

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

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

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

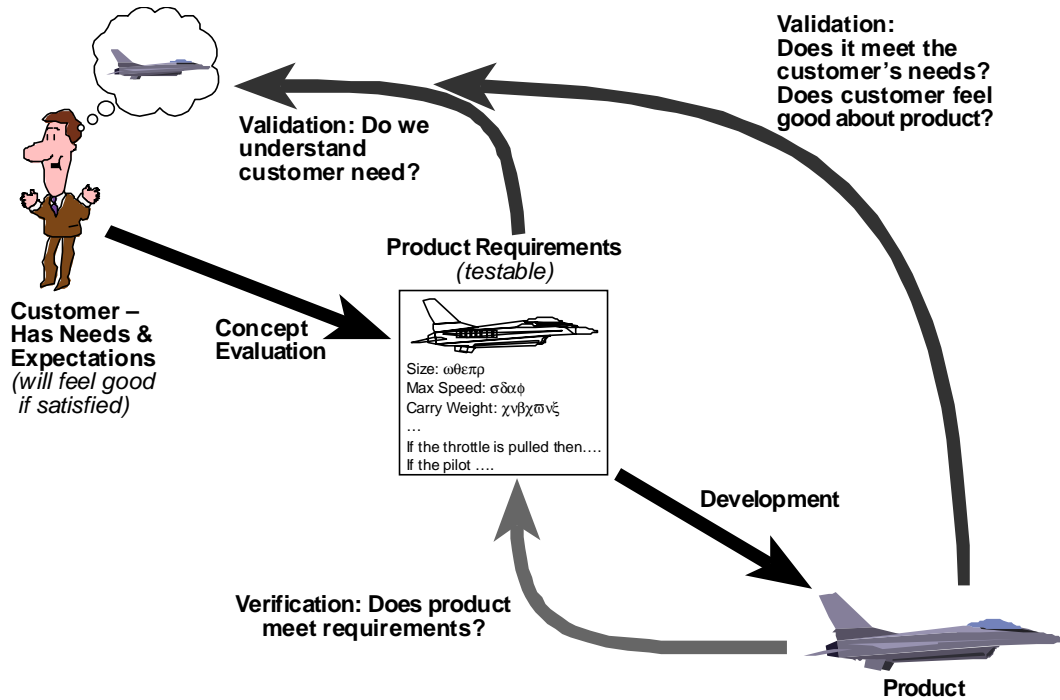
Key Concepts

Method Overview

Summary

Why UML

Developer's Goal — Satisfied Customer(s)



Why UML (cont.)

To Satisfy A Customer, We Must...

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

<p>Operational System Objectives*</p> <ul style="list-style-type: none"> • Reliability • Efficiency • Suitability¹ <p>* Must be verifiable & testable</p>	<p>Lifecycle Objectives</p> <ul style="list-style-type: none"> • Understandability • Adaptability <ul style="list-style-type: none"> – Portability – Re-usability – Tunability – Plasticity • Maintainability
<p>Development Process's (Economic) Objective</p> <ul style="list-style-type: none"> • Cost-Effectiveness <ul style="list-style-type: none"> – Productivity – Return on investment 	

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

¹ Also called “appropriateness”

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:

Hardware

Hand ax

etc.

Programming Languages

Simula 67/SmallTalk

C++

Object-Oriented Languages

Design Methods

"Program Design by Informal English Descriptions", Abbott

Booch

Ken Orr ERDs

Chen ERDs

Database

Flat-File Databases

Hierarchical Databases

Relational Databases

Object-Oriented Databases

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 languages²
- 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

² Only partially succeeded.

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 community³
 - Intended for a wide range of systems

UML is not

- A visual *programming* language
 - Doesn't have all the necessary visual and semantic support
 - Does not 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

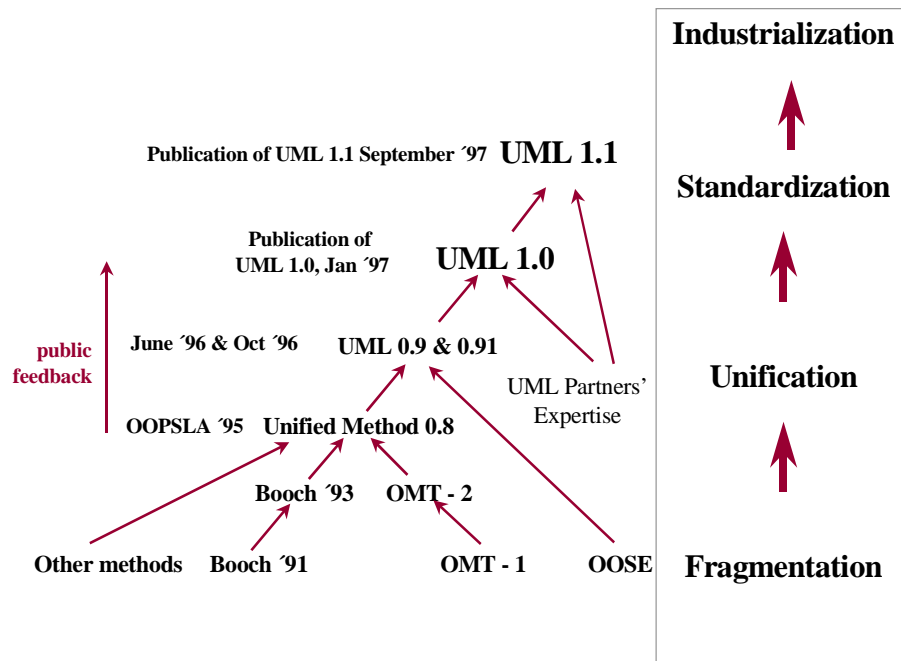
³ There is a second standard, called OPEN, that is supported by others in the community.

UML Scope (cont.)

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



from *UML Summary*

UML History (cont.)

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

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

Extensions & Variants (cont.)

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

Outline

Background

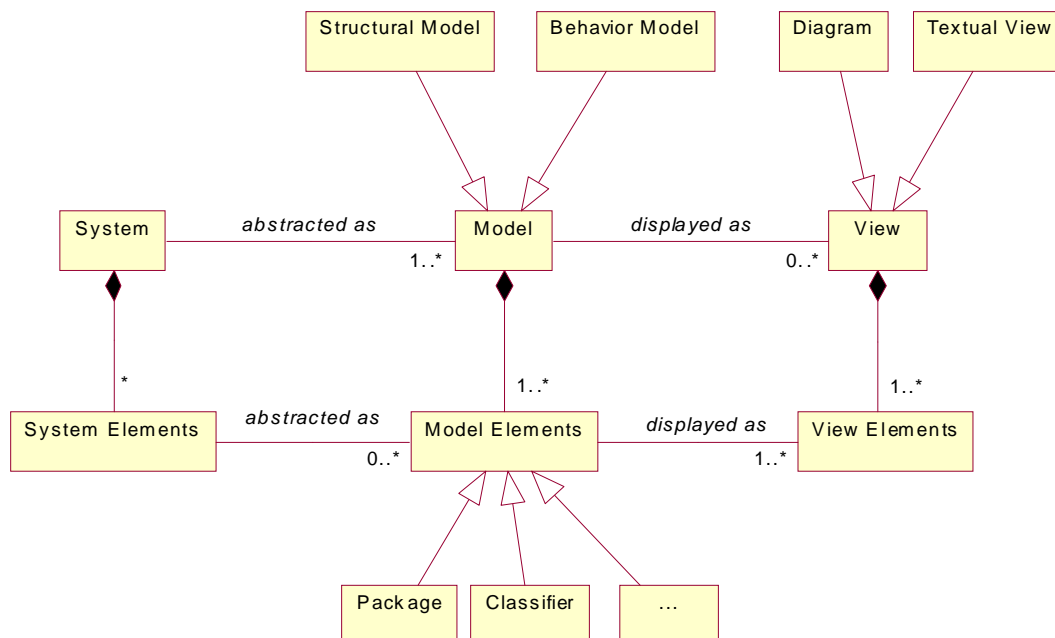
Key Concepts

Method Overview

Summary

Models & Views

UML Representation of A System



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

Structural Models

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 & tables⁴

⁴ Format of these forms and tables is not defined by UML.

Behavioral Models

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 & tables⁵

⁵ Format of these forms and tables is not defined by UML.

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*

Outline

Background

Key Concepts

Method Overview

Summary

Method Overview

Outline

Problem Description

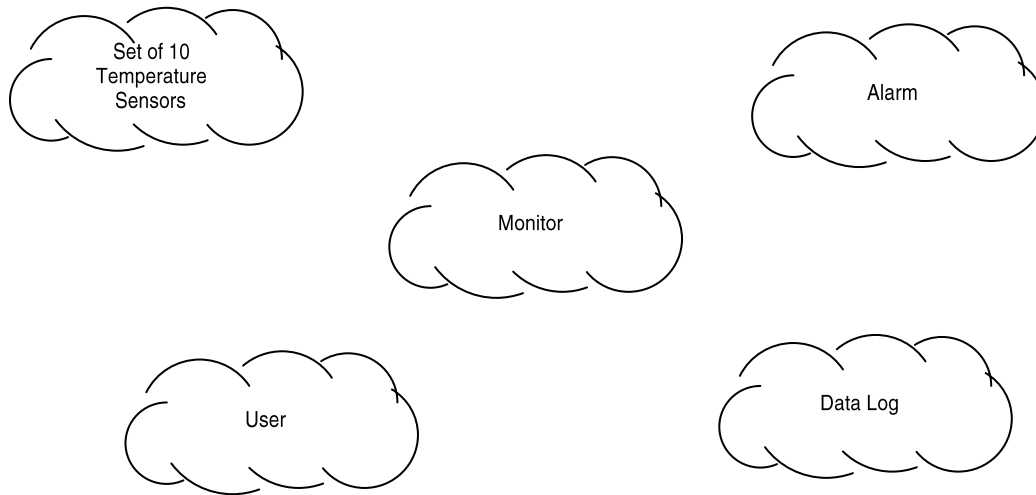
Process Overview

Planning & Elaboration

Building Cycle 1

Case Study Problem Description

Customer's Concept



Case Study Problem Description (cont.)

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.

Description⁶

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.

⁶ 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

Outline

Problem Description

Process Overview

Planning & Elaboration

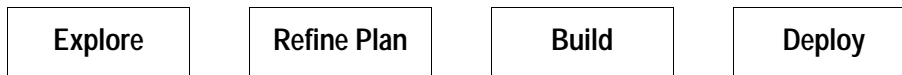
Building Cycle 1

Process

Evolutionary development recommended

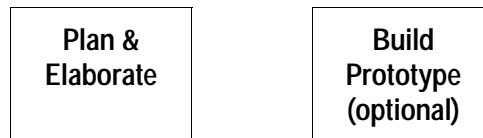


Activities within each evolutionary cycle



Process (cont.) Evolutionary Cycle Activities Details

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

Process (cont.) 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

Outline

Problem Description

Process Overview

Plan & Elaborate

Building Cycle 1

Plan & Elaborate Activity

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

Plan & Elaborate (cont.) Define Requirements

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

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

System Functions (cont.)

Statistics Gathering Functions

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

System Functions (cont.)

Profile Management Functions

Requirement			Details & Constraints		
No.	Statement	Category	Type	Description	Category
R3.1	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.	Hidden			

Initialization Functions

Requirement			Details & Constraints		
No.	Statement	Category	Type	Description	Category
R4.1	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.	Hidden			

Questions For Customer/System Engineers

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

#	Question	Questioner	Date	Applies To	Response	Responder	Response Date
Q1	Are the sensors intelligent?	R1.1, R1.2, R1.3
Q2	Number of fault lights?	R2.1.2	H/W limits
Q3	Turn off lights?	Problem stmt.	Yes
Q4	Conditions for Detecting Faulty Sensors?	R1.4.2	Anytime detect
Q5	Number of limit lights?	R2.1.2	H/W limits
Q6	What period should sensors be tested?	R2.1.2	5 seconds
Q7	What period should read profiles	R4.1	30 minutes

Questions For Customer/System Engineers (cont.)

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

Questions For Customer/System Engineers (cont.)

Field	Description
Question ID	Q8
Question:	How do we know if a sensor value has been written?
Originator:	...
Date:	...
Applies To:	R2.3
Notes:	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
Response Date:	...
Responder:	Fred the System Engineer
Response:	Write a 16-bit -1 (16#FF#) to the memory location after reading it.
Part Of	
Parts:	
Original Questions:	Q1
Derived Questions:	

Draft System Architecture

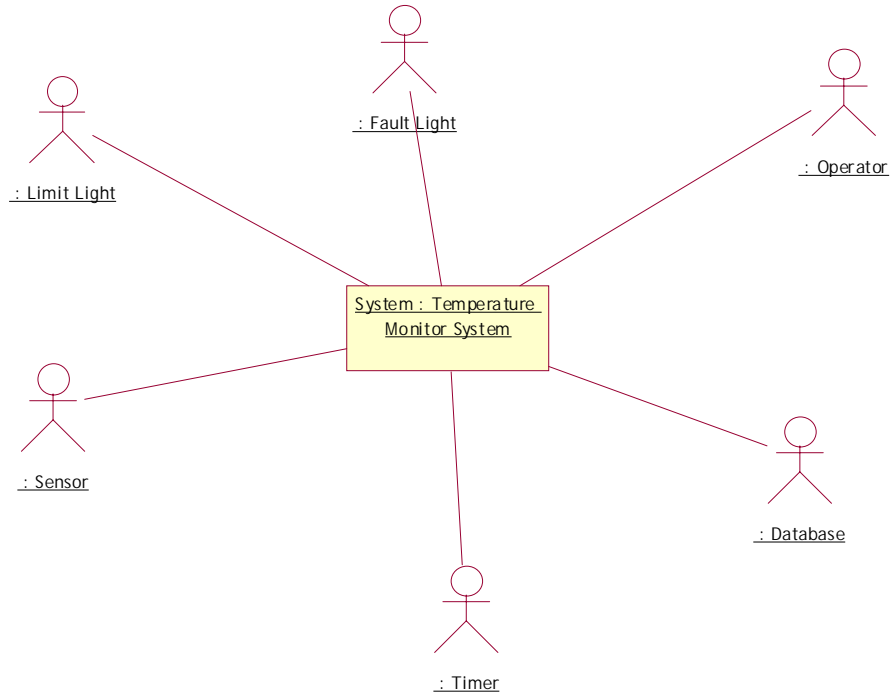
Purpose:

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

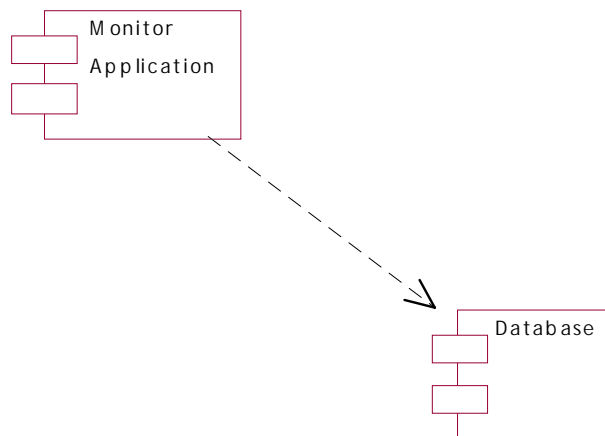
Artifacts:

- Collaboration Diagram
- Run-time Component Diagram
- Deployment Diagram
- Glossary entries

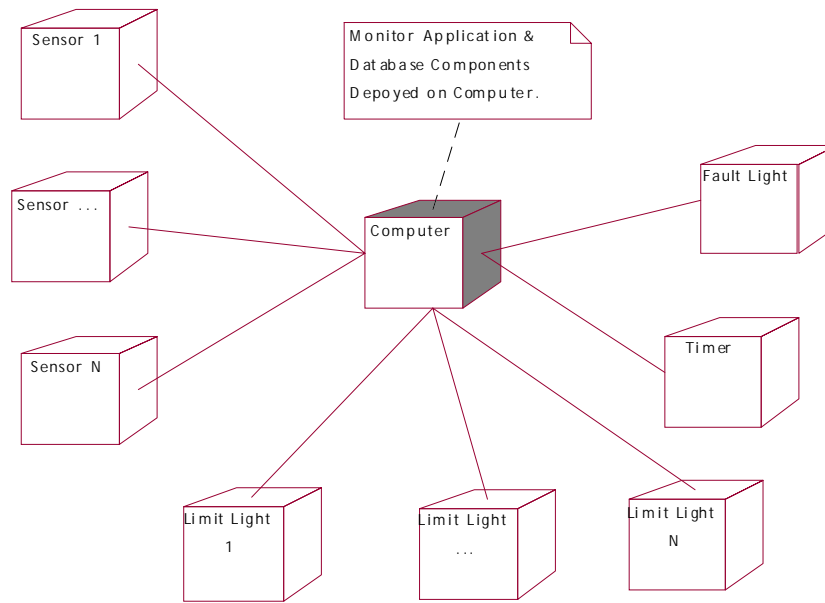
Temperature Monitoring System Architectural Context



Temperature Monitoring System Run-time Component Diagram



Temperature Monitoring System Deployment Diagram



Note: In UML, component icons are normally drawn in processor nodes; but tool didn't support, so I used a Note.

Define Use-Cases

Purpose:

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

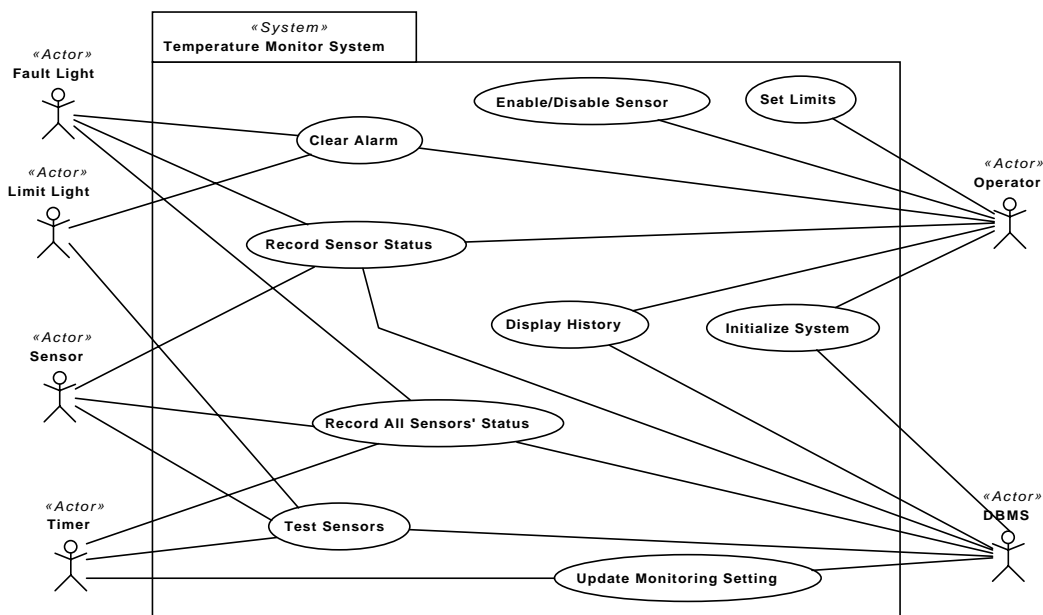
Artifacts:

- Use-cases
 - Summary
 - High-level descriptions of each
 - Extended descriptions of “interesting” ones
- Use-case Diagram(s)

Temperature Monitoring System Use-Cases

Name	Purpose	Importance	Requirements
Enable/Disable Sensors	Set the monitoring state of a sensor.	Primary	R1.1, R1.2
Set Limits	Set the upper & lower temperature bounds, which cause an alarm if the sensor value exceeds.	Primary	R.1.3
Test Sensors	Test whether any sensors are out of limits.	Primary	R1.4.1, R.1.4.2, R2.4
Clear Alarm	Turn off selected alarm lights.	Primary	R1.5
Record Sensor Status	Record the status of a sensor on operator command.	Primary	R2.2, R2.3, R2.4
Record All Sensors' Status	Periodically record the status of all sensors.	Primary	R2.1, R2.3, R2.4
Display History	Display the history of events & sensor statuses	Primary	R2.5, R2.6
Update Monitoring Settings	Set the monitoring state & limits for each sensor based on stored profile	Secondary	R3.1
Initialize System	Initialize the system on startup/reboot	Primary	R4.1

Use-Case Diagram



High-Level Descriptions⁷

Use Case Name	Enable/Disable Sensor
Purpose	Set the monitoring state of a sensor.
Actors	Operator
Importance	Primary
Overview	Operator selects a sensor and sets it's state to enabled or disable.
Requirements	R1.1, R1.2
State	High-Level Conceptual

Use Case Name	Set Limits
Purpose	Set the upper & lower temperature bounds, which cause an alarm if the sensor value exceeds.
Actors	Operator
Importance	Primary
Overview	Operator selects a sensor and sets it's upper & lower bounds.
Requirements	R1.3
State	High-Level Conceptual

Etc.

⁷ 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

Use Case Name	Record All Sensors' Status
Purpose	Periodically record the status of all sensors.
Actors	Timer, Sensor, Fault Light, DBMS
Importance	Primary
Overview	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 & limits, and time stamp.
Requirements	R2.1, R2.3, R2.4
State	Essential
Uses	

Typical Course of Action

	Actor Actions	System Response
1.	Timer signal occurs	
2		[time to record all sensors' status] for each sensor
2.1		[Sensor is enabled] read sensor value.
2.2		[Value is available] store status (name, value, state, limits, and time stamp) in database.

Use-Case Expanded/Essential Description (cont.)

Alternate Course of Action: *Disabled Sensor*

	Actor Actions	System Response
2.1		[Sensor is disabled] store status (name, state, and time stamp) in database.
2.2		Skip

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

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

Define Draft Conceptual Model

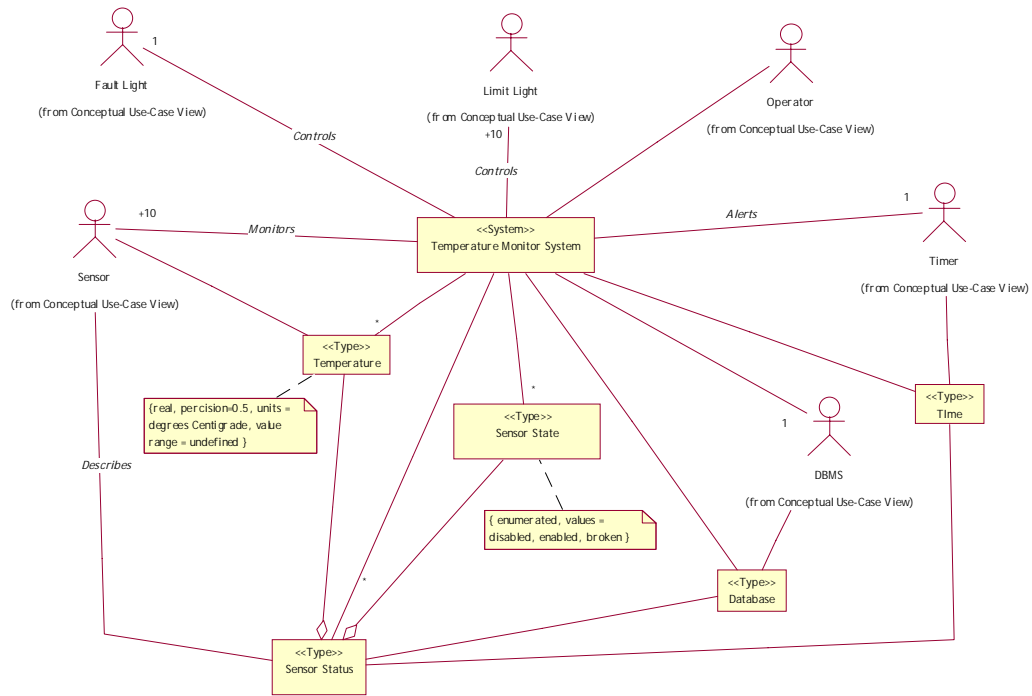
Purpose:

- 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

Plan & Elaborate – Define Conceptual Model
Temperature Monitoring System Conceptual Model



Plan & Elaborate
Record Terms in Glossary⁸

Purpose

- Gather definitions of terms for easy lookup

Example

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

⁸ 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
- Budget⁹
- Schedule⁹

⁹ Not part of this course

Plan & Elaborate – Planning (cont.)

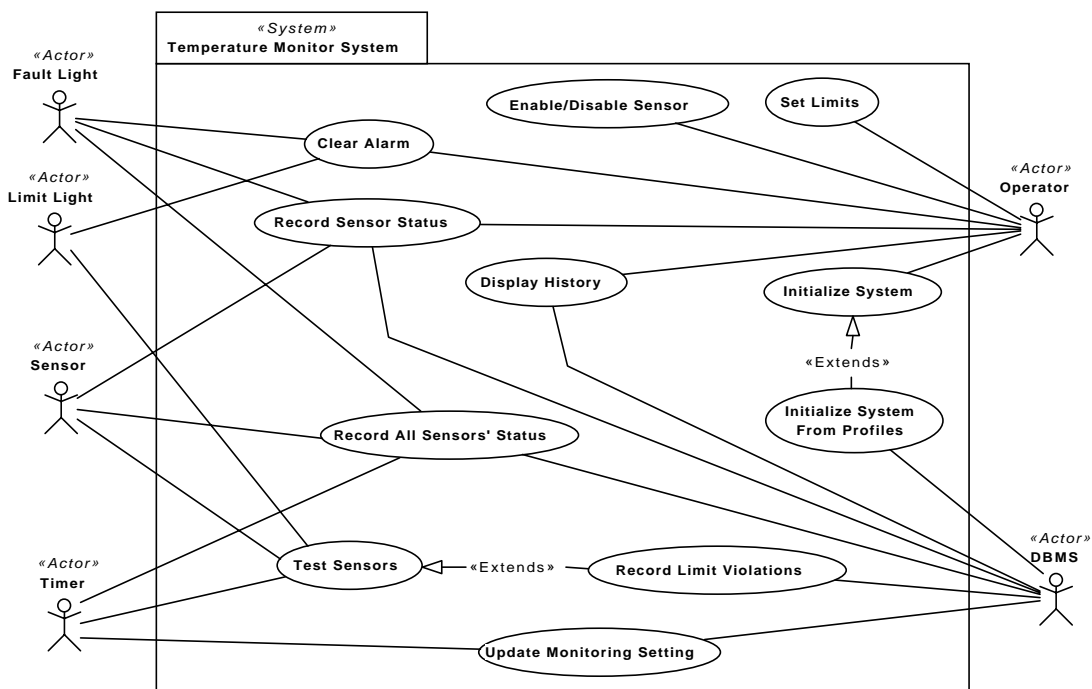
**Temperature Monitor System Use-Case/Requirement
Ranking**

Use-Case Name	Requirements	Rank	Justification
Test Sensors	R1.4.1, R.1.4.2, R2.4	Highest	Fundamental capability of system. Significant impact on architecture. Significant insight with relatively small effort.
Initialize System	R4.1	High/Low	Basic initialization is absolutely necessary, Initialization from profiles is low priority
Enable/Disable Sensors	R1.1, R1.2	High	Basic capability of system. Significant impact on architecture (UI).
Set Limits	R.1.3	High	Basic capability of system. Significant impact on architecture (UI).
Clear Alarm	R1.5	High	Basic capability of system. Significant impact on architecture (UI).
Record Sensor Status	R2.2, R2.3, R2.4	Medium	Significant impact on architecture. Supports increased revenue.
Record All Sensors' Status	R2.1, R2.3, R2.4	Medium	Significant impact on architecture. Supports increased revenue.
Display History	R2.5, R2.6	Low	Significant impact on architecture (UI). Short-term work around.
Update Monitoring Settings	R3.1	Low	Option of system

Temperature Monitor System Planned Cycles

Cycle	Name	Requirements	Rank	Notes
1	Test Sensors	R1.4.1, R.1.4.2	Highest	No database support Basic Initialization Only
1	Initialize System		High	
2	Enable/Disable Sensors	R1.1, R1.2	High	Provides basic temperature monitoring capability
2	Set Limits	R.1.3	High	
2	Clear Alarm	R1.5	High	
3	Record Limit Violations	R2.4		Adds basic statistics monitoring
3	Record Sensor Status	R2.2, R2.3, R2.4	Medium	
3	Record All Sensors' Status	R2.1, R2.3, R2.4	Medium	
4	Display History	R2.5, R2.6	Low	Final product. Add profile options & program display of history.
4	Update Monitoring Settings	R3.1	Low	
4	Initialize System From Profiles	R4.1	Low	

Revised Use-Case Diagram



Outline

Problem Description

Process Overview

Planning & Elaboration

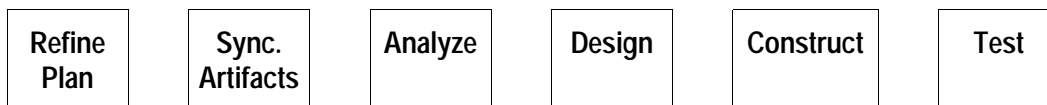
Building Cycle 1

Build Activity

Iterative development recommended



In each cycle



- 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

Build

Refine Plan

Synchronize Artifacts

Analyze

Design

Construct¹⁰

Test¹⁰

¹⁰ Not covered in this course

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

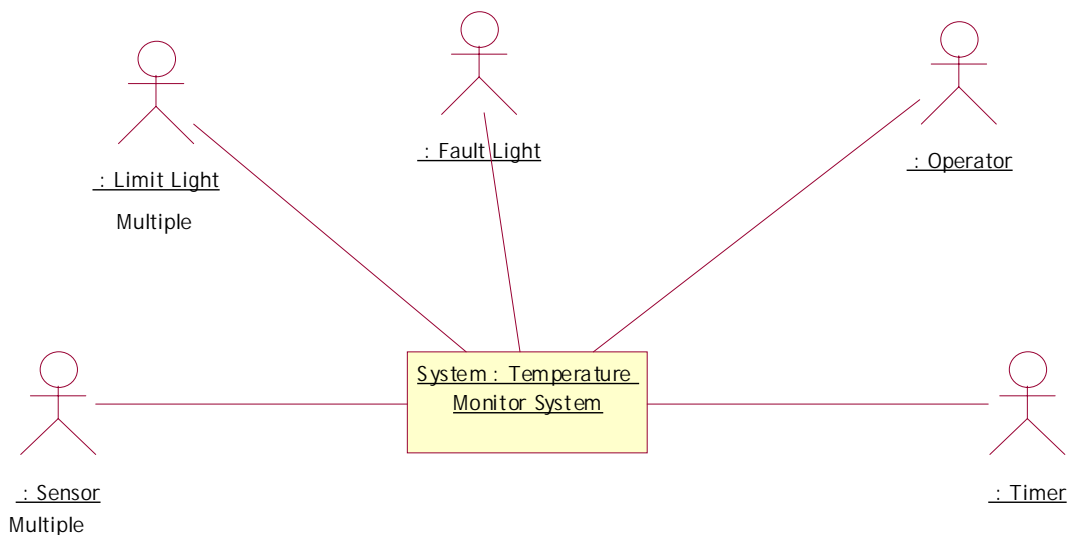
Purpose:

- 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



Define Use-Cases

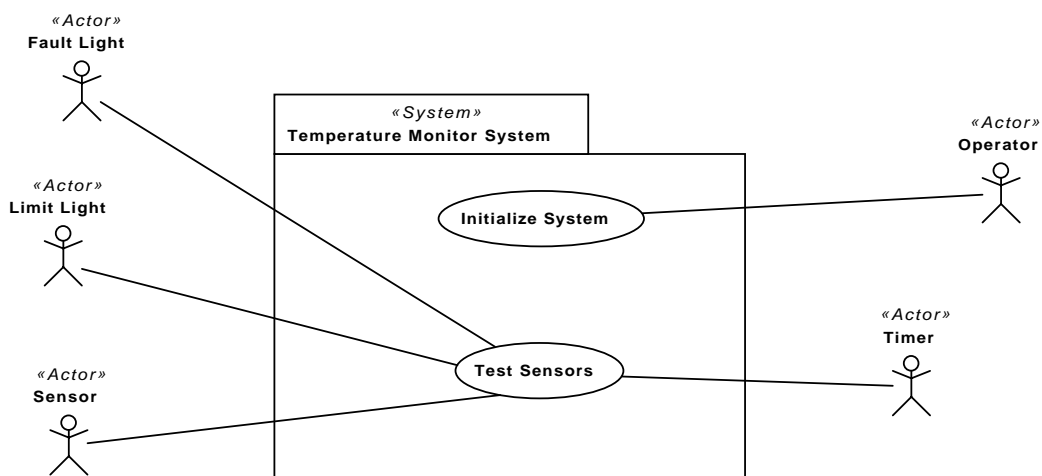
Purpose:

- 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



Use-Case Expanded/Essential Description

Use Case Name	Test Sensors
Purpose	Test whether any sensors are out of limits.
Actors	Timer, Sensor, Fault Light, Limit Light
Importance	Primary
Overview	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.
Requirements	R1.4.1, R.1.4.2, R2.3 (see Q4)
State	Essential
Uses	

Typical Course of Action

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

Use-Case Expanded/Essential Description (cont.)

Alternate Course of Action: Value is Out Of Limits

	Actor Actions	System Response
2.4		[Value is in limits] turn on limit light for sensor

Exceptional Course of Action: Sensor May be Broken

	Actor Actions	System Response
2.2		[Value is not available]
2.3		[This is second attempt] turn on fault light

Define Conceptual Model

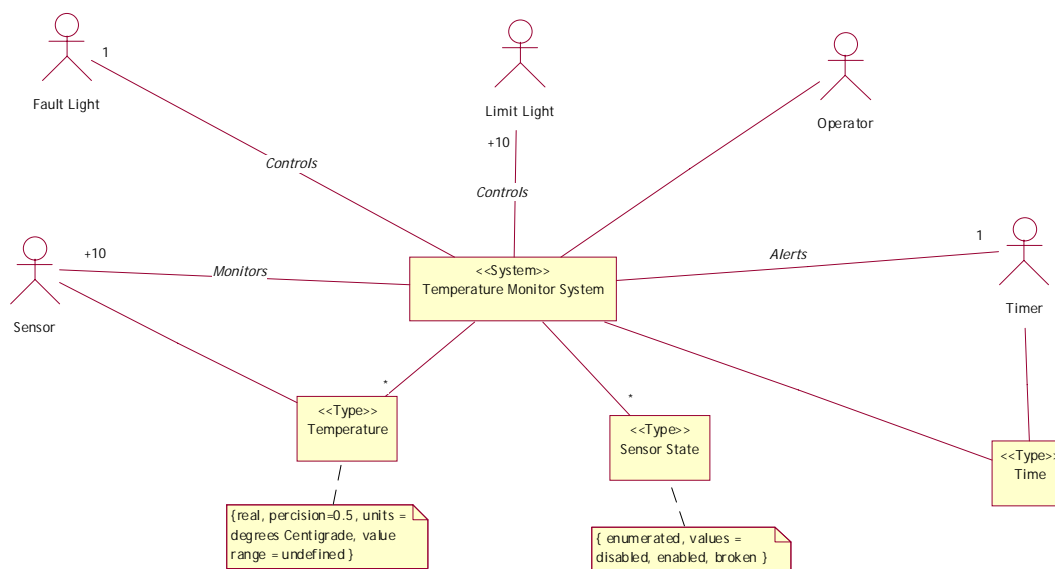
Purpose:

- 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



Define System Behavior

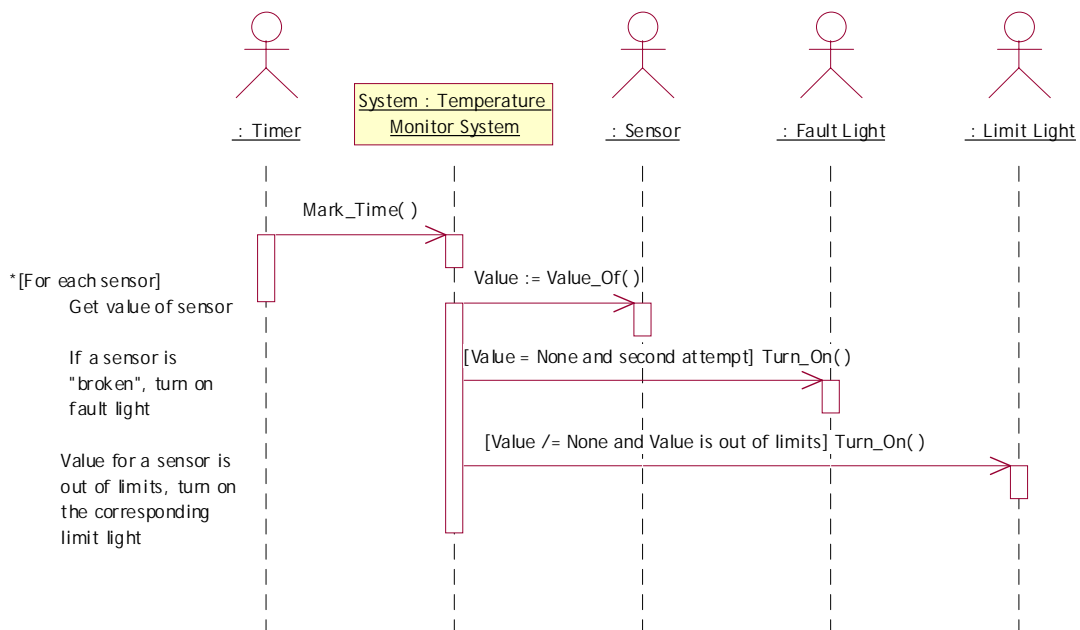
Purpose:

- 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

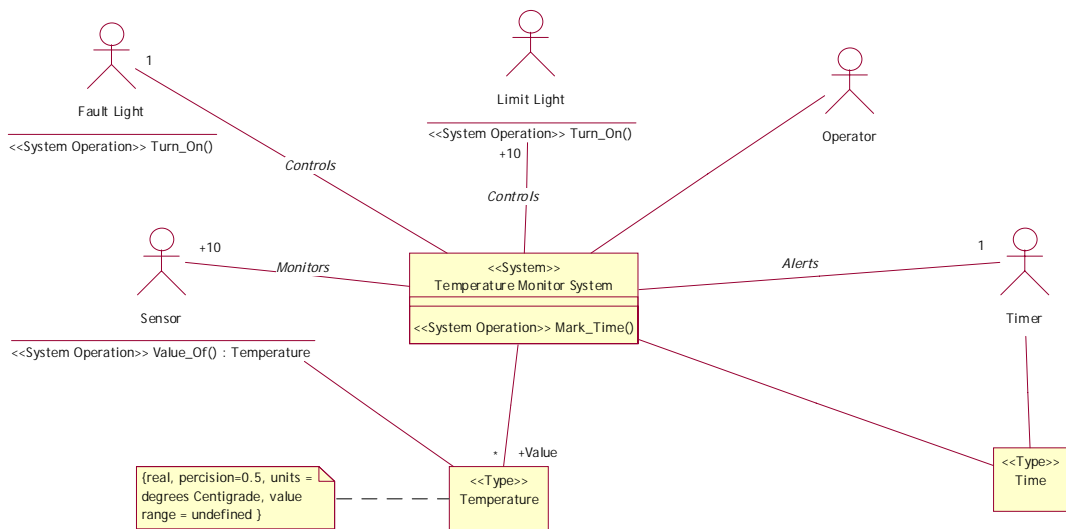
Temperature Monitor System Test Sensor Sequence Diagram



Temperature Monitor System — Mark_Time Operation Contract

Name:	Mark_Time ()
Responsibilities:	Read sensors and verify that they are working and in limits
Requirements:	R1.4.1, R.1.4.2, R2.3 (see Q4)
Use-Cases:	Test_Sensor
Notes:	Since this is called periodically, must complete operation within period (5 seconds)
Exceptions:	None
Output:	None
Pre-conditions:	None
Post-conditions:	<ul style="list-style-type: none"> [For each Sensor that return value of "None"] a flag indicates that last read failed

Temperature Monitor System — Revised Conceptual Model



Build 1
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

Build 1 – Design (cont.)
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

Temperature Monitor System — Test Sensor Real Use-Case Description

Use Case Name	Test Sensors
Purpose	Test whether any sensors are out of limits.
Actors	Timer, Sensor, Fault Light, Limit Light
Importance	Primary
Overview	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.
Requirements	R1.4.1, R.1.4.2, R2.3 (see Q4)
State	Real
Uses	

Typical Course of Action

	Actor Actions	System Response
1.	Timer signal occurs	
2		For each sensor
2.1		Read sensor value from sensor's memory location
2.2		[Value != "None" value] test the value against current limits
2.3		[Value is in limits] Write "None" value to sensor's memory location (16#FF# -- see Q8)
2.4		Set Last_Value to value read

Test Sensor Real Use-Case Description (cont.)

Alternate Course of Action: Value is Out Of Limits

	Actor Actions	System Response
2.5		[Value is in limits] turn on limit light for sensor by writing to light's port

Exceptional Course of Action: Sensor May be Broken

	Actor Actions	System Response
2.2		[Value = "None" value]
2.3		[Last_Value = "None" value] turn on fault light by writing to light's port

Design Component Interactions

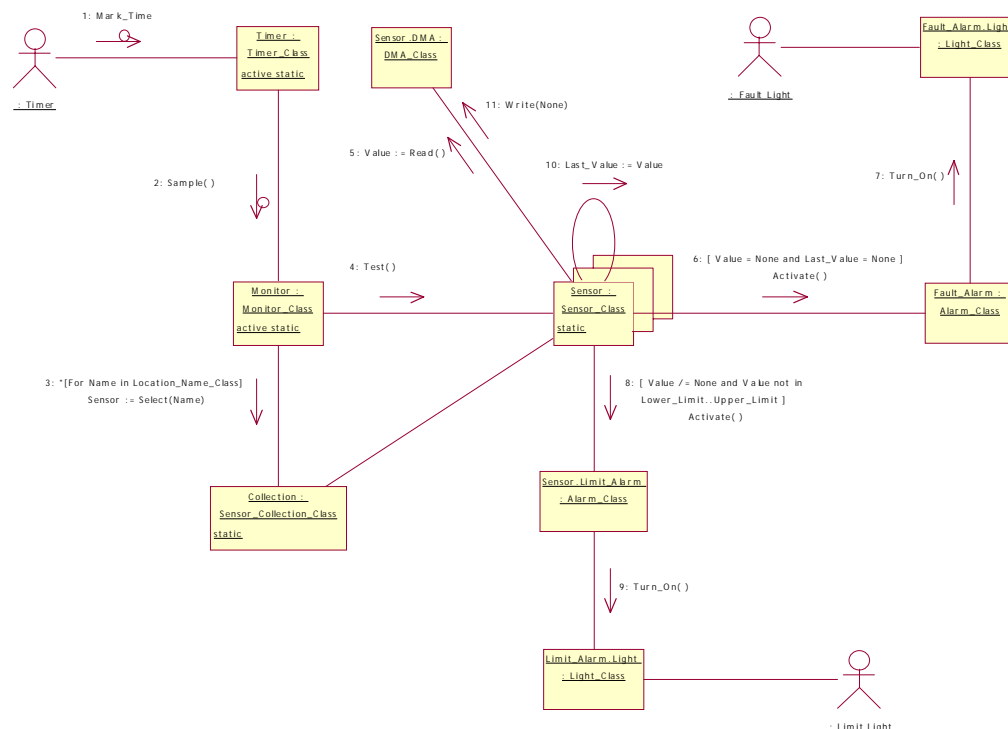
Purpose:

- 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

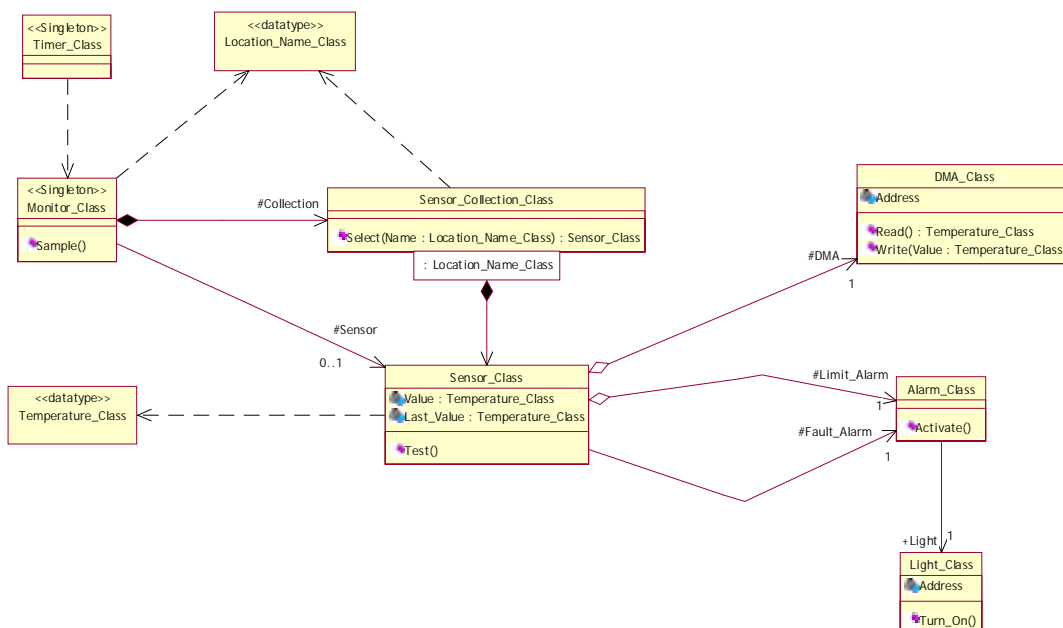
Purpose:

- 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

Temperature Monitor System — Class Diagram



Define Component Behavior

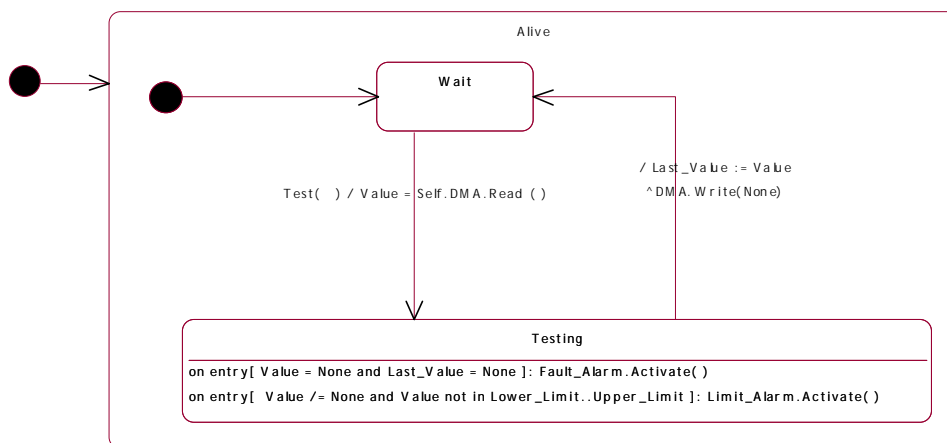
Purpose:

- 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



Build 1
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

Build 1 – Construction (cont.)
Refine Class Model

Purpose:

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

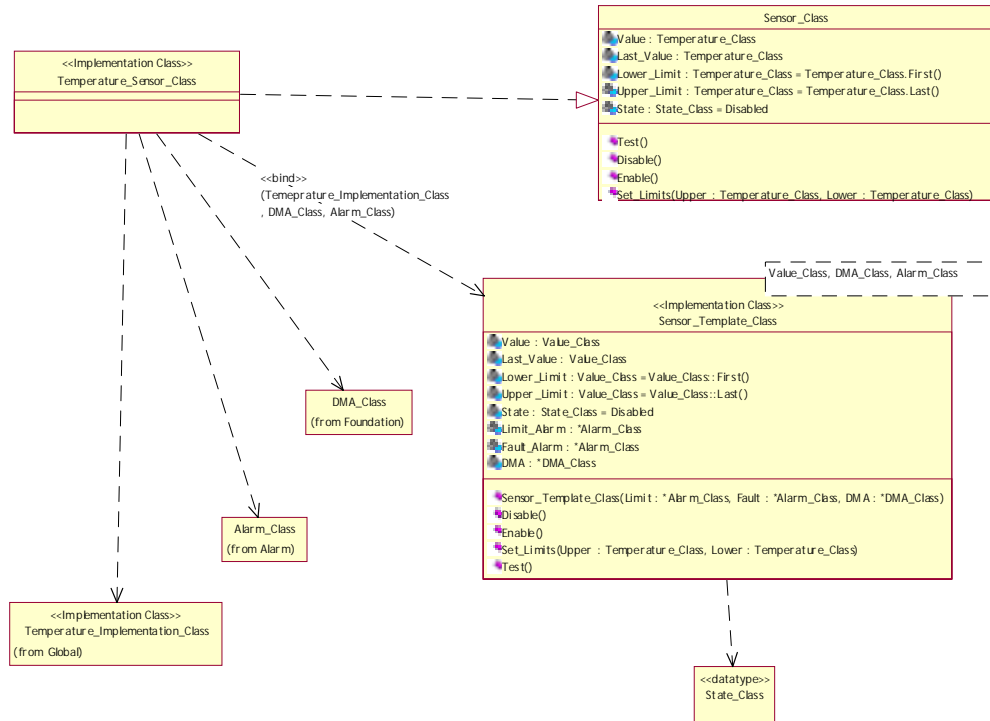
Artifacts:

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

Note:

- May update Design of Components & Interactions
- May update Design of Component Behavior

Temperature Monitoring System – Sensor Package



Design Components

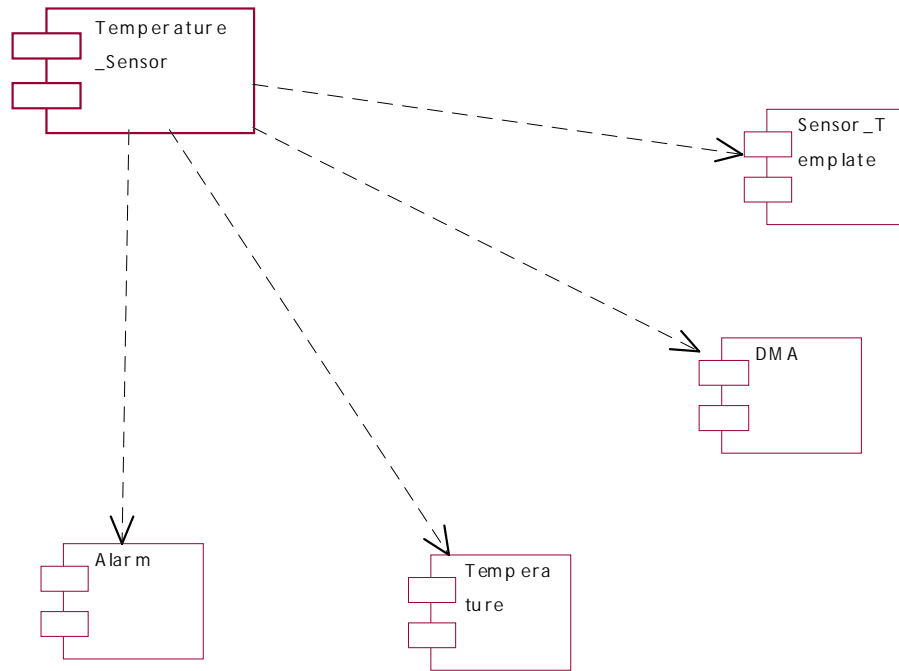
Purpose:

- 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”

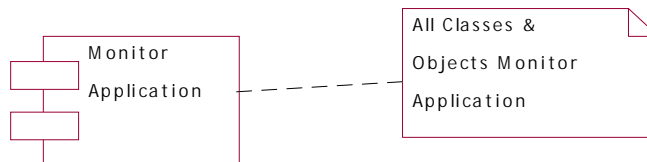
Artifacts:

- Component Diagram(s)
- Glossary entries

Temperature Monitoring System – Source Code: Sensor Package



Temperature Monitoring System Run-time Component Diagram



Note: In UML, object icons are normally drawn in appropriate components; but tool didn't support, so I used a Note.

Design Run-Time Deployment

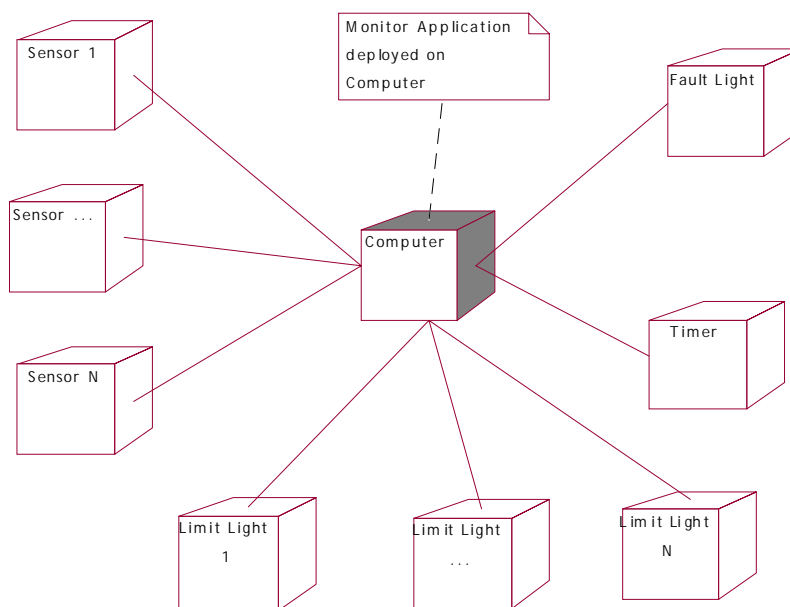
Purpose:

- 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



Outline

Background

Key Concepts

Method Overview

Summary

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

UML Is

An industry standard¹¹

- 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

¹¹ Others in the community support a second standard, "OPEN".

UML Definition Is

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

- Abbott, R. J. (1983). "Program Design by Informal English Descriptions", *Communications of the ACM*. Vol. 26, No. 71: pp. 882–895.
- Booch, G. (1983). *Software Engineering with Ada*, First ed., Benjamin/Cummings: Menlo Park, CA.
- Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*, Second ed. (previous edition entitled *Object-Oriented Design with Applications*), Benjamin/Cummings: Menlo Park, CA.
- Colbert, E. (1997). *Object-Oriented Software Development Method*, Absolute Software Co., Inc.: Los Angeles, CA.
- Fowler, M. (1997). *UML Distilled*, Addison Wesley Longman, Inc.: Reading, MA.
- Gilb, T. (1988). *Principles of Software Engineering Management*, Addison-Wesley: Reading, MA.
- Harel, D. (1987). "Statecharts: A Visual Formalism for Complex Systems", *Science of Computer Programming*, No. 8: pp. 231–274.
- Harel, D. (1988). "On Visual Formalisms", *Communications of the ACM*. Vol. 31, (May): pp. 514–530.
- Intermetrics, I. (1994). *Ada Reference Manual (INTERNATIONAL STANDARD ISO/IEC 8652:1995(E))*, Intermetrics, Inc.: Boston, MA.
- Larman, C. (1998). *Applying UML and Patterns*, Prentice Hall PTR: Upper Saddle River, New Jersey.
- Rational Software Corporation (1997). *Object Constraint Language Specification*, 1.1 ed., Unified Modeling Language (UML) Specification: Santa Clara, CA.
- Rational Software Corporation (1997). *UML Extension for Business Modeling*, 1.1 ed., Unified Modeling Language (UML) Specification: Santa Clara, CA.
- Rational Software Corporation (1997). *UML Extension for Objectory Process for Software Engineering*, 1.1 ed., Unified Modeling Language (UML) Specification: Santa Clara, CA.

References (cont.)

Rational Software Corporation (1997). *UML Notation Guide*, 1.1 ed., Unified Modeling Language (UML) Specification: Santa Clara, CA.

Rational Software Corporation (1997). *UML Semantics*, 1.1 ed., Unified Modeling Language (UML) Specification: Santa Clara, CA.

Rational Software Corporation (1997). *UML Summary*, 1.1 ed., Unified Modeling Language (UML) Specification: Santa Clara, CA.

Ross, D. T., J. B. Goodenough, et al. (1975). "Software Engineering Process, Principles, and Goals", *IEEE Computer* (May): pp. 17–27.

Rumbaugh, J. (1996). *OMT Insights*, SIGS Books: New York.

Rumbaugh, J., M. Blaha, et al. (1991). *Object-Oriented Modeling and Design*, Prentice-Hall: Englewood Cliffs, NJ.

Stroustrup, B. (1997). *The C++ Programming Language*, Third ed., Addison-Wesley: Reading, MA.