Learning Report – Embedded software – Design Development Processes and Standards

GLOBAL ENGINEERING ACADEMY

Genesis

L&T Technology Services

SHANDHIYA.V. S
99002477

Document History

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	Approved By	Remarks/Revision Details
1	10/10/2020	SHANDHIYA.V. S 99002477		Bhargav N	FIRST DRAFT

Table of Contents

Table of figures

ACTIVITY-1

ACTIVITY 1- TYPES OF TESTING:

BLACK BOX TESTING:

- Black box testing is a method of testing where the input is given to the system and the output is checked, no matter how it was generated. It is done at an outer level.
- It can be done without any knowledge of the system as of how it works. However, it is a less good way of testing when compared to white box testing as no internal knowledge is concerned it this. Any glitch cannot be corrected as there is no architectural or code knowledge involved.
- The system is tested by the team just for the suitable output for the given input. Hence the application is tested here.

Figure 1- BLACK BOX TESTING

APPLICATIONS AND BENEFITS:

- The tester doesn't need any technical knowledge to test the system. It is essential to understand the user's perspective.
- Testing is performed after development, and both the activities are independent of each other.
- It works for a more extensive coverage which is usually missed out by testers as they fail to see the bigger picture of the software.
- Test cases can be generated before development and right after specification.
- Black box testing methodology is close to agile.

WHITE BOX TESTING:

- White Box (or glass box) testing is the process of giving input code to the system and checking how the system processes it how the output is generated using that code. This is the test where the team tests the internal logic of the code written.
- It is the process of checking how the system processes input to give a suitable output by giving it the required input.
- It involves testing a system with full access to source code and other architecture documents. This testing reveals bugs and vulnerabilities as quickly as possible. In comparison to an authentic trial and error method, this white box testing is way quicker.
- By exactly knowing what to test, more testing coverage is ensured. This type of testing involves testing of the application. It needs knowledge of code and test cases chosen if the system is implemented as expected it is verified.
- It is basically checking whether the given code gives out expected results.

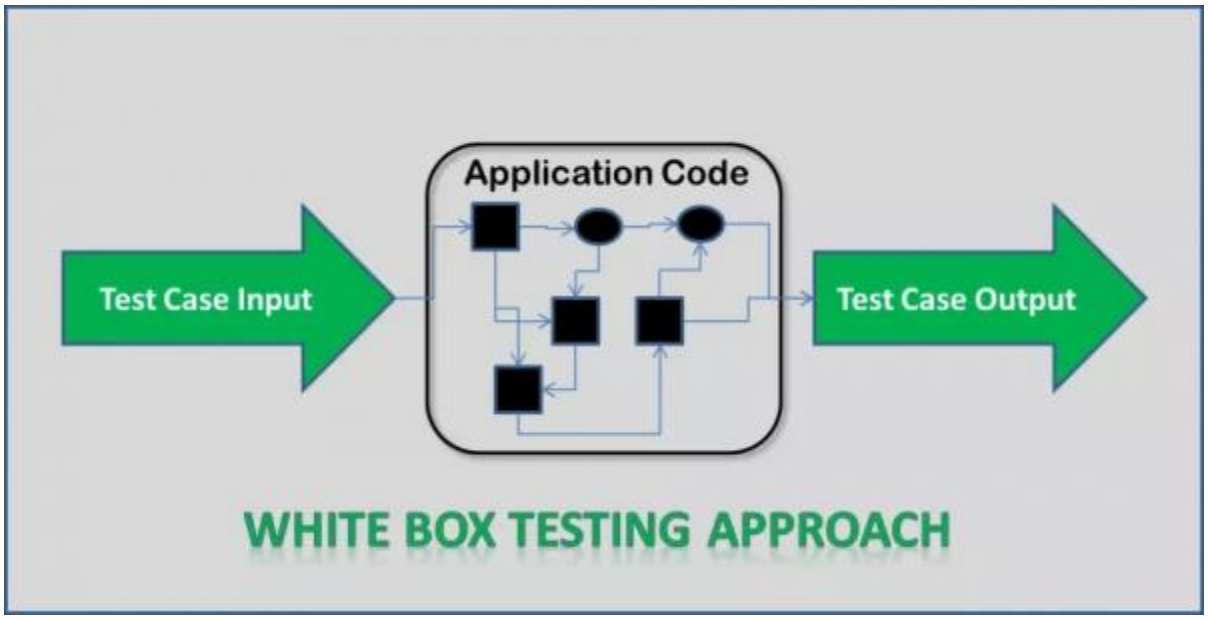

Figure 2- WHITE BOX TESTING

APPLICATIONS AND BENEFITS:

- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

GRAY BOX TESTING:

- This is the combination of both white box and black box testing.
- In this type of testing the tester has limited knowledge of the system and the internal applications. Just the output is checked as per the given inputs.
- Gray Box testing is a combination of White Box and Glass Box Testing. In this Gray box testing, the tester need not have the knowledge about the internal working of the software.
- Only the output, the process of how the output has arrived is tested in this method.

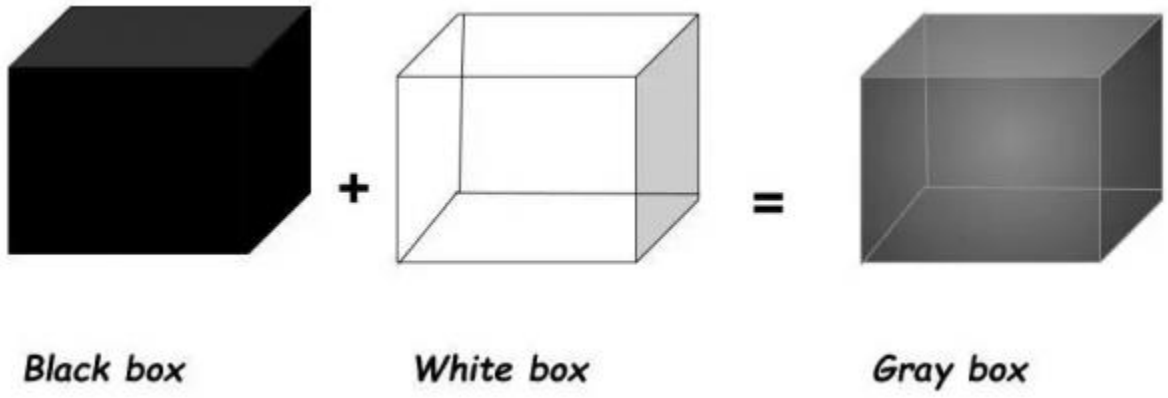

Figure 3- GREY BOX TESTING

APPLICATIONS AND BENEFITS:

- Users and developers have clear goals while doing testing.
- Gray box testing is mostly done by the user perspective.
- Testers are not required to have high programming skills for this testing.
- Gray box testing is non-intrusive.
- Overall quality of the product is improved.
- In gray box testing, developers have more time for defect fixing.
- By doing gray box testing, benefits of both black box and white box testing is obtained.
- Gray box testing is unbiased. It avoids conflicts between a tester and a developer.
- Gray box testing is much more effective in integration testing.

ACTIVITY-2

ACTIVITY 2- INDUSTRIAL STANDARDS

AERO INDUSTRY STANDARDS:

The National Aerospace Standards (NAS) are voluntary standards developed by the aerospace industry. Subject matter experts from AIA member companies participate in committees and working groups to develop and maintain the NAS library, which currently contains over 1400 active standards. These standards cover a wide variety of subject areas including:

- NAS parts (bolts, rivets, washers, screws, nut plates, pins, knobs, etc.)
- Safety Management Systems (NAS9927)
- Nondestructive Test Personnel certification (NAS410)
- Hazardous materials management (NAS411)
- Foreign Object Debris (FOD) prevention (NAS412)
- Cutting tools (drills, reamers, end mills)
- Airport Operations (NAS3306)
- Trade Compliance Standards (TCS)

The NAS parts are most well-known for state-of-the-art, high strength, precision fasteners, electrical connectors, splices and terminations, rod end bearings, and many other types of hardware and components. Most of the NAS parts are available as 3D CAD models (see below for more information). In addition to the NAS standards, AIA supports standardization work at the International Organization for Standardization (ISO). AIA holds the secretariat for ISO/TC 20, Aircraft and Space Vehicles, and its subcommittee SC 16, Unmanned Aircraft Systems.

AUTOMOTIVE INDUSTRY STANDARDS:

he most common standards related to the automotive industry include:

- IATF 16949**:** We work with the automotive industry to support the manufacturing of safe and reliable products, which are produced and continually improved to meet or exceed customer and regulatory authority requirements. Most organizations manufacturing for the automotive industry are required to be certificated to IATF 16949, which was developed by the International Automotive Task Force (IATF).
- ISO 9001**:** Since IATF 16949 isn't designed to be a self-sufficient quality standard but instead works best in conjunction with a comprehensive QMS, ISO 9001 certification makes a great deal of sense for automotive companies looking to demonstrate improvement in customer satisfaction, operating costs, stakeholder relationships, legal compliance, risk management, business credentials and attracting new business.

- ISO 14001**:** As the international standard for environmental management systems — or EMS — ISO 14001 is the primary EMS certification for more than 250,000 organizations around the world. As the global standard for any business that wants to manage and positively control all aspects of its environmental impact, ISO 14001 certification is a proven way to demonstrate that you're serious about your business's environmental and economic sustainability.
- ISO 45001**:** Along with offering reliable products and services, automotive manufacturers must constantly strive to provide their workers and visitors with a safe and healthy business environment. With the ultimate goal of providing businesses with a framework for controlling and eliminating factors that can lead to illness, injuries and — in worst-case scenarios — death, obtaining ISO 45001 Health and Safety certification is a prudent move for any organization's senior management to support.

RAIL INDUSTRY STANDARDS:

- The International Organization for Standardization (ISO)
- The European Rail Industry – UNIFE
- GS1
- International Rail Industry Standard (IRIS)
- American National Standards Institute (ANSI)
- The Rail Committee on Information Standards (RailCIS)
- Association of American Railroads (AAR)
- Federal Railroad Administration (FRA)
- Railway Age
- International Railway Journal

ACTIVITY-3

ACTIVITY 3- SYSML

OVERVIEW OF SYSML

This SysML Diagram Tutorial is a Systems Modeling Language (SysML) primer that provides an overview of the nine (9) SysML diagram types and complementary Allocation Tables that constitute this de facto architecture modeling language standard for Model-Based Systems Engineering (MBSE) applications.

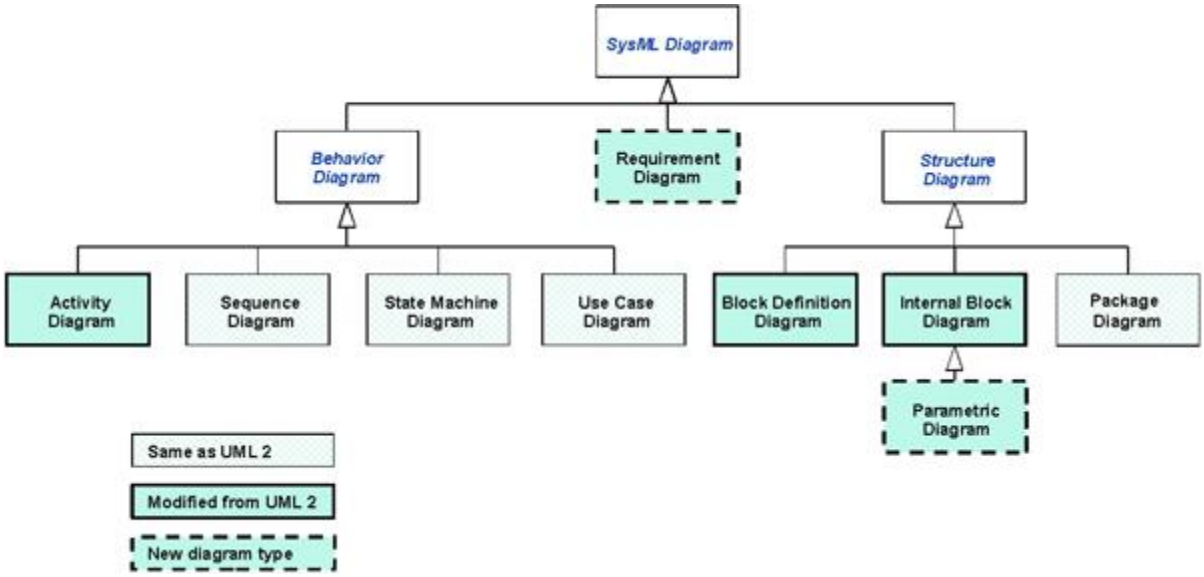

Figure 4- OVERVIEW OD SYSML

The SysML is composed of nine (9) diagram types and Allocation Tables for mapping language elements across diagram types:

1. Requirement diagram (req)
2. Structure Diagrams
    - Block Definition Diagram (bdd)
    - Internal Block Diagram (ibd)
    - Parametric Diagram (par)
    - Package diagram (pkg)
3. Behavior Diagrams

- Activity diagram (act)
- Sequence diagram (seq)
- State Machine diagram (stm)
- Use Case diagram (uc)

4. Allocation Table

The SysML Diagram Taxonomy comparison table below explains the similaries and differences among the various SysML diagram types.

The block is the basic unit of structure in SysML and can be used to represent hardware, software, facilities, personnel, or any other system element. The system structure is represented by block definition diagrams and internal block diagrams. A block definition diagram describes the system hierarchy and system/component classifications. The internal block diagram describes the internal structure of a system in terms of its parts, ports, and connectors. The package diagram is used to organize the model.

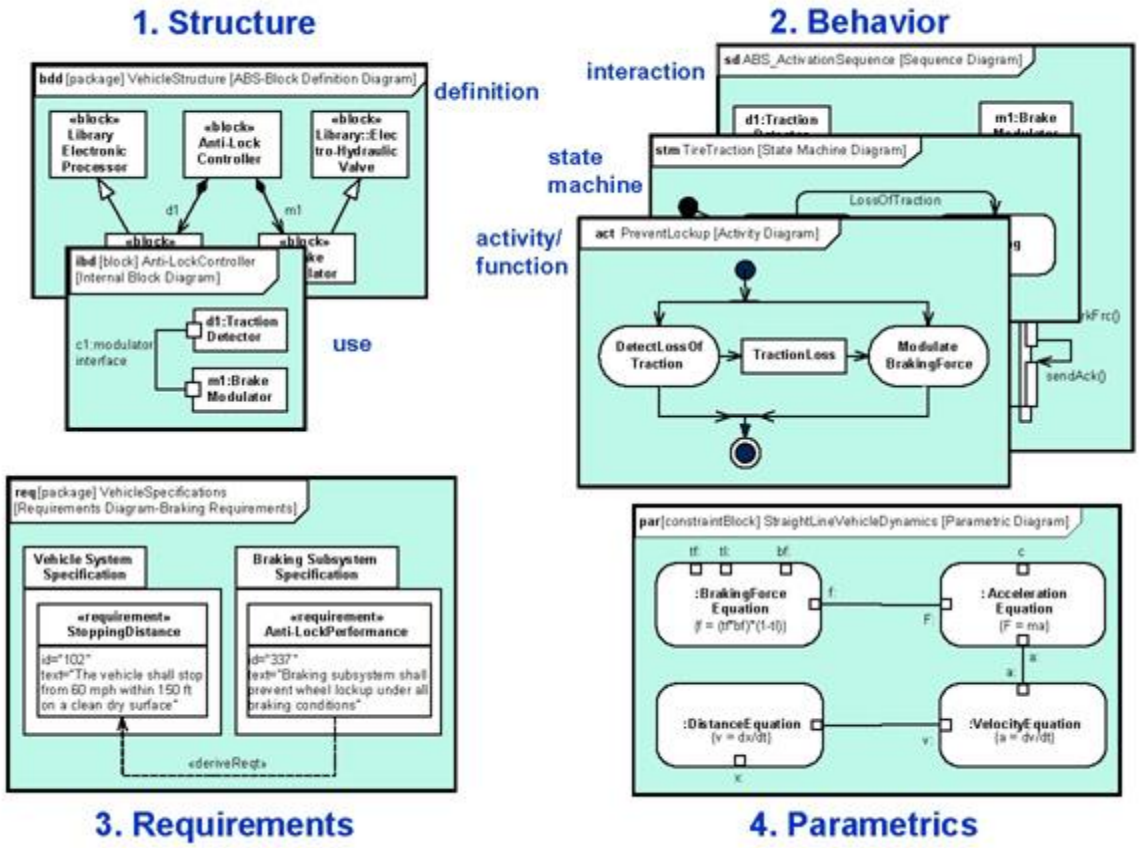

Figure 5- TYPES OF SYSML

REQUIREMENT DIAGRAM:

Requirement Diagram is particularly valuable when you want to demonstrate the traceability from the requirements to the elements in your system model that are dependent on them. This diagram provides modeling constructs to represent text-based requirements and relate them to other modeling elements. These requirement modeling constructs are intended to provide a bridge between traditional requirement management tools and other SysML models.

Requirements diagrams display requirements, packages, other classifiers, test cases, rationales, and relationships. Possible relationships available for Requirements diagrams are containments, deriveReqt and requirement dependencies ('Copy', 'Refine', 'Satisfy', 'Trace', and 'Verify'). The callout notation can also be used to reflect the relationships of other models.

Requirements can also be shown on other diagrams to illustrate their relationships to other modeling elements.

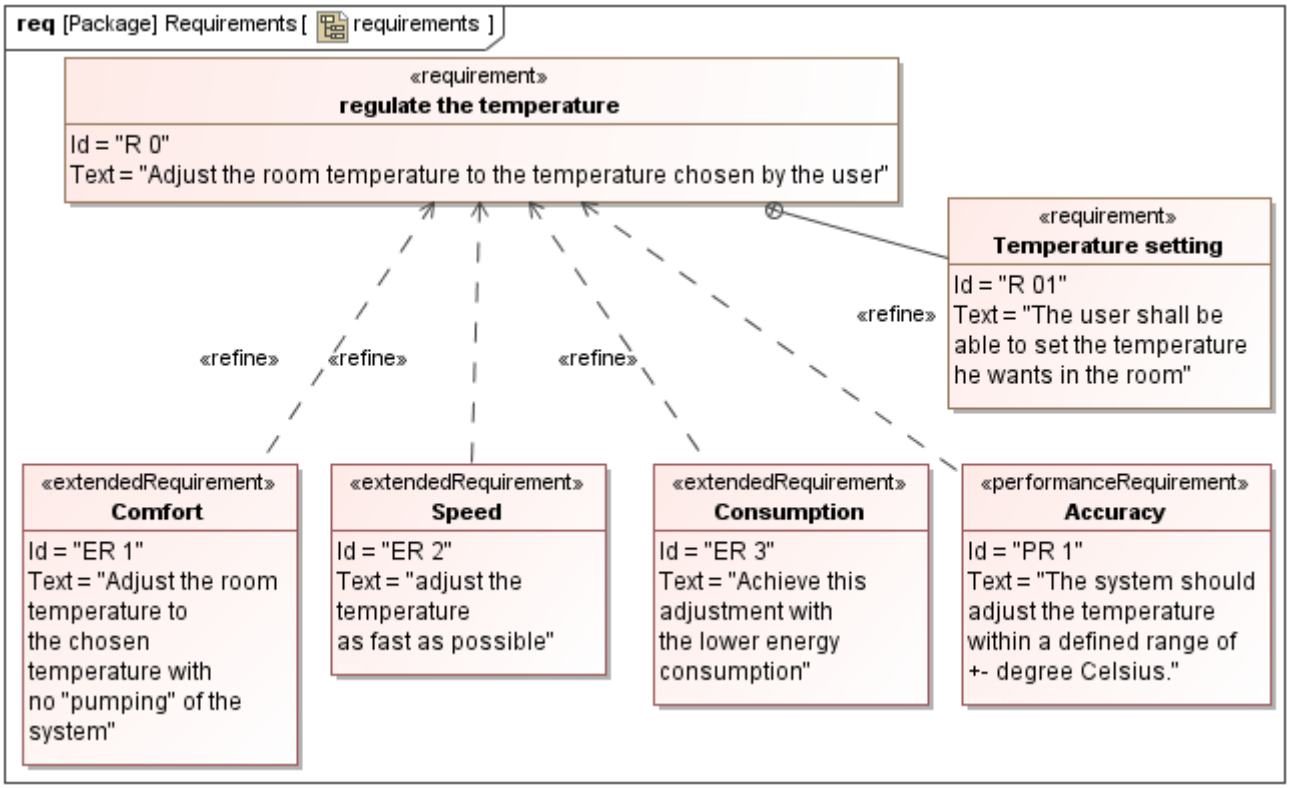

Figure 6- REQUIREMENT DIAGRAM

BLOCK DEFINITION DIAGRAM:

Block Definition Diagram defines the features of a block and any relationships between blocks such as associations, generalizations, and dependencies, in terms of properties, operations, and relationships (for example, a system hierarchy or a system classification tree).

Block Definition Diagrams are based on UML class diagrams and include restrictions and extensions as defined by SysML. They are generally used to display systems of blocks or show a system dictionary and/or extensions.

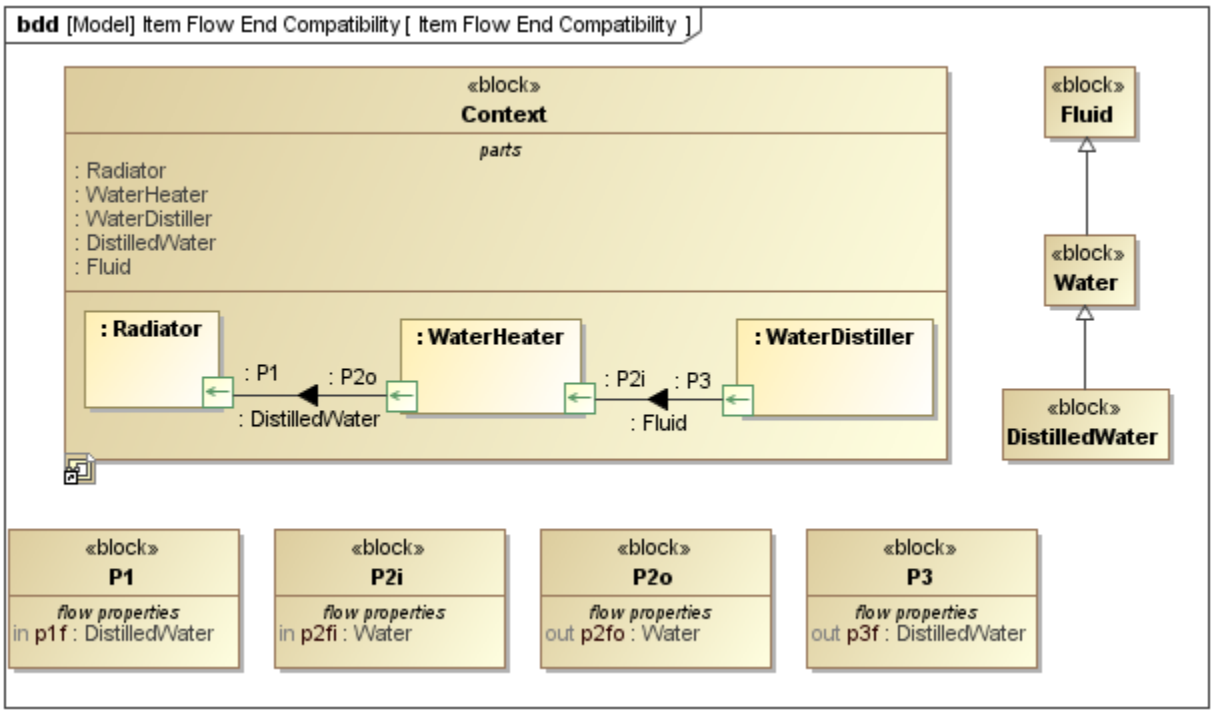

Figure 7- BLOCK DEFINITION DIAGRAM

INTERNAL BLOCK DIAGRAM:

Internal Block Diagram is based on UML composite structure diagram and includes restrictions and extensions as defined by SysML. An Internal Block Diagram captures the internal structure of a Block in terms of properties and connections among properties. A Block includes properties so that its values, parts, and references to other blocks can be specified. However, whereas an Internal Block Diagram created for a Block (as an inner element) will only display the inner elements of a classifier (parts, ports, and connectors), an Internal Block Diagram created for a package will display additional elements (shapes, notes, and comments).

All properties and connectors that appear inside an Internal Block Diagram belong to (are owned by) a Block whose name is written in the diagram heading. That particular Block is the context of the diagram. SysML allows any property (part) to be shown in an Internal Block Diagram to display compartments within the property (or part) symbol.

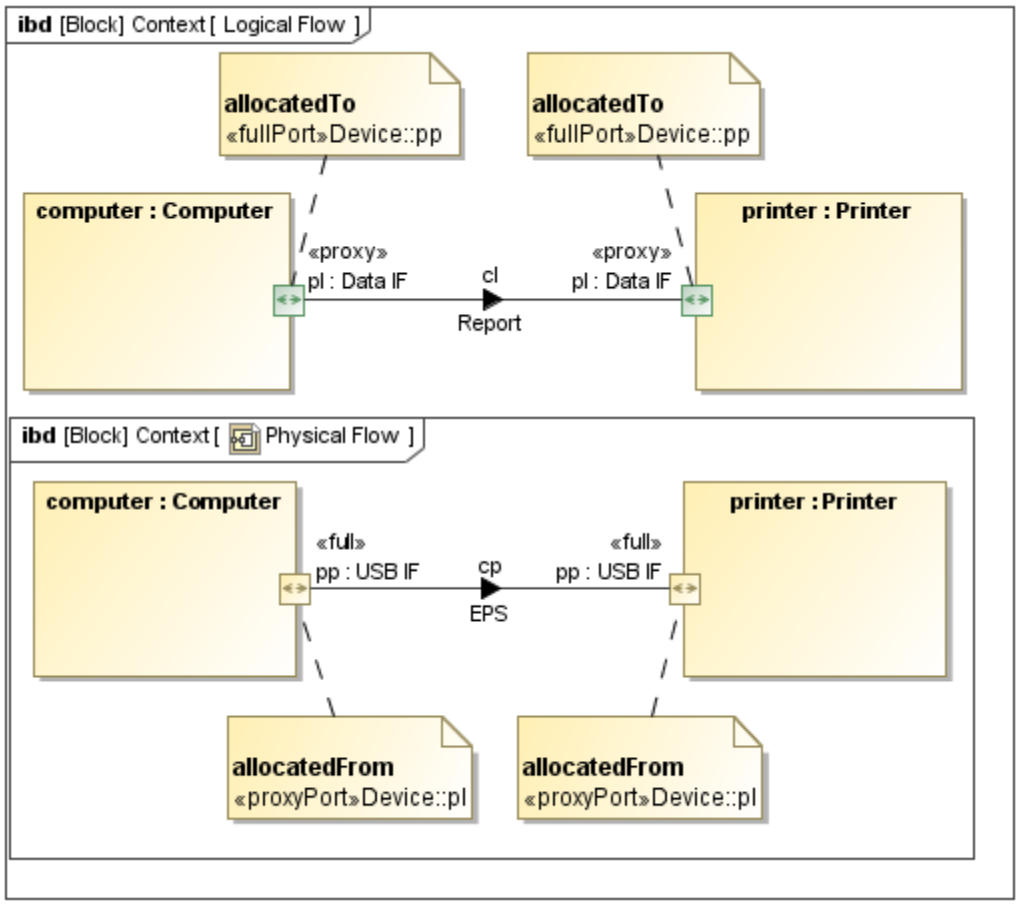

Figure 8- INTERNAL BLOCK DIAGRAM

PACKAGE DIAGRAM:

Package diagram typically enables you to organize models by partitioning model elements into packageable elements and establishing dependencies between packages and/or model elements within these packages. Since Package diagram is used to organize models in packages and views, it can include a wide array of packageable elements.

A package is a construct that enables you to organize model elements, such as use cases or classes, into groups. Packages define namespaces for packageable elements. Model elements from one package can be imported and/or accessed by another package. This organizational principle is intended to help establish unique naming of the model elements and avoid overloading a particular model element's name. Packages can also be shown on Block Definition diagrams or Requirement diagrams.

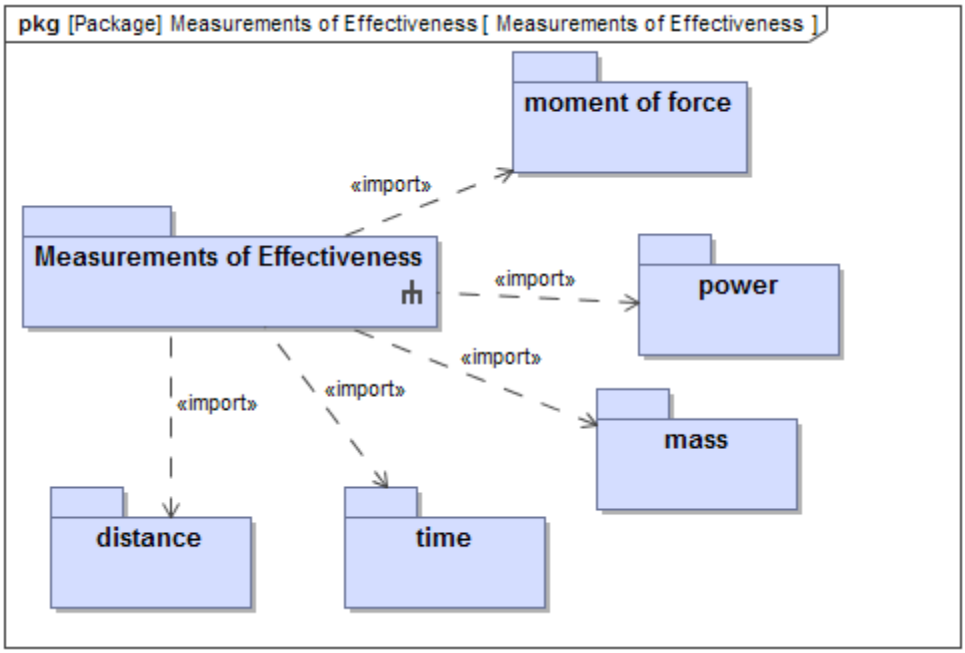

Figure 9- PACKAGE DIAGRAM

PARAMETRIC DIAGRAM:

Parametric diagram is designed to describe mathematical equations by modeling elements, e.g. to count breaking distance of a vehicle.

Parametric diagram can be defined as restricted forms of Internal Block diagrams. They are similar to Internal Block diagrams except that the only connectors allowed are Binding Connectors, each having at least one end connected to a Constraint Parameter.

A Parametric diagram includes the usage of a Constraint Block to constrain the properties of another Block. It contains Constraint Properties and constraint parameters as well as other properties from within that internal block context. All properties displayed, other than the constraints themselves, must either be bound directly to a Constraint Parameter or contain a property that is bound to a Constraint Parameter (through any number of containment levels). A Constraint Block generally contain many constraints, each of them containing many Constraint Parameters.

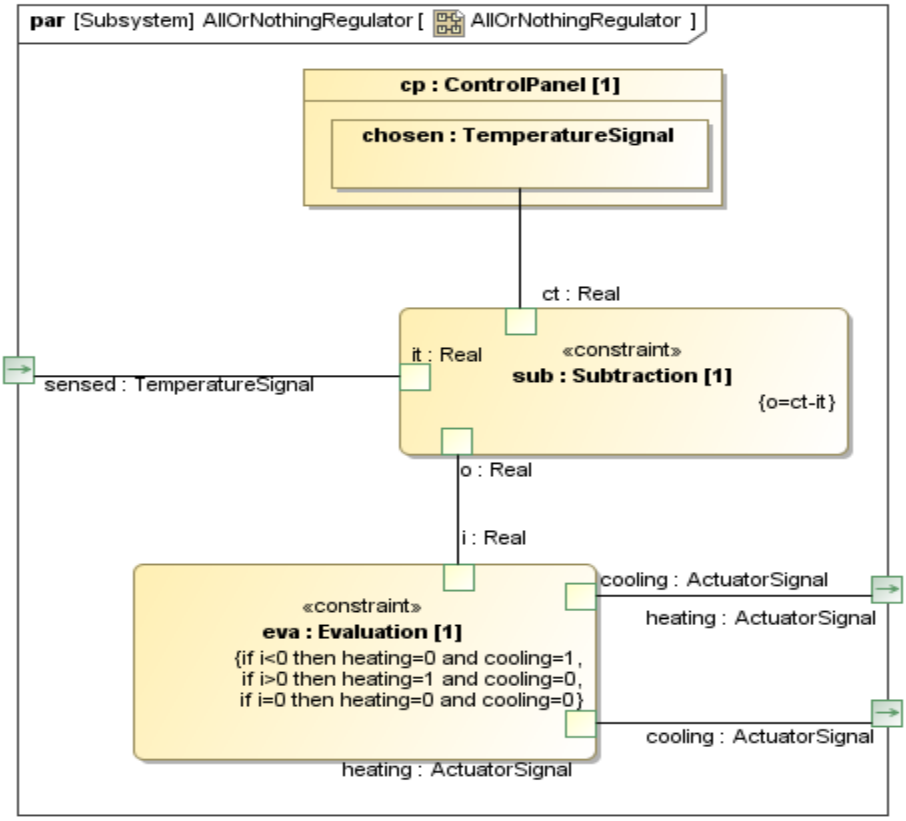

Figure 10- PARAMETRIC DIAGRAM

USE CASE DIAGRAM:

The purpose of a Use Case Diagram is to give a graphical overview of the functionalities provided by a system in terms of actors, their goals (represented as use cases), and any dependencies among those use cases.

A Use Case Diagram describes the usage of a system. The associations between actors and use cases represent the communications that occur between the actors and the subjects to accomplish the functionalities associated with the use cases. The subject of a use case can be represented through a system boundary. The use cases enclosed in the system boundary represent the functionalities performed by behaviors (activity diagrams, sequence diagrams, and state machine diagrams).

Actors may interact either directly or indirectly with the system. They are often specialized so as to represent a taxonomy of user types or external systems. The only relationship allowed between actors in a use case diagram is generalization. This is useful in defining overlapping roles between actors. Actors are connected to use cases through communication paths, each represented by a relationship. There are four use case relationships:

- communication
- include
- extend
- generalization

Communication

A communication path represents an association between two Deployment Targets. It connects actors to use cases.

Include

An include relationship provides a mechanism for factoring out a common functionality that is shared among multiple use cases and is always performed as part of the base use case.

Extend

An extend relationship provides an optional functionality, which extends the base use case at defined extension points under specified conditions.

Generalization

A generalization relationship provides a mechanism to specify variants of the base use case.

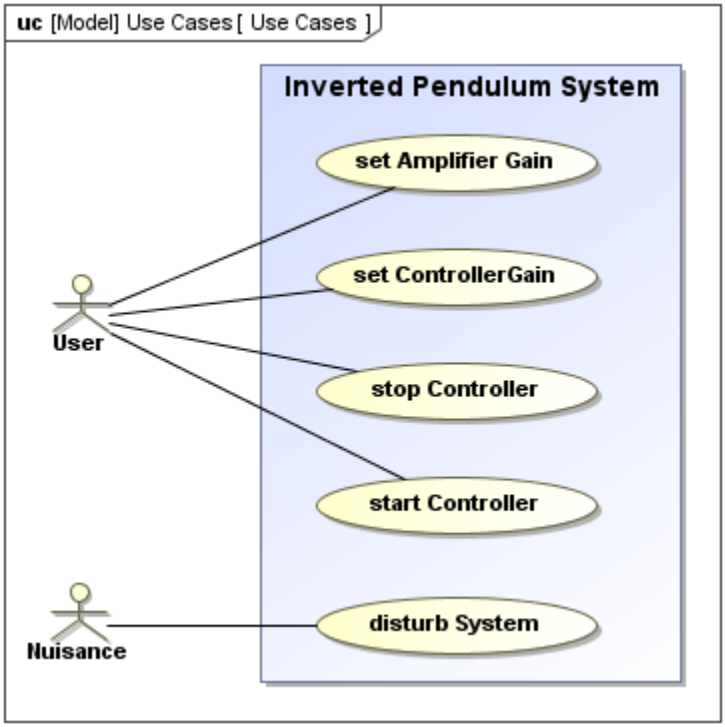

Figure 11- USE CASE DIAGRAM

REFERENCES

- https://www.mpofcinci.com/blog/rail-industry-standards-resources/
- https://www.google.com/search?q=OVERVIEW+OF+DIAGRAMS+IN+SYSML&source=lmns&bih=789&biw=1600&rlz=1C1NHXL_enIN890IN891&hl=en&sa=X&ved=2ahUKEwifqI2YtqfsAhVx5nMBHSgwArwQ_AUoAHoECAEQAA
- https://docs.nomagic.com/display/SYSMLP190/Diagrams+ownership
- https://omgsysml.org/what-is-sysml.htm
- https://www.nqa.com/en-in/certification/sectors/automotive#:~:text=The%20most%20common%20standards%20related,customer%20and%20regulatory%20authority%20requirements.