CLEMSON UNIVERSITY

SMART THERMOSTAT SYSTEM
Software Architecture Document (SAD)

CONTENT OWNERS: Siri Uday Shastry and Sravani Chatrathi

DOCUMENT NUMBER:	**RELEASE/REVISION:**	**RELEASE/REVISION DATE:**
• 1	• 1.0	• 04/25/2017

Table of Contents

List of Figures

Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 ("Document Management and Configuration Control Information") explains revision history. This tells you if you're looking at the correct version of the SAD.

- Section 1.2 ("Purpose and Scope of the SAD") explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you're seeking is likely to be in this document.

- Section 1.3 ("How the SAD Is Organized") explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.

- Section 1.4 ("Stakeholder Representation") explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.

- Section 1.5 ("Viewpoint Definitions") explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 ("Views"). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.

- Section **Error! Reference source not found.** ("**Error! Reference source not found.**") explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

- Revision Number: 1.0

- Revision Release Date: 25/4/2017
- Purpose of Revision: To create a document
- Scope of Revision: This document describes the aspects of Smart Thermostat System design that are considered to be architecturally significant; that is, those elements and behaviors that are most fundamental for guiding the construction and working of a Thermostat and for understanding this project as a whole. Stakeholders who require a technical understanding of Thermostat are encouraged to start by reading this document, then reviewing the thermostat use cases and the architectural model, and then by reviewing the source code.

1.2 Purpose and Scope of the SAD

This SAD specifies the software architecture for Thermostat. This document describes the aspects of Thermostat design that are considered to be architecturally significant; that is, those elements and behaviors that are most fundamental and also required for understanding this project as a whole. All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and po-

[1] Here, a system may refer to a system of systems.

diatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

This SAD is organized into the following sections:

- **Section 0 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 ("Architecture Background") explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (Views") and Section 4 ("Relations Among Views") specify the software architecture**. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections** Error! Reference source not found. **("**Error! Reference source not found.**") and 6 ("Directory") provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5.

1.5 Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stake-holder's concerns**
Architect	Module Decomposition Viewpoint
Developer	Reliability Viewpoint
Tester	Accuracy and safety assurance Viewpoint
User	Temperature settings Viewpoint

1.5.1 Component and Connector(Communications) Viewpoint Definition

1.5.1.1 Abstract

Views conforming to the component and connector viewpoint categories the system into Executing components and their interactions among them i.e the data movement.

1.5.1.2 Stakeholders and Their Concerns Addressed

• project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules

• testers and integrators who ensure that data is flown correctly through modules

• system build engineers who use the elements to produce a running version of the system;

- maintainers, who are tasked with modifying the software elements;
- implementers, who are required to implement the elements;
- software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;

• the user, who is concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.5.1.3 Elements, Relations, Properties, and Constraints

Elements of the Component connector (communication) viewpoint are modules, which are units of implementation that provide defined functionality.

1.5.1.4 Language(s) to Model/Represent Conforming Views

Views conforming to the communication viewpoint may be represented by (a) plain text using indentation or outline form (b) UML diagrams (c) Use case Diagrams

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Completeness/consistency criteria include (a)Major executing components (b) Major data stores. Applicable evaluation/analysis techniques include (a)Change in system structures as it executes (b) Parts of the system running in parallel

1.5.1.6 Viewpoint Source

It is about the Communication and data flow associated with different modules in the pacemaker system.

2 Architecture Background

2.1 Problem Background

To design a working pacemaker architecture for installing it into the heart.

2.1.1 System Overview

The system should contain all the systems and devices required. It should also satisfy all functional requirements and make sure that security is enhanced between them.

2.1.2 Goals and Context

Verify that all functional requirements, quality attribute requirements, and design constraints assigned to the parent element have been allocated to one or more child elements in the decomposition. Translate any responsibilities that were assigned to child elements into functional requirements for the individual elements.

2.1.3 Significant Driving Requirements

- There are always too many requirements for the team to consider. The driving requirements are those things that are most important to the most important stakeholders. The output of QAW involves a list of architectural drivers, the raw scenarios, the prioritized list of raw scenarios, the refined scenarios.

2.2 Solution Background

The architecture satisfies all the requirements. The quality attribute workshop helps in satisfying both functional and non-functional requirements and all those requirements of the stakeholders.

2.2.1 Architectural Approaches

Evaluate the design against the architectural drivers. If necessary, use models, experiments, simulations, formal analysis, and architecture evaluation methods. Determine if there are any architectural drivers that were not considered. Evaluate alternative patterns or apply additional tactics, if the design does not satisfy the architectural drivers. Evaluate the design of the current element against the design of other elements in the architecture and resolve any inconsistencies. For example, while designing the current element, you may discover certain properties that must be propagated to other elements in the architecture.

2.2.2 Analysis Results

The services and properties required and are provided by the software elements in our design. These services and properties are referred to as the element's interface. Interfaces describe the PROVIDES and REQUIRES assumptions that software elements make about one another. Observe any information that is produced by one element and consumed by another. The interfaces from the perspective of different views. For example, a Module view will allow you to reason about information flow; a Concurrency view will allow you to reason about performance and availability; and a Deployment view will allow you to reason about security and availability.

2.2.3 Requirements Coverage

The functional requirements, quality attribute requirements, and design constraints assigned to the parent element have been allocated to one or more child elements in the decomposition. Translate any responsibilities that were assigned to child elements into functional requirements for the individual elements. The quality attribute requirements for individual child elements as necessary.

2.2.4 Summary of Background Changes Reflected in Current Version

- The elements will be instantiated and individual properties and structural relations they will possess

• The computational elements will be used to support the various categories of system use

• The elements will support the major modes of operation

• the quality attribute requirements have been satisfied within the infrastructure and applications

• The functionality is divided and assigned to software elements including that the functionality is allocated across the infrastructure and applications

2.3 Product Line Reuse Considerations

The modules mode can be reused according to the requirements.

3 Views

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?

- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?

- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)

- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?

- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Name of view	**Viewtype that defines this view**	**Types of elements and relations shown**		**Is this a module view?**	**Is this a component-and-connector view?**	**Is this an allocation view?**
Module view	Logical view			Yes	No	No
Component and Connector View	Data view and Process view			No	Yes	No
Allocation View	implementation and deployment view			No	No	Yes

3.1 Module View

3.1.1 View Description

Module views. The elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module views tells us about: (a) The primary functional responsibility assigned to each module. (b) The software elements a module is allowed to use (c)The software does it uses (d) The modules are related to other modules by generalization or specialization (i.e., inheritance) relationships

3.1.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

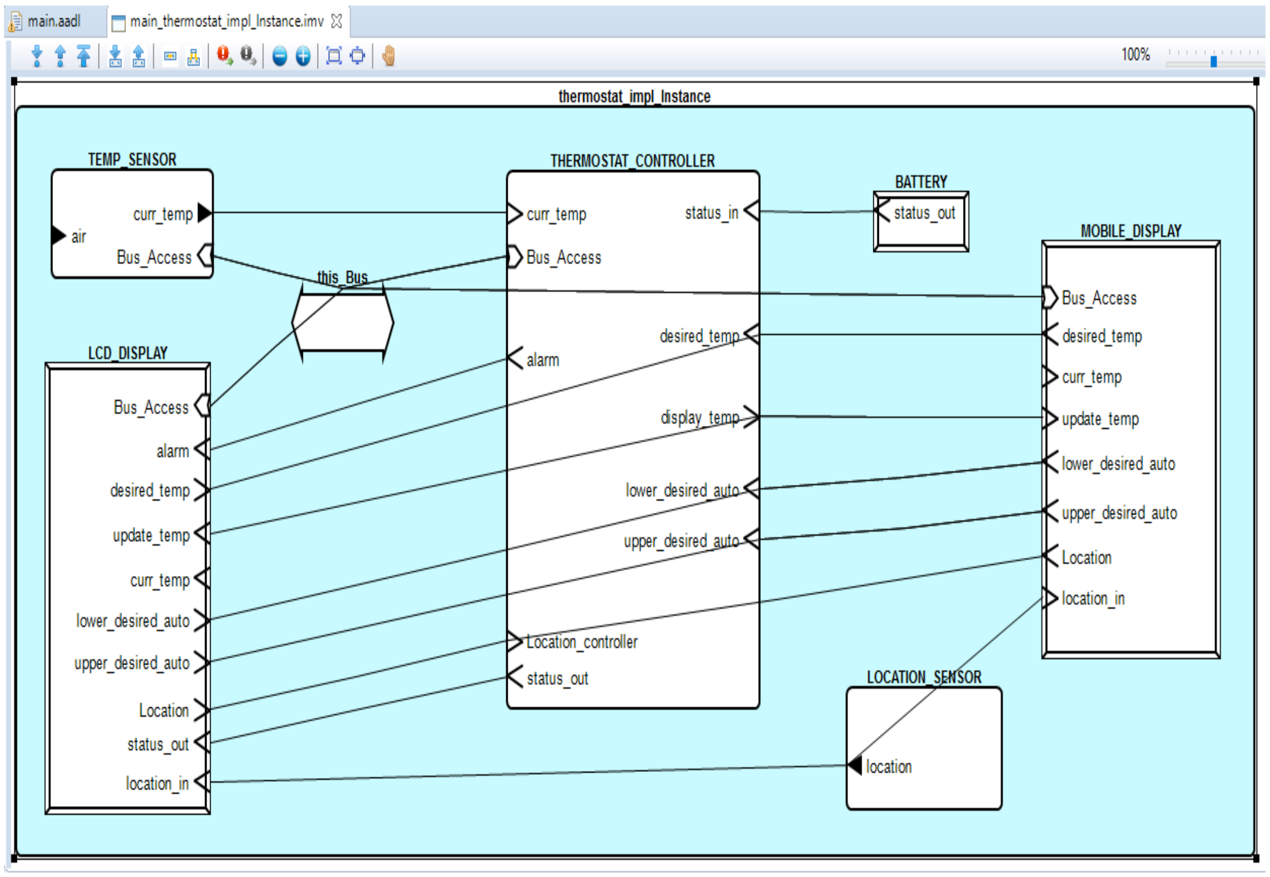

Fig 1 : Smart Thermostat System Architecture

3.1.3 Architecture Background

3.1.4 Variability Mechanisms

3.1.5 View Packets

3.1.5.1 View packet # Module View

3.1.5.1.1 Primary Presentation : It represents the Modules initially and the way they are decomposed later on for further convenience.

3.1.5.1.2 Element Catalog

3.1.5.1.2.1 Elements: Temperature sensor, LCD display, Bus, Thermostat controller, battery, Mobile display, Location sensor

3.1.5.1.2.2 Relations: Shown in the diagram above.

3.1.5.1.2.3 Interfaces: They are the once which allow interactions between different modules and components

3.1.5.1.2.4 Behavior: It shows both normal and erroneous behavior

3.1.5.1.2.5 Constraints:

When battery power is low, the device will not function as desired.

3.1.5.1.3 Context Diagram :

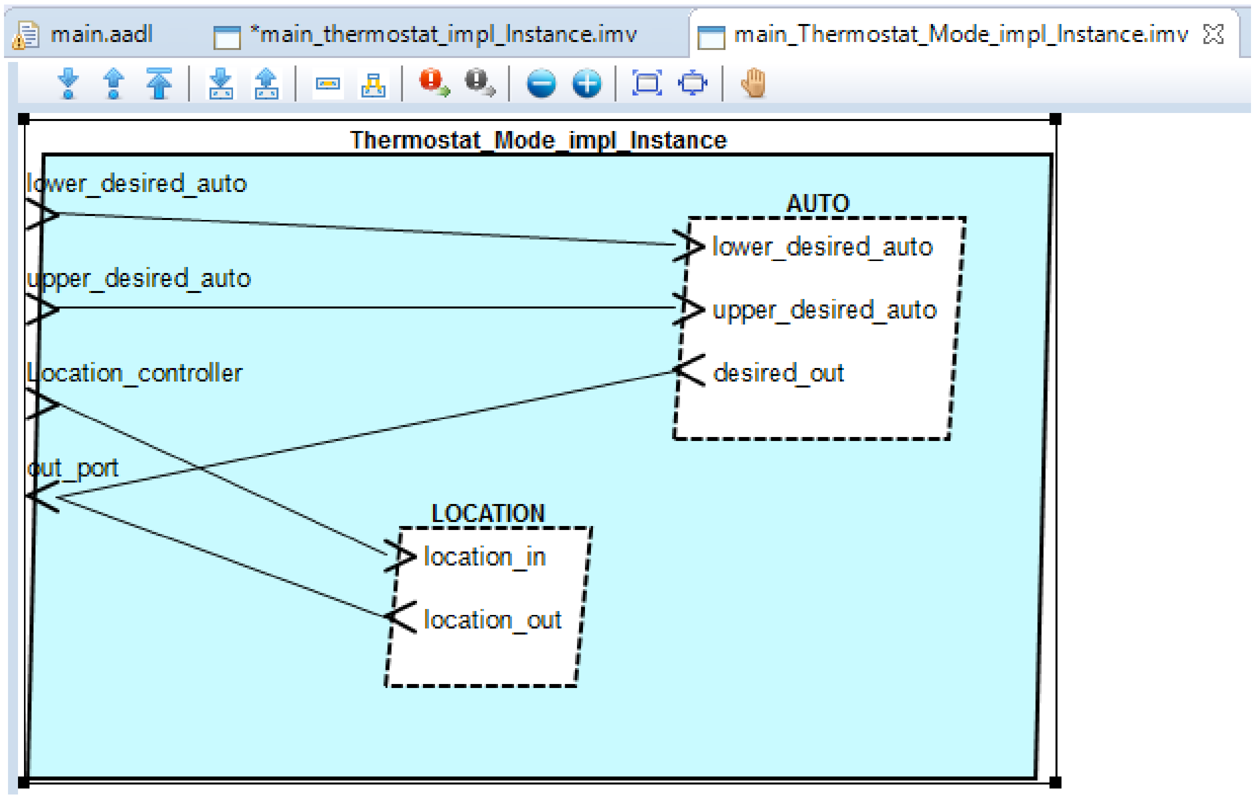

Fig 2 : Thermostat mode instance

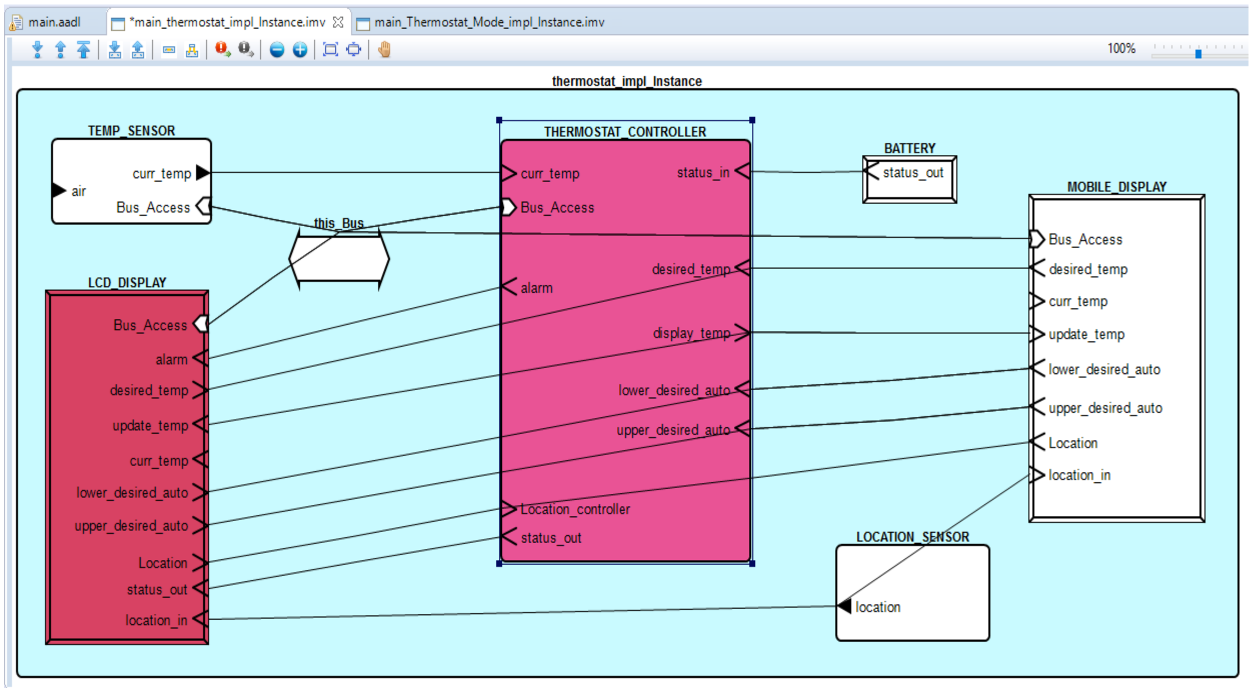

Fig 3 : Error flow from Thermostat controller to LCD display

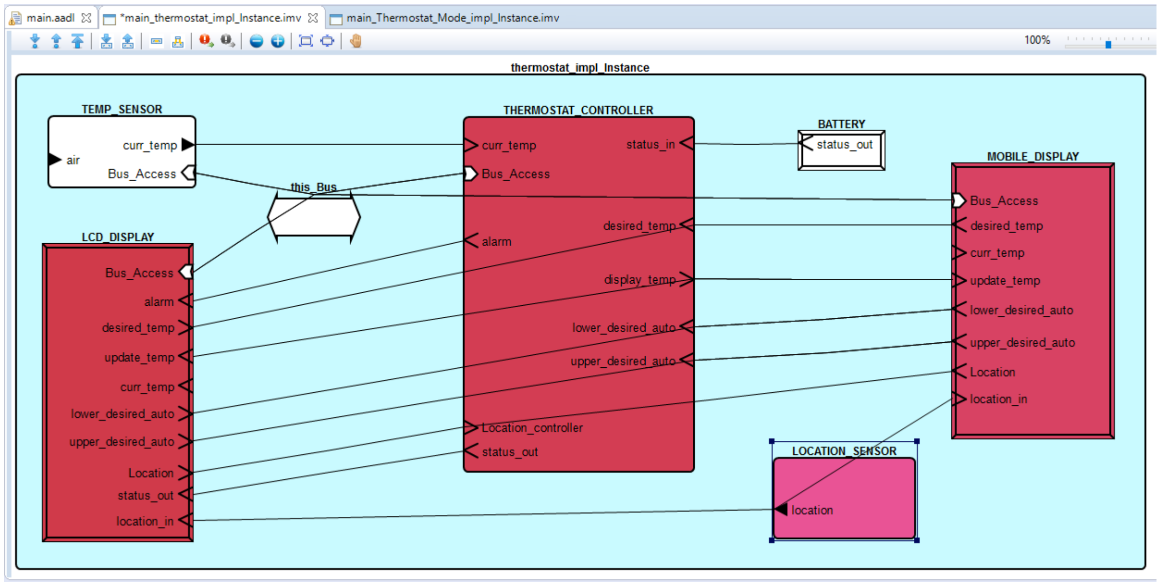

Fig 4 : Error flow for an error in the location sensor

3.1.5.1.4 Variability Mechanisms : The way the flow differs in few cases.

3.1.5.1.5 Architecture Background: The initial has been changes in many ways to enable all possible requirements and flows.

3.1.5.1.6 Related View Packets: The related view packets are component and connector views and allocation deployment structures.

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

4.1 General Relations Among Views

A module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego.

A component / connector view used to link several modules. It helps in communication between the modules.

Allocation view show the relationship between the software elements and elements in one or more external environments in which the software is created and executed.

4.2 View-to-View Relations

In general views are many-to-many relations. All the three views are interrelated to each other.

5 Referenced Materials

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice*, second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.
ACEEE 2016	Jen Robinson, Ram Narayanamurthy, Bienvenido Clarin, Christine Lee, Pranshu Bansal, *National Study of Potential of Smart Thermostats for Energy Efficiency and Demand Response, 2016*
ACM 2010	Jiakang Lu, Tamim Sookoor, Vijay Srinivasan, Ge Gao, Brian Holben, John Stankovic, Eric Field, Kamin Whitehouse, *The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes, 2010*
Matthew Burrough	Matthew Burrough, Jonathan Gill, *Smart Thermostat Security: Turning up the Heat*

6 Directory

6.1 Index

6.2 Glossary

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
View	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
Viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for

	a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
IEEE	Institute of Electrical and Electronics Engineers
OO	Object Oriented
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
UML	Unified Modeling Language