Results
- Source code of the system
- Components according to the component diagram
- Installers of the different component (only for SW)
- Unit Test Cases
- System synchronization of model with implementation

Testing workflow
Steps
- Creation of a reproducible test environment
- Specification of integration and system test cases:
  - based on analysis, design and implementation criteria:
    - specification of functional, usability, reliability and performance
test cases
- User Guide consistency check
- Installation Manual consistency check
- Scripts for test automation
  (regression testing)

Testing: Results
- Integration Test Results
- System Test Results
- Suggestion for modifications
- User Guide: final version
- Installation Manual: final version

Outline
- UML meaning
- Benefits provided by UML
- UML Notation
- Advantages and disadvantages by using UML
- Object Oriented Process Streamlined
- Case study
"System-Level Design of Embedded Applications by UML, the wireless meter reading case"

Srinivas Mankan, Aris Martinola, Antonio Minoli, Mauro Prevostini

Cooperation with partners

The project has been supported by:

And presented at:

Project Goals

- Figure out one possible implementation of the Wireless Meter Reading (WMR) system.
- Use the Unified Modeling Language (UML) for analysing and modeling the hardware-software systems.
- Simulate data transmission between the different protocols involved in the WMR.
Problem Description

- **Wireless Meter Reading (WMR)** as a reading system to help Service Providers:
  - to get consumer consumption data regarding gas, electricity and water
  - automatically via wireless technology.
- WMR must be a low-power and low-cost device.
Objectives

- Real-Time determination of energy consumption
- Use wireless technology.
- Security: authentication and data integrity

Benefits:
- Realtime billing
- Operational costs minimization
- Theft monitoring

Outline

- Problem Description
- Development Process Overview
- UML Formalization

Development Process

- Inception Phase (Requirements Specification)
- Elaboration Phase (Requirements Analysis & Design)
- Implementation Phase (System Simulation)
- Test & Validation Phase (To be performed)
Inception Phase (Requirements Specification)
- specification of the project vision;
- develop an Operational Concept Description (OCD) document;
- gather companies feedback/reviews on the OCD document.

Elaboration Phase (Requirements Analysis & Design)

Implementation Phase (System Simulation)

Test & Validation Phase (To be performed)
**Development Process**

- **Inception Phase (Requirements Specification)**
  - planning the necessary activities;
  - analysing the system requirements;
  - specifying the system architecture.

- **Elaboration Phase (Requirements Analysis & Design)**
  - Requirements Analysis & Design

- **Implementation Phase (System Simulation)**
- **Test & Validation Phase - (To be performed)**

**Outline**

- **Problem Description**
- **Development Process Overview**
- **UML Formalization**
Analysis workflow

- Analysis
  - Object Analysis
  - Architectural Design
  - Mechanistic Design
  - Detailed Design

Testing

- Testing
  - Unit Testing
  - Iterative Prototypes
  - Integration Testing

Steps

- Dynamic Analysis (Behavioural)
  - Actor
  - Use Case
  - Relation between actor and Use Case
  - Scenarios
  - Detailed Analysis of scenarios
Scopo dello studio di caso è di formalizzare tramite UML le seguenti specifiche di un lettore di contatori wireless (WMR = Wireless Meter Reader).

Il WMR è un dispositivo a bassa potenza che misura i dati relativi al consumo di elettricità, gas e acqua. L'operazione avviene ogni 15 minuti. Una volta al giorno il WMR spedisce i dati al concentratore (DC = Data Concentrator) attraverso un collegamento senza fili (ad esempio Bluetooth). Se il DC non riesce ad elaborare il messaggio spedito dal WMR, la comunicazione viene abbandonata dopo un timeout predefinito.

Uno dei compiti del DC è verificare l'integrità dei dati. In caso di errori il DC spedisce una richiesta di rianimazione al WMR. In seguito il DC memorizza i dati in un buffer locale.

Una volta al mese i dati vengono spediti mediante rete wireless (ad esempio via GSM) alla centrale della società erogatrice dove vengono verificati e, in caso di errori, verrà spedita una richiesta di rianimazione al DC. Al termine del procedimento di verifica, i dati vengono depositati nella banca dati locale.

La società erogatrice inoltre deve poter leggere in qualsiasi momento sia i dati depositati nel DC sia, in tempo reale, i dati del contatore.