An Overview of Object–Oriented Software Development with the Unified Modeling Language (UML)

Edward Colbert
Absolute Software Co., Inc.
1444 Sapphire Dr.
Carlsbad, CA  92009-1200
Phone:  (760) 929-0612
FAX:  (760) 929-0236
E-mail:  colbert@abssw.com
Website:  www.abssw.com

Copyright

Copyright 1998–99 Absolute Software Co., Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the copyright holder.

This presentation was given in substantially similar form to the Southern California Software Process Improvement Network on May 28, 1999, at Irvine, California. Permission to copy without fee is granted provided that the copies are not made or distributed for direct commercial advantage, the Absolute Software copyright notice and the title of the publication and its date and author appear, and notice is given that copying is by permission of Absolute Software. To copy otherwise, or to republish, requires a fee and specific permission.

About permission to copy, corporate licenses to use course materials, or for other information, please call Absolute Software at (760) 929-0612 or send E-mail to info@abssw.com.
Goals of Presentation

You should understand

- The motivation for UML
- The goals, scope, & history of UML
- What UML is & is not
- The basic structure the Unified Modeling Language (UML)
  - Models
  - Views
  - Diagrams
  - Model & View Elements
- The primary view of OO supported by UML

Note:

- Absolute Software’s class fully teaching the UML notation, with lecture & exercise, runs about 4 days
- Absolute Software’s method class with lecture and lab runs 5 days.

Outline

- Background
- Key Concepts
- Method Overview
- Summary
**Why UML**

Developer’s Goal — Satisfied Customer(s)

Customer — Has Needs & Expectations (will feel good if satisfied)

Concept Evaluation

Product Requirements (testable)

Development

Validation: Do we understand customer need?

Validation: Does product meet requirements?

Product

Verification: Does product meet requirements?

If the throttle is pulled then….

If the pilot ….

**Build a software product that satisfies a human need & meets:**

### Operational System Objectives
- Reliability
- Efficiency
- Suitability

* Must be verifiable & testable

### Lifecycle Objectives
- Understandability
- Adaptability
  - Portability
  - Re-usability
  - Tunability
  - Plasticity
- Maintainability

### Development Process’s (Economic) Objective
- Cost–Effectiveness
  - Productivity
  - Return on investment

i.e., achieve “Software Engineering & System Building Goals”

1 Also called “appropriateness”
Why UML (cont.)

**Need Modeling Language**

So can model system before construction
- Acts like blueprint for a large building
- Supports communication among project team & customer(s)
- Supports verification & validation of construction

Must have:
- Model elements (concepts with semantics)
- Notation for visualizing model elements
- Guides for usage of model elements

**OO Notation Standard Needed**

OO methodology evolved in parallel in different areas of development:

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand ax</td>
<td>Simula 67/SmallTalk</td>
</tr>
<tr>
<td>etc.</td>
<td>C++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design Methods</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Program Design by Informal English Descriptions”, Abbott</td>
<td>Flat–File Databases</td>
</tr>
<tr>
<td>Booch</td>
<td>Hierarchical Databases</td>
</tr>
<tr>
<td>Ken Orr ERDs</td>
<td>Relational Databases</td>
</tr>
<tr>
<td>Chen ERDs</td>
<td>Object–Oriented Databases</td>
</tr>
</tbody>
</table>

Many notations evolved
- Not all as well defined as others
### UML Goals

Provide a visual modeling language that is:

- Ready to use
- Expressive
- Independent of particular programming languages
- Independent of development process
- Extensible
  - Core concepts can be extended or specialized by users
- Supportive of higher-level concepts
  - e.g. collaborations, frameworks, patterns, components

Provide a modeling language that:

- Integrates best practices
- Encourages growth of OO tools market

---

### UML Scope

UML is:

- An integration of Booch, OMT (Rumbaugh), & OOSE (Jacobson) concepts
- A standard modeling language
  - Defined by consensus of many in the OO community
  - Intended for a wide range of systems

UML is not:

- A visual *programming* language
  - Doesn’t having all the necessary visual and semantic support
  - Does have a tight mapping to a family of OO languages
    - In practice, you are driven to using your language of choice for things like parameter declaration syntax
- An tool *interface, storage, or run-time model*
  - Although these should be fairly close to *[semantic] model*
UML is not

- A Method
  - A method requires a process
  - Defining a standard process was not a goal of UML or OMG’s RFP
    - No consensus for standardization, yet
  - Authors felt processes must be tailored to the organization, culture, and problem domain at hand
    - Building shrink-wrapped software is different from building hard-real-time avionics
  - Promotes processes that are
    - Use-case driven
    - Architecture centric
    - Iterative & incremental
**UML History (cont.)**

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct ’94</td>
<td></td>
<td>Grady Booch &amp; Jim Rumbaugh begin merging Booch &amp; OMT</td>
</tr>
<tr>
<td>Oct ’95</td>
<td>0.8 draft</td>
<td>Release</td>
</tr>
<tr>
<td>Fall ’95</td>
<td></td>
<td>Ivar Jacobson joins Rational &amp; UML effort, adding OOSE</td>
</tr>
<tr>
<td>’96</td>
<td></td>
<td>Object Management Group (OMG) issues Request for Proposal for a modeling language</td>
</tr>
<tr>
<td>Oct ’96</td>
<td>0.9 draft</td>
<td>Release</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rational establishes consortium, including DEC, HP, I-Logix, IntelliCorp, IBM, ICON Computing, MCI System House, Microsoft, Oracle, TI, &amp; Unisys</td>
</tr>
<tr>
<td>Jan ’97</td>
<td>1.0</td>
<td>Released &amp; Submitted to OMG.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IBM/ObjecTime Team, Platinum Technology, Ptech, Taskon &amp; Reich Technologies, Softeam submit separate responses, then join the UML Partners</td>
</tr>
<tr>
<td>Sep ’97</td>
<td>1.1</td>
<td>Published.</td>
</tr>
<tr>
<td>Feb ’98</td>
<td>1.1</td>
<td>Standard adopted by OMG</td>
</tr>
</tbody>
</table>

---

**An Overview of Object-Oriented Software Development with the Unified Modeling Language (UML)**

**UML Definition**

Defined by 3 documents

- **UML Semantics**
  - Defines the language
  - Uses very detailed *metamodel*
  - Includes two appendices: *Standard Elements* and *UML Glossary*

- **UML Notation Guide**
  - Describes the UML notation
  - Provides examples

- **UML Object Constraint Language Specification**
  - A formal language to express side effect-free constraints
UML Extensions & Variants

Companies & projects can define extensions

Extensions will not be as universally agreed upon

Terms are defined in order to reduce confusion

- **UML Variant**
  - a language with well-defined semantics that is built on top of the UML metamodel
  - It specializes the UML metamodel, without changing any of the UML semantics or redefining any of its terms.
    
    e.g. cannot reintroduce a class called State

- **UML Extension**
  - A predefined set of Stereotypes, Tagged Values, Constraints, and notation icons that extend and tailor the UML for a specific domain or process

Rational Corporation Had Defined 2 Extensions

- **UML Extension for Business Modeling**
- **UML Extension for Objectory Process for Software Engineering**

For most projects

- Variants should be rare
- A few extension may be desirable
An Overview of Object-Oriented Software Development with the Unified Modeling Language (UML)

Outline

Background

Key Concepts

Method Overview

Summary

Models & Views

UML Representation of A System

- Structural Model
- Behavior Model
- Diagram
- Textual View

System
  - System Elements
  - abstracted as
  - 0..*

Model
  - Model Elements
  - abstracted as
  - 0..*
  - displayed as
  - 1..*

View
  - View Elements
  - abstracted as
  - 1..*
  - displayed as
  - 1..*

Package
Classifier
...

© 1998–99 Absolute Software Co., Inc.
**What Is A UML Model?**

A model is

- an abstraction of a system, possibly from a certain viewpoint
  - e.g.
    - *Structural and Behavioral Models*
    - Conceptual, Design, & Deployment Models
    - ROSE uses Use–Case, Logical, and Deployment Models

- complete & self-contained from that viewpoint at the chosen level of abstraction

- a subclass of *package*
  - the model package contains model elements that describe the system

Note: the UML Glossary isn’t usually as helpful as the main body of the text

  e.g. the glossary just says “a model is a semantically closed abstraction of a system” (*Semantics*, p315)

**What Is A View?**

A view is

- “A projection of a model, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective” (*Semantics*, p372)

- A diagram or text form illustrating some aspect of the model
  
  e.g.,
  - a Use–Case Diagram
  - a class specification
Also known as *static* models

Emphasizes the structure of objects in a system, including their classes, interfaces, attributes and relations

Viewed using a combination of

- Static Structure Diagrams
  - Class Diagrams
  - Object Diagrams
- Implementation Diagrams
  - Component Diagrams
  - Deployment Diagrams
- Text forms & tables

Format of these forms and tables is not defined by UML.

---

Also known as *dynamic* models

Emphasizes the behavior of objects in a system, including their methods, interactions, collaborations and state histories

Viewed using a combination of

- Use–Case Diagrams
- Statechart Diagrams
  - Activity Diagrams
- Interaction Diagrams
  - Sequence Diagrams
  - Collaboration Diagrams
- Text forms & tables

Format of these forms and tables is not defined by UML.
Common Elements

Diagrams

Are 2-dimensional graphs containing *node* and *paths*

Size, shape, & placement of symbols is not important
- Exception: sequence diagram has a time axis

3 kinds of visible relations
- Connection — usually paths to nodes
- Containment — symbols within the boundary of nodes
- Visual attachment — one symbol near another on a diagram

4 kinds of graphical constructs
- Icon
- 2-dimensional Symbol
- Path
- Strings

Extension Mechanism

UML authors

- Realized that the UML model needed to be extensible to handle new concepts not conceived at time of development

- Discovered that visualization & tool constraints limited
  - Amount of information that could be displayed graphically
  - Number of 2-dimensional symbols that could be used

- Defined means for extending base UML model
  - *Comments & Constraints*
  - *Element Properties*
  - *Stereotypes*
An Overview of Object-Oriented Software Development with the Unified Modeling Language (UML)

Outline

Background

Key Concepts

Method Overview

Summary

Problem Description

Process Overview

Planning & Elaboration

Building Cycle 1
Case Study Problem Description

Customer’s Concept

- Set of 10 Temperature Sensors
- Alarm
- Monitor
- User
- Data Log

Purpose

This application is to perform real-time monitoring of a collection of temperature sensors, and to record information in a database for later trend analysis.

Description

There exists a collection of ten independent sensors that continually monitor temperatures in an office building. Initially, all sensors are disabled. We may explicitly enable or disable a particular sensor, and we may also force its status to be recorded. Furthermore, we may set the lower and upper limits of a given sensor. In the event that any of the enabled sensors register an out-of-limits value, the system must post an alarm condition. Additionally, it must request and record the status of all the sensors every 15 minutes (set by a timer hardware interrupt). If we do not get a response from any sensor within 5 seconds after this time, we must assume that the sensor is broken, and immediately post another alarm. Asynchronously, we may get a user command to enable or disable a specific sensor, set the temperature limits, or force the status of a given sensor to be recorded. In any case, failure of the user interface must not affect the monitoring of any currently enabled sensors.

The status of the sensors at each reading, and the occurrence of any alarms are to be stored in a database with appropriate time and date information. The system must be able to reconstruct and display the historic values of the sensors.

---

Based on a problem described in G. Booch, Software Engineering with Ada, pp. 287-304 (Benjamin/Cummings, 1st ed. 1983; the improved treatment of this problem in the 2nd ed. 1985 is less interesting for the present purpose).
Case Study Problem Description (cont.)

An optional feature of the system is to set the state and limits of the sensors from profiles, which are stored in the database. The profiles may include exceptional days or weeks where different limits may be set, for example building shutdowns or maintenance (e.g., carpet cleaning or paint drying). The profiles may also be used to initialize the system if necessary to restart it.

The physical sensors are located in the lobby, main office, warehouse, stock room, terminal room, library, computer room, lounge, loading dock, and cleaning room.

The physical sensors use memory-mapped I/O ports to write integer values to the ten 16-bit words starting at address 16#0100#. This integer value when multiplied by the accuracy of the sensor (0.5°C) results in the external temperature value.

When a fault is detected or a sensor goes out of limits, the system shall turn on a warning light, which the user must manually clear. To activate a warning light, the system must place all 1's (16-bits) in the memory-mapped I/O ports:

- 16#0010# -- address of fault warning
- 16#0011# to 16#001A# -- address of out-of-limits warning on
  -- sensors 1 to 10
Evolutionary development recommended

Product Release 1  Product Release 2  ...  Product Release N

Activities with in each evolutionary cycle

Explore  Refine Plan  Build  Deploy

Exploratory Phase

- Plan & Elaborate
  - Problem Definition
  - Concept exploration
  - Planning of development effort
  - Defining requirements

- Build Prototype
  - Short development of system prototype to explore critical or high risk issues
  - May not be required for each release
Evolutionary Cycle Activities Details (cont.)

Refine Plan
- Modify overall schedule, budget based on results of exploration

Build
- Design, coding, & testing of system release

Deploy
- Putting the system to use
  - User Documentation
  - Training
  - Installation Instructions
  - Technical Support
  - etc.

This overview focuses on Plan & Elaborate and Build activities
Plan & Elaborate Activity

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Artifacts</th>
<th>Business/Technical</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Draft Plan</td>
<td>Schedule, resource, budget, etc.</td>
<td>Business</td>
<td></td>
</tr>
<tr>
<td>Create Preliminary Investigation Report</td>
<td>Motivation, alternatives, business needs</td>
<td>Business</td>
<td></td>
</tr>
<tr>
<td>Define Requirements</td>
<td>Declarative statement of requirements</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>Define Use–Cases</td>
<td>Descriptions of domain processes &amp; Diagrams to illustrate use–case relations</td>
<td>Technical</td>
<td></td>
</tr>
<tr>
<td>Define Draft Conceptual Model</td>
<td>Preliminary model of concepts</td>
<td>Technical</td>
<td></td>
</tr>
<tr>
<td>Define Draft System Architecture</td>
<td>Preliminary model of system &amp; actors</td>
<td>Technical</td>
<td>Yes</td>
</tr>
<tr>
<td>Define Behavior Prototype</td>
<td>Prototype system to aid understanding of problem risks &amp; requirements</td>
<td>Technical</td>
<td>Yes</td>
</tr>
<tr>
<td>Record Terms in Glossary</td>
<td>Dictionary of terms along with related information (e.g. constrains &amp; rules)</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>Refine Plan</td>
<td>Schedule, resource, budget, etc.</td>
<td>Business</td>
<td></td>
</tr>
</tbody>
</table>

Purpose:
- Document needs and desires for product in form that supports communication among clients and developers
- Reduce risks by clearly defining requirements

Artifacts:
- Purpose statement
- Customers
- Goals
- System functions
- System operational characteristics
Temperature Monitoring System Requirements

Purpose
This application is to perform real-time monitoring of a collection of temperature sensors, and to record information in a database for later trend analysis.

Customers
Building manager(s)

Goals
- Notify building managers when temperatures of rooms in building get outside defined ranges.
- Gather statistics about temperatures throughout building so building manager can better control heating/cooling costs.

System Functions

Basic Functions – Monitoring

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Category</th>
<th>Type</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1</td>
<td>System shall enable sensors on user command.</td>
<td>Evident</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.2</td>
<td>System shall disable sensors on user command.</td>
<td>Evident</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.3</td>
<td>System shall set sensors alarm limits on user command.</td>
<td>Evident</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.4.1</td>
<td>System shall automatically test current sensor value against limits set for that sensor while sensor is enabled.</td>
<td>Hidden Interface</td>
<td>Sensor DMA's ten 16-bit words starting at 16#0100#.</td>
<td>H/W Req</td>
<td></td>
</tr>
<tr>
<td>R1.4.2</td>
<td>System shall turn on “alarm” light corresponding to sensor whenever sensor value is out of limits and sensor is enabled.</td>
<td>Evident Interface</td>
<td>Memory mapped I/O ports 16#0011# to 16#001A# (16-bit)</td>
<td>H/W Req</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>On Value =&gt; 16#FF#</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Statistics Gathering Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Category</th>
<th>Type</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2.1</td>
<td>System shall periodically record the status of all sensors in database.</td>
<td>Hidden</td>
<td>Period</td>
<td>15 seconds</td>
<td>Must</td>
</tr>
<tr>
<td>R2.2</td>
<td>System shall record the status of selected sensor in database on user command.</td>
<td>Evident</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.3</td>
<td>System shall turn on &quot;alarm&quot; light indicating that a sensor is broken when the system detects that sensor didn't provided status within a fixed time limit from request.</td>
<td>Evident</td>
<td>Interface</td>
<td>Memory mapped i/O ports 16#0010#</td>
<td>Must</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Deadline</td>
<td>On Value =&gt; 16#FF#</td>
<td>Must</td>
</tr>
<tr>
<td>R2.4</td>
<td>System shall record the occurrence of each alarm event.</td>
<td>Hidden</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.5</td>
<td>System shall be able to display the status history of selected sensor</td>
<td>Evident</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.6</td>
<td>System shall be able to display the history of alarm events</td>
<td>Evident</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Profile Management Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Category</th>
<th>Type</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3.1</td>
<td>System shall periodically enable or disable sensors and set their limits based on the profile, which is stored in the database, that is applicable to the current date and time.</td>
<td>Hidden</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Initialization Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Requirement</th>
<th>Category</th>
<th>Type</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4.1</td>
<td>System shall enable or disable sensors and set their limits based on profile, which is stored in the database, that is applicable to the current date and time.</td>
<td>Hidden</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questions will arise that need response

- Especially if group formalizing the requirements doesn’t include customer, marketers, system engineers who gave original problem statement
- These should be documented

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
<th>Questioner</th>
<th>Date</th>
<th>Applies To</th>
<th>Response</th>
<th>Responder</th>
<th>Response Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Are the sensors intelligent? I.e., can they be enabled, disabled, told when to send values, do they testing of their value against set limits?</td>
<td>...</td>
<td>...</td>
<td>R1.1, R1.2, R1.3</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Q2</td>
<td>Number of fault lights?</td>
<td>...</td>
<td>...</td>
<td>R2.1.2</td>
<td>H/W limits</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Q3</td>
<td>Turn off lights?</td>
<td>...</td>
<td>...</td>
<td>R2.1.2</td>
<td>Problem stmt.</td>
<td>Yes</td>
<td>...</td>
</tr>
<tr>
<td>Q4</td>
<td>Conditions for Detecting Faulty Sensors?</td>
<td>...</td>
<td>...</td>
<td>R1.4.2</td>
<td>Anytime detect</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Q5</td>
<td>Number of limit lights?</td>
<td>...</td>
<td>...</td>
<td>R2.1.2</td>
<td>H/W limits</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Q6</td>
<td>What period should sensors be tested?</td>
<td>...</td>
<td>...</td>
<td>R2.1.2</td>
<td>5 seconds</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Q7</td>
<td>What period should read profiles</td>
<td>...</td>
<td>...</td>
<td>R4.1</td>
<td>30 minutes</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Field | Description
Question ID | Q1
Question: | Are the sensors intelligent? I.e., can they be enabled, disabled, told when to send values, do they testing of their value against set limits?
Originator: | ...
Date: | ...
Applies To: | R1.1, R1.2, R1.3
Notes: | The initial part of the problem description implies that the sensors are intelligent, e.g., “we may explicitly enable or disable a particular sensor”; but latter it describes only 1 interface the I/O port that it writes its values to.
Response Date: | ...
Responder: | Fred the System Engineer
Response: | No, they're dumb sensors. Sensor monitoring should be enabled & disabled. Software is responsible for testing values written by sensors to specified memory locations.
Part Of: |
Parts: |
Original Questions: |
Derived Questions: | Q8
An Overview of Object–Oriented Software Development with the Unified Modeling Language (UML)

Questions For Customer/System Engineers (cont.)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question ID</td>
<td>Q8</td>
</tr>
<tr>
<td>Question:</td>
<td>How do we know if a sensor value has been written?</td>
</tr>
<tr>
<td>Originator:</td>
<td>…</td>
</tr>
<tr>
<td>Date:</td>
<td>…</td>
</tr>
<tr>
<td>Applies To:</td>
<td>R2.3</td>
</tr>
<tr>
<td>Notes:</td>
<td>The response to Q1 was that the sensors are dumb. If so, than I do we know that a sensor has written a value to the memory location</td>
</tr>
<tr>
<td>Response Date:</td>
<td>…</td>
</tr>
<tr>
<td>Responder:</td>
<td>Fred the System Engineer</td>
</tr>
<tr>
<td>Response:</td>
<td>Write a 16-bit –1 (16#FF#) to the memory location after reading it.</td>
</tr>
<tr>
<td>Part Of</td>
<td></td>
</tr>
<tr>
<td>Original Questions:</td>
<td>Q1</td>
</tr>
<tr>
<td>Derived Questions:</td>
<td></td>
</tr>
</tbody>
</table>

Draft System Architecture

Purpose:

- Improve understanding of requirements
- Developing an understanding of context in which system will operate
  - What external objects (actors) the system needs to communicate

Artifacts:

- Collaboration Diagram
- Run-time Component Diagram
- Deployment Diagram
- Glossary entries
Temperature Monitoring System Architectural Context

- Fault Light
- Operator
- Limit Light
- System: Temperature Monitor System
- Sensor
- Database
- Timer

Temperature Monitoring System Run-time Component Diagram

- Monitor Application
- Database
Purpose:

- Improve understanding of requirements
- Develop an understanding of the domain processes ("use-cases") & relations between processes
- Support planning of development cycles

Artifacts:

- Use-cases
  - Summary
  - High-level descriptions of each
  - Extended descriptions of "interesting" ones
- Use-case Diagram(s)
## Plan & Elaborate – Define Use-Cases (cont.)

### Temperature Monitoring System Use-Cases

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Importance</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable/Disable Sensors</td>
<td>Set the monitoring state of a sensor.</td>
<td>Primary</td>
<td>R1.1, R1.2</td>
</tr>
<tr>
<td>Set Limits</td>
<td>Set the upper &amp; lower temperature bounds, which cause an alarm if the sensor value exceeds.</td>
<td>Primary</td>
<td>R1.3</td>
</tr>
<tr>
<td>Test Sensors</td>
<td>Test whether any sensors are out of limits.</td>
<td>Primary</td>
<td>R1.4.1, R1.4.2, R2.4</td>
</tr>
<tr>
<td>Clear Alarm</td>
<td>Turn off selected alarm lights.</td>
<td>Primary</td>
<td>R1.5</td>
</tr>
<tr>
<td>Record Sensor Status</td>
<td>Record the status of a sensor on operator command.</td>
<td>Primary</td>
<td>R2.2, R2.3, R2.4</td>
</tr>
<tr>
<td>Record All Sensors’ Status</td>
<td>Periodically record the status of all sensors.</td>
<td>Primary</td>
<td>R2.1, R2.3, R2.4</td>
</tr>
<tr>
<td>Display History</td>
<td>Display the history of events &amp; sensor statuses</td>
<td>Primary</td>
<td>R2.5, R2.6</td>
</tr>
<tr>
<td>Update Monitoring Settings</td>
<td>Set the monitoring state &amp; limits for each sensor based on stored profile</td>
<td>Secondary</td>
<td>R3.1</td>
</tr>
<tr>
<td>Initialize System</td>
<td>Initialize the system on startup/reboot</td>
<td>Primary</td>
<td>R4.1</td>
</tr>
</tbody>
</table>

### Use-Case Diagram

![Use-Case Diagram](image-url)
Plan & Elaborate – Define Use-Cases – Temperature Monitoring System (cont.)

High-Level Descriptions

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Enable/Disable Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Set the monitoring state of a sensor.</td>
</tr>
<tr>
<td>Actors</td>
<td>Operator</td>
</tr>
<tr>
<td>Importance</td>
<td>Primary</td>
</tr>
<tr>
<td>Overview</td>
<td>Operator selects a sensor and sets its state to enabled or disable.</td>
</tr>
<tr>
<td>Requirements</td>
<td>R1.1, R1.2</td>
</tr>
<tr>
<td>State</td>
<td>High-Level Conceptual</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Set Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Set the upper &amp; lower temperature bounds, which cause an alarm if the sensor value exceeds.</td>
</tr>
<tr>
<td>Actors</td>
<td>Operator</td>
</tr>
<tr>
<td>Importance</td>
<td>Primary</td>
</tr>
<tr>
<td>Overview</td>
<td>Operator selects a sensor and sets its upper &amp; lower bounds.</td>
</tr>
<tr>
<td>Requirements</td>
<td>R1.3</td>
</tr>
<tr>
<td>State</td>
<td>High-Level Conceptual</td>
</tr>
</tbody>
</table>

Etc.

Typical tools will automatically track Use-Case Name & Actors, and will provide forms where the rest of the information can be entered.

Use-Case Expanded/Essential Description

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Record All Sensors' Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Periodically record the status of all sensors.</td>
</tr>
<tr>
<td>Actors</td>
<td>Timer, Sensor, Fault Light, DBMS</td>
</tr>
<tr>
<td>Importance</td>
<td>Primary</td>
</tr>
<tr>
<td>Overview</td>
<td>A timer signal occurs and it's time to record the status of all sensors. The system reads (the memory location assigned to each) sensor for which monitoring is enabled. If a value is not available, it waits (5 seconds) and tries again, if a value is still not available, it turns on the Fault light and records the event in the database. If a value is available, it stores it along with the current sensor state &amp; limits, and time stamp.</td>
</tr>
<tr>
<td>Requirements</td>
<td>R2.1, R2.3, R2.4</td>
</tr>
<tr>
<td>State</td>
<td>Essential</td>
</tr>
<tr>
<td>Uses</td>
<td></td>
</tr>
</tbody>
</table>

Typical Course of Action

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Timer signal occurs</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[time to record all sensors' status] for each sensor</td>
</tr>
<tr>
<td>2.1</td>
<td>[Sensor is enabled] read sensor value.</td>
</tr>
<tr>
<td>2.2</td>
<td>[Value is available] store status (name, value, state, limits, and time stamp) in database.</td>
</tr>
</tbody>
</table>
### Alternate Course of Action: Disabled Sensor

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 [Sensor is disabled]</td>
<td>store status (name, state, and time stamp) in database.</td>
</tr>
<tr>
<td>2.2 Skip</td>
<td></td>
</tr>
</tbody>
</table>

### Exceptional Course of Action: Enabled Sensor May be Broken

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2 [Value is unavailable]</td>
<td>wait for next interrupt</td>
</tr>
<tr>
<td>3. Timer signal</td>
<td></td>
</tr>
<tr>
<td>4. For all enabled sensors whose values weren’t available</td>
<td></td>
</tr>
<tr>
<td>4.1 Read value</td>
<td></td>
</tr>
<tr>
<td>4.2 [Value is available]</td>
<td>store status (name, value, state, limits, and time stamp) in database.</td>
</tr>
<tr>
<td>4.3 [Value is unavailable]</td>
<td></td>
</tr>
<tr>
<td>4.3.1 Turn on Fault light</td>
<td></td>
</tr>
<tr>
<td>4.3.2 Store status (name, state = broken, limits, and time stamp) in database.</td>
<td></td>
</tr>
</tbody>
</table>

---

### Define Draft Conceptual Model

**Purpose:**
- Improve understanding of requirements
- Developing an understanding of the key concepts
  - i.e. types of objects that need to be developed

**Artifacts:**
- Class Diagram of preliminary concepts
- Glossary entries
An Overview of Object–Oriented Software Development with the Unified Modeling Language (UML)

Plan & Elaborate – Define Conceptual Model

Temperature Monitoring System Conceptual Model

Plan & Elaborate

Record Terms in Glossary

Purpose

• Gather definitions of terms for easy lookup

Example

| Term                    | Category | Stereotype | Description                                      | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Classifier</td>
<td>Type</td>
<td>Were we record status values.</td>
<td></td>
</tr>
<tr>
<td>DBMS</td>
<td>Classifier</td>
<td>Type</td>
<td>Software that will buy to manage our database.</td>
<td></td>
</tr>
<tr>
<td>Enable/Disable Sensor</td>
<td>Use–Case</td>
<td>Actor</td>
<td>Set the monitoring state of a sensor.</td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Classifier</td>
<td>Actor</td>
<td>The role of the &quot;person&quot; who controls the system.</td>
<td></td>
</tr>
<tr>
<td>Record All Sensors' Status</td>
<td>Use–Case</td>
<td>Type</td>
<td>Periodically record the status of all sensors.</td>
<td></td>
</tr>
<tr>
<td>Sensor State</td>
<td>Classifier</td>
<td>Type</td>
<td>The current monitoring state of a sensor.</td>
<td></td>
</tr>
<tr>
<td>Sensor Status</td>
<td>Classifier</td>
<td>Type</td>
<td>Information that describes what we know about a sensor at a given point in time.</td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>Classifier</td>
<td>Type</td>
<td>The kind of values we are monitoring.</td>
<td></td>
</tr>
<tr>
<td>Temperature Monitor System</td>
<td>Classifier</td>
<td>System</td>
<td>The system we are building.</td>
<td></td>
</tr>
</tbody>
</table>

Typical tools will produce some form of glossary.
Plan & Elaborate

Planning

Purpose:
- Develop preliminary plan of development phase
  - Estimate of number cycles
  - Understand what capabilities (use cases) will be developed in which cycle
  - Estimate schedule for each cycle
  - Estimate budget for each cycle
  - Identify resources (e.g., people)

Artifacts:
- Description of cycles & use cases that will be implemented in each
- Budget
- Schedule

Not part of this course

Temperature Monitor System Use–Case/Requirement Ranking

<table>
<thead>
<tr>
<th>Use–Case Name</th>
<th>Requirements</th>
<th>Rank</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Sensors</td>
<td>R1.4.1, R1.4.2, R2.4</td>
<td>Highest</td>
<td>Fundamental capability of system. Significant impact on architecture. Significant insight with relatively small effort.</td>
</tr>
<tr>
<td>Initialize System</td>
<td>R4.1</td>
<td>High/Low</td>
<td>Basic initialization is absolutely necessary. Initialization from profiles is low priority</td>
</tr>
<tr>
<td>Enable/Disable Sensors</td>
<td>R1.1, R1.2</td>
<td>High</td>
<td>Basic capability of system. Significant impact on architecture (UI).</td>
</tr>
<tr>
<td>Set Limits</td>
<td>R1.3</td>
<td>High</td>
<td>Basic capability of system. Significant impact on architecture (UI).</td>
</tr>
<tr>
<td>Clear Alarm</td>
<td>R1.5</td>
<td>High</td>
<td>Basic capability of system. Significant impact on architecture (UI).</td>
</tr>
<tr>
<td>Record Sensor Status</td>
<td>R2.2, R2.3, R2.4</td>
<td>Medium</td>
<td>Significant impact on architecture. Supports increased revenue.</td>
</tr>
<tr>
<td>Record All Sensors’ Status</td>
<td>R2.1, R2.3, R2.4</td>
<td>Medium</td>
<td>Significant impact on architecture. Supports increased revenue.</td>
</tr>
<tr>
<td>Display History</td>
<td>R2.5, R2.6</td>
<td>Low</td>
<td>Significant impact on architecture (UI). Short-term work around.</td>
</tr>
<tr>
<td>Update Monitoring Settings</td>
<td>R3.1</td>
<td>Low</td>
<td>Option of system</td>
</tr>
</tbody>
</table>
## Temperature Monitor System Planned Cycles

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Name</th>
<th>Requirements</th>
<th>Rank</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Test Sensors</td>
<td>R1.4.1, R1.4.2</td>
<td>Highest</td>
<td>No database support</td>
</tr>
<tr>
<td>1</td>
<td>Initialize System</td>
<td></td>
<td>High</td>
<td>Basic Initialization Only</td>
</tr>
<tr>
<td>2</td>
<td>Enable/Disable Sensors</td>
<td>R1.1, R1.2</td>
<td>High</td>
<td>Provides basic temperature monitoring capability</td>
</tr>
<tr>
<td>2</td>
<td>Set Limits</td>
<td>R1.3</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Clear Alarm</td>
<td>R1.5</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record Limit Violations</td>
<td>R2.4</td>
<td></td>
<td>Adds basic statistics monitoring</td>
</tr>
<tr>
<td>3</td>
<td>Record Sensor Status</td>
<td>R2.2, R2.3, R2.4</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record All Sensors’ Status</td>
<td>R2.1, R2.3, R2.4</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Display History</td>
<td>R2.5, R2.6</td>
<td>Low</td>
<td>Final product. Add profile options &amp; program display of history.</td>
</tr>
<tr>
<td>4</td>
<td>Update Monitoring Settings</td>
<td>R3.1</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Initialize System From Profiles</td>
<td>R4.1</td>
<td>Low</td>
<td></td>
</tr>
</tbody>
</table>

## Revised Use–Case Diagram

The use-case diagram for the Temperature Monitor System shows the interactions between actors and the system. The diagram includes use cases such as testing sensors, initializing the system, enabling/disabling sensors, setting limits, clearing alarms, recording sensor status and all sensors' status, displaying history, and updating monitoring settings. The system also extends from other systems and interfaces with a DBMS.
Outline

Problem Description

Process Overview

Planning & Elaboration

Building Cycle 1

Build Activity

Iterative development recommended

Development Cycle 1  Development Cycle 2  ...  Development Cycle N

In each cycle

- Refine Plan
- Sync. Artifacts
- Analyze
- Design
- Construct
- Test

- Developers take on a relatively small subset of the requirements
  - Keeps complexity down
- Developers complete all steps in development process

System grows incrementally

- Ideally, each cycle should produce a product that someone uses
- Early builds emphasize on architectural issues
An Overview of Object–Oriented Software Development with the Unified Modeling Language (UML)

**Build**

- Refine Plan
- Synchronize Artifacts
- Analyze
- Design
- Construct\(^\text{10}\)
- Test\(^\text{10}\)

\(^{10}\) Not covered in this course

---

An Overview of Object–Oriented Software Development with the Unified Modeling Language (UML)

**Build 1**

**Analysis**

**Purpose:**
- Improve understanding of requirements for build

**Activities:**
- Define System Architecture
- Define Use–Cases
- Define Conceptual Model
- Define System Behavior
- Record Terms in Glossary
Define System Architecture

Purpose:
- Improve understanding of requirements for build
- Define the context in which system will operate during this build
  - What external objects (actors) the system needs to communicate

Artifacts:
- Collaboration Diagram
- Glossary entries

Temperature Monitoring System Context

- System: Temperature Monitor System
  - : Sensor (Multiple)
  - : Limit Light (Multiple)
  - : Fault Light
  - : Operator
  - : Timer
Define Use–Cases

Purpose:

• Improve understanding of requirements
• Develop an understanding of the domain processes ("use–cases") & relations between processes for this build

Artifacts:

• Use–cases
  – Summary
  – Extended descriptions
• Use–case Diagram(s)

Temperature Monitor System Use–Cases

- «Actor» Fault Light
- «Actor» Limit Light
- «Actor» Sensor
- «Actor» Operator
- «Actor» Timer
- «System» Temperature Monitor System
  - Initialize System
  - Test Sensors
### Use Case Expanded/Essential Description

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Test Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Test whether any sensors are out of limits.</td>
</tr>
<tr>
<td>Actors</td>
<td>Timer, Sensor, Fault Light, Limit Light</td>
</tr>
<tr>
<td>Importance</td>
<td>Primary</td>
</tr>
<tr>
<td>Overview</td>
<td>A timer signal occurs. The system reads (the memory location assigned to each) sensor for which monitoring is enabled. If the value is not available, the system waits until the next signal (5 seconds) and tries again, if a value is still not available, the system turns on the Fault light. If the value is available, the system checks whether the value is in the current limits. If the value is out of limits, the system turns on the Limit Light for the specified sensor.</td>
</tr>
<tr>
<td>Requirements</td>
<td>R1.4.1, R1.4.2, R2.3 (see Q4)</td>
</tr>
<tr>
<td>State</td>
<td>Essential</td>
</tr>
<tr>
<td>Uses</td>
<td></td>
</tr>
</tbody>
</table>

#### Typical Course of Action

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Timer signal occurs</td>
<td></td>
</tr>
<tr>
<td>2. For each sensor</td>
<td>Read sensor value.</td>
</tr>
<tr>
<td>2.1 [Value is available] test the value against current limits</td>
<td></td>
</tr>
<tr>
<td>2.2 [Value is in limits] mark sensor as read (see Q8)</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td></td>
</tr>
</tbody>
</table>

#### Alternate Course of Action: Value is Out Of Limits

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4 [Value is in limits] turn on limit light for sensor</td>
<td></td>
</tr>
</tbody>
</table>

#### Exceptional Course of Action: Sensor May be Broken

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2 [Value is not available]</td>
<td></td>
</tr>
<tr>
<td>2.3 [This is second attempt] turn on fault light</td>
<td></td>
</tr>
</tbody>
</table>
Define Conceptual Model

Purpose:
- Improve understanding of requirements
- Developing an understanding of the key concepts in this build
  i.e. types of objects that need to be developed

Artifacts:
- Class Diagram of concepts used in this build
- Glossary entries

Temperature Monitor System Conceptual Model

Temperature Monitor System

- Operator
  - real, precision=0.5, units = degrees Centigrade, value range = undefined
- Limit Light
  - enumerated, values = disabled, enabled, broken
- Fault Light
- Sensor
  - Temperature
  - Sensor State
- Timer
- Time
- Controls
  - 1
  - +10
- Monitors
  - 1

Alerts

Temperature Monitor System Conceptual Model (cont.)
Define System Behavior

Purpose:

- Improve understanding of requirements
- Understand the exact “conceptual” behavior required of the system for this build
- Identify system events & operations

Artifacts:

- Sequence Diagram(s) for each Use-case
- Statechart model of the system behavior (optional)
- Contracts for each system operation

Temperature Monitor System Test Sensor Sequence Diagram

- Mark_Time( )
- Value := Value_Of( )
- [Value = None and second attempt] Turn_On( )
- [Value /= None and Value is out of limits] Turn_On( )

* [For each sensor]
- Get value of sensor
- If a sensor is "broken", turn on fault light
- Value for a sensor is out of limits, turn on the corresponding limit light
## Temperature Monitor System — Mark_Time Operation Contract

<table>
<thead>
<tr>
<th>Name:</th>
<th>Mark_Time ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities:</td>
<td>Read sensors and verify that they are working and in limits</td>
</tr>
<tr>
<td>Requirements:</td>
<td>R1.4.1, R1.4.2, R2.3 (see Q4)</td>
</tr>
<tr>
<td>Use-Cases:</td>
<td>Test_Sensor</td>
</tr>
<tr>
<td>Notes:</td>
<td>Since this is called periodically, must complete operation within period (5 seconds)</td>
</tr>
<tr>
<td>Exceptions:</td>
<td>None</td>
</tr>
<tr>
<td>Output:</td>
<td>None</td>
</tr>
<tr>
<td>Pre-conditions:</td>
<td>None</td>
</tr>
<tr>
<td>Post-conditions:</td>
<td>[For each Sensor that return value of “None”] a flag indicates that last read failed</td>
</tr>
</tbody>
</table>

### Revised Conceptual Model

- **Temperature Monitor System**
  - **Timer**
  - **Sensor**
    - Value_Of() : Temperature
  - **Fault Light**
    - Turn_On()
  - **Limit Light**
    - Turn_On()
  - **Operator**
  - **Controller**
  - **Alerts**
    - +10
  - **Monitors**
  - **Values**
    - +Value

### Temperature
- **Type**: Temperature
- **Value**: [real, precision=0.5, units = degrees Centigrade, value range = undefined]
**Design**

**Purpose:**
- Design an implementation of the system that implements the current requirements
  - Components
  - Interactions among components
  - Classes of components

**Activities:**
- Design "Real" Use-Cases
- Design Components & Interactions
- Design Class Model
- Design Component Behaviors
- Record Terms in Glossary

---

**Design Real Use-Cases**

**Purpose:**
- Describe how the use-case will be realized in terms of
  - Input / Output technology
    - **e.g.** Use of graphical user interface
  - Overall implementation

**Artifacts:**
- Real Use-cases
Test Sensor Real Use–Case Description

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Test Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Test whether any sensors are out of limits.</td>
</tr>
<tr>
<td>Actors</td>
<td>Timer, Sensor, Fault Light, Limit Light</td>
</tr>
<tr>
<td>Importance</td>
<td>Primary</td>
</tr>
<tr>
<td>Overview</td>
<td>A timer signal occurs. The system reads (the memory location assigned to each) sensor for which monitoring is enabled. If the value is not available, the system waits until the next signal (5 seconds) and tries again, if a value is still not available, the system turns on the Fault light. If the value is available, the system checks whether the value is in the current limits. If the value is out of limits, the system turns on the Limit Light for the specified sensor.</td>
</tr>
<tr>
<td>Requirements</td>
<td>R1.4.1, R1.4.2, R2.3 (see Q4)</td>
</tr>
<tr>
<td>State</td>
<td>Real</td>
</tr>
<tr>
<td>Uses</td>
<td></td>
</tr>
</tbody>
</table>

**Typical Course of Action**

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Timer signal occurs</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>For each sensor</td>
</tr>
<tr>
<td>2.1</td>
<td>Read sensor value from sensor's memory location</td>
</tr>
<tr>
<td>2.2</td>
<td>[Value /= &quot;None&quot; value] test the value against current limits</td>
</tr>
<tr>
<td>2.3</td>
<td>[Value is in limits] Write &quot;None&quot; value to sensor's memory location (16#FF# -- see Q8)</td>
</tr>
<tr>
<td>2.4</td>
<td>Set Last_Value to value read</td>
</tr>
</tbody>
</table>

**Alternate Course of Action:** Value is Out Of Limits

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>[Value is in limits] turn on limit light for sensor by writing to light's port</td>
</tr>
</tbody>
</table>

**Exceptional Course of Action:** Sensor May be Broken

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>[Value = &quot;None&quot; value]</td>
</tr>
<tr>
<td>2.3</td>
<td>[Last_Value = &quot;None&quot; value] turn on fault light by writing to light's port</td>
</tr>
</tbody>
</table>
Design Component Interactions

Purpose:
- Design the objects in this system will implement requirements of this build
- Design the interactions among the objects

Artifacts:
- Collaboration Diagram (or Sequential Diagrams)
  - Collaboration Diagrams are slightly more expressive
- Glossary entries

Temperature Monitoring System — Test Sensor Interaction Diagram
Design Class Model

Purpose:
- Design classes for each object, & required concept
  - i.e. class of objects that need to be developed
    - Identify attributes
    - Identify relations
    - Identify operations

Artifacts:
- Class Diagram of concepts used in this build
- Glossary entries
Define Component Behavior

Purpose:
- Describe state logic for components with complex behavior based on state

Artifacts:
- Statechart model of the objects (classes) behavior

Temperature Monitor System — Sensor_Class Statechart

Alive

Wait

Testing

on entry [ Value = None and Last_Value = None ]: Fault_Alarm.Activate()
on entry [ Value /= None and Value not in Lower_Limit..Upper_Limit ]: Limit_Alarm.Activate()
Construct

Purpose:
- Create code that implements design in selected programming language

Activities:
- Refine Class Model
- Design Components
- Design Run–Time Deployment
- Write Code
- Record Terms in Glossary

Refine Class Model

Purpose:
- Design language–specific implementations of each class
  - i.e. Code–level descriptions of each class
    - Identify attributes
    - Identify relations
    - Identify operations

Artifacts:
- Class Diagram showing classes implemented in this build
- Glossary entries

Note:
- May update Design of Components & Interactions
- May update Design of Component Behavior
An Overview of Object-Oriented Software Development with the Unified Modeling Language (UML)

Build 1 – Construction – Refine Class Model (cont.)

Temperature Monitoring System – Sensor Package

Purpose:
- Design COM/CORBA/JAVA Beans components
- Design language–specific components
  - e.g.
    - In C++, Files
    - In Ada, Packages, Tasks, Subprograms
- Allocate classes, objects, operations, etc. to components
- Determine component dependencies
  - e.g.
    - In C++, “includes”
    - In Ada, “withs”

Artifacts:
- Component Diagram(s)
- Glossary entries
Temperature Monitoring System – Source Code: Sensor Package

Note: In UML, object icons are normally drawn in appropriate components; but tool didn’t support, so I used a Note.
Purpose:
- Design run–time deployment of components to hardware
  i.e. What software is running on what hardware

Artifacts:
- Run–Time Deployment Diagram(s)
- Glossary entries

Temperature Monitoring System

Monitor Application deployed on Computer

Sensor 1

Sensor ...

Sensor N

Fault Light

Timer

Limit Light 1

Limit Light ...

Limit Light N
Outline

Background

Key Concepts

Method Overview

Summary

An Overview of Object–Oriented Software Development with the Unified Modeling Language (UML)

UML Is Not

A visual programming language

- Doesn’t have all the necessary visual and semantic support
- But does have a tight mapping to a family of OO languages

A tool interface, storage, or run-time model

- Should be fairly close to semantic model

A Method

- UML does not define a process
  - Was not a goal of UML or OMG’s RFP
  - No consensus for standardization, yet
- Processes must be tailored to the organization, culture, and problem domain at hand
- Promotes processes that are
  - Use–case driven
  - Architecture centric
  - Iterative & incremental
An industry standard

- Object–Management Group (OMG)

A nice integration of

- Booch, OMT (Rumbaugh), & OOSE (Jacobson) concepts
- Many best practices in OO community

A visual modeling language that met most of its design goals

- Ready to use
- Expressive
- Relatively independent of particular programming languages
- Independent of development process
- Extensible
- Supportive of higher–level concepts
- Supportive of a variety of systems

---

Others in the community support a second standard, "OPEN".

---

Reasonably well-defined

- Good definitions of
  - Model elements (concepts with semantics)
  - Notation for visualizing model elements
  - Guides for usage of model elements

- Still needs some work
  - Default textual representations of model
  - Clean up of documentation inconsistencies
  - More guidance for some concepts
    - e.g. Patterns

Better defined than any non-programming or natural language
Tool Support

Relatively weak

- Incomplete support for notation
- Misunderstanding of semantics
  - Partially caused by inconsistencies in UML documentation
- Standard is still new
  - About 6 months old

Tool developers should be able to rapidly improve support

- They can focus on 1 (or 2) notations, not 30
- Stability of standard should allow developers to catch up
- Semantic model is relatively well defined

Software Development with UML

UML provides good supports development that is

- Object-Oriented
  - Emphasizes “responsibility–based collaboration” concept
    - Modularization of responsibility/behavior
    - Co–operating components
- Evolutionary
- Use–case driven
- Early Architecture Definition
- Early validation
- Wide variety of systems
Evolutionary Process Benefits

Reduces risk
- Manages complexity
  - System grows incrementally
- Early validation

Provides early feedback
- At end of each build
  - Product testing verifies product meets requirements & qualities for build
  - Management compares cost & schedule estimates against reality
    - Feed into planning for next build
    - Feed into planning for release
- At end of release
  - Customer starts using product & validates that meets needs
  - Customer feedback to developers for next release
    - What’s unsatisfactory
    - What’s great
    - New ideas

Use–Case Driven Process Benefits

Good way to capture & organize customer requirements

Supports early validation
- Easier for non–technical Customers & Reviewers to understand
  - Don’t need to know complex notation
  - Informal natural–language descriptions of system behavior
- Customers (& other reviewers) can give better feedback
- Earlier agreement among Customers & Developers on what the system needs to do

Supports verification
- Acceptance tests of system
  - Can be designed concurrent to design of software
  - Based on behavior defined by use–case courses of action
Conclusions

UML is an OMG standard language for OO analysis & design

UML is not a method; but, supports good object-oriented methods!

UML can help you achieve primary goals

- Software Engineering & System Goals
- Satisfied customer(s)

References


References (cont.)


