Architecture Description

Anomaly Management Platform for Manufacturing Plants

Software Platform for Detecting & Remedying Anomaly occurrences of Manufacturing Plants

May 2023

Revision History

Version	Date	Author	Revisions Made
0.8	4/05(Wed)	S.D. Kim	(Interim) Refining SRS & Context View
0.9	4/18(Tue)	S.D. Kim	(Pre-final) Skeleton Architecture and Viewpoints
1.0	**5/02(Tue)**	**S.D. Kim**	**(Final) Design for NFRs and Validation**

삼성전자 첨단기술아카데미

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Architecture Description of Anomaly Management Platform

1. Introduction

1.1. Purpose of the Document

The purpose of this document is to specify the architecture design for the target system. It describes all the essential architectural aspects of the target system including its structure, functional components, data components, their relationships, runtime behavior, and deployment.

1.2. System of Interest

The target system, Anomaly Management Platform for Manufacturing Plants, is a software planform that provides a comprehensive set of functionalities for detecting and handling various anomaly types in manufacturing plants.

- Anomaly in Manufacturing Domain

  An anomaly in the manufacturing domain refers to any deviation or irregularity in the performance of a machine or a process that is unexpected or differs from the normal behavior or performance.

- Key Functionality of the Anomaly Management Platform
  - Registering Manufacturing Plants
  - Registering Hardware Devices
  - Registering Use Profiles
  - Registering Anomaly Types
  - Registering Remedy Methods
  - Detecting Anomaly Occurrences
  - Remedying Anomaly Occurrences
  - Generating Anomaly Management Reports
- Process to Manage Anomaly Occurrences

  The typical process to manage anomaly occurrences is the following.
  - Step 1. Initialize the hardware devices and check for their connectivity.
  - Step 2. Identify the data sources that can be used to detect anomalies. These may include data from sensors, production equipment, quality control systems, and maintenance logs.
  - Step 3. Acquire data from the sources. The data acquisition is done by utilizing

effective data acquisition schemes such as pulling, pushing, register-notify-fetch, shared basket, and broadcasting.

- Step 4. Preprocess data to ensure accuracy and validness by applying data cleaning, normalization, and noises filtering.
- Step 5. Choose the appropriate method to detect anomaly occurrences. There can be several methods that can be used for anomaly detection including statistical methods such as Z-score and standard deviation, machine learning algorithms such as clustering and decision trees, and deep learning algorithms such as autoencoders.
- Step 6. Upon an anomaly occurrence, choose the appropriate metho to remedy the anomaly. There can be several methods that can be used to handle anomaly occurrences. A remedy method may utilize the installed actuators to control machines and their environment, communication schemes such as SMS messages, phone calls, and posting on bulletin boards, and alarming.
- Step 7. Apply the selected remedy method and check for the result of the remedy action. This is to execute the remedy method.
- Step 8. Check the result of the remedy action. If the anomaly is not successfully handled, then report the incident to the system operator.
- Step 9. Repeat steps 3 through 8.

- Deployment of the System

  The platform is developed as a programming library, which can be imported in application programs.

1.3. Definitions, Acronyms, and Abbreviations

- Manufacturing Plant

  A manufacturing plant is a facility or building where raw materials are transformed into finished target products through a series of production processes. The production of target products typically includes a combination of machinery, equipment, labor, and other resources.

- Managed Object

  A managed object can be any participating machine, part-carrying vehicle, supply components, and any other resources needed for production. The target system manages the anomaly that occurred on the registered managed object.

- Worker

  A worker is a person who participates in the production process. The system maintains the identification, contact, login, and credit card information of the workers.

- Manager

  A manager is a person who manages the overall operation of the manufacturing plant.

The system maintains the identification, contact, login, and credit card information of the workers.

- Anomaly

  An anomaly in the manufacturing domain refers to any deviation or irregularity in the performance of a machine or a process that is unexpected or differs from the normal behavior or performance.

- Anomaly Type

  Anomaly Type is a specific type of anomaly, and it is specified in an Anomaly Type Profile. This profile captures the meta-information of each anomaly type. The classification of anomaly types largely depends on the target manufacturing plant, and hence a tree-type hierarchical classification could be applied in specifying anomaly types.

- Anomaly Detection

  Anomaly Detection is a task of identifying occurrences of anomaly on managed objects and its environment. Anomaly detection can be done by utilizing a variety of techniques including statistical analysis, pattern matching, and machine learning.

- Anomaly Remedy

  Anomaly Remedy is a task of handling an anomaly occurrence to reduce or remove the negative consequences of the occurred anomaly. The system should first determine the remedy method(s) to apply for the occurred anomaly, perform the method(s), and store the results.

- Anomaly Management

  Anomaly management is a set of tasks managing anomaly on manufacturing plants, and it includes the task of detecting anomaly occurrences, analyzing the causes, determining remedy methods, applying the methods, and evaluating the results of applying the methods.

- Software Platform

  Software platform can be defined as the underlying infrastructure that allows software applications to be developed, deployed, and run. It may include the operating system, programming libraries, frameworks, or tools that are necessary to create and run software applications. For example, Microsoft .NET platform provides developers with a framework and tools to create software applications for Windows operating system.

1.4. References

[Kim 23a] Soo Dong Kim, Associate Architect Program, 2023-A2, CEP Specification of Anomaly Management Platform for Manufacturing Plants, Version 0.9, 삼성전자 첨단기술 아카데미, March 2023.

[Kim 23b] Soo Dong Kim, Associate Architect Program, 2023-A2, CEP Template for Anomaly Management Platform for Manufacturing Plants, 삼성전자 첨단기술 아카데미, March 2023.

[Kim 23c] Soo Dong Kim, Architecture Design Process & Instructions, 2023-A2, Lecture Note #1, 삼성전자 첨단기술 아카데미, March 2023.

[Kim 23d] Soo Dong Kim, 2023-A2, CEP Specification of Anomaly Management Platform for Manufacturing Plants, Version 1.0, 삼성전자 첨단기술 아카데미, March 2023.

[ISO 42010] ISO/IEC/IEEE, *Systems and software engineering - Architecture description*, pp. 46, Dec. 2011.

1.5. Process applied to Architecture Design

The process applied to designing software architecture in this sample solution is given [KIM 22c]. It consists of the following six activities as shown in Figure 1.

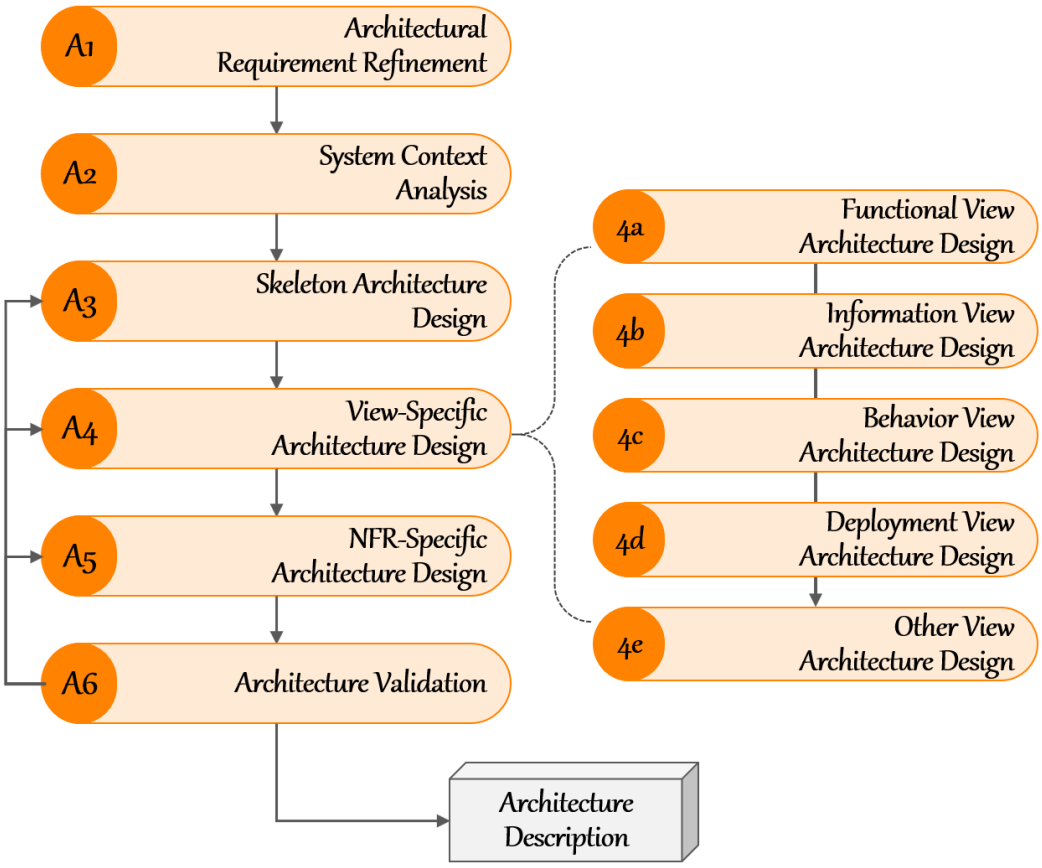


Figure 1. Process to design Software Architecture

- Activity 1. Architectural Requirement Refinement

  This activity is to refine the given requirements for developing a target system.

  Software Requirement Specification should be refined before designing the architecture of the target system. The principles of requirement engineering can be well applied in this activity.

- Activity 2. System Context Analysis

  This activity is to analyze the given requirements for comprehending the target system before making any architectural decisions. The initial comprehension of the target system is specified in the context model of the target system.

- Activity 3. Skeleton Architecture Design

  This activity is to design the initial and high-level architecture of the target system, called skeleton architecture. The skeleton architecture mainly specifies the structural aspect of the target system and becomes the stable basis for making additional architectural decisions and defining more detailed architectural elements accordingly.

The skeleton architecture can effectively be derived by utilizing architectural styles.

- Activity 4. View-specific Architecture Design

  This activity is to specify more detailed architectural elements for different views. It is advantageous to separate architecture design activities by view, backed by the principle of separate of concerns. Essential Views of Software Architecture are Functional view, Information view, Behavior view, and Deployment view. Utilize viewpoints.

- Activity 5. NFR-specific Architecture Design

  This activity is to refine the architecture with additional architectural decisions for each NFR item. Each NFR is thoroughly analyzed, and effective architectural tactics are defined to fulfill the NFR. Then, the existing architecture is refined with defined architectural tactics.

- Activity 6. Architecture Validation

  This activity is to validate the resulting architecture design of the target system for both functional and non-functional aspects.

  Architecture description becomes a concrete baseline document on which detailed system design is made for implementation. Hence, this activity is essential to confirm the fulfillment of both the functional and non-functional requirements.

1.6. Template used for Architecture Description

The template used for writing this architect description is given in [Kim 23b].

2. Activity 1. Architectural Requirement Refinement

This chapter describes the refinements made over the initial requirements of the target system.

2.1. [Step 1] Identify Stakeholders

A stakeholder can be an individual, a group, or an organization. Stakeholders have interests on the target system and concerns that are used as key drivers for designing architecture.

- Stakeholder 1. System Manager

  This represents the user group that manages the operation of manufacturing plants.

- Stakeholder 2. Production Manager

  This represents the user group who manages the manufacturing of target products and controls the quality of products.

- Stakeholder 3. Worker

  This represents the user group who participate in the production process at manufacturing plants.

- Stakeholder 4. Client

  This represents the user group who provides the funding and other resources for developing and operating the target system.

The profile of each stakeholder is summarized in Table 1.

Table 1. Profiles of Stakeholders

Stakeholder Group	Representative Name	Contact Information	Availability
System Manager	Linda Johnson	251-546-9442 Gulf Shores, AL	After 2pm, only on F, Phone Only
Production Manager	James Brown	415-546-4478 San Francisco, CA	Before Noon, MWF, Phone Only
Worker	Susan Tayler	949-569-4371 Santa Clara, CA	10-Noon, T. Th, Office Visits
Client	David Harris	408-925-1352 San Jose, CA	All Day, M-F, Phone Only

2.2. [Step 2] Refining Functional Requirements

Utilize the *SRS Refinement Table* to document the results of requirement refinement.

❑ Deficiency #1

Deficiency ID	FR.DEF.01	**Deficiency Type**	Ambiguity	**Location**	SRS 4.3
Original Context	"This functionality is to register hardware devices in the plant that are used to gather information about and control the managed objects."				
Questioning	The SRS states sensors, camera, and actuator. Can there be other types of hardware devices to interact?				
Refined Context	These are the 3 basic hardware types, and there can be other forms of devices such as IoT devices and drones.				

❑ Deficiency #2

Deficiency ID	FR.DEF.02	**Deficiency Type**	Incomplete	**Location**	SRS 4.9
Original Context	"The system should first determine the remedy method(s) to apply for the occurred anomaly, perform the method(s), and store the results."				
Questioning	Should the system keep track of the results of applying remedy methods on anomaly occurrences?				
Refined Context	Yes. The system should evaluate the result of performing remedy methods. If the evaluation is not automatically performed by the system, managers can determine the effectiveness of the methods and enter the results.				

2.3. [Step 3] Architectural Concerns

An architectural concern is a feature or a characteristic of the target system that are raised and defied by stakeholder(s). Hence, architectural concerns represent the stakeholders' view on the target system and its architecture. Consequently, architectural concerns are expressed in the application domain language, rather than technology languages.

Many of the architectural concerns are requirements and expectations about the target system. And, in fact, many of the concerns in a target system may already be represented in the SRS of the target system in the forms of functional and non-functional requirement items.

The following concerns are acquired from the stakeholders.

❑ Concern-1. Wide Applicability of the Platform

The system should be designed to be highly applicable to building a wide ranges of anomaly management applications.

- Concern-2. High Safety of the Manufacturing Plants

  This system should be designed to provide a high safety.

- Concern-3, Low Cost of Managing Anomaly

  The system should be designed to yield a low cost for managing anomaly.

Merge newly derived NFR items and the NFR items of the SRS. We now have 2 NFR items.

- Concern-1. Wide Applicability of the Platform → (NFR #1) High Configurability of the Anomaly Management Platform
- Concern-2. High Safety of the Manufacturing Plants → NFR #2 (Newly Added)
- Concern-3, Low Cost of Managing Anomaly → NFR #3 ((Newly Added)

We define the relevance of NFR items to the identified stakeholder using the template in Table 2.

Table 2. Relevance of NFRs to Stakeholders

NFR Items	Relevance to Stakeholders				Average Relevance	Standard Deviation	Selection (Y/N)
	User	Staff	Client	Scientist			
NFR-1	2	1	2	1	1.5	0.58	Y
NFR-2	2	0	0	2	1	1.15	Y
NFR-3	2	-1	-1	2	0.5	1.73	N

We apply a 5-level Relevance rating scheme as shown in Figure 2 to fill in the table.

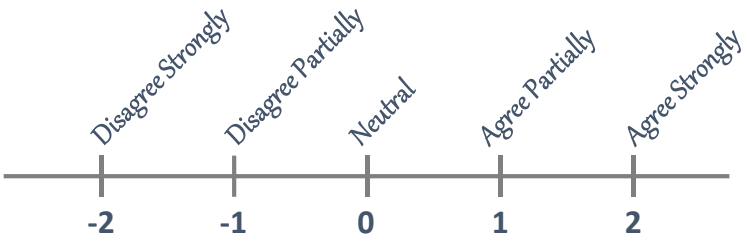


Figure 2. Degree of Relevance of NFRs to Stakeholders

We apply the following guidelines for choosing NFR items

- Case 1) High Average Relevance & Low Standard Deviation ⇒ Choose!
- Case 2) Medium Average Relevance & Low Standard Deviation ⇒ May choose with justification!
- Case 3) Medium Average Relevance & High Standard Deviation ⇒ May not choose with justification!
- Case 4) Low Average Relevance ⇒ Do not choose!
- Case 5) High Average Relevance & High Standard Deviation ⇒ Would not occur.
- Case 6) Low Average Relevance & High Standard Deviation ⇒ Would not occur.

As the result of quantitative assessment on NFR items, we choose NFR-1 and NFR-2.

2.4. [Step 4] Refine Non-Functional Requirements

Utilize the *SRS Refinement Table* to document the results of requirement refinement.

❑ Deficiency #1

Deficiency ID	NFR.DEF.01	**Deficiency Type**	Incompleteness	**Location**	SRS 5.1
Original Context	"The key variation points to handle in the platform are the following."				
Questioning	The SRS states the areas of key variation points. Is this a complete list of variation points?				
Refined Context	No. Additional variation points might be identified by the system designers.				

❑ Deficiency #2

Deficiency ID	NFR.DEF.02	**Deficiency Type**	Ambiguity	**Location**	SRS 5.1
Original Context	"Configurability of the platform is the degree of effectiveness in customizing the platform for a target anomaly management application system."				
Questioning	How to measure the configurability? Should it be managed in a quantitative manner?				
Refined Context	There is no metric defined for the configurability. Therefore, a quantitative measure should be defined by the developing team.				

The resulting SRS now becomes more complete and well-aligned with stakeholders' concerns.

2.5. [Step 5] Write Refined Software Requirement Specification

The revised SRS is available here [KIM 23d].

3. Activity 2. System Context Analysis

This chapter specifies the context of the target system in terms of the followings.

- Target System and Its Boundary
- Functionality provided by the system
- Information manipulated in the system
- Runtime behavior of the system

Additional type of the context can be described.

3.1. [Step 1] System Boundary Context

The target system may interact with users, hardware devices, external systems or other sources in the operational environment. *System Boundary Context* describes the boundary of the system and elements in the environment which interact with the target system. This helps architect and developers to clearly understand the scope of the system.

3.1.1. Representing the Boundary Context

We use *Context Diagram*, i.e., Level 0 of Data Flow Diagram (DFD), which shows each tier of the target system as a process and relationships with its environment.

3.1.2. DFD for Context View

"The target system is a versatile software platform designed to support the development of various applications. The architectures of anomaly management applications developed on top of this platform may vary. Some applications may have a single tier where other applications may adopt multiple tiers.

We define the context view with a single tier as shown in Figure 3.

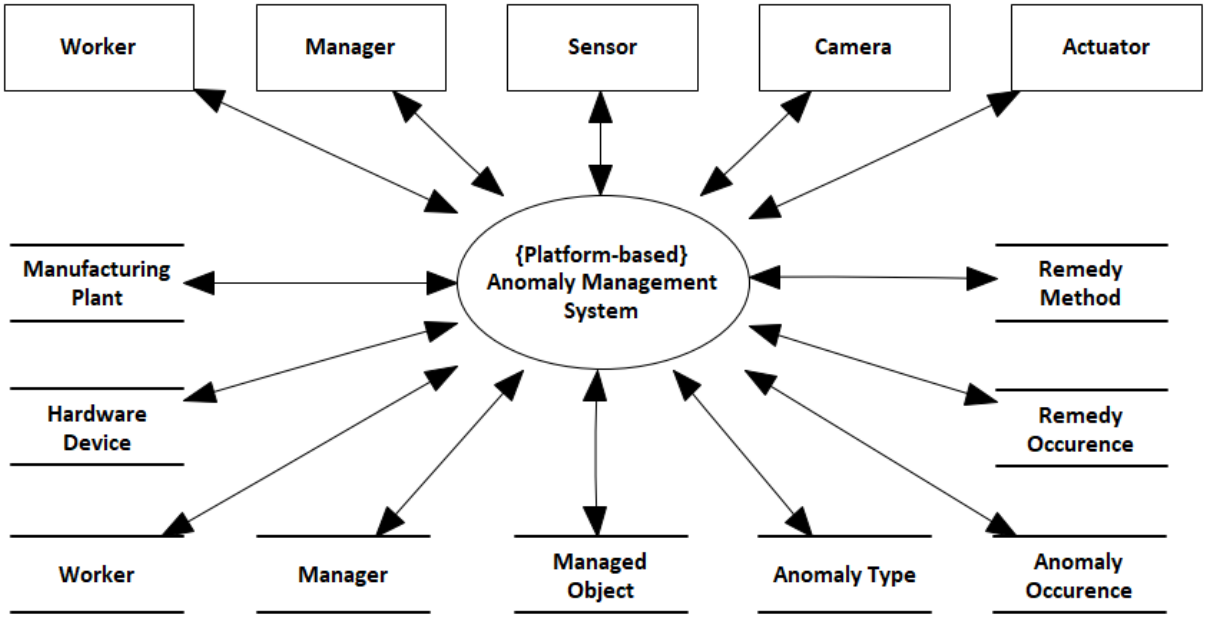


Figure 3. DFD Diagram for showing Boundary Context

3.1.3. Description of the Context Diagram

- Process
  - There is only one process, *Anomaly Management System*, which depicts an application developed with the platform.
- Terminals
  - The terminals of human user types are *Worker* and *Manager*.
  - The terminals of hardware devices are sensor, camera, and actuator.

  Note that each application may adopt external systems or services such as SMS system, and Emergency Alert Services.
- Store
  - The application maintains a number of distinct data stores. All the essential persistent information is captured in the data store set.
- Data Flow

  An arrow between two elements depicts a flow of data, and the names of the data on arrows are omitted in the diagram.

3.2. [Step 2] Functional Context

3.2.1. Representing the Functional Context

The functional context of the target system can be well described with a use case diagram and descriptions of the use cases. A use case diagram shows the whole functionality of the target system. It is specified with Include actors, use cases, and their relationships.

The following use case diagram shows the whole functionality of the target system. It is specified with Include actors, use cases, and their relationships.

We do not attempt to show the control flow in the use case diagram; rather show only the use cases and invocation relationships. We do not consider tiers; rather we consider the whole system.

3.2.2. Defining Functional Groups

We apply a scheme for numbering the use cases by considering functional groups. A functional group is a collection of *closely related* use cases. And we assign a two-character prefix to each functional group. A use case diagram with use case identification numbers becomes easier to comprehend and to manage.

Based on the given SRS, we identify the following functional groups and their prefixes.

- ❑ Registering Manufacturing Plants → RMP
- ❑ Registering Managed Objects → RMO
- ❑ Registering Hardware Devices → RHW
- ❑ Registering Worker Profiles → RWP
- ❑ Registering Manager Profiles → RMP
- ❑ Registering Anomaly Types → RAT
- ❑ Registering Remedy Methods → RRM
- ❑ Detecting Anomaly Occurrences → DAO
- ❑ Remedying Anomaly Occurrences → RAO
- ❑ Generating Operation Reports → GRP

3.2.3. Defining Actors

We define actors that interact with the use cases. Each functional group is given its relevant actors as shown below.

Actors / Functional Groups	Active Actors	Passive Actors
Registering Manufacturing Plants	Manager	
Registering Managed Objects	Manager	
Registering Hardware Devices	Manager	Sensor, Camera, Actuator
Registering Worker Profiles	Manager	
Registering Manager Profiles	Manager	
Registering Anomaly Types	Manager	
Registering Remedy Methods	Manager	
Detecting Anomaly Occurrences	Detection Agent	Sensor, Camera
Remedying Anomaly Occurrences	Remedy Agent, Manager	Actuator
Generating Operation Reports	Report Agent, Manager	

Note that an application developed with this platform may employ additional types of passive actors such as SMS System and Emergency Alert Service.

3.2.4. Defining Use Cases

The use cases in the diagram can be derived from the functional requirement of SRS. The name of each use case begins with a prefix which indicates the functional group it belongs to. The use cases in a functional group are placed together in the diagram for readability.

A use case typically has an active actor and optional passive actors.

3.2.5. Context-level Use Case Diagram

The use case diagram for the functional context is shown in Figure 4.

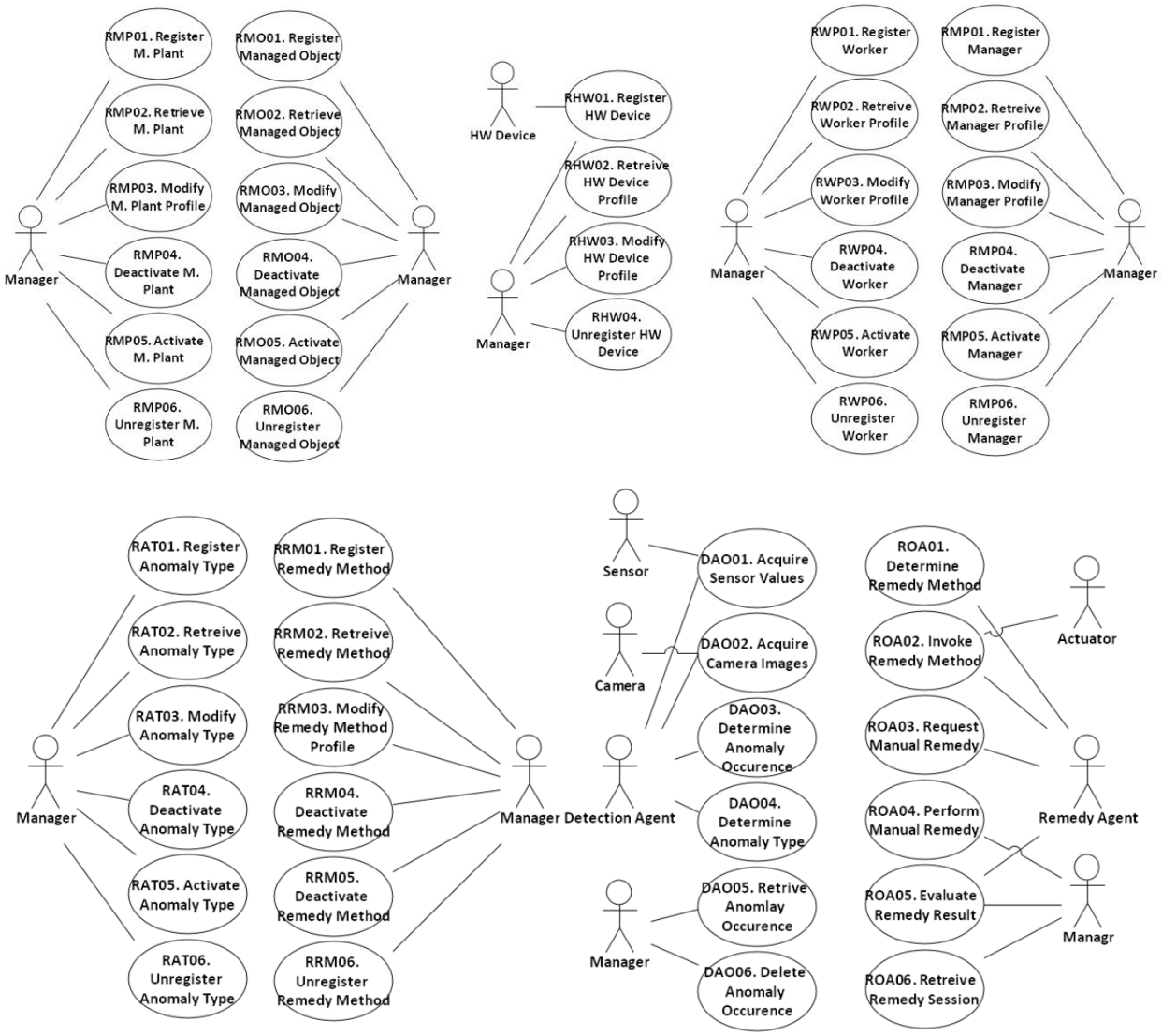


Figure 4. Use Case Diagram for the Functional Context

3.3. [Step 3] Information Context

3.3.1. Representing the Information Context

The information context of the system shows the datasets manipulated by the system. Class Diagram can be effectively used to capture the information context.

In this context-level class diagram, we only show the entity-type classes, their relationships, and the cardinalities. No need to specify attributes and methods at this stage.

3.3.2. Identifying Persistent Object Classes

- There are several classes of human user type.
- There are several classes of hardware devices.
- The core physical class of the class diagram is *Manufacturing Plant*.
- The core logical class of the class diagram is *A.M. Session*, which captures the results of detecting and remedying anomaly occurrences.
- The class, *Report*, may or may not be included in the class diagram. The decision is made on how each instance of *Report* is managed and traced.

3.3.3. Context-level Class Diagram

The class diagram for acquiring the information context consists of only classes and their relationships. A relationship is defined with cardinalities.

The context-level class diagram of the target system is shown in Figure 5.

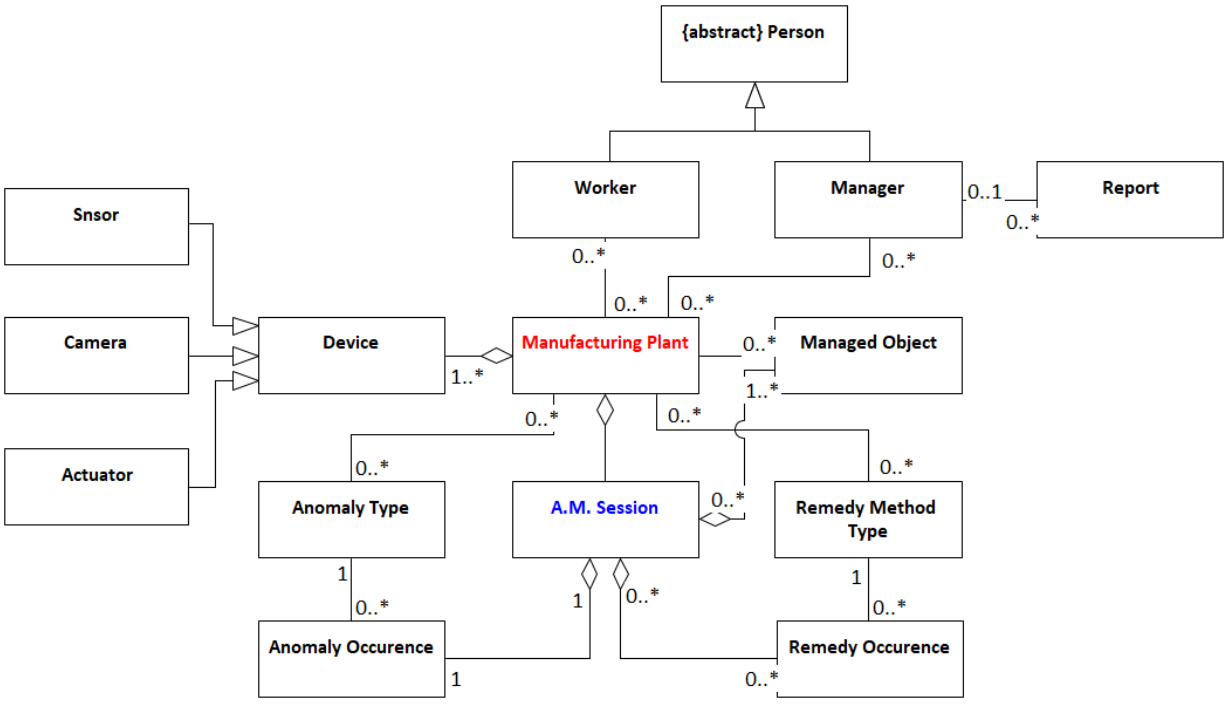


Figure 5. Class Diagram for showing Information Context

- Relationships

  Various types of relationships are defined between classes.

- Cardinalities

  Cardinalities are defined on every relationship.

3.4. [Step 4] Behavioral Context

Behavioral context of the system shows the execution and control flow at runtime. Behavioral context may be more important for systems with complex workflows, parallel processing, and timing constraints.

3.4.1. Representing the Behavior Context

Activity Diagram of UML provides a rich set of constructs that can be used to model the runtime behavior of the system.

3.4.2. Allocate Functionality over Tiers

This task is to allocate the system functionality over the tiers. Since the target platform/system adopts a single tier, we do not apply this task.

For reference, a system adopting multiple tiers may show the functionality allocation as in in Table 3.

Table 3. Allocation of Functionality over Tiers (Applicable to C-S architecture)

Tiers / Functional Groups	**Client**	**Server**
Registering Manufacturing Plants	✓	✓ (Retrieve)
Registering Managed Objects	✓	✓ (Retrieve)
Registering Hardware Devices	✓	
Registering Worker Profiles	✓	✓ (Retrieve)
Registering Manager Profiles	✓	✓ (Retrieve)
Registering Anomaly Types		✓
Registering Remedy Methods		✓
Detecting Anomaly Occurrences	✓	✓ (Retrieve)
Remedying Anomaly Occurrences	✓	✓ (Retrieve)
Generating Operation Reports		✓

3.4.3. Define Invocation Patterns

We define appropriate invocation patterns of the allocated functional groups. Each functional group is assigned with one or more invocation patterns. The common types of invocation patterns are Explicit Invocation, Event-driven, Timer-based, and Closed Loop.

The event-driven invocation may occur in two different ways:

- **Event I** is for handling events within a tier, i.e., intra-tier event-driven invocation.
- **Event II** is for handling events among multiple tiers, i.e., inter-tier event-driven invocation.
  - This is not appliable to 1-tier system.

The invocation patterns defined on the functional groups is shown in Table 4.

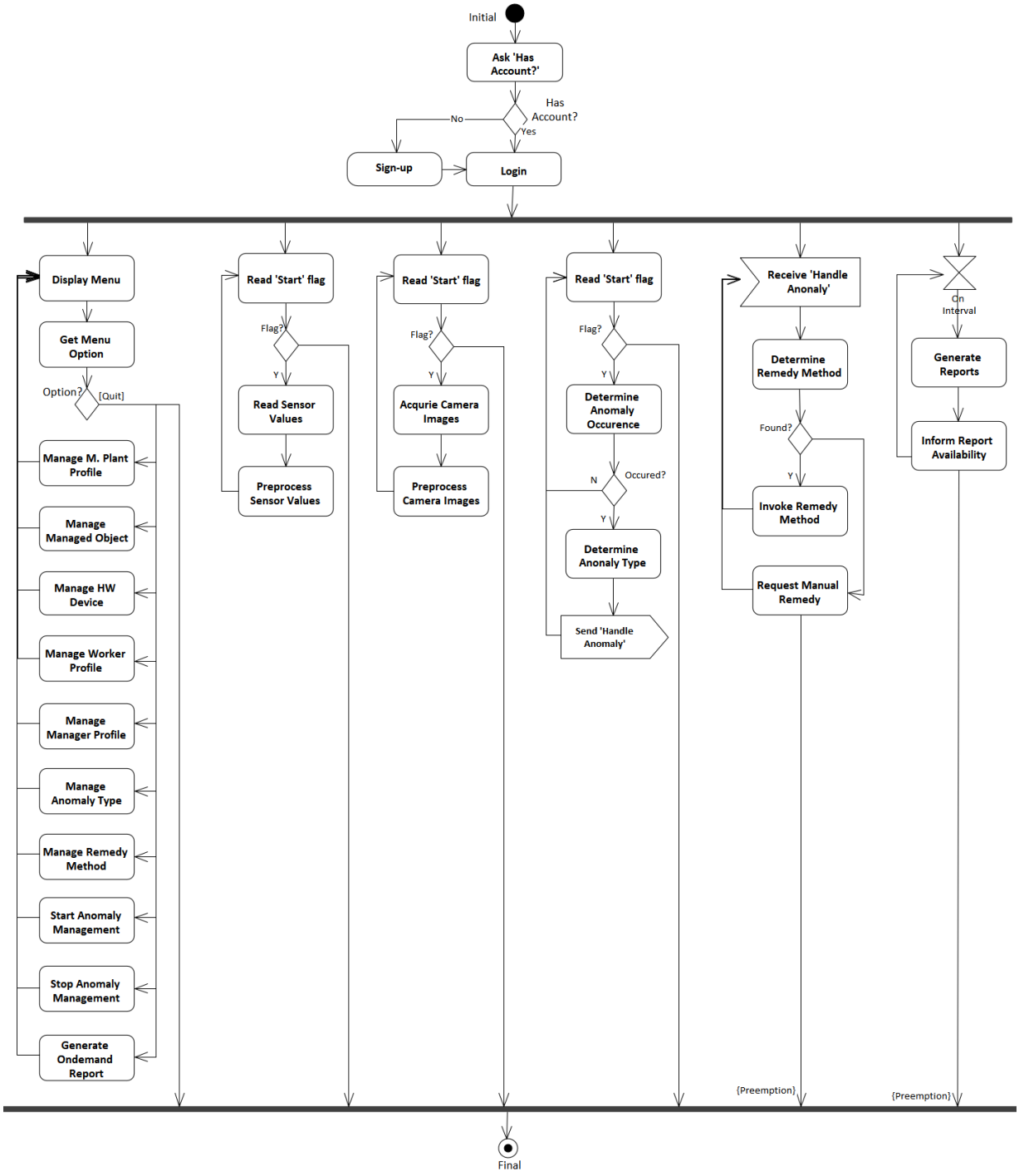
Table 4. Invocation Patterns defined for Functional Groups

	A.M. System	
Registering Manufacturing Plants	Explicit	Not Applicable
Registering Managed Objects	Explicit	
Registering Hardware Devices	Explicit	
Registering Worker Profiles	Explicit	
Registering Manager Profiles	Explicit	
Registering Anomaly Types	Explicit	
Registering Remedy Methods	Explicit	
Detecting Anomaly Occurrences	Closed Loop, Event-based	
Remedying Anomaly Occurrences	Closed Loop, Event-based	
Generating Operation Reports	Closed Loop, Explicit	

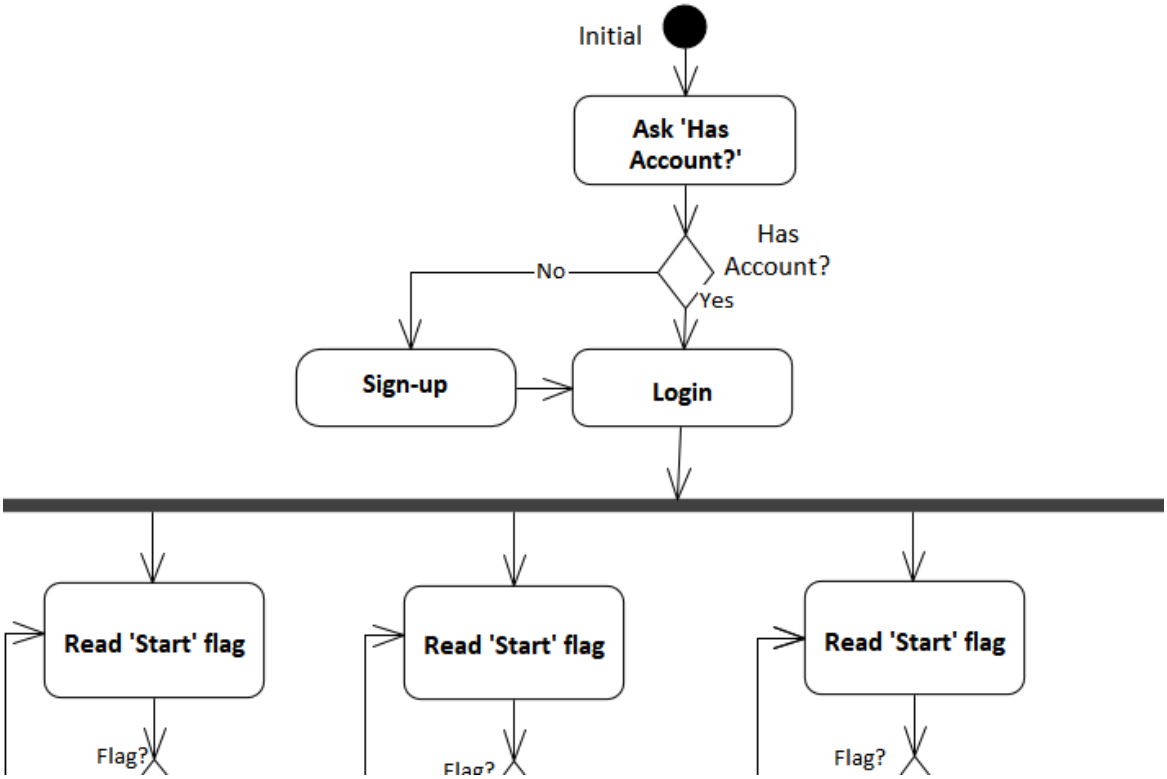
Now, the control flow of the target system can be well modeled based on the specified invocation patterns.

3.4.4. Context-level Activity Diagram

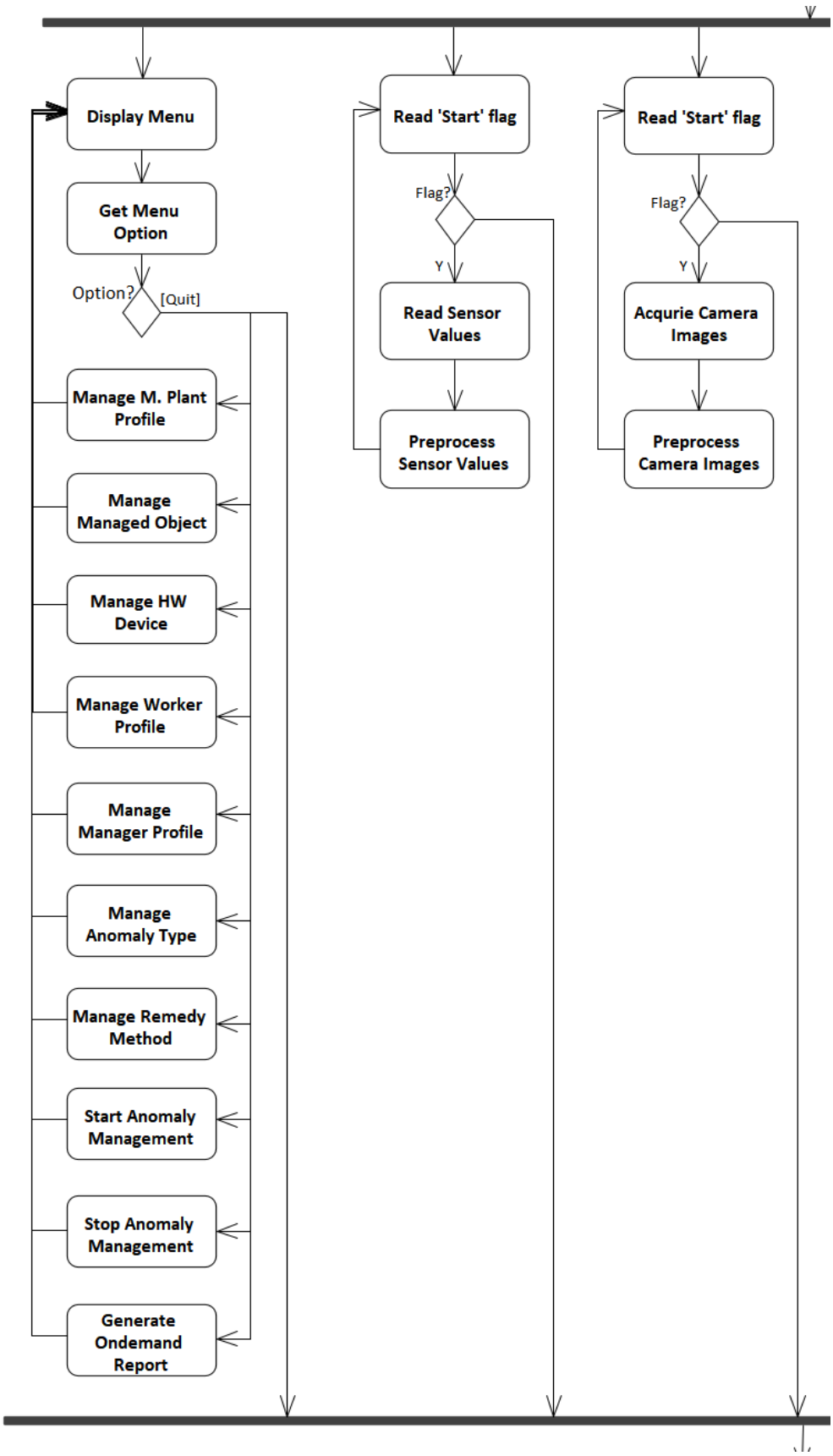
Based on the invocation patterns defined over the tiers, we draw an activity diagram for the system as shown in Figure 6.



- Part 1


Initial
Ask 'Has Account?'
Has Account?
No
Yes
Sign-up
Login
Read 'Start' flag
Flag?
Read 'Start' flag
Flag?
Read 'Start' flag
Flag?

- Part 2


Display Menu
Get Menu Option
Option?
[Quit]
Manage M. Plant Profile
Manage Managed Object
Manage HW Device
Manage Worker Profile
Manage Manager Profile
Manage Anomaly Type
Manage Remedy Method
Start Anomaly Management
Stop Anomaly Management
Generate Ondemand Report
Read 'Start' flag
Flag?
Y
Read Sensor Values
Preprocess Sensor Values
Read 'Start' flag
Flag?
Y
Acqurie Camera Images
Preprocess Camera Images

- Part 3

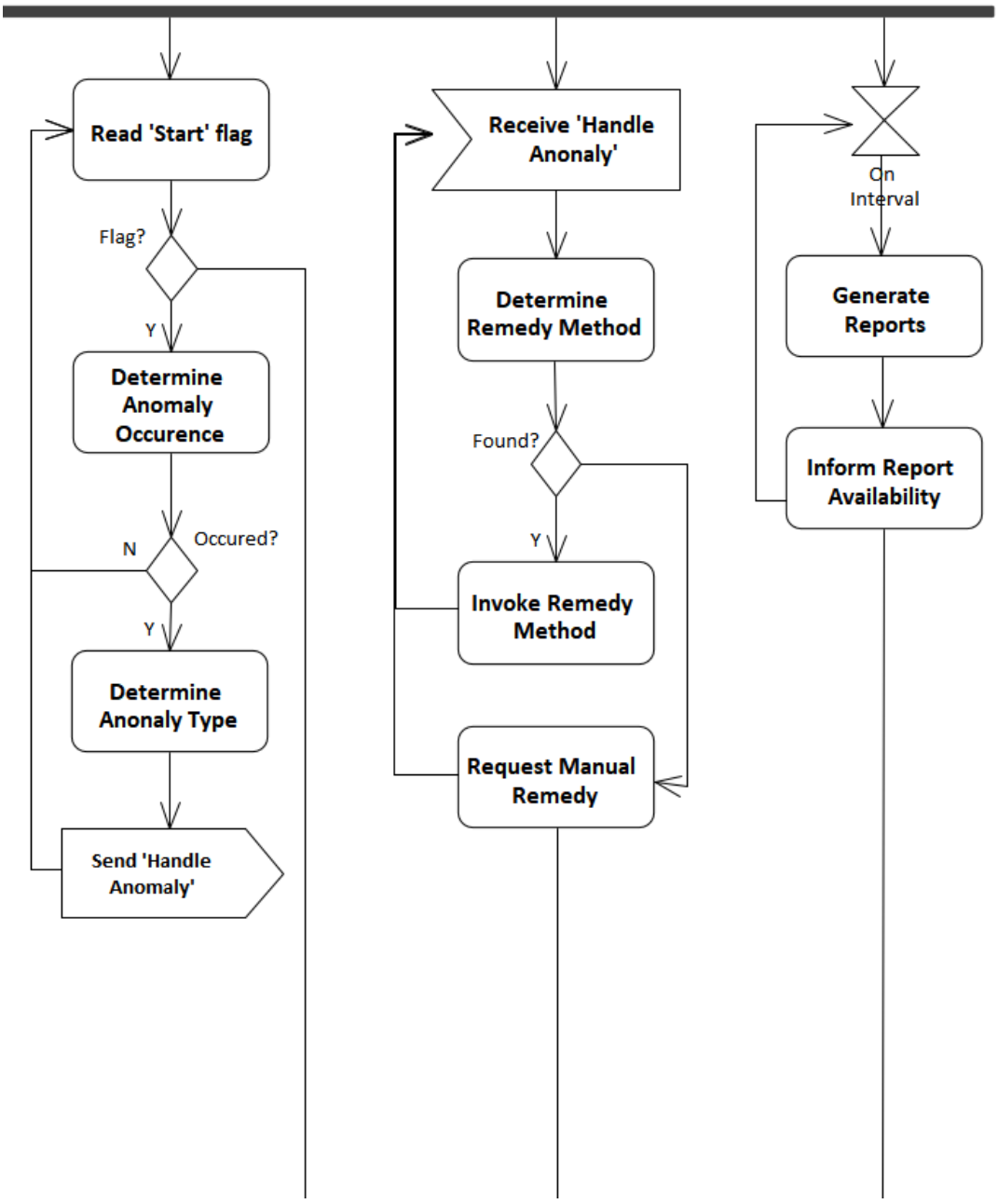


Figure 6. Activity Diagram for showing Behavior Context of Mobile App

The control flow begins with a login/sign-on process. Then, it runs 6 parallel threads.

- The thread #1 is to handle the functionality with explicit invocation pattern using a menu.
- The thread #2 and #3 are to acquire the environment context of the target manufac turing plant using sensors and cameras.
- The thread #4 is to detect occurrences of anomaly using the anomaly type profiles.
- The thread #5 is to handle each occurrence of anomaly by finding and invoking app ropriate remedy methods.

- The thread #6 is to generate periodical reports using a timer.

The behavior context shows all the functionality of the mobile app as defined in its functional context. That is, all the use cases of the mobile app are reflected in this activity diagram.

3.5. [Step 5] Additional Contexts

Any additional contexts of the target system can be described.

- None

4. Activity 3. Skeleton Architecture Design

A skeleton architecture is a description of the structural aspect of the target system without fully describing the key components and their properties. It can be effectively derived by applying architectural style(s). Each architectural style is a named collection of architectural decisions that are applicable in a given development context.

4.1. [Step 1] Candidate Architecture Styles

We make the following observation on the target and derive candidate architecture styles from the observation.

- System consisting with Distinct Roles with Virtual Machine → Layered Architecture Style

  The system has a number of distinct roles what can be organized hierarchically.

- System containing High Variability → Micro Kernel Architecture Style

  There exist several variation points in the system where application-specific variants should be bound.

- Others Architecture Styles
  - N-Tier Architecture Style

    When the platform itself has a strong requirements of having multiple tiers
  - Microservice Architecture Style

    When the platform is to utilize the functionality provided by external services
  - Event-driven Architecture Style

    When the platform has a considerably large portion of the functionality that is invoked with event exchanges.
  - Pipe-n-Filter Architecture Style

    When the platform requires a number of data manipulation components that transform data in streaming mode.

    If the platform has a small number of data manipulation components, such as 2 or 3, the justification for using this style will be largely limited.
  - Blackboard Architecture Style

    When the platform has some functionality that may not be the optimal solution
  - Dispatch Architecture Style

    When the platform is provided as a separate node of server form.

4.2. [Step 2a] Evaluating 'Candidate 1. Layered Architecture Style'

The layered architecture consists of multiple layers which are arranged as the virtual machine dependency.

4.2.1. Evaluate the Applicable Situations

Applicable Situations	Match	Demands on the System
Multiple Distinct Roles in the System which can be ordered vertically for their dependency. That is, a layer can access its immediately lower layer but no other layers.	❍	A.M. Platform consists of several distinct roles that can be modeled as layers. And they can be well organized vertically.
Each layer of the system provides its well-defined interface that is exposed to its immediate upper layer.	❍	The distinct roles of A.M. Platform can be defined as layers and the functionality of each layer is exposed through a well-defined interface.
Each layer of the system is seen as a virtual machine to its immediate upper layer. The design complexity of a layer can be simplified with its sole dependency on the virtual machine.	❍	The role of each layer can be realized with the functionality provided by its immediate lower layer, and the lower layer is seen as a virtual machine.
The modification of internal details of a layer except its interface has no influence on other layers. Hence, the maintainability becomes enhanced.	❍	A.M. Platform is maintained over time, therefore, it should be design to minimize the propagation of maintenance impacts.

4.2.2. Evaluate the Benefits

Advantages of the Style	Match	Benefits Applicable to the System
The dependency of each layer is minimal; depending on its immediately lower layer.	❍	Each layer of A.M. Platform should depend only on its immediate lower layer, reducing the dependency on other parts.
The overall design of the system can be logically simplified with the virtual machine relationship.	❍	The design of each layer becomes simplified by relying on its lower layer and providing functionality to its upper layer. The complexity of each layer becomes minimal.
The maintenance on one layer except its interface has minimal influence on other layers. Hence, the maintainability of the system becomes high.	❍	The need for maintaining A.M. Platform would be minimal.

4.2.3. Evaluate the Drawbacks

Cons of the Style	Match	Handling the Drawbacks
The performance overhead can be substantial due to vertical communications among layers.	❍	The performance overhead with multiple layers in A.M. Platform is minimal due to the small number of layers. The minimal overhead of the system can be justified with the benefits.
It could be difficult to define the right set of layers and organize them in a virtual machine relationship.	❍	Due to the intrinsic and distinct roles of the A.M. Platform, the design of the layers can be defined with significant difficulty.

4.2.4. Result of the Evaluation

Layered architecture style is well applicable to the target system according to the justification. There is no significant issue which prevents the application of this style.

4.3. [Step 2b] Evaluating 'Candidate 2. Micro Kernel Architecture Style'

Micro Kernel Architecture Style is characterized by the high variability of the system functionality, which can be fulfilled by incorporating plug-in components. Consequently, the flexibility of the Micro Kernel-based applications is achieved by the openness of the variable functionality.

4.3.1. Evaluate the Applicable Situations

Applicable Situations	Match	Demands on the System
The target system has a minimal functionality which is stable, fixed, and hence closed, called 'kernel' functionality.	❍	A.M. Platform has a relatively minimal functionality which is core and stable. That functionality is mainly for user profile management and medical image management.
The target system has a large portion of functionality which is fulfilled by externally developed 'plug-in' components. That is, the system has a high variability on its functionality.	❍	The key feature of A.M. Platform is the capability of detecting and remedying anomalies. This key feature should be designed to be 'open' since there are a number of different approaches to the detection and remedy methods. Moreover, new and improved methods could appear in the future, which should be effectively incorporated into the system.
There is a high potential to develop and acquire a number of variants for a commonly required functionality. That is, there could potentially exist a number of different plug-in components which could fulfill the required functionality.	❍	There exist a number of common interfaces on A.M. Platform. There can be different implementations of the interface. Some can be developed internally, and others may appear with newer technologies.
The system has a requirement for high adaptability by adopting current and future variants for a common required functionality.	❍	There is a specific requirement for adopting newer and improved methods for medical analytics.

4.3.2. Evaluate the Benefits

Advantages of the Style	Match	Benefits Applicable to the System
The kernel portion of the system becomes stable and widely applicable to various application instances.	❍	The functionality of profile management and medical image management is stable and hence widely applicable to various A.M. applications.
The system has a high extendibility by incorporating most appropriate plug-in components statically and dynamically.	❍	The plug-in components for A.M. Platform can be flexibly adopted, and hence the system become extensible.
The high separation of the kernel from the plug-in components provides a high maintainability since the dependency between the two is minimal.	❍	The maintenance of plug-in components does not influence on the kernel part, and vice versa.

4.3.3. Evaluate the Drawbacks

Cons of the Style	Match	Handling on Cons on the System
It could be technically hard to define well-defined interfaces for plug-in components.	Δ	The technical hardship in defining well-defined interfaces for medical analytics remains on A.M. applications.
The validation of plug-in components becomes an issue and could be technically hard.	Δ	The validation of plug-in components is also an important issue and should be well applied.

4.3.4. Result of the Evaluation

Micro Kernel Architecture style is well applicable to the target system according to the justification. There is no significant issue which prevents the application of this style.

4.4. [Step 2c] List of Selected Architecture Styles

All the candidate architecture styles are chosen for defining the skeleton architecture.

- ❑ Layered Architecture Style
- ❑ Micro Kernel Architecture Style

4.5. [Step 3] Applying Architecture Styles

We apply the selected architecture styles incrementally.

4.5.1. Applying Layered Architecture Style

By considering the common structural properties of anomaly management applications, we define four layers as shown in Figure 7.

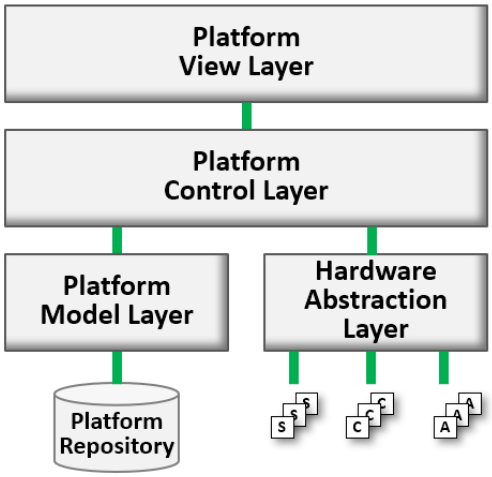


Figure 7. Applying Layered Architecture Style

- Platform View Layer

  This layer does not represent the user interface of various anomaly applications; rather it models the common user interface elements which can be applied to various applications.

- Platform Control Layer

  This layer consists of functional components that carry the core functionality of anomaly management systems.

- Platform Model Layer

  This layer defines the interface of data components that manipulate the persistent datasets.

- Platform Repository

  This is the master data repository of persistent datasets that are managed by the platform. This repository contains the common set of datasets and hence each anomaly management application may maintains its own database that contains the application-specific datasets.

- Hardware Abstraction Layer (HAL)

  This layer defines the abstract interfaces of various hardware device types. Especially, it includes the common interfaces for accessing sensors, cameras, and actuators. This layer allows the development of functional components that does not depend on

specific hardware models.

4.5.2. Applying Micro Kernal Style

By modeling the commonality and variability among various anomaly management applications, we define two partitions in the control layer as shown in Figure 8.

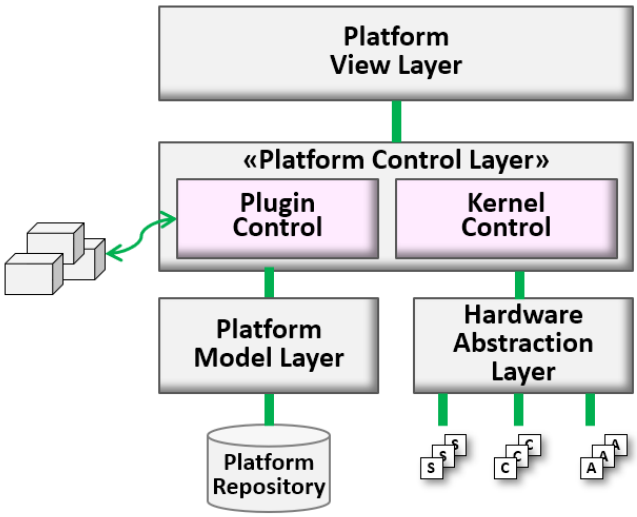


Figure 8. Applying Micro Kernel Architecture Style

- Kernel Control Layer

  This layer is to model and deliver the fixed functionality among various anomaly management applications. The functionality delivered by this layer has no variable features.

- Plugin Control Layer

  In contrast to Kernel Control Layer, this layer models the variability among various anomaly management applications and allows the customization of the variation points for each application.

- Plugin Objects (External)

  The plugin objects are externally development objects that implement the (required) interface of *Plugin Control Layer*. Each plugin object implements the interfaces for its own requirements.

4.5.3. Resulting Skeleton Architecture

The resulting skeleton architecture of applying all the selected architecture styles is shown in Figure 9.

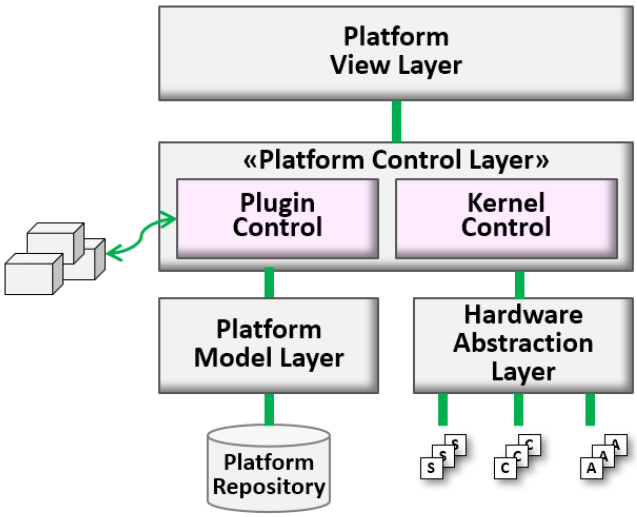


Figure 9. Resulting Skeleton Architecture

The resulting architecture shows the application of the selected architecture styles. And it serves the stable basis on which view-specific architectural designs can be appended.

4.6. [Step 4] Refining Interaction Paths

Define interaction paths among places in the skeleton architecture. An interaction path can be casual dependency or persistent relationship. It provides paths for making function calls or sending messages for communications among components.

4.6.1. Interaction Paths derived from the Styles

All the interaction paths defined in each architecture style are adopted and remain unchanged in the skeleton architecture.

- Interaction Paths from Layered Architecture Style
- Interaction Paths from Micro Kernel Architecture Style

4.6.2. Refinements on the Default Interaction Paths

- None

  The target platform does not require refinements on the interaction paths. Therefore, the default interaction paths remain the same.

4.7. [Step 4] Elaborating the Skeleton Architecture

4.7.1. Strengths

Specify the advantages of the proposed skeleton architecture.

- Separation of Concern

  Each component or layer represents a unique and separate concern. It yields a logically well-defined architecture with high modularity.

- Complexity of the System Design and Implementation

  Due to the independence of each component or layer, the complexity design is low, and effort to implement the system can be greatly reduced.

- High Maintainability

  Due to the key principles applied to designing the skeleton architecture, the impact of modification would be minimal.

4.7.2. Drawbacks

Specify the drawbacks and risks of the proposed skeleton architecture.

- Not Anticipated.

5. Activity 4. View-specific Architecture Design

This chapter describes the results of applying essential architecture viewpoints. The skeleton architecture is now refined with additional architectural decisions made with viewpoints.

5.1. Functional View

5.1.1. [Step 1] Refine Functional Groups

We use the same list of functional groups used in the functional context view.

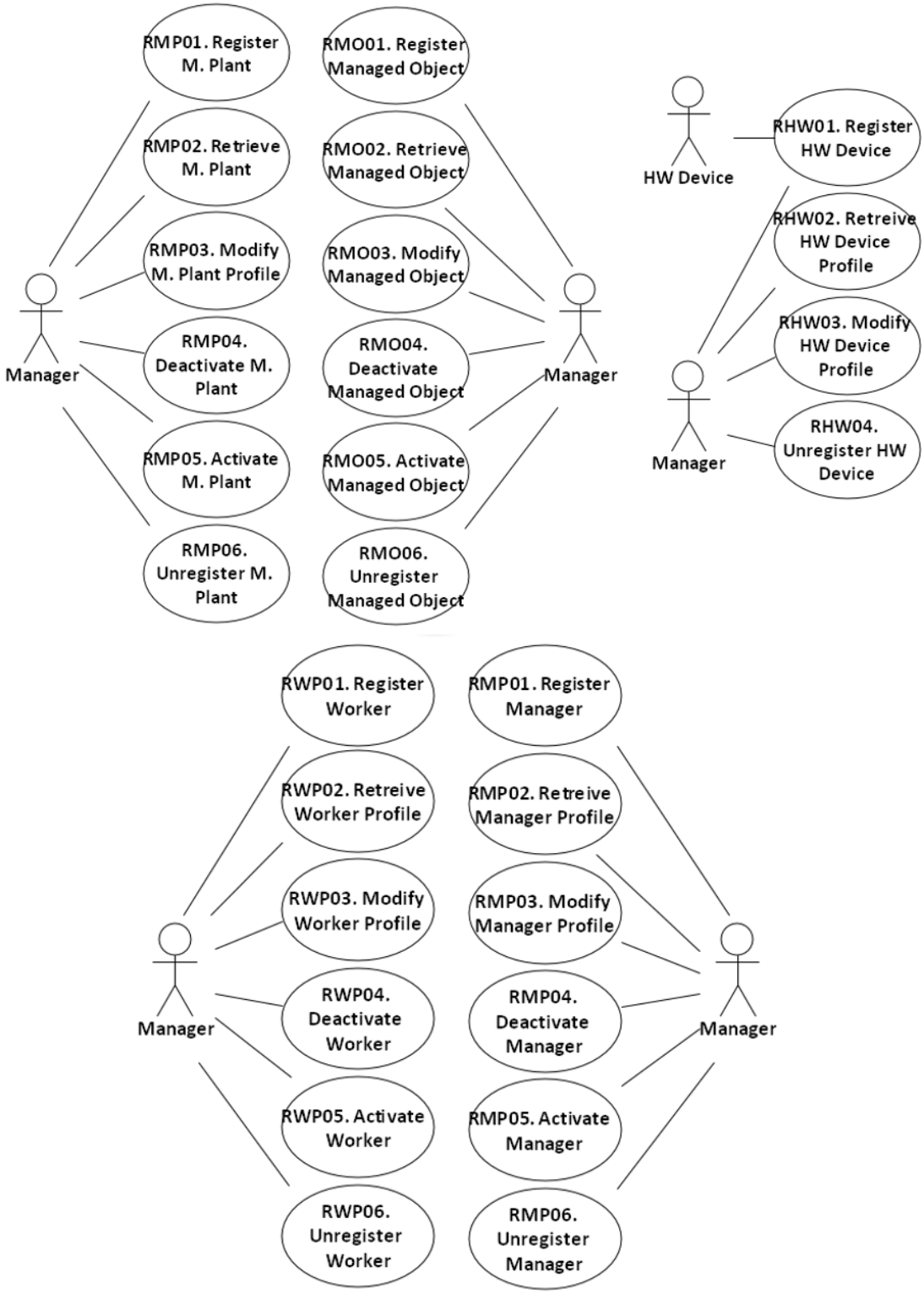
- ❑ Registering Manufacturing Plants → RMP
- ❑ Registering Managed Objects → RMO
- ❑ Registering Hardware Devices → RHW
- ❑ Registering Worker Profiles → RWP
- ❑ Registering Manager Profiles → RMP
- ❑ Registering Anomaly Types → RAT
- ❑ Registering Remedy Methods → RRM
- ❑ Detecting Anomaly Occurrences → DAO
- ❑ Remedying Anomaly Occurrences → ROA
- ❑ Generating Operation Reports → GRP

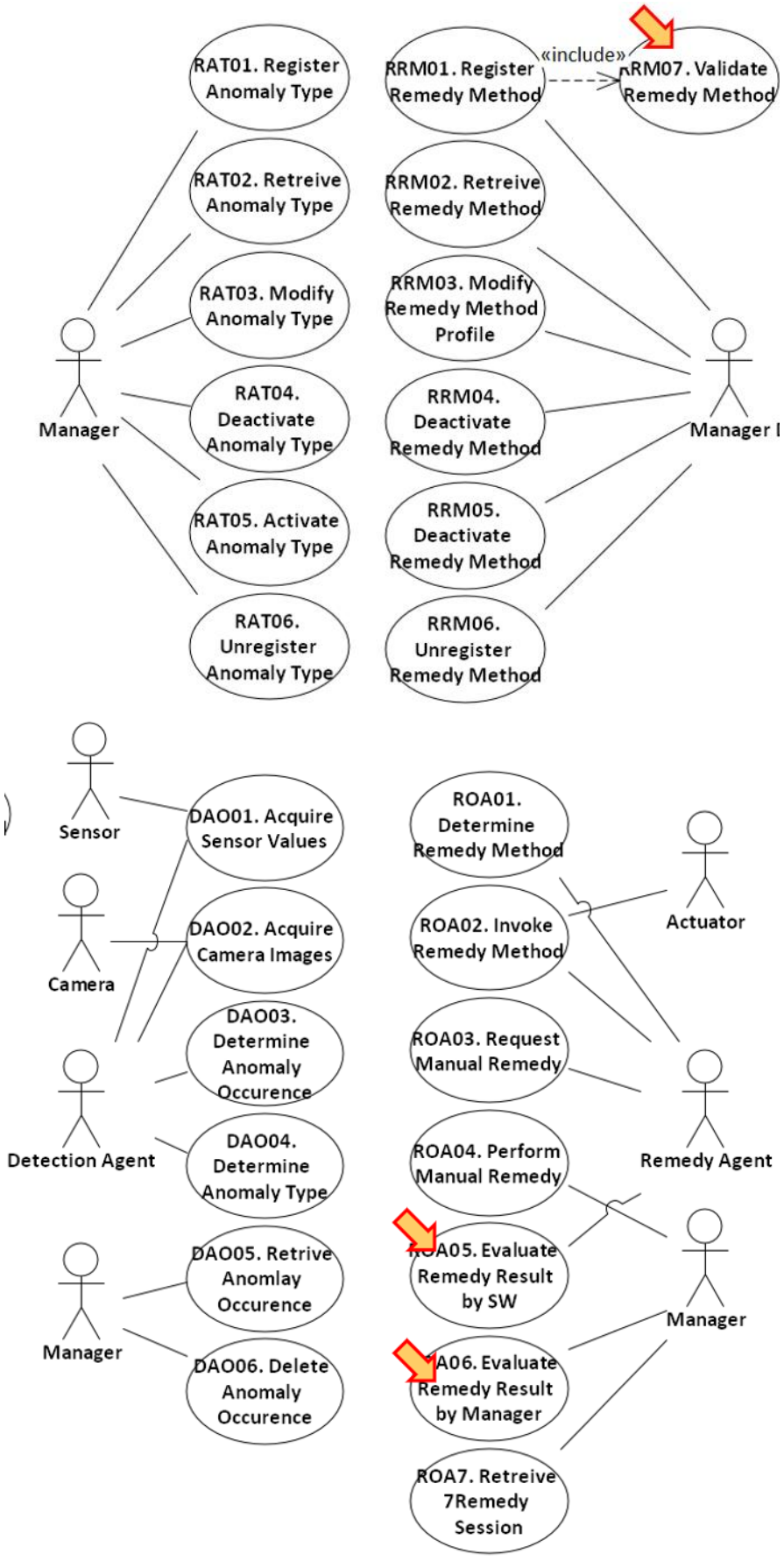
5.1.2. [Step 2] Refine Use Case Diagram

Before identifying functional components, we refine the context-level use case diagram with corrections and greater details.

- Refined Use Case Diagram

  The refined use case diagram is shown in Figure 10.




RAT01. Register Anomaly Type
RAT02. Retreive Anomaly Type
RAT03. Modify Anomaly Type
RAT04. Deactivate Anomaly Type
RAT05. Activate Anomaly Type
RAT06. Unregister Anomaly Type
Manager
RRM01. Register Remedy Method
«include»
RRM07. Validate Remedy Method
RRM02. Retreive Remedy Method
RRM03. Modify Remedy Method Profile
RRM04. Deactivate Remedy Method
RRM05. Deactivate Remedy Method
RRM06. Unregister Remedy Method
Manager I
Sensor
Camera
Detection Agent
Manager
DAO01. Acquire Sensor Values
DAO02. Acquire Camera Images
DAO03. Determine Anomaly Occurence
DAO04. Determine Anomaly Type
DAO05. Retrive Anomlay Occurence
DAO06. Delete Anomaly Occurence
ROA01. Determine Remedy Method
ROA02. Invoke Remedy Method
ROA03. Request Manual Remedy
ROA04. Perform Manual Remedy
ROA05. Evaluate Remedy Result by SW
ROA06. Evaluate Remedy Result by Manager
ROA7. Retreive 7Remedy Session
Actuator
Remedy Agent
Manager

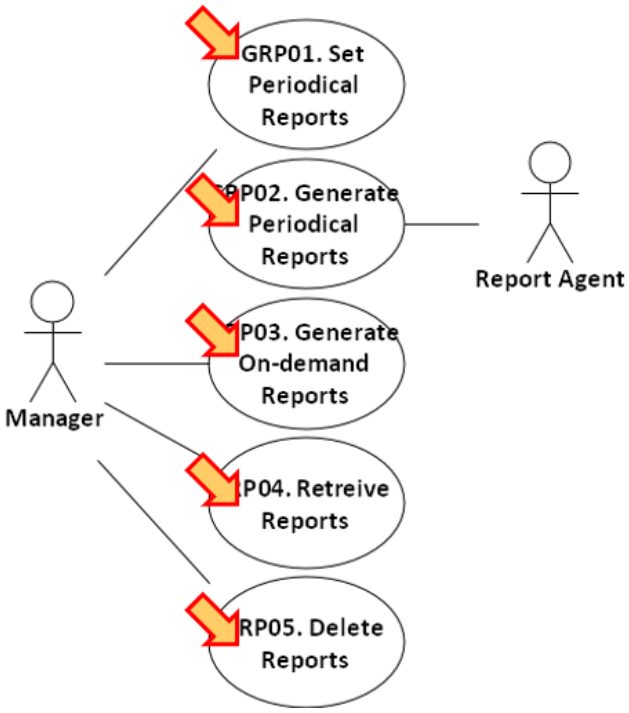


Figure 10. Refined Use Case Diagram

- Refinements made

  The refinements made on the initial use case diagram are shown with arrow markers.

  - A new use case 'RRM07 Validate Remedy Method' was added to check the effectiveness and validity of remedy methods.
  - There are two use cases of evaluating the remedy result: one by software and the other by manager.
  - The set of 5 use cases for generating reports are now defined.

5.1.3. [Step 3] Derive Functional Components

There are three categories of functional components to consider.

- Category 1. Functional Components derived from the SRS

  The functional components are mainly derived from the system-intrinsic functionality, which is well modeled in its use case diagram. That is, the functional components can be systematically derived by clustering relevant use cases.

- Category 2. Functional Components derived from Skeleton Architecture

  A skeleton architecture is typically designed by applying architecture styles. An architecture style consists of components and connectors. Some of the components and connects may need to be modeled as functional components.

- Category 3. Interface-centric Functional Components

  An interface-centric functional component specifies a stable and public interface, which will be realized/implemented.

The functional Components derived are shown in Figure 11.

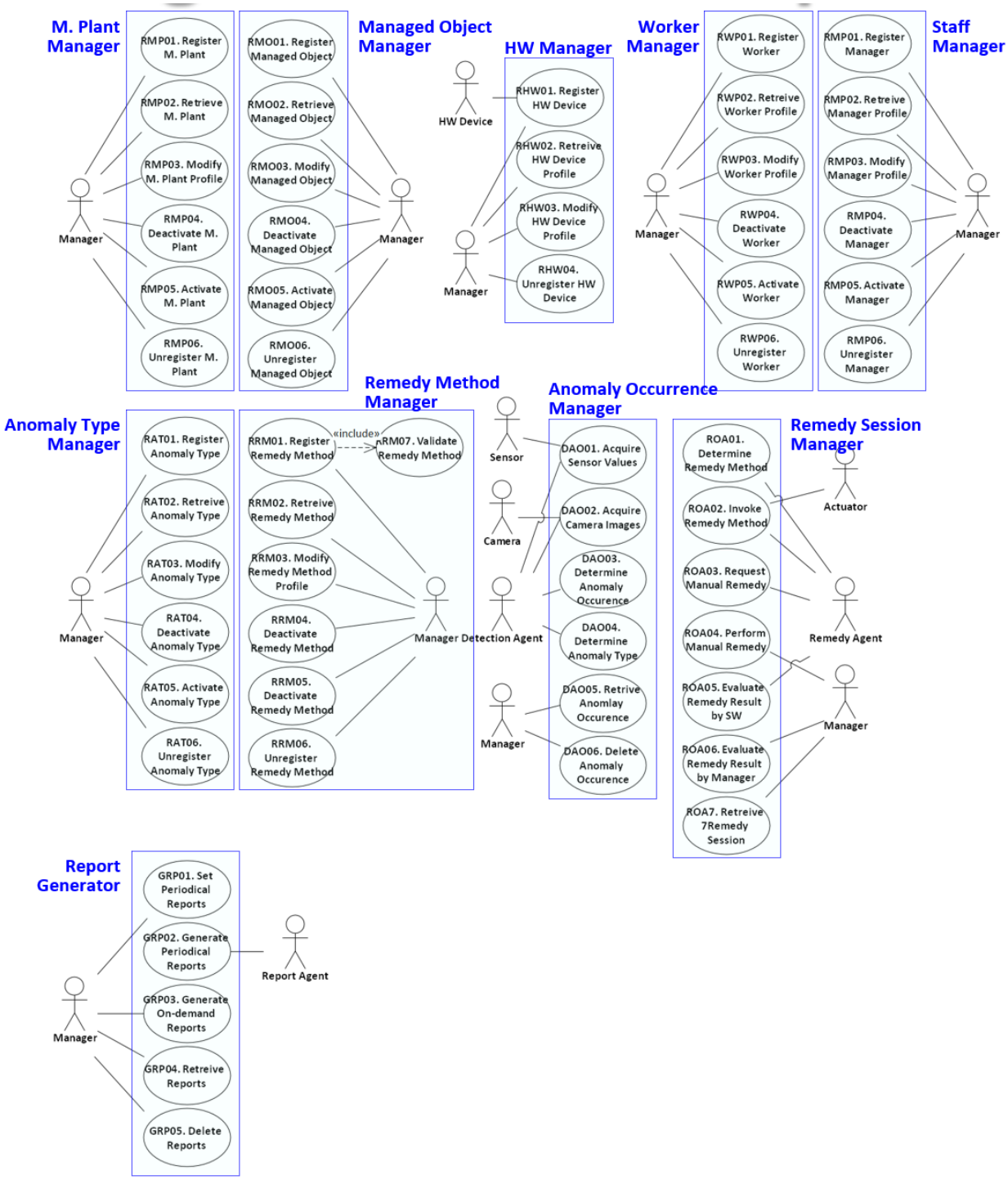


Figure 11. Deriving Functional Components

The summary of the functional components and their relevant use cases is shown in Table 5.

Table 5. Functional Components and their Use Cases

Functional Components	Use Cases
M. Plant Manager	RMP01. Register Manufacturing Plant RMP02. Retrieve Manufacturing Plant RMP03.Modify Manufacturing Plant RMP04. Deactivate Manufacturing Plant RMP05. Activate Manufacturing Plant RMP06. Unregister Manufacturing Plant
Managed Object Manager	RMO01. Register Managed Object RMO02. Retrieve Managed Object RMO03. Modify Managed Object RMO04. Deactivate Managed Object RMO05. Activate Managed Object RMO06. Unregister Managed Object
HW Manager	RHW01. Register HW Device RHW02. Retrieve HW Device Profile RHW03. Modify HW Device Profile RHW04. Unregister HW Device
Worker Manager	RWP01. Register Worker RWP02. Retrieve Worker Profile RWP03. Modify Worker Profile RWP04. Deactivate Worker RWP05. Activate Worker RWP06. Unregister Worker
Staff Manager	RMP01. Register Manager RMP02. Retrieve Manager Profile RMP03. Modify Manager Profile RMP04. Deactivate Manager RMP05. Activate Manager RMP06. Unregister Manager
Anomaly Type Manager	RAT01. Register Anomaly Type RAT02. Retrieve Anomaly Type RAT03. Modify Anomaly Type RAT04. Deactivate Anomaly Type RAT05. Activate Anomaly Type RAT06. Unregister Anomaly Type
Remedy Type Manager	RRM01. Register Remedy Method

	RRM02. Retrieve Remedy Method RRM03. Modify Remedy Method Profile RRM04. Deactivate Remedy Method RRM05. Activate Remedy Method RRM06. Unregister Remedy Method RRM07. Validate Remedy Method
Anomaly Occurrence Manager	DAO01. Acquire Sensor Values DAO02. Acquire Camera Images DAO03. Determine Anomaly Occurrence DAO04. Determine Anomaly Type DAO05. Retrieve Anomaly Occurrence DAO06. Delete Anomaly Occurrence
Remedy Session Manager	ROA01. Determine Remedy Method ROA02. Invoke Remedy Method ROA03. Request Manual Remedy ROA04. Perform Manual Remedy ROA05. Evaluate Remedy Result by SW ROA06. Evaluate Remedy Result by Manager ROA07. Retrieve Remedy Session
Report Generator	GRP01. Set Periodical Reports GRP02. Generate Periodical Reports GRP03. Generate On-demand Reports GRP04. Retrieve Reports GRP05. Delete Reports

- Components derived from Architecture Styles

  There are no functional components derived from the Architecture styles.

- Interface Components

  The target system has the following interface-centric Functional Components. These components are to effectively handle the heterogeneity of sensor, camera, and actuator devices from different manufacturers.

  - Sensor Abstraction Layer
  - Camera Abstraction Layer
  - Actuator Abstraction Layer

5.1.4. [Step 4] Refine Functional Components for Tiers

This step is to refine functional components on the tiers of skeleton architecture. Since the skeleton architecture of the target system has only one tier, this step is not applicable.

- ❑ Not Applicable

5.1.5. [Step 5] Allocate Functional Components

This step is to find the Functionality Place Holders and allocate functional components onto functionality place holders in the skeleton architecture.

- ❑ Functionality Place Holders

  A functional place holder is a layer, a partition, or any place which is defined to host some functionality. Often, the control layer of each tier becomes the functionality place holder.

  The functionality place holders for the target system are shown in Figure 12.

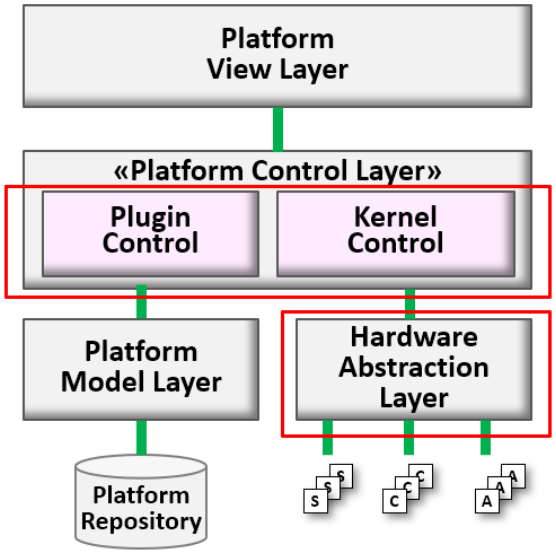


Figure 12. Functionality Place Holders of the Target System

- ❍ Plugin Control Layer
- ❍ Kernel Control Layer
- ❍ Hardware Abstraction Layer

❑ Allocating Functional Components

Since the skeleton architecture is defined with Micro Kernel Architecture style, we classify the functional components into two partitions: Kernel Control and Plugin Control.

The classification of the functional components is shown in Table 6

Table 6. Partitioning Functional Components

Place Holders / Functional Components	Kernel Control	Plugin Control
M. Plant Manager	M. Plant Manager	
Managed Object Manager		Managed Object Manager
HW Manager	HW Manager	
Worker Manager	Worker Manager	
Staff Manager	Staff Manager	
Anomaly Type Manager		Anomaly Type Manager
Remedy Type Manager		Remedy Type Manager
Anomaly Occurrence Manager		Anomaly Occurrence Manager
Remedy Session Manager	Remedy Session Manager	
Report Generator		Report Generator

As shown in the table, the functional components with no variability are allocated to Kernel Control layer, and the components with variability are allocated to Plugin Control layer.

❍ Remedy Session Manager

Note that 'Remedy Session Manager' is allocated on the kernel control layer since the functionality of this component is to invoke other components in a fixed workflow.

➢ Observation 1.

If the workflow of managing remedy session is variable, it should be placed on Plugin Control layer to allow bindings of various workflow variants.

➢ Observation 2.

If some part of the workflow of is fixed and the other part of the workflow is variable, it should be placed on both layers.

Remedy Session Manager	kRemedy Session Manager	pRemedy Session Manager

Then, *kRemedy Session Manager* is designed to provide the fixed and the default workflow of session management and the *pRemedy Session Manager* is defined with its required interface that can be implemented for the variants of the workflow.

- Components in Plugin Control layer

  The functionality of these components is highly dependent on the target manufacturing plant, its managed objects, and anomaly types to handle. Therefore, they are allocated to Plugin Control layer.

We now allocate the refined functional components on the functionality holders according to the table of refined functional components, as shown in Figure 13.

Place Holders / Functional Components	Kernel Control	Plugin Control
M. Plant Manager	M. Plant Manager	
Managed Object Manager		Managed Object Manager
HW Manager	HW Manager	
Worker Manager	Worker Manager	
Staff Manager	Staff Manager	
Anomaly Type Manager		Anomaly Type Manager
Remedy Type Manager		Remedy Type Manager
Anomaly Occurrence Manager		Anomaly Occurrence Manager
Remedy Session Manager	Remedy Session Manager	
Report Generator		Report Generator

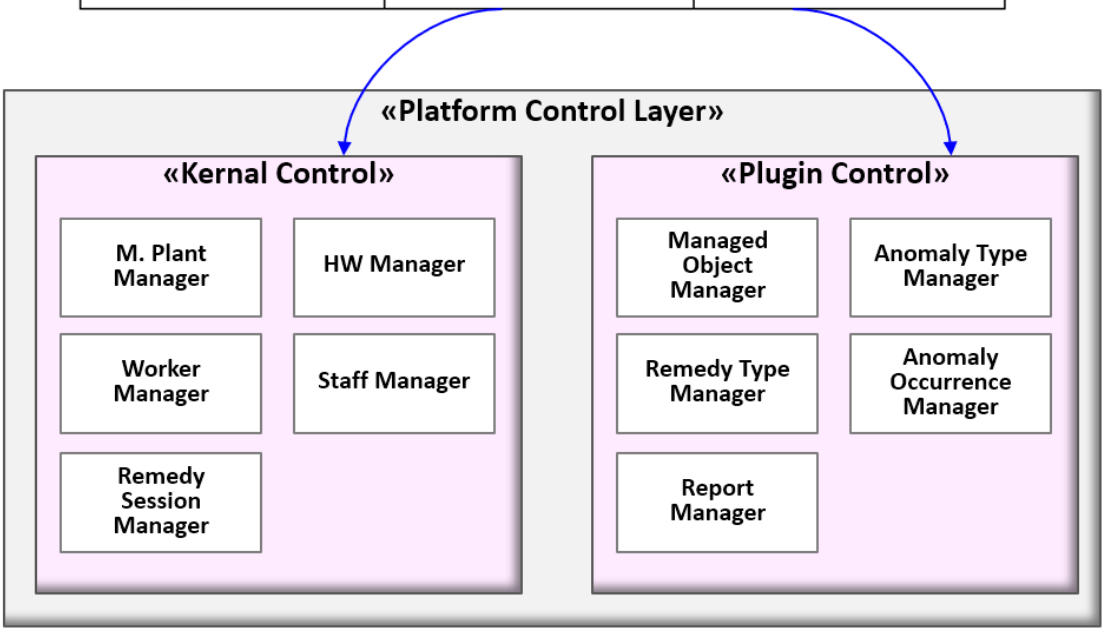


Figure 13. Allocaiton of Components on Functionality Place Holders

Each place holder is defined with its functional components. The skeleton architecture is now more complete with functional components. The functional components in *Plugin Control* will be defined with its interfaces for plugin objects.

- Defining Interface Components

  According to the SRS, we define three types of interfaces as shown in Figure 14.

Figure 14. Interface Components for HAL

Each interface would be specialized for the types of hardware. For example, *Sensor Interface* can be specialized into types of sensors such as *Light Sensor, Smog Sensor,* and *Vibration Sensor*.

5.1.6. [Step 6] Design Functional Components

Each functional component must be designed in greater detail. There are different options for designing functional components in detail. We use the following criteria for determining the type of each functional component.

- Criterion 1. Visibility of Functional Component

  Whitebox Component or Blackbox Component

- Criterion 2. Type of Interface (for Blackbox Components)

  Façade-type or Mediator-type

- Criterion 3. Variability of Functional Component

  Closed Component or Open Component

- Criterion 4. Variation Points (for Open Component)

  Variation point where the variability occurs.

We use the table of Functional Component Design Decisions as shown in Table 7.

Table 7. Design of Functional Components

Refined Functional Components	Visibility	Interface Type	Open/Closed	Variant Points
M. Plant Manager	Blackbox	Façade	Closed	N/A
Managed Object Manager	Blackbox	Mediator	Open	Target M. Object
HW Manager	Blackbox	Façade	Closed	N/A
Worker Manager	Blackbox	Façade	Closed	N/A
Staff Manager	Blackbox	Façade	Closed	N/A
Anomaly Type Manager	Blackbox	Mediator	Open	Algorithm
Remedy Type Manager	Blackbox	Mediator	Open	Algorithm
Anomaly Occurrence Manager	Blackbox	Mediator	Open	Algorithm
Remedy Session Manager	Blackbox	Mediator	Closed	N/A
Report Generator	Blackbox	Mediator	Open	Scope, Format

- Blackbox Components

  All the functional components derived from the SRS are defined as the type of blackbox component for reusability and maintainability.

- Whitebox Components

  None

- Components with Openness

  The components with openness shows the variability and hence they are designed with open-design schemes.

5.1.7. [Step 7] Define Interfaces of Functional Components

A functional component provides its functionality through a *provided* interface. A functional component with 'open' design may also need a *required* interface if it accepts a pluggable object as a variant. Note that the 'Required Interface' is only one of various ways of designing components with openness.

We first define the names of *provided* Interfaces for functional components. We use a prefix of 'i' to indicate an interface name.

Table 8. Interfaces of Functional Components

Tiers / Functional Components	A.M Platform
M. Plant Manager	iM. Plant Manager
Managed Object Manager	iManaged Object Manager
HW Manager	iHW Manager
Worker Manager	iWorker Manager
Staff Manager	iStaff Manager
Anomaly Type Manager	iAnomaly Type Manager
Remedy Type Manager	iRemedy Type Manager
Anomaly Occurrence Manager	iAnomaly Occurrence Manager
Remedy Session Manager	iRemedy Session Manager
Report Generator	iReport Generator

We define the interface for each essential component. In this CEP, we define the interface of only one component, *Anomaly Occurrence Manager*. The component has the following use cases.

- DAO01. Acquire Sensor Values
- DAO02. Acquire Camera Images
- DAO03. Determine Anomaly Occurrence
- DAO04. Determine Anomaly Type
- DAO05. Retrieve Anomaly Occurrence
- DAO06. Delete Anomaly Occurrence

- Provided Interface of *iAnomaly_Occurrence_Manager*

  By referring the use cases in the component, we define the following method signatures.

  - get_sensor_values(List_Sensors): Measurement;
    This method is to read the sensor values for the given set of sensors. The result of reading sensor values is stored in an object of 'Measurement' class.
  - get_camera_Images(List_Cameras): list of captured images;
    This method is to acquire images from the given cameras.
  - detect_anomaly (List of Managed Objects): List of Anomaly Occurrence;
    This method is to detect the occurrence of anomaly on the given list of managed objects.
  - determine_anomaly_type(Anomaly_Occurrence): Anomaly_Type;
    This method is to identify the sub-type and intensity of the given anomaly occurrence
  - retrieve_anomaly(search_condition): List of Anomaly occurrences;
    This method is to search and return the list of anomaly occurrences for the given condition.
  - delete_anomaly(): Boolean;
    This method is to delete the current anomaly occurrence.

5.2. Information View

Architecture design for Information View is to make decisions about persistent datasets, properties, and their management. This activity includes a number of tasks including identifying data components, allocating data components, defining their data contents, ownership, data distribution, replication, migration, data security, and data timeliness.

5.2.1. [Step 1] Observe Informational Characteristics

We first observe the informational characteristics of the target system.

- ❑ Adding Session-related Classes
- ❑ Refining Relationships for various Conceptual Object Classes

5.2.2. [Step 2] Refine Persistent Object Model

In object-oriented paradigm, persistent datasets are modeled as entity-type classes. Hence, we refine the context-level class diagram with greater details, which becomes the basis for deriving data components.

- ❑ Refined Class Diagram

  The revised class diagram shows additional classes and refined relationships as shown in Figure 15. Note that attributes are omitted in this sample solution.

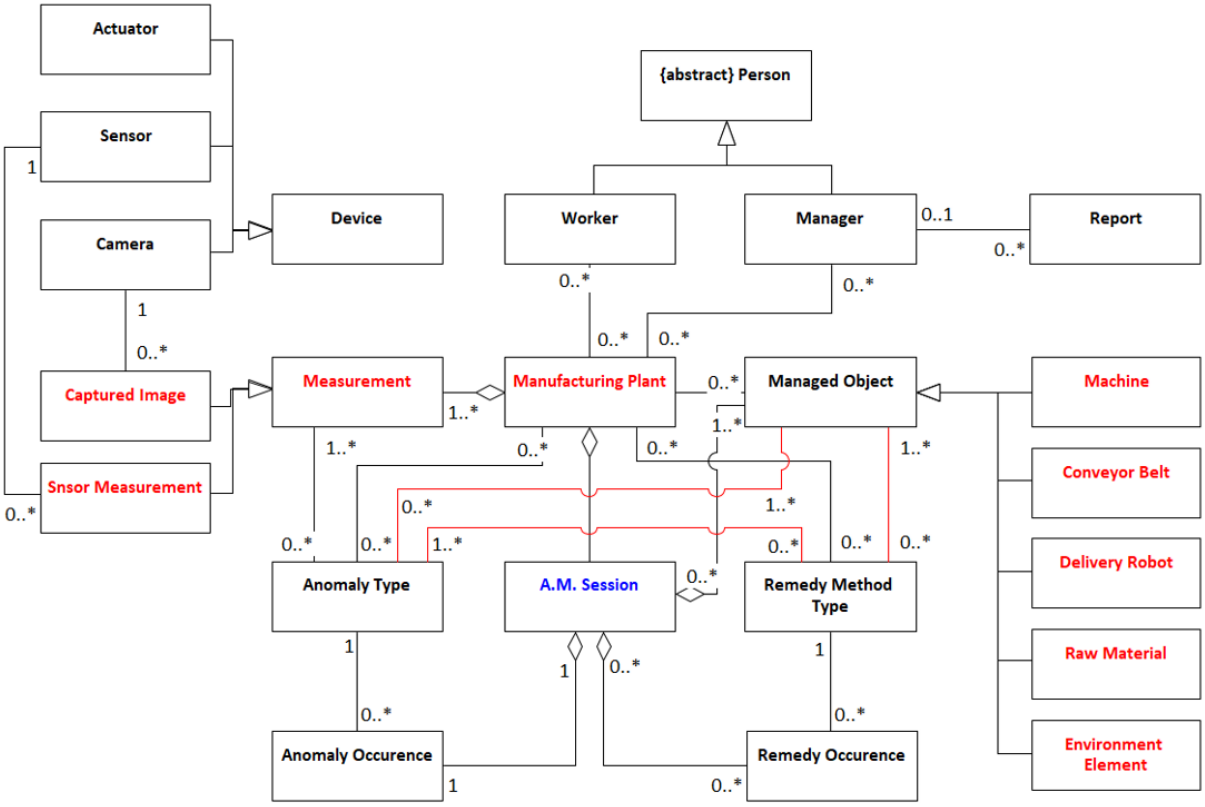


Figure 15. Refined Class Diagram

The refined class diagram shows refinements in colors.

- Managed Object is specialized in specific types.
- 'Measurement Class' is added with its sub-classes. This is to store the acquired measurements from sensors and cameras on persistent storage for further analysis.
- Manufacturing Plant now has a relationship with 'Measurement' class, which is a newly added class.
- Refinements on Relationships
  - Relationships between *Managed Object* and *Anomaly type*.
  - Relationships between Managed Object and Remedy Method type.
  - Relationships between Anomaly Type and Remedy Method type.

- Each class is specified with a textual description.

  Omitted in this sample solution.

5.2.3. [Step 3] Derive Data Components

By considering the strengths of inter-class relationships, we group a set of related classes into a data component as shown in Figure 16.

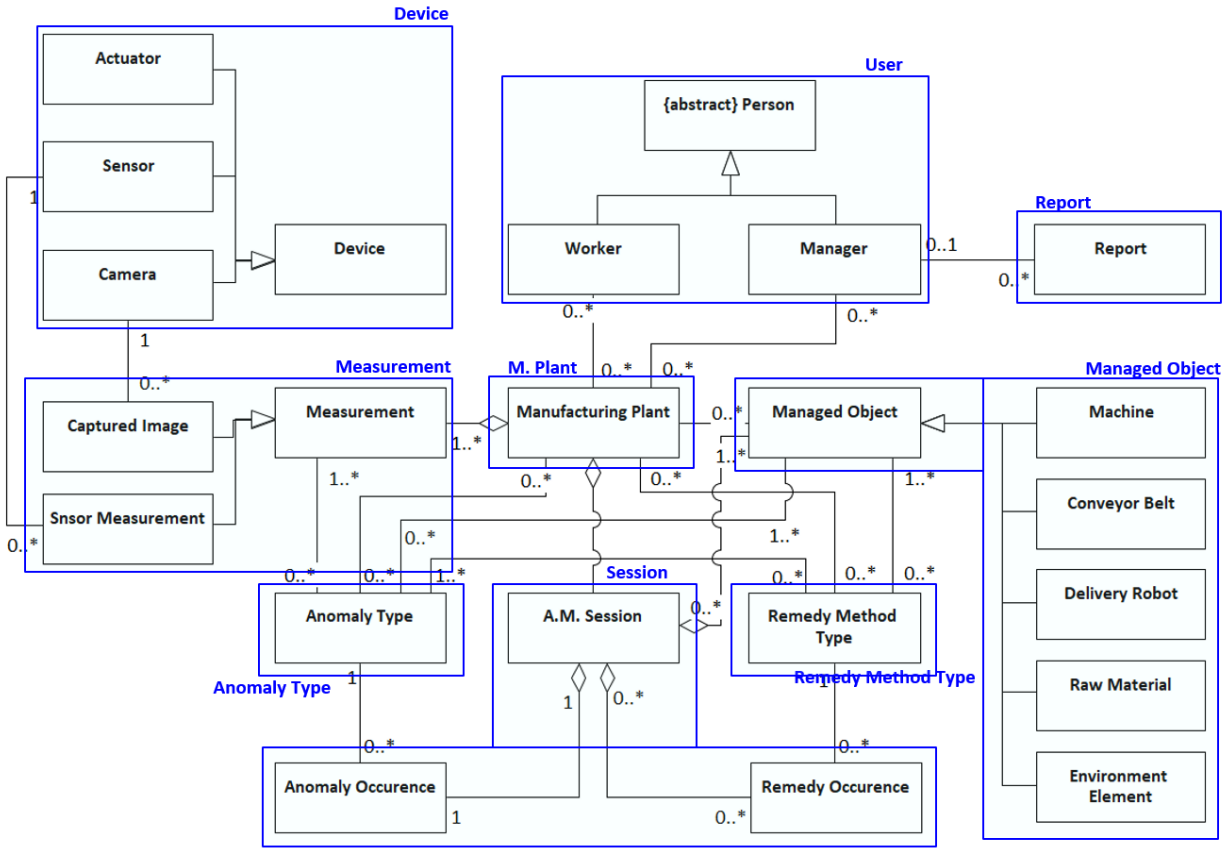


Figure 16. List of Data Components

As shown in the figure, classes with strong dependency such as inheritance and compositions are grouped into the same data component. The data components that are derived by considering the strengths of relationships among classes.

❑ Data Components derived from Architecture Style

None

5.2.4. [Step 4] Refine Data Components for Tiers

The skeleton architecture of the target system has multiple tiers, and hence we need to refine them for the tiers. Table 9 shows the allocation of data components on tiers.

❍ In fact, the A.M. platform has only one tier.

Table 9. Data Components allocated on Tiers

Tier / Data Components	A.M. Platform
M. Plant	M. Plant
User	User
Device	Device
Managed Object	Managed Object
Anomaly Type	Anomaly Type
Remedy Method Type	Remedy Method Type
Measurement	Measurement
Session	Session
Report	Report

❑ Considering the Consistency with Allocation of Functional Components

The allocation of data components is well aligned with the allocation of functional components. Hence, the inter-tier access between functional components and data components is not presented.

5.2.5. [Step 5] Allocate Data Components

Allocate the data components to the appropriate place holders of the architecture. The data component holders for the target system are shown in Figure 17.

❑ Place Holders for Data Components

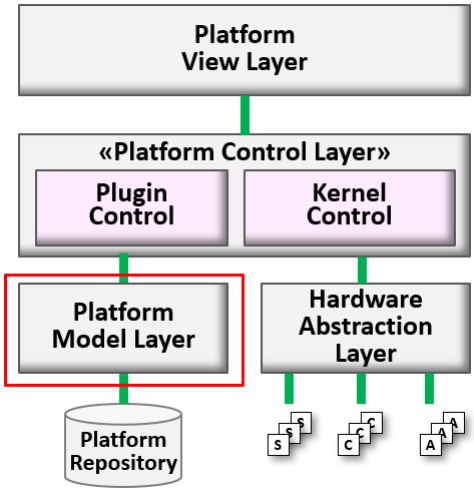


Figure 17. Data Component Holders of the Target System

There is only one data component holder in the system: *Platform Model Layer*.

We now allocate the data components on the data component placeholders on each tier. We use the table of refining data components over tiers as shown in Figure 18.

Data Components \ Tier	A.M. Platform
M. Plant	M. Plant
User	User
Device	Device
Managed Object	Managed Object
Anomaly Type	Anomaly Type
Remedy Method Type	Remedy Method Type
Measurement	Measurement
Session	Session
Report	Report

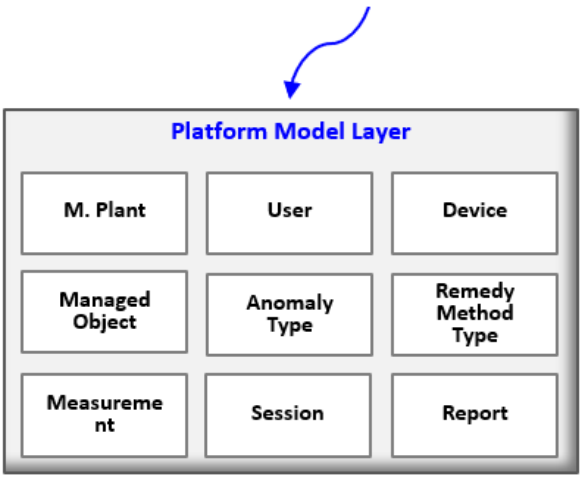


Figure 18. Allocation of Data Components

The allocation of data components is made according to the table for 'Data Components Refinement'. Each model layer is allocated with an appropriate set of data components.

5.2.6. [Step 6] Design Data Components

This step is to design the internal details of data components. This is trivial since each data component consists of classes and each class is defined with persistent attributes.

- ❑ Omitted in Sample Solution

5.2.7. [Step 7] Define Interfaces of Data Components

This step is to define the interface of each data component. The interface for data components is mostly for CRUD-type data manipulation.

- ❑ Omitted in Sample Solution

5.3. Behavioral View

The behavioral view of the architecture describes the dynamic aspect of the system, focusing on the runtime behavior of the system.

5.3.1. [Step 1] Observe Behavioral Characteristics

The observations made in the behavioral context are applicable in this view-level design. The system behavior exhibits the following invocation types.

The invocation patterns for functional groups are refined as shown in **Error! Reference source not found.**.

Table 10. Invocation Patterns defined for Functional Groups

	A.M. System
Registering Manufacturing Plants	Explicit
Registering Managed Objects	Explicit
Registering Hardware Devices	Explicit
Registering Worker Profiles	Explicit
Registering Manager Profiles	Explicit
Registering Anomaly Types	Explicit
Registering Remedy Methods	Explicit
Detecting Anomaly Occurrences	Closed Loop, Event-based
Remedying Anomaly Occurrences	Closed Loop, Event-based
Generating Operation Reports	Closed Loop, Explicit. Event-based

Now, the control flow of the target system can be well modeled based on the specified invocation patterns.

5.3.2. [Step 2] Refining Control Flow of the Platform

We design the overall control flow of the target system. This is done by refining the context-level activity diagram. If the target system has multiple tiers, each tier has its own control flow and the interaction between the tiers should also be designed.

All the use cases in the functional view are reflected in the activity diagram, and all the actions and activities have their corresponding use cases. Hence, the consistency between the use case diagram and the activity diagram is well-maintained.

The refined control flow of the platform is shown in Figure 19.

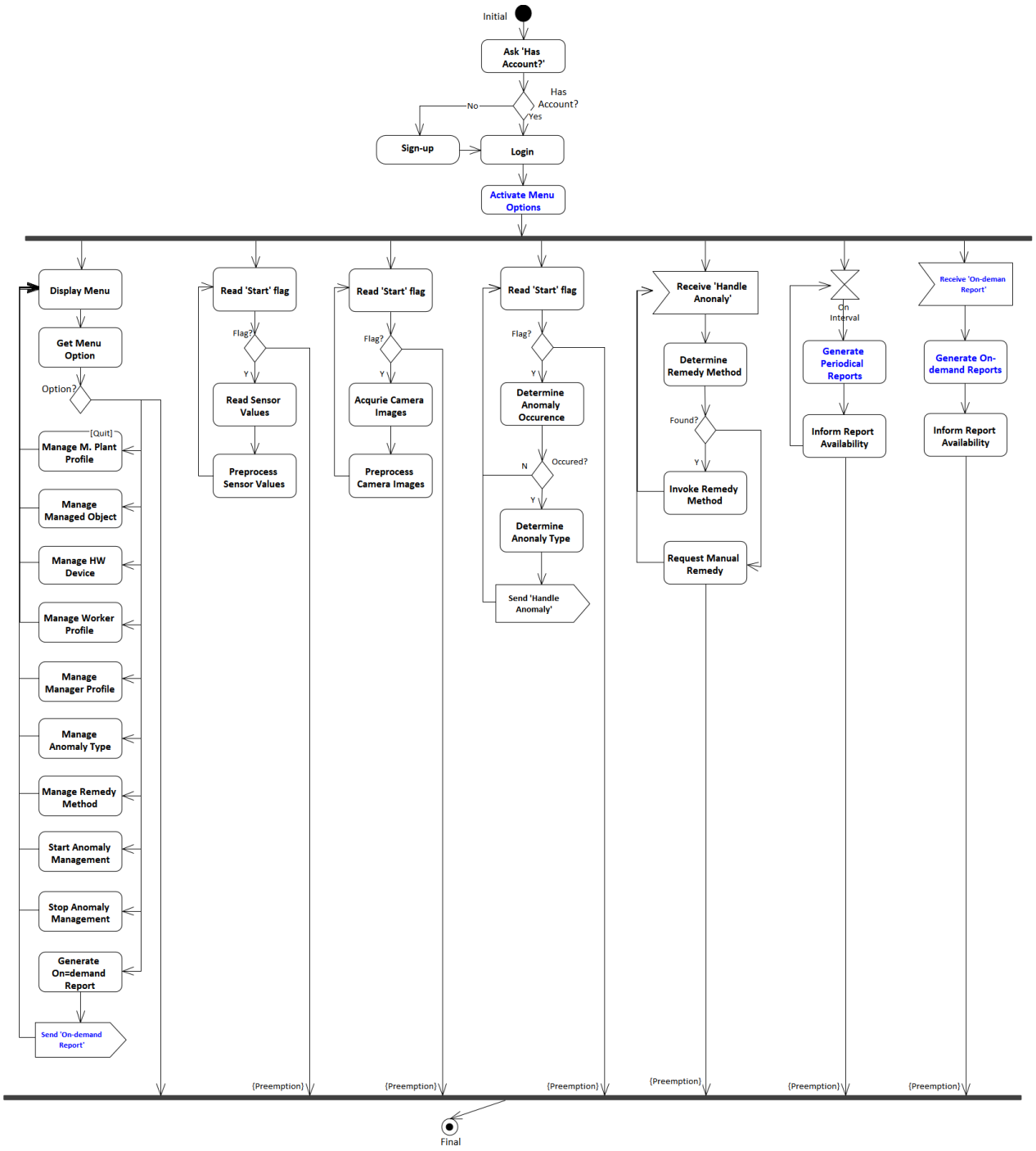


Figure 19. Refined Control Flow of A.M Platform

- Refinements made on the Control Flow
  - Right after a successful login, the system activates the menu options that are available to the user. For example, the manage is provided with only the accessible menu options.
  - Adding ‘event-based’ generation of on-demand reports. On the thread #1, an event is sent whenever ‘Generate On-demand Report’ is trigged, and the event is handled by a separate thread, the rightmost thread, which is generate the requested report and inform the availability of the new report to the user.

This is to consider the generation of complex on-demand reports that might takes minutes and hours to generate.

5.3.3. [Step 3] Choosing Element for Detailed Control Flow

In this step, we choose the functionality with complex control flows. That is, we chose use cases in Use Case Diagram, functional components, or actions and/or activities in Activity Diagram. Then, we perform a detailed behavior design for each element chosen.

For the target system, we choose the following elements for detailed control flow as shown in Figure 20.

- Action, i.e., 'DAO003, Determine Anomaly Occurrence', on the activity diagram.

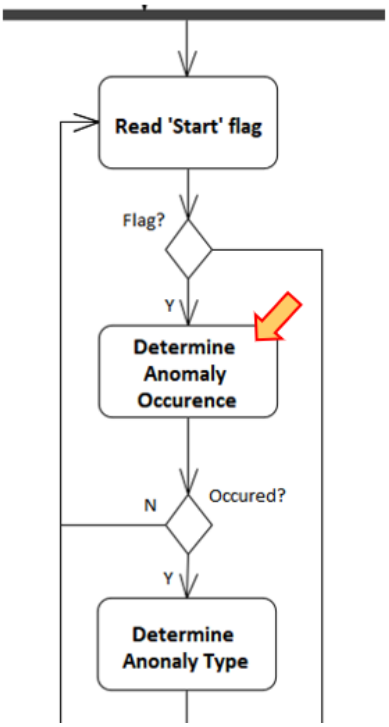


Figure 20. Target Element for Designing Detailed Control Flow

5.3.4. [Step 4] Detailed Control Flow for 'Determine Anomaly Occurrence.'

- Functionality

  The functionality of this action is to determine the occurrences of anomaly on manage objects. The detection of anomaly can be done by observing the measures from sensors and the camera images.

- Input
  - Measurements from Sensors
  - Captured Images from Cameras
  - Optionally

    Environmental Contexts from Public Information Providers such as Weather System, Traffic System, and Disaster Alert System.

- Output
  - A List of Anomaly Occurrences

    Each anomaly occurrence is a profile that contains the details of the anomaly such as date, time, managed object, anomaly type, and anomaly intensity.

- Method to Determine Anomaly Occurrence → Classification Model

  We adopt a classification machine learning model for detecting anomalies.

  - Machine Learning Algorithm to apply → Support vector machine (SVM)

    SVM is a classification algorithm that can classify objects into classes, and it often performs better than conventional regression models due to the hyperplane-based classification.

  - Dataset for Training the Model

    We will acquire the previous history of anomaly occurrences on target managed objects. The training set can be configured by considering the currently available datasets, which may be in-house datasets or publicly available datasets.

  - Feature Set

    A feature set is a set of features on which a classification model is trained. Hence, the selection of the features is essential in producing high performing classification models.

    However, the feature set depends on the target managed object and its anomaly type. Hence, it should be defined as a variation point.

  - Performance Metric

    There exist different metrics for evaluating SVM models including Accuracy, Confusion Matrix, Log-Loss, and AUC-ROC. The system can be designed to support all the commonly used metrics for flexibility.

- Algorithm
  - For each managed object, OBJ, in List_Managed_Object

    For each applicable anomaly type, AnoType DO {

    Step 1. Activate the right Anomaly Detection model for (OBJ, AnoType);

    Step 2. Acquire environmental contexts from sensors and cameras

    Step 3. Validate the acquired environment contexts.

    If invalid, break the sequence.

    Step 4. Preprocess the environment contexts for the SVM model.

    Step 5. Detect the occurrence of anomaly on (OBJ, AnoType).

    If no occurrence, break the sequence.

    Step 6. Compute the intensity and other descriptors for the occurrence.

    // This can be done in various ways.

    Step 7. Add the new anomaly occurrence to List_Anomaly_Occurrence

    }

  }

- Open Design for Step 1

  We add a new class, 'Anomaly Detection Model' class, and define its relationships with other classes, as shown in Figure 21

  - An instance of Anomaly Detection Model, *ADM-ID*, is created for each pair of (OBJ, AnoType)
    - where OBJ is an instance of a concrete Managed Object and
    - AnoType is an instance of a concrete Anomaly Type.

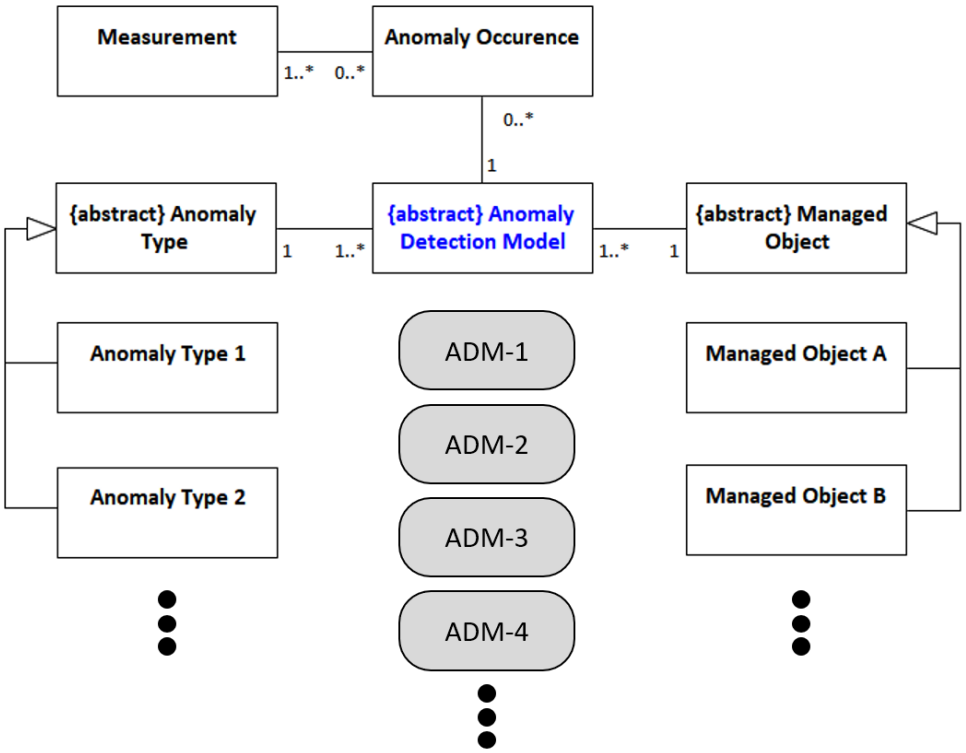


Figure 21. Refined Class Diagram with Anomaly Detection Model

The open design scheme with specializing abstract classes allows the flexible additions and modifications of anomaly detection models.

- Open Design for Step 3

  The variation point is on the way the environment context is validated.

  Note that a classification model such as SVM is associated with a feature set and the validation should be done by the valid value ranges of the features in the set. Each classification model is defined with its own feature set, i.e., variable.

  To allow variants of the validation method, we define an abstract method, and let it implement for each SVM model.

  - abstract validateContext(): Boolean.

- Open Design for Step 6

  The variation point is on the way the intensity of occurred anomaly is computed.

  Note that the computation of intensity depends on the target managed object and its anomaly type for each Anomaly Detection Model. That is, the computation is done by observing the sensor measurements and camera images used by the pair of (OBJ, Anotype).

  To allow variants of the computation method, we define an abstract method.

  - Abstract computeIntensity*():*

    A float number between 0 and 1 where a value near '0' for a light level of occurrence severity and a value near '1' for a severe level occurrence.

5.4. Deployment View

Deployment view of the architecture is concerned with the topology of software components on the physical layer, as well as the physical connections between these components.

5.4.1. [Step 1] Observe Deployment Characteristics

- Programming Languages to support

  A.M. Platform will initially be available for Python implementations.

  - Python

- Execution Environment

  The execution environment will depend on the programming language.

  - For Python implementation

- Operating System
  - Ubuntu

5.4.2. [Step 2] Define Nodes

The skeleton architecture of the target system consists of a single node. Each node is configured with a hardware specification and its execution environment.

- Node 1. A.M. Platform
  - Hardware Specification

    Server Computer
    - CPU: Intel Zeon Cascade Lake Platinum 8280 Processor
    - RAM: 128GB
    - SSD: 4TB
    - HDD: 32TB
    - GPU: GeForce RTX 4090
  - Execution Environment
    - Operating System: Ubuntu

5.4.3. [Step 3] Define Network Connectivity

There is no network connectivity within the A.M Platform, as it is configured as a single-tier architecture.

5.4.4. [Step 4] Define Artifacts to Deploy

- All the Functional Components specified in Functional View-design
- All the Data Components specified in Information View-design

5.4.5. [Step 5] Allocate Artifacts on Nodes

This step is to allocate all the deployable artifacts and show the network connections.

- Deployment for Python version

  We use a Deployment Diagram to represent the nodes and artifacts as shown in Figure 22.

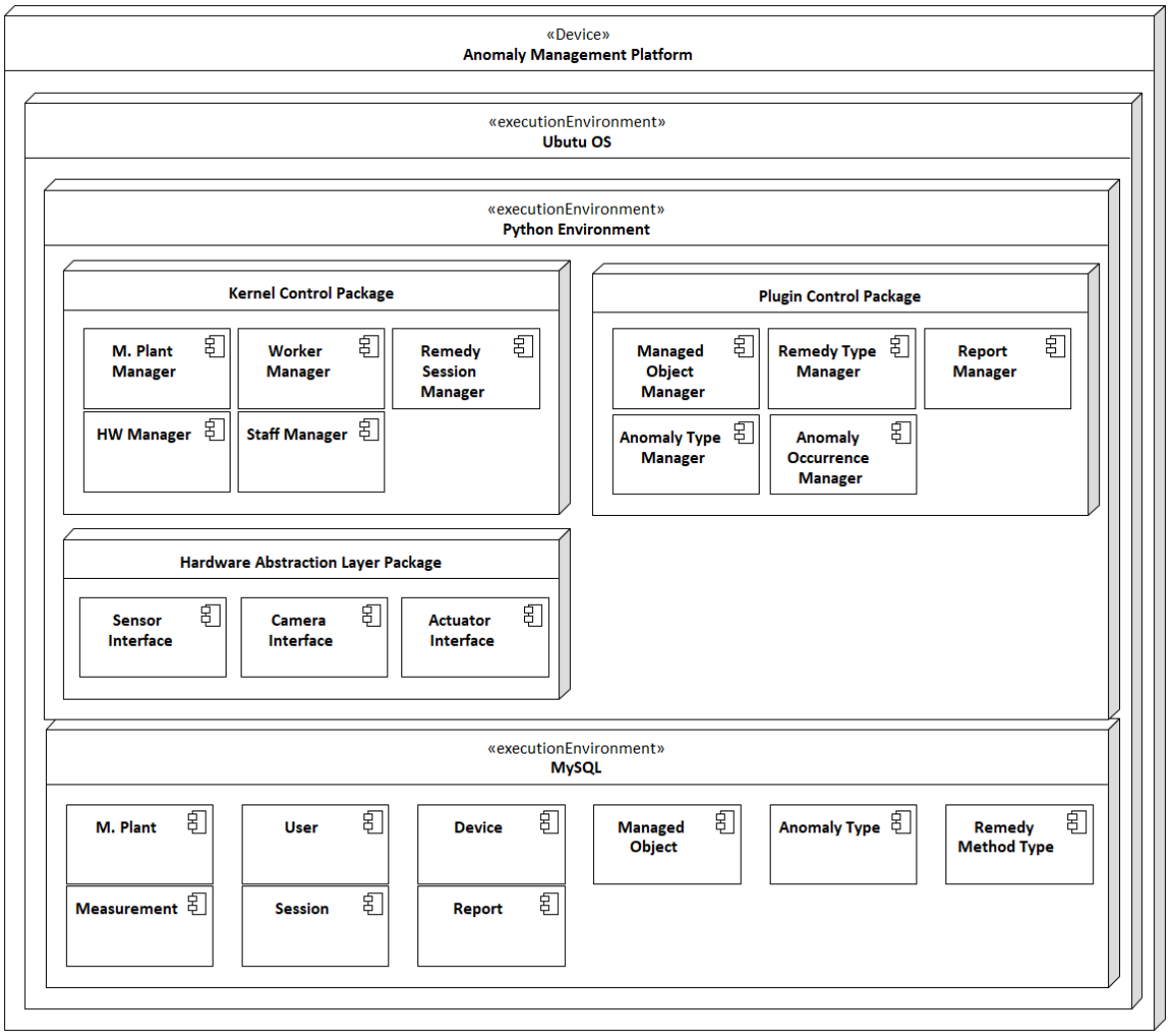


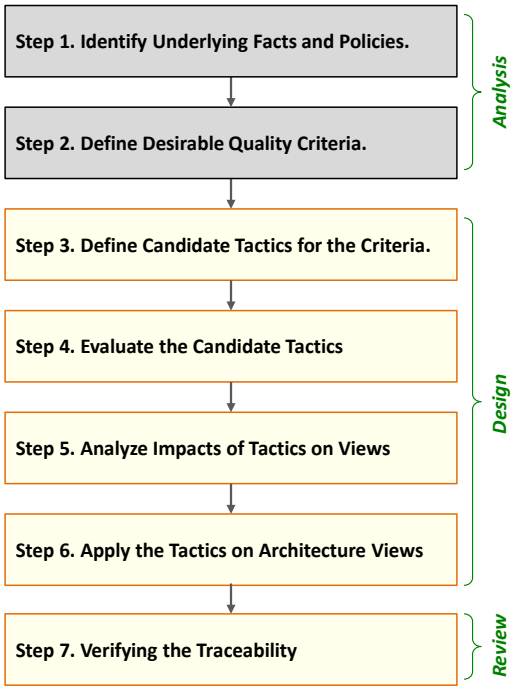
Figure 22. Deployment Diagram for the Target System

- Deployment for Java version

  Omitted in this sample solution.

6. Activity 5. NFR-specific Architecture Design

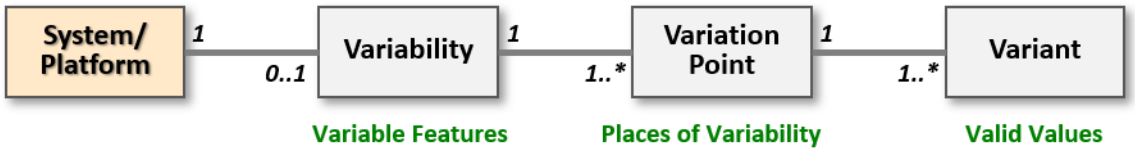
This chapter shows the architectural design for the given NFRs. The following process is used for applying NFR-based design.



6.1. Design for NFR-1. High Configurability of the Anomaly Management Platform

Anomaly Management Platform is a software framework, which should be generic and configurable in developing various types of anomaly management systems for different manufacturing domains and plants. That is, the variability among the target application systems should be well-modeled in advance and the platform should be designed by considering all the essential variation points. Doing so, the platform can be widely applied to developing various target systems.

The relationship between variability, variation points and variants are here.



- A system may have a variability, which is defined as one or more variation points. Each variation point is associated with one or more variants, which is a valid value that can be bound to a variation point.

Configurability of the platform is the degree of effectiveness in customizing the platform for a target anomaly management application system. That is, the platform should be designed to handle all the essential variation points and effectively bind the application-specific variants on the variation points.

The key variation points to handle in the platform are the following.

- Heterogeneity on Manufacturing Domains
- Heterogeneity on Manufacturing Plants
- Heterogeneity on Target Products
- Heterogeneity on Hardware Devices
- Heterogeneity on Anomaly Types
- Heterogeneity on Methods to detect Anomaly Occurrences
- Heterogeneity on Remedy Methods
- Heterogeneity on Metrics to evaluate the Anomaly Management Performance
- Heterogeneity on Types of Reports

6.1.1. [Step 1] Identify Facts and Policies

Given the NFR of Platform Configurability, we can reasonably establish the following facts.

- (F1) High Coverage of Platform

  By examining the given requirement in its entirety, we can conclude that a platform with high coverage will enhance its configurability, allowing it to accommodate a broad range of anomaly management applications.

- (F2) High Customizability of Platform

  Platform configurability is defined as the degree of effectiveness in customizing the platform for various anomaly management applications. The effectiveness of this customization is determined by the cost and time required to tailor the platform for specific use cases of target applications.

  A platform that offers effective customization schemes for variation points can reduce the cost and time needed for customization. We can infer the following variation points from the given NFR.

  - (F2a) Variation Point on Manufacturing Domains
  - (F2b) Variation Point on Manufacturing Plants
  - (F2c) Variation Point on Hardware Devices
  - (F2d) Variation Point on Anomaly Types
  - (F2e) Variation Point on Methods to detect Anomaly Occurrences
  - (F2f) Variation Point on Remedy Methods
  - (F2g) Variation Point on Method to evaluate results of applying Remedy Methods
  - (F2h) Variation Point on Report Types
  - Note

    The NFR specifies the heterogeneity of target products produced by the manufacturing plants. However, the manufacturing domain and the plant together determine the target products. Hence, we do not consider the variability of target products.

  - Note

    While there may be additional variation points that could be incorporated into the platform, we have not identified them at this stage of the architecture design process. Any additional variation points should be addressed in future versions of the platform.

- (F3) High Modularity of Platform

  Software modularity refers to the level of functional independence among components in a system. This independence is measured by evaluating the cohesion of each component and the coupling between components.

When the cohesion of each component is high, it can contribute to the overall effectiveness of the platform. Similarly, when the coupling between components is low, it can also enhance the effectiveness of the platform.

6.1.2. [Step 2] Define Criteria for Tactics

- (C1) Modeling the Scope of Platform (Relevant to F1)

  The platform should be designed by analyzing the commonality among various anomaly management systems and reflecting the commonality in the platform design.

- (C2) Apply OCP for Variability on Manufacturing Domains (Relevant to F2a)

  The variability on manufacturing domains should be analyzed and the design with OCP should be applied to provide high customizability.

- (C3) Apply OCP for Variability on Manufacturing Plants (Relevant to F2b)

  The variability on manufacturing plants should be analyzed and OCP should be applied to handle the variability.

- (C4) Apply OCP for Variability on Hardware Devices (Relevant to F2c)

  The variability on hardware devices should be analyzed and OCP should be applied to handle the variability.

- (C5) Apply OCP for Variability on Anomaly Types (Relevant to F2d)

  The variability on anomaly types should be analyzed and OCP should be applied to handle the variability.

- (C6) Apply OCP for Variability on Detection Methods (Relevant to F2e)

  The variability on detection methods should be analyzed and OCP should be applied to handle the variability.

- (C7) Apply OCP for Variability on Remedy Methods (Relevant to F2f)

  The variability on remedy methods should be analyzed and OCP should be applied to handle the variability.

- (C8) Apply OCP for Variability on Evaluation Methods (Relevant to F2g)

  The variability on evaluation methods should be analyzed and OCP should be applied to handle the variability.

- (C9) Apply OCP for Variability on Report Types and OCP (Relevant to F2h)

  The variability on report types should be analyzed and OCP should be applied to handle the variability.

- (C10) Design with High Modularity (Relevant to F3)

  The platform should be designed to provide high cohesion of components and low coupling among components.

6.1.3. [Step 3] Define Candidate Tactics

The following tactics are proposed for the identified criteria.

- (T1) Commonality Analysis for the Platform Scope (Relevant to C1)

  Commonality analysis in software design is a process used to identify common requirements or features that can be shared among multiple software systems or components. It is a technique that helps software designers and developers to identify the similarities and differences among different applications or systems, and to determine the most efficient way to reuse or adapt existing components.

  - Apply Commonality Analysis of Product Line Engineering

    Commonality analysis is often used in the context of software product line engineering, where a family of related software products is developed using a common set of features and components.

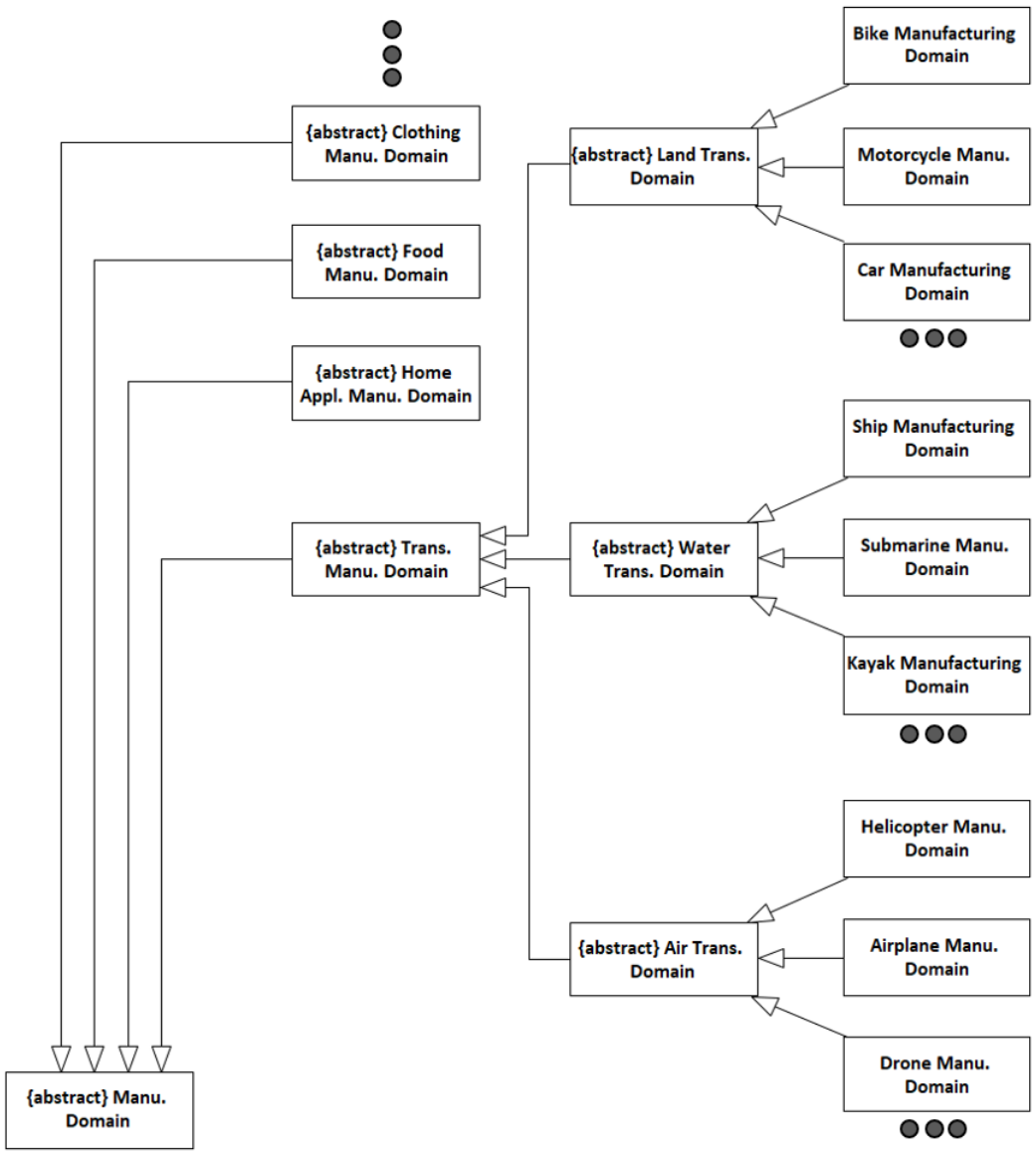
    - Step 1. Define the common features among various anomaly management systems.
    - Step 2. Identify the common features that exhibit variability within the features. Note that a common feature may not exhibit any variability.
    - Step 3. Define variation points in each of common features revealing variability and identify variants for each variation point. Some of the features with variability are already provided in the SRS.

  The application of this tactic results in the commonalty and variability (C&V) analysis of the platform.

- (T2) Apply Specialization for Manufacturing Domains (Relevant to C2)

  There exists a number of well-known manufacturing domains and there could be additional manufacturing domains to be added after the platform is released. Hence, we need to apply the *open-scope* for this variation point.

  We apply *Object Specialization* to define the taxonomy of various manufacturing plants. Some known manufacturing domains are pre-defined as concrete subclasses, and new manufacturing domains can be defined by subclassing an appropriate superclass. The design with object specialization is shown in the following figure.



- Benefits
  - Extendibility → The hierarchy of manufacturing plants can flexibly be expanded with specializing appropriate classes.
  - Reusability → The common attributes and methods can be defined in a class and its subclasses can inherit and reuse them, saving the development a nd maintenance effort.
  - Customizability → In each subclass of a new manufacturing plant, adding additi onal attributes and methods can be added. Also, the inherited methods of a sup erclass can be overridden for the plant-specific features.
- Limitation
  - Rebuilding the System → This tactic requires the rebuilding of the system when ever a new subclass is added.

- (T3) Apply Specialization, Association and Realization for Manufacturing Plants (Relevant to C3)

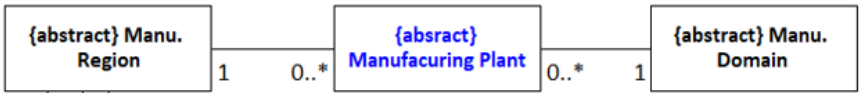
  Manufacturing plants vary large on (1) manufacturing domain and (2) manufacturing region. Different regions typically have different laws, policies, industry regulations, standards, voltages, tax system, and environments including weather. Hence, we define an abstract class of 'Manufacturing Plant' and let it have associations with 'Manufacturing Region' and 'Manufacturing Domain'.

  - Specializing 'Manufacturing Region'

    The abstract class, Manufacturing Region' can be specialized into different regions.

  - Associations of 'Manufacturing Plant'

    An abstract class, Manufacturing Plant, is defined with two association relationships as shown in the following figure.



  - Realization of 'Manufacturing Plant'

    Each manufacturing plant is defined by realizing 'Manufacturing Plant' class. Then, each manufacturing plant is linked to an instance of Manufacturing Region and an instance of Manufacturing Domain.

  - Benefits

    Each of the hierarchy, Manufacturing Domain and Manufacturing Region, can independently expanded and evolved without affecting each other. This enhances the extendibility and maintainability of the platform.

    A new manufacturing plant can efficiently be defined by simply choosing links to the two instances.

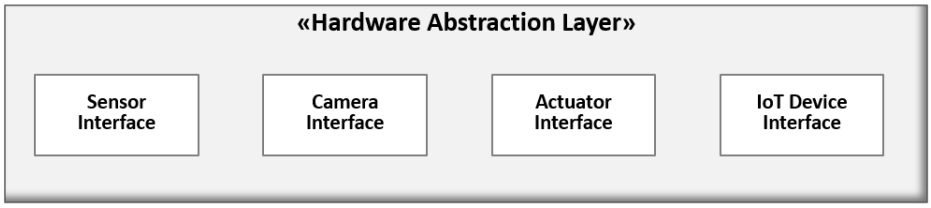
- (T4) Apply a HAL layer for Hardware Devices (Relevant to C4)

  This tactic is to define a Hardware Abstraction Layer (HAL) to accommodate the variability of hardware devices such as the APIs of hardware devices.

  Hardware Abstraction Layer (HAL) is a layer of software that sits between physical hardware devices and the higher-level functional components that interact with it. The purpose of the HAL is to abstract away the low-level details of the hardware, providing a standard interface that can be used by higher-level software without needing to know the specific details of the underlying hardware.

- Interfaces of Hardware Devices in HAL

  The three interfaces of HAL have already been defined in the functional view design. We add a new interface for 'IoT device' for its potential usages in manufacturing plants.



  Note that each interface can be specialized into sub-types of the hardware. For exa mple, the Sensor Interface can be specialized into different types of sensor interfac es such as Temperature Sensor interface, Humidity Sensor interface, Vibration Sens or interface, Light Sensor interface, etc.

- Benefits

  A HAL layer provides a standardized interface that can be used by functional components regardless of the specific hardware devices they are running on. This makes it easier to develop software that can run on multiple platforms without needing to be modified for each individual platform.

  HAL also helps to isolate higher-level functional components from the details of the hardware. This can make the system more modular and easier to maintain, as changes to the underlying hardware can be handled by updating the HAL without needing to modify the higher-level software components.

- (T5) Apply Specialization and Required Interfaces for Anomaly Types (Relevant to C5)

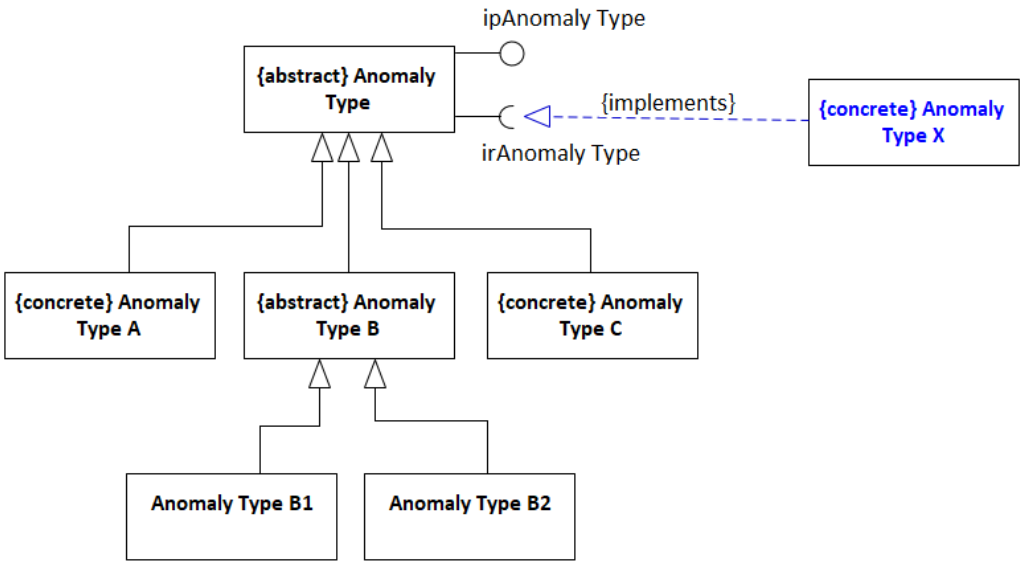
  This tactic is to apply specialization for common and known anomaly types and required interfaces for accepting new Anomaly Types.

  - Specialization of Anomaly Types

    A hierarchy of anomaly types can be defined for known anomaly types, and its structure is similar to that of 'Manufacturing Domain'.

  - Required Interfaces for New Anomaly Types

    A required interface in UML specifies a set of methods that must be implemented by other components to be able to work with the component that defines the interface. The methods in the required interface represent the expected behavior or functionality of the component, and therefore it is utilized as an open design scheme to accept variants in plugin-object form.



Note that the requirement interface, 'irAnomaly Type', defines the set of common methods that must be implemented by a class of new anomaly type. Once the class is implemented using the interface, an instance of the class is created and bound to the 'Anomaly Type' by a method such as *setAnomalyType(irAnomalyType)*.

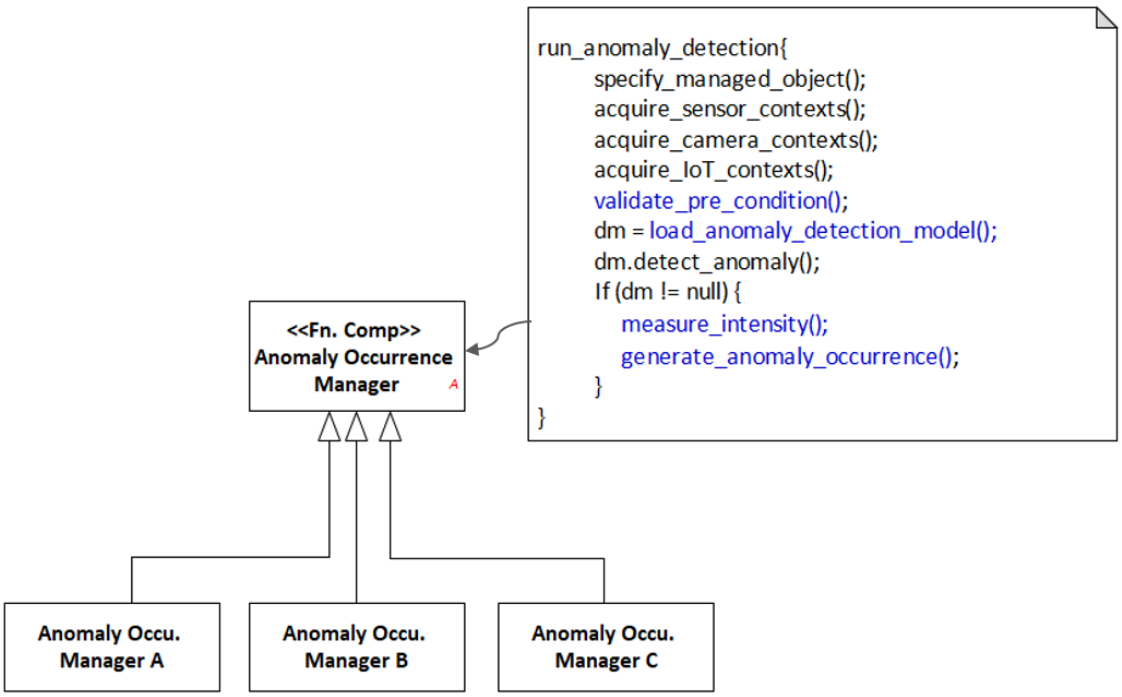
- Customizing the Platform target Anomaly Types
  - Method 1.
    Identity an appropriate abstract class for the target anomaly type. Then, create an instance of the use and use it.
  - Method 2.
    Implement an external class that implements the required interface. And, bound an instance of the external class, i.e., a plug-in object.

- (T6) Apply Template Method pattern for Detection Methods (Relevant to C6)

  This tactic is to apply Template Method pattern to capture the common algorithm of detecting anomaly in a superclass, and let the subclasses implement some steps of the algorithm for the target detection method.

  Template Method pattern provides a template for defining the steps of an algorithm (Common Part) while allowing subclasses to override some of the steps (Variable Part) without changing the overall structure of the algorithm.

  The superclass of this pattern consists of two types of methods: *abstract methods* and *concrete methods*. The abstract methods define the basic steps of the algorithm, while the concrete methods provide default implementations for those steps.



Note that the methods in blue color are defined as abstract methods, that must be im plemented by subclasses.

- Benefit

  We can define a common algorithm for detecting various anomaly types but allow each subclass to customize some steps in the algorithm.

  Also, the concrete methods in the superclass can also be overridden by subclasses to customize the behavior of the algorithm.

- (T7) Apply Template Method pattern for Remedy Methods (Relevant to C7)

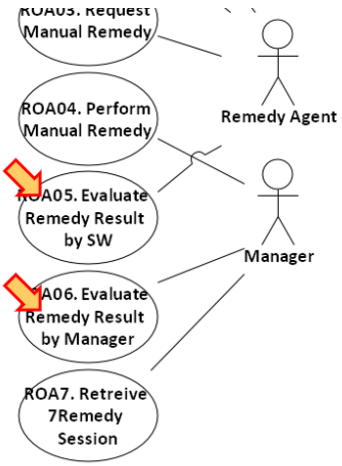
  This tactic is essentially identical to (T6), where a template method pattern is applied except for the following modifications.

  - Define 'Remedy Method Manager' for the superclass.
  - Define the common algorithm for applying remedy tasks on each occurred anomaly. It consists of the following steps.
    - Step 1. Set a target anomaly occurrence to be remedied.
    - Step 2. Identify the managed object that is associated with the anomaly occurrence.
    - Step 3. Identify the causes of the anomaly occurrence by using a causal effect analysis method.
    - Step 4. Run a software method to remove the cause or disable the effect of the cause.
    - Step 5. Run an evaluation method to measure the result of applying the remedy method.
  - Define subclasses that implement the variable steps in the common algorithm.

- (T8) Apply Strategy pattern for Evaluation Methods (Relevant to C8)

  This tactic is to apply *Strategy* pattern to define the common methods in the abstract strategy class and let concrete strategies, i.e., subclasses, implement the method for each type of evaluation method.

  Note that there are two ways of evaluating the result of remedying anomaly as shown in the following figure.



  Hence, we need to define two hierarchies of evaluation methods because the evaluation methods used for remedy-by-software and the evaluation methods used for remedy-by-manager would not be identical.

- (T9) Apply Specialization and Required Interfaces for Report Types (Relevant to C9)

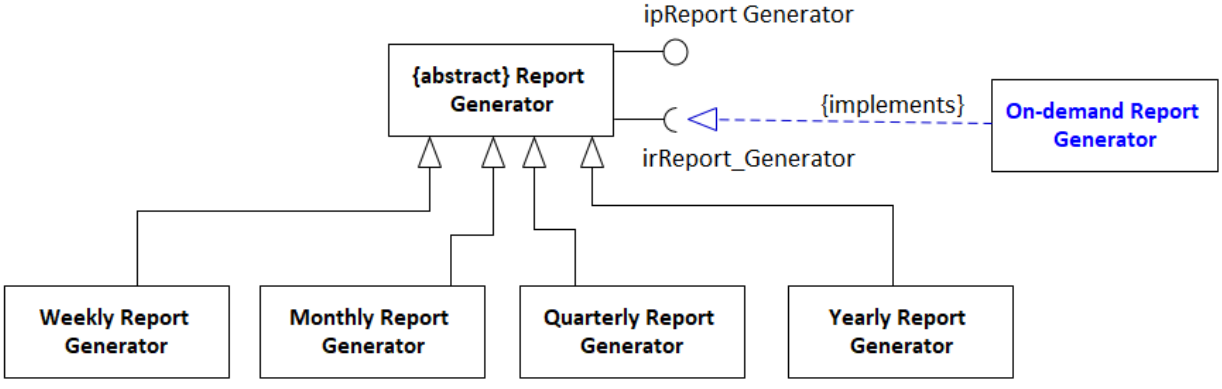
  This tactic is to apply specialization for periodical reports such as weekly report, monthly report, quarterly report, and yearly report. And, it is to define required interfaces for accepting report generators for on-demand reports.

  - Specialization of Periodical Reports

    A hierarchy of periodical report classes can be defined. Under each periodical report class, we can also define its subclasses to reflect the variants on the periodical report.

  - Required Interfaces for On-demand Reports

    We define a set of methods that are commonly applied to generating on-demand reports as shown in the following figure.



Note that the external component, *On-demand Report Generator* can also be specialized for various on-demand report generators.

- (T10) Design with High Modularity (Relevant to C10)

  This tactic is to apply the principle of modular design for the platform. That is, we need to design components to provide high cohesiveness and low coupling with other components.

  - Already Applied in View-based Design

    This tactic of applying modularity design has also been used during the architecture design for functional, information, and behavior views.

6.1.4. [Step 4] Evaluate Candidate Tactics

We evaluate the proposed candidate tactics in terms of their benefit and cost.

We evaluate the proposed candidate tactics in terms of their benefit and cost.

ID	Tactics	Y/N	Justification
T1	Commonality Analysis for the Platform Scope	Y	Benefit) Commonality Analysis can compute the common features of Anomaly Management systems. Cost) The cost for analyzing and designing the commonality is minimal. Decision) The tactic is essential to ensure the high applicability of A.M. platform while the cost is low.
T2	Apply Specialization for Manufacturing Domains	Y	Benefit) Specialization of manufacturing domain allows modeling various manufacturing domains. Cost) The development cost for the specialization occurs only once and it is minimal. Decision) The tactic is essential to support various manufacturing domains while the cost is low.
T3	Apply Specialization, Association and Realization for Manufacturing Plants	Y	Benefit) This tactic allows the modeling of various manufacturing plant which are known or unknown. Cost) The technical hardship for implementing this tactic is somewhat high. Decision) The tactic is already designed by the architect, and the implementing can be done with appropriate guidance.
T4	Apply a HAL layer for Hardware Devices	Y	Benefit) This tactic ensures the decoupling of functional components from their hardware devices. Cost) It is trivial to design and implement this tactic. Decision) The tactic is essential to effectively adopt various types of sensors, cameras, IoT devices, and actuator of A.M. Platform
T5	Apply Specialization and Required Interfaces for Anomaly Types	Y	Benefit) This tactic allows the modeling of various anomaly types which are known or unknown. Cost) The technical hardship for implementing this tactic is somewhat high. Decision) The tactic is already designed by the architect, and the implementing can be done with

			appropriate guidance.
T6	Apply Template Method pattern for Detection Methods	Y	Benefit) This tactic allows the closed-design for the common algorithm for detecting anomaly, and the open-design for specific methods for detecting anomaly. Cost) The cost for applying this tactic is minimal as the pattern is well-known. Decision) The tactic is essential to support various types of detection methods.
T7	Apply Template Method pattern for Remedy Methods	Y	Benefit) This tactic allows the closed-design for the common algorithm for remedying anomalies, and the open-design for specific methods for remedying anomalies. Cost) The cost for applying this tactic is minimal as the pattern is well-known. Decision) The tactic is essential to support various types of remedy methods.
T8	Apply Strategy pattern for Evaluation Methods	Y	Benefit) This tactic defines the common methods in the superclass and lets the common interface be implemented for various evaluation methods. Cost) The cost for applying this tactic is minimal as the pattern is well-known. Decision) The tactic is essential to support various types of evaluation methods.
T9	Apply Specialization and Required Interfaces for Report Types	Y	Benefit) This tactic allows the modeling of periodical reports and dynamic modeling of on-demand reports. Cost) The technical hardship for implementing this tactic is somewhat high. Decision) The tactic is already designed by the architect, and the implementing can be done with appropriate guidance.
T10	Design with High Modularity	Y	Benefit) The platform design with modularity increases the adoption and expansion of the A.M. platform. Cost) The design cost for modularity is moderate. Decision) The benefit of modular design largely exceeds the cost.

6.1.5. [Step 5] Analyze Impacts of Tactics

We analyze the impacts of each selected tactic.

Note that 'Deployment View' is not used, instead 'Development View' is added.

ID	Tactics	Functional View	Information View	Behavior View	Development View
T1	Commonality Analysis for the Platform Scope	Check if the common features are all reflected in the use case diagram.			Apply C&V Analysis during the system modeling.
T2	Apply Specialization for Manufacturing Domains		Modify the object model to allow the specialization of manufacturing domains.	Accept an instance of concrete manufacturing domain classes through dynamic binding.	Define the set of common manufacturing domains.
T3	Apply Specialization, Association and Realization for Manufacturing Plants		Modify the object model to support the specialization, association, and realization for Manufacturing Plants.	Accept an instance of concrete Manufacturing Plants or a plug-in object that implements the required interface.	
T4	Apply a HAL layer for Hardware Devices	Add a new interface 'IoT Device' in the HAL layer.	Add a new class 'IoT Device' as a subclass of Hardware Device.		
T5	Apply Specialization and Required Interfaces for Anomaly Types		Modify the object model to support the specialization, association, and realization for Anomaly Types.	Accept an instance of concrete Anomaly Types or a plug-in object that implements the required interface.	
T6	Apply Template Method pattern for Detection Methods			Invoke the template method for detecting anomalies and	Define the common algorithm for detecting various anomaly types.

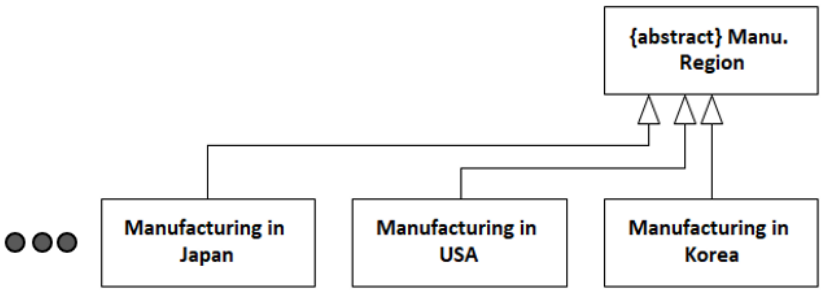
				bind instances of subclasses through dynamic binding.	
T7	Apply Template Method pattern for Remedy Methods			Invoke the template method for remedying anomalies and bind instances of subclasses through dynamic binding.	Define the common algorithm for remedying various anomaly types.
T8	Apply Strategy pattern for Evaluation Methods	Define the common methods for evaluating remedy results.		Accept an instance of concrete Evaluation Method class through dynamic binding.	
T9	Apply Specialization and Required Interfaces for Report Types		Modify the object model to support the specialization, association, and realization for Reports.	Accept an instance of concrete Periodical Report classes or a plug-in object that implements the required interface.	
T10	Design with High Modularity				Design components to ensure high cohesion and low coupling with other components.

6.1.6. [Step 6] Apply Tactics

We design the detailed control flow for selected tactics. In CEP, we choose only one tactic.

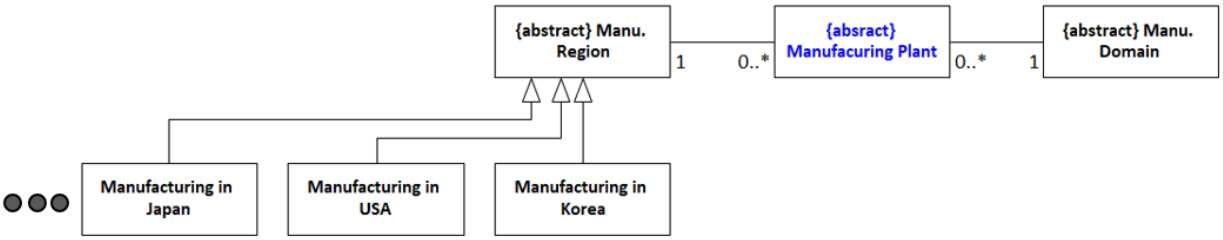
- Applying T3) Apply Specialization, Association and Realization for Manufacturing Plants
  - We first specialize 'Manufacturing Region' by considering the regions of manufacturing plants.

    The abstract class, Manufacturing Region' can be specialized into different regions, as shown in the following figure.



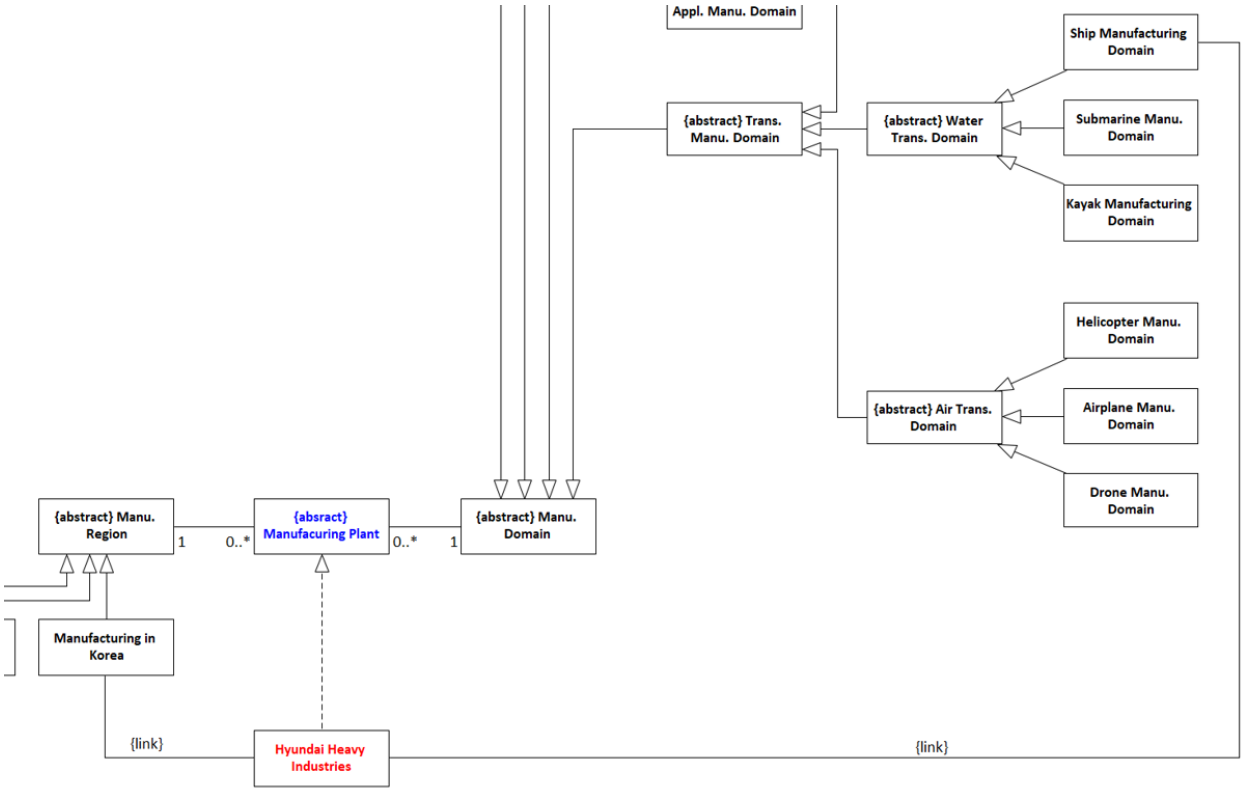
  - Associations of 'Manufacturing Plant'

    An abstract class, Manufacturing Plant, is defined with two association relationships as shown in the following figure.



❍ Realization of 'Manufacturing Plant'

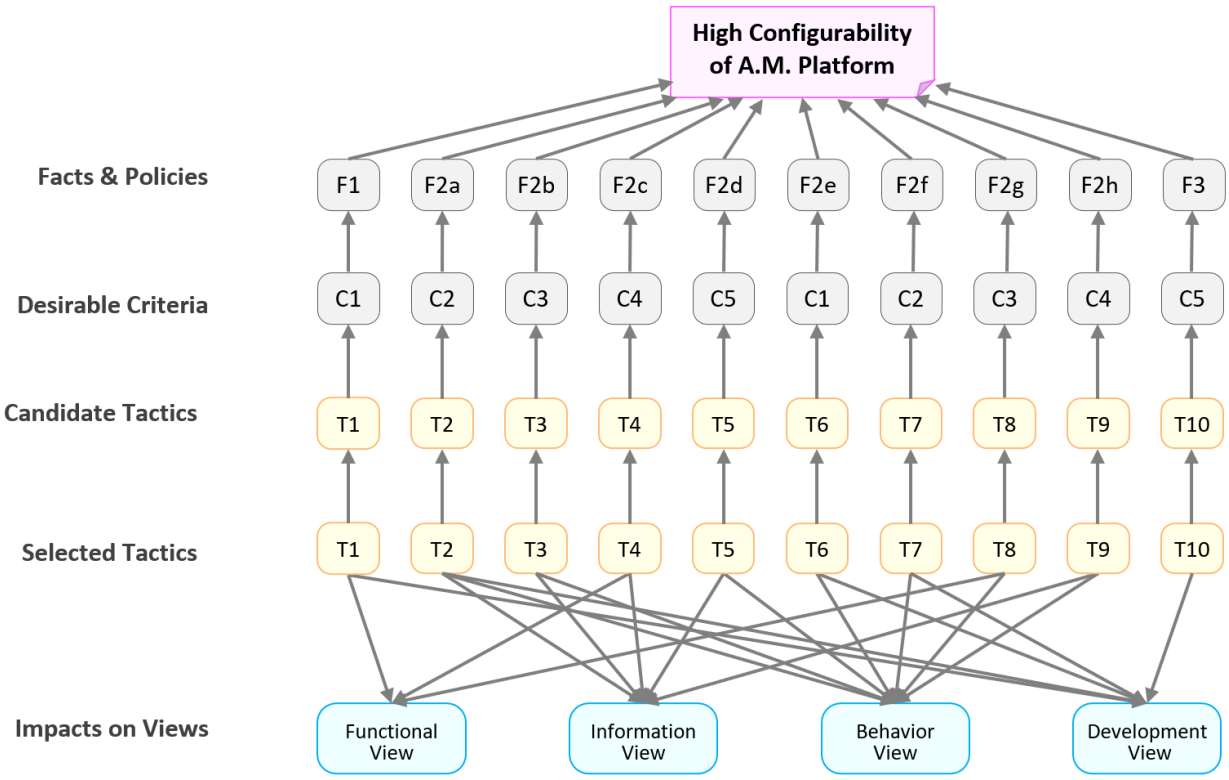
Each manufacturing plant is defined by realizing 'Manufacturing Plant' class. Then, each manufacturing plant is linked to an instance of Manufacturing Region and an instance of Manufacturing Domain, as shown in the following figure.



In the figure, 'Hyundai Heavy Industries' plant is linked to an instance of 'Manufacturing in Korea' and an instance of 'Ship Manufacturing Domain'.

6.1.7. [Step 7] Validate Conformance

It is important to enforce the traceability among the facts/policies, criteria, tactics and impacts on views. The following figure shows the trace links among facts, criteria, tactics and their impacts on views.



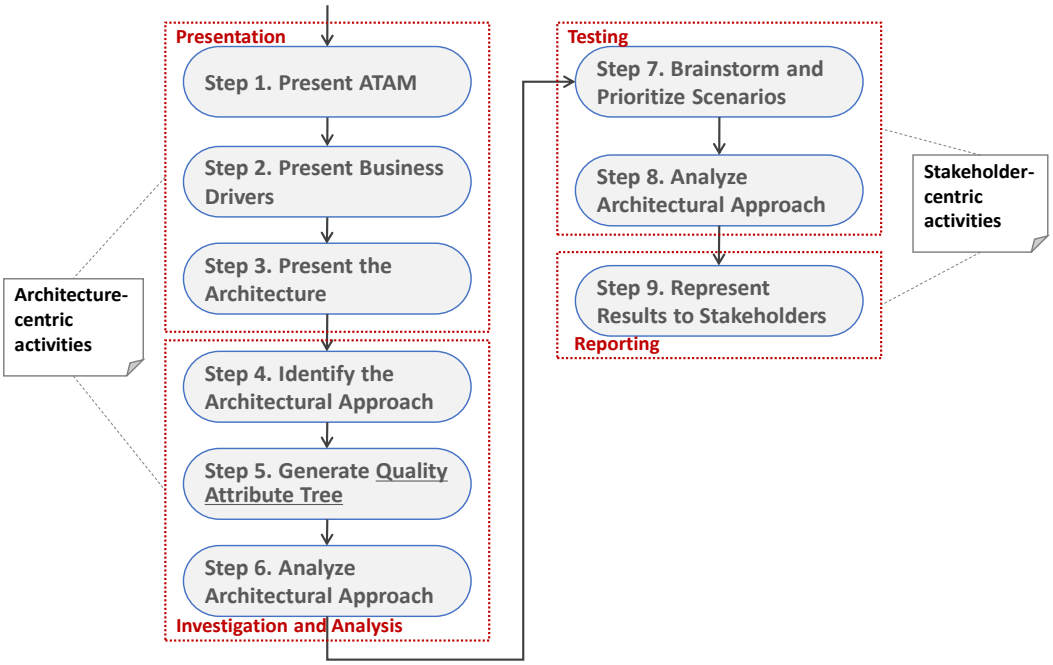
As shown in the figure, all the elements defined with ‘conforms-to’ relationships, which yields a high traceability and consistency.

6.2. NFR-2. (Not Applicable)

Not Included in this CEP.

7. Activity 6. Architecture Validation

❑ Steps of ATAM



7.1. [Step 1] Presenting ATAM

This step is to present the ATAM process to the assembled stakeholders, typically customer representatives, the architect or architecture team, user representatives, maintainers, administrators, managers, testers, integrators, etc.

❑ Already Presented

7.2. [Step 2] Presenting Business Drivers

This step is for the project manager to present what business goals are motivating the development effort and hence what will be the primary architectural drivers.

❑ Already Specified in the CEP SRS.

❑ Already Specified in this AD

The architecture drivers are further specified in chapter 2 of 'Activity 1. Architectural Requirement Refinement'

7.3. [Step 3] Presenting the Architecture

This step is for the architect to present the designed architecture, focusing on how the architecture addresses the business drivers.

- ❑ Already Presented in this AD.
  - ❍ Chapter 4 of Skeleton Architecture
  - ❍ Chapter 5 for View-specific Architecture Design
  - ❍ Chapter 6 for NFR-specific Architecture Design

7.4. [Step 4] Identifying the Architectural Approaches

This step is for the architect to specific architectural approaches in the AD.

- ❑ Already Presented in this AD.
  - ❍ Specific Design Decisions in Chapter 5 for View-specific Architecture Design
  - ❍ Specific Design Decisions in Chapter 6 for NFR-specific Architecture Design

7.5. [Step 5] Generating Quality Attribute Tree

This step is for the architect to define the core business and technical requirements of the system and map them to an appropriate architectural property. This is done by elicitating the quality factors that comprise system "utility" (performance, availability, security, modifiability, etc.), specify down to the level of scenarios, annotated with stimuli and responses, and prioritized.

A scenario is a specific situation or use case that demonstrates how the architecture will perform under different conditions. Each scenario should be specific and detailed, describing the actions of the user or system and the expected outcome.

- ❍ Use Case Scenarios
- ❍ Growth Scenarios
- ❍ Exploratory Scenarios

Quality	Refinements	Scenarios	Importance	Difficulty
Functionality	NFR-1. High Configurability of the Anomaly Management Platform	NFR-1-1. Evaluate the percentage of how much functionality of the existing systems (which are considered in the architecture design) is covered by this platform without any modification. The rate should be higher than 80%.	High	Low
		NFR-1-2. Evaluate the percentage of how much functionality of the existing systems (which are considered in the architecture design) is covered by this platform	High	Medium

		with slight modification. The rate should be higher than 85%.		
		NFR-1-3. Measure the percentage of how much functionality of the existing systems (which are considered in the architecture design) is NOT covered by the platform.	High	Medium
		NFR-1-5. Evaluate what portion of the functionality is additionally implemented to apply this system to a new system (which is not considered in the architecture design).	High	Medium
		NFR-1-6. Evaluate how long dev work is needed to apply this platform to a new manufacturing domain.	High	Medium
		NFR-1-7. Evaluate how long dev work is needed to apply this platform to a new manufacturing plant.	High	Medium
		NFR-1-8. Evaluate how long dev work is needed to apply this platform to a system handling a new target product.	Medium	Medium
		NFR-1-9. Evaluate how long dev work is needed to apply this platform to a system handling a new hardware device.	Medium	Low
		NFR-1-9. Evaluate how long dev work is needed to apply this platform to a system processing a new anomaly type.	High	Medium
		NFR-1-10. Evaluate how long dev work is needed to apply this platform to a system with a new method to detect anomaly occurrence.	Medium	Low
		NFR-1-11. Evaluate how long dev work is needed to apply this platform to a system offering a new remedy method.	High	Medium
		NFR-1-12. Evaluate how long dev work is needed to apply this platform to a system measuring a new metric to evaluate the anomaly management performance.	Medium	Medium
		NFR-1-13. Evaluate how long dev work is needed to apply this platform to a system generating a	Medium	Low

		new type of reports.		

Based on this analysis, priotize the factors.

7.6. [Step 6] Analyzing Architectural Approaches

This step is to analyze the architectural approaches that address those factors are elicited and analyzed, based upon the high-priority factors identified in Step 5. Identify architectural risks, sensitivity points, and tradeoff points.

The following table describes an analysis result of architectural approaches addressing a scenario, NFR-1-2.

Analysis of Architectural Approach				
Scenario #	NFR-1-2. Evaluate the percentage of how much functionality of the existing systems (which are considered in the architecture design) is covered by this platform with slight modification.			
Attribute	Functionality			
Environment	Normal Operation.			
Stimulus	A system is run under normal condition.			
Architectural Decision	**Sensitivity**	**Trade-Off**	**Risk**	**Nonrisk**
D1. Commonality Analysis for the Platform Scope	S1, S2			NR1
D2. Apply Specialization for Manufacturing Domains	S1, S4	T2		
D3. Apply Specialization, Association and Realization for Manufacturing Plants	S2, S4	T2		
D4. Apply a HAL layer for Hardware Devices	S3, S4			
D5. Apply Specialization and Required Interfaces for Anomaly Types	S4, S5		R1	
D6. Apply Template Method pattern for Detection Methods	S6, S7	T1, T2		
D7. Apply Template Method pattern for Remedy Methods	S6, S7	T1, T2		
D8. Apply Strategy pattern for Evaluation Methods	S6, S7	T1, T2		

D9. Apply Specialization and Required Interfaces for Report Types	S4, S8	T2		
D10. Design with High Modularity				NR1
Reasoning	The decisions are made for meeting this scenario since the chosen decisions are commonly used for improving accuracy of the results of running machine learning algorithms.			
Architectural Diagram	Refer to refined component diagram in 6.1.			

- Sensitivity Points
  - S1. Representativeness of the manufacturing domains considered in the analysis
  - S2. Representativeness of the manufacturing plants considered in the analysis
  - S3. The diversity / range of hardware devices
  - S4. Right Level of abstraction
  - S5. Soundness of Anomaly Types Classification
  - S6. The range of predicted variants for each considered variation point
  - S7. The degree of openness & easiness of the customization
  - S8. The diversity / range of Report types
- Trade-off
  - T1. Extensibility (+) vs. Performance (-): Applying design patterns ends up with having multiple layers of the classes, which results in large occurrence of method invocations among the classes. There would be some degree of performance degradation.
  - T2. Extensibility (+) vs. Maintainability (-): Design with close/open design principle ends up defining a larger number of smaller classes in order to separate closed parts from variable parts. Without sufficient documentation, this makes more difficult to maintain this system.
- Risk
  - R1. The risk is caused since the high degree of variability for anomaly types would be expected. In addition, there is also higher variability on dealing with different anomaly types, which can limit the applicability of this platform.
- Nonrisk
  - NR1. There are well-known, well-established design tactics for designing components with high modularity. Hence, defining with high modularity would be less challenging.

7.7. [Step 7] Brainstorming and Prioritizing Scenarios

This step is to elicit a set of scenarios from the entire group of stakeholders and prioritize the scenarios via a voting process.

- List of scenarios collected by all stakeholders (i.e. system managers, production managers, workers, and clients)
  - About 10 scenarios are additionally acquired for evaluating high configurability of the *Anomaly Management Platform*. Most scenarios are gathered from system managers and production managers.
    - Need to elaborate scenarios for NFR-1-8 to NFR-1-13. The current scenarios are identified at a coarse-grained level. Finer-grained details are needed.
    - Measure how effectively or technically complicated the platform is incorporated with brand-new manufacturing domains/plants .
    - And others.

7.8. [Step 8] Analyzing Architectural Approaches

This step reiterates step 6, but here the highly ranked scenarios from Step 7 are considered to be test cases for the analysis of the architectural approaches determined thus far. These test case scenarios may uncover additional architectural approaches, risks, sensitivity points, and tradeoff points which are then documented.

Since the result of this step is same as the one of step 6, we do not include the table.

7.9. [Step 9] Presenting the Results

This step I sto for the ATAM team to present the findings to the assembled stakeholders and potentially write a report detailing this information along with any proposed mitigation strategies.

- The evaluation team concludes the following results;
  - Concerns on the functionality, specifically on high configurability of the *Anomaly Management Platform*, need to be re-considered.
    - There might be a wide variety on anomaly types, anomaly detection methods, and report types. Currently, the evaluation is more focused on whether the current design is addressing variability of manufacturing domains and manufacturing plants. However, more diverse cases need to be considered in the design.
    - Current scenarios for evaluating platform configurability regarding anomaly types, anomaly detection methods, and report types are too coarse-grained so that it is not enough to evaluate the current architecture design with the scenarios.

8. Concluding Remarks

The architecture description in this document is to meet both the functional and non-functional requirements for the system. It is the result of applying the proposed core process of designing software architecture.

It is believed that this architecture description is practically implementable with current technologies and such implementation would yield a high level of quality-in-use.

➢ END OF ARCHITECTURE DESCRIPTION ≺