INTEGRATED SYSTEM ARCHITECTURE DEVELOPMENT AND ANALYSIS FRAMEWORK APPLIED TO A DISTRICT COOLING SYSTEM

by

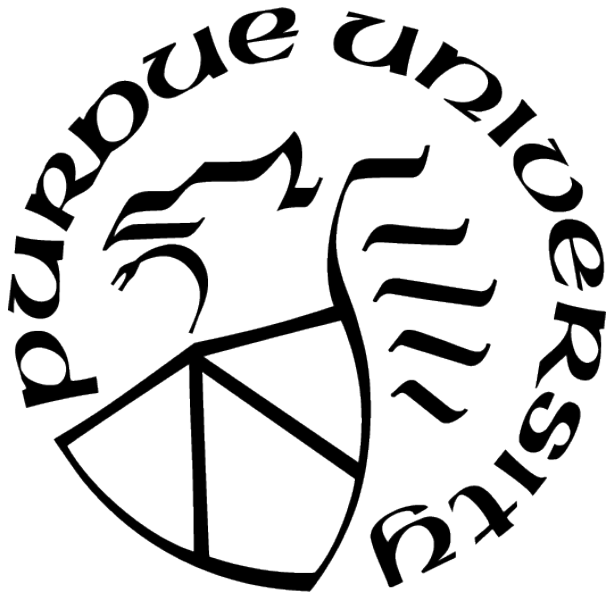
Akshay Satish Dalvi

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Mechanical Engineering



Department of Mechanical and Energy Engineering at IUPUI

Indianapolis, Indiana

December 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Hazim El-Mounayri, Chair

Department of Mechanical and Energy Engineering

Dr. Ali Razban

Department of Mechanical and Energy Engineering

Dr. Sohel Anwar

Department of Mechanical and Energy Engineering

Approved by:

Dr. Jie Chen

Dedicated to my family

ACKNOWLEDGMENTS

I express my deepest gratitude towards Dr. Hazim El-Mounayri, as he has always motivated and guided me in the right direction throughout my research. Though this research field was entirely new for me, my academic advisor Dr. Hazim El-Mounayri, kept complete confidence in me. He supported me in implementing mathematical aspects into systems engineering. I would also like to thank my committee members Dr. Ali Razban and Dr. Sohel Anwar, for providing insights into my research. I would also like to thank Jerry Mooney for assisting with all the administration concerns throughout my time at IUPUI and reviewing my thesis. I want to thank my colleague at Ford Motor Company to share knowledge and build my forte in the industry. I would also like to thank my friends Ashwin Gaonkar, Shashank Alai, Kalpak Kalvit, Jeevana Madinneni, Vikhil Dessai, Niraj Kumbhare, Nikhil Patil, Atish Dahitule, Abhishek Gawade, Shantanu Shinde, Harsh Saksena, and Hrishi Zhurange, for supporting me in my journey. I would like to thank Fher and Hudson families for making my journey in the US smooth. I thank my colleague Adrian (Ford) for peer reviewing of my system architecture model and Alexander (Claytex) for peer reviewing my dynamic model. Finally, I would like to thank my family for all the support and care.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\hat{C}$	complexity index
C	complexity number
$\hat{C}_s$	structural complexity index
$\hat{C}_s$	behavioral complexity index
S	set of structural elements
U	set of use cases
I	set of interfaces
P	set of functions
P_d	set of white-box functions
i	complexity index of nth component
j	complexity index of pth function
T	program time
M_{i}	inlet mass flow
H	enthalpy
P_{el}	electrical power consumption
$P_{(\mathrm{el},nom)}$	nominal power consumption
$\dot{m}$	mass flow rate
$\dot{m}_nom$	nominal mass flow rate
f_{el}	efficiency correction factor for part-load operation
f_p	fraction of electrical compressor power added to the refrigerant
$\dot{Q}_s$	heat loss rate
$T_{\mathrm{in},h}$	inlet hot condenser fluid temperature
$T_{\mathrm{in},c}$	inlet cold atmospheric air temperature
m_{ev}	mass flow rate of chilled water, kg/s
c_p	specific heat of water, J/ (kg K)
Q	heat removed by an evaporator, J
$T_{ev,\mathrm{in}}$	chilled water inlet temperature, °C
T_{target}	target temperature, °C

ABSTRACT

The internal and external interactions between the complex structural and behavioral characteristics of the system of interest and the surrounding environment result in unpredictable emergent behaviors. These emergent behaviors are not well understood, especially when modeled using the traditional top-down systems engineering approach. The intrinsic nature of current complex systems has called for an elegant solution that provides an integrated framework in Model-Based Systems Engineering. A considerable gap exists to integrate system engineering activities and engineering analysis, which results in high risk and cost. This thesis presents a framework that incorporates indefinite and definite modeling aspects that are developed to determine the complexity that arises during the development phases of the system. This framework provides a workflow for modeling complex systems using Systems Modeling Language (SysML) that captures the system's requirements, behavior, structure, and analytical aspects at both problem definition and solution levels. This research introduces a new level/dimension to the framework to support engineering analysis integrated with the system architecture model using FMI standards. A workflow is provided that provides the enabling methodological capabilities. It starts with a statement of need and ends with system requirement verification. Detailed traceability is established that glues system engineering and engineering analysis together. Besides, a method is proposed for predicting the system's complexity by calculating the complexity index that can be used to assess the complexity of the existing system and guide the design and development of a new system.

To test and demonstrate this framework, a case study consisting of a complex district cooling system is implemented. The case study shows the framework's capabilities in enabling the successful modeling of a complex district cooling system. The system architecture model was developed using SysML and the engineering analysis model using Modelica. The proposed framework supports system requirements verification activity. The analysis results show that the district chiller model developed using Modelica produces chilled water below 6.6 degrees Celsius, which satisfies the system requirement for the district chiller system captured in the SysML tool. Similarly, many such requirement verification capabilities using

dynamic simulation integration with the high-level model provides the ability to perform continuous analysis and simulation during the system development process. The systems architecture complexity index is measured for the district cooling case study from the black-box and white box-perspective. The measured complexity index showed that the system architecture's behavioral aspect increases exponentially compared to the structural aspect. The systems architecture's complexity index at black-box and white-box was 4.998 and 67.3927, respectively.

1. INTRODUCTION

The most fundamental building block of our world, if isolated in void space, has no existence. It does not exhibit any behavior. Such an entity's mere existence is impossible because when we say the entity is isolated, we imply that it exists relative to us. This underlying principle of existence supports the importance of interactions. Interactions are the essence of reality (existence). Here evolves the concept of a 'system'; a system consists of more than one entity and has at least one interaction or connection. An 'interaction' represents an exchange of information. This exchange from an external perspective is called a behavior. When external observers (humans mostly, but not limited to) of the system started taking an interest in the system's behavior that emerges from the interaction, it gave rise to the development of systems engineering discipline. Based on this, it can be inferred that the system's behavior can become more complicated as the number of participants (parts) increases. 'Complex' behavior emerges in the observer's mind when he/she attempts to comprehend the increasing interactions. In the system engineer's community, it is called the system's complexity. Complexity is the behavior that emerges when the observer interacts with the system. A simple arithmetic operation of one-digit numbers can be simple to compute for an adult, but it might be troublesome for a first-grade kid and non-existent for an infant child. The infant child does not interact with the calculation; therefore, there is no emergent complexity behavior between the system (equation system) and an infant child.

The increasing need to extract the benefits from a dynamic system complies systems engineers to develop a systematic approach that can realize the successful development of a system. Earlier, many organizations used the document-based approach for systems engineering. This traditional approach does not provide a concrete platform for system development that's spread across multi-domains. Model-based system engineering (MBSE) provides a model-oriented platform that can be shared and transformed across multi-domain. MBSE provides completeness and a consistent relationship between requirements, design, and analysis.

System Modeling Language (SysML) [1], is the graphical language created as an international standard to support MBSE [2]. The SysML is a standard modeling language developed

by the Object Management Group (OMG) to support systems engineering activities. There is a need to deploy activities that link system engineering activities and engineering analysis to enable a system. Kim, et al. [3], show that the gap between systems engineering and engineering analysis results in inefficiencies and quality issues that can be very expensive. The current SysML modeling tools have limitation to system-level analysis and is limited to simple parametric equations. SysML model describes a system with a high level of detail, and it is difficult to evaluate how well the design meets the requirements or performs essential trade-offs between performance, cost, and risk [4]. SysML is merely a language describing system specification, and it is not a framework or a method. According to Kim et al. [5], MBSE cannot be implemented based on SysML alone; it needs a methodology to provide insight into the starting modeling process. A modeling language, when combined with the right methodology, can initiate system development [5]. A framework is a canvas on which the activities can be placed; it provides a workflow to complete modeling activity. To Morkevivius et al. [6], MBSE Grid [7] is a simplified approach for modeling with SysML that gives ambiguous guidelines for the modeling process.

This thesis introduces an innovative approach to fill these gaps. A framework inspired by the Zachman style matrix [8], and the MBSE grid [6], is introduced to capture the system's complexity aspect. Adjoining the existing three pillars, such as requirements, structure, and behavior, a new complexity pillar (column) is introduced in the framework matrix to fill this gap. A new engineering analysis row is added to the framework matrix to support engineering analysis and integration with the system architecture model. A workflow is provided that provides us the enabling methodological capabilities. The workflow starts with the elicitation of the 'statement of need' and ends with 'system requirement verification.' Detailed traceability is established that glues system engineering and engineering analysis together. A case study on a complex district cooling system is implemented to test how well this framework adapts to solving the existing problems.

A part of the thesis work is published in the ECOS paper [9]. Increasing energy demands have raised interest in exploring technological solutions for achieving efficient large-scale energy systems. The cooling needs are at their peak, and the district cooling system (DCS) technology provides an efficient way to satisfy this vast need [10]. The DCS, as shown in

Figure 1.1, consists of a district cooling plant (DCP), a distribution system (DS), and an energy transfer station (ETS). The DC plant was modeled to capture the system's requirements, structure, and behavior at one end. On the other end, a computational model of the DC plant was modeled for verification purposes. DCP is the most crucial part of the overall system that produces chilled water at the desired (target) temperature. This chilled water is supplied through the DC network to the buildings for air conditioning purposes. The DCS offsets the need for mechanical rooms in each building. Thermal energy storage for optimal chiller loading has shown energy saving as high as 9.4% and cost-saving as 17.4% compared to the conventional cooling system [11]. The DCS needs a transformation in managing operational parameters and predicting system behavior along with financial and water savings. The DCS is a complex dynamic system that is driven by customer needs. Most importantly, the total efficiency of the DCS is dependent on the various subsystem. A systematic approach is needed to allocate an operational system to satisfy the system requirements and top-level mission requirements.

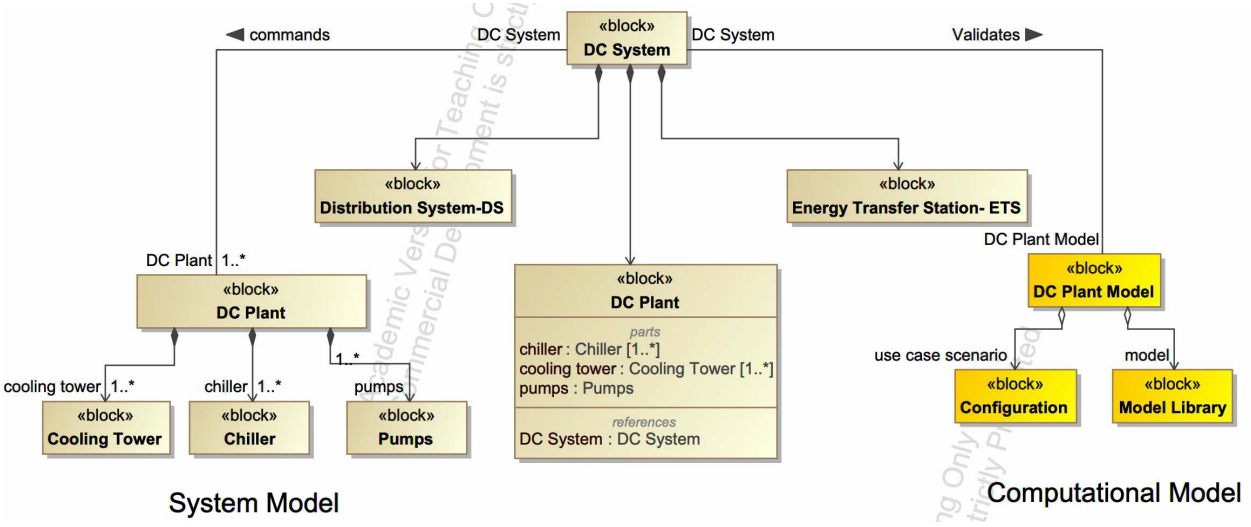


Figure 1.1. High-Level System and Computational Model of the District Cooling System

The multi-plant DCS presents several challenges in integrating system level requirements with engineering analysis for verification and optimization. Systems engineering provides the top-down approach for system synthesis and its development, where the system is considered as a 'whole' rather than 'distinct' and unrelated parts. Systems engineering processes are

applied for the development of the system [2]. A model-based systems engineering (MBSE) approach allows developing a system architecture model that supports system requirements, design, analysis, and validation activities throughout the system's life cycle phases [12]. The scenario-based approach used in developing system architecture models helps in evaluating alternative options to meet stakeholder's objectives. The system architecture model provides an abstraction to the operating system's structural and behavioral concepts that are displayed with various viewpoints using a graphical modeling language like SysML. Dynamic modeling and simulation capabilities are needed during the analysis and development phase of the system. SysML has limitations over providing such capabilities; hence domain-specific simulation tools are used.

Modelica is an acausal, object-oriented language used to model and simulate the system's dynamic behavior using differential and algebraic equations (DAEs) [13]. Both SysML and Modelica are based on the same principles. The model represents the top-level system's decomposition into simpler subsystems/components at the logical and physical level, which are later integrated into a complete system model for validation and verification.

SysML leverages model integration by abstracting domain-specific aspects into the systems architecture model. This thesis presents a SysML-Modelica integration using functional mock-up interface (FMI) standards. This SysML-Modelica integration framework helps in managing different modeling languages used for solving systems engineering problems. Dynamic modeling of a DCP helps predict behavior and performance parameters such as chilled water temperature difference, energy consumption, and chiller plant COP of the multi-chiller plant. Dymola is a Modelica based modeling and simulation environment that allows describing the dynamic behavior of a system using mathematical equations [14]. The performance requirements of DCP are verified through the simulation results. The developed integrated model using the system architecture model in SysML and dynamic/simulation model in Modelica demonstrates MBSE tools and method's capabilities to help meet the inter-dependability of system engineering and engineering analysis requirements. Closed-loop information flow was developed to map SysML constructs with their respective Modelica models to support simulated experiments with SysML constructs. The platform shown in Figure 1.2 captures the contingency plan for achieving the ultimate goal, where the model

synchronously configures, designs, simulates, and analyses based on the new requirements and needs. A real-time simulator updates these new needs and requirements monitored, controlled, and commanded by the existing system.

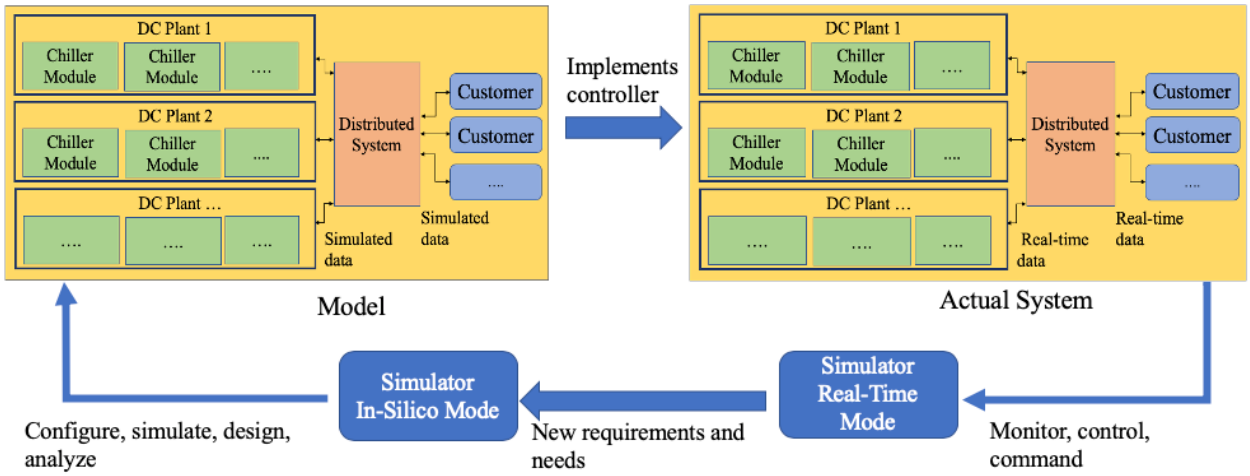


Figure 1.2. District Cooling Simulator Mode

1.1 Background

This section provides a brief overview of some of the most important concepts related to systems engineering. The thesis's interdisciplinary scope makes it necessary to emphasize the details supplied in the background.

1.1.1 System Engineering

Engineers with specific domain expertise generally design components. These components behave well in a predictable fashion when tested and operated in an isolated environment. The concern arises when the components start interacting with external components whose behavior is complex, undetermined, and unpredictable. A component solely cannot satisfy complex needs. Hence it needs to interact and operate with other components that can provide inputs to it to produce desirable outcomes. The level of complexity and unpredictability increases not only when the components interact with an unknown external environment but also when the component interacts with known or well understood (here

it means well-developed whose behavior can be modeled and predicted) components. For instance, no matter how much the instrumentalist may be skilled in an orchestra, they would not produce a pleasant symphony without a conductor.

Who orchestrates the development of a product from a technical and managerial perspective throughout the system's life? How can we assure the success of the product? Who has the bigger picture? Before looking for existing answers to these questions, the reader should understand the term 'system'.

Systems

A combination of interacting elements organized to achieve one or more stated purposes [15] . A system is a human-made collection of elements whose sole purpose is to satisfy the stakeholder's needs. The elements can be physical, non-physical (an idea), living, non-living to perform some activities to produce an expected outcome.

Systems Engineering

An interdisciplinary approach governing the total technical and managerial effort required to transform a set of stakeholder needs, expectations, and constraints into a solution and support that solution throughout its life [15]. Systems engineering is the art developed to address the problem stated above. It is art characterized by having a bigger picture perspective and developing a solution for an operable system that meets requirements within conflicting constraints. In concise, systems engineering is about making a good functioning system that satisfies stakeholder needs and effectively designing the system using the right process and technology.

The systems engineer is responsible for implementing systems engineering at the organization. Systems engineers must possess the art and instinct to identify and target the development effort needed by a particular subsystem/component for optimizing the complete system. Systems engineers ensure that the system/subsystem/components engineering process function correctly and the system evolves from concept to development phase.

1.1.2 MBSE – Systems Engineering Methodology

As we introduce what systems engineering is, it becomes necessary to identify the approach to implement this art. Traditionally, a document-based approach was used, but as the system became more complex, it raised more questions than it could solve. It makes it necessary to call for an approach that could reduce the complexity of handling the systems and support different development aspects throughout the life cycle. This problem raises the questions stated below.

What method would improve communications between stakeholders, system engineers, testing, and specialty engineering teams? What method can manage system complexity by capturing and reusing the existing knowledge?

As stated in INCOSE systems engineering vision 2020, Model-Based Systems engineering is "The formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and latter cycle phases." The current technological timeline cannot be much better for utilizing the MBSE's maximum potential capability. The model-based approach provides a solution for significant questions realized during the implementation of systems engineering. This thesis's work shows how the MBSE approach is enriched by providing simulation models, representing the radical shift in merely utilizing the models for more productive and effective use.

1.1.3 System Life Cycle

As we deal with system development problems, we should capture the concerns that might lie in the latter stages of the systems life cycle. Every human-made system has a purpose that brings the systems into existence—developed to satisfy the stakeholder's needs, produced, and utilized until it serves its purpose and is finally retired. Every human-made system goes through these phases in its lifetime, and the phases are termed as 'life cycle stages' and its journey as 'system life-cycle.' ISO 15288 states system life-cycle stages as "A system progresses through its life cycle as the result of actions, performed and manage by people in organizations, using processes for execution of these actions" [15]. As per ISO/IEC 24748,

general categories of the stages are concept, development, production, utilization, support, and retirement [16]. This technical, management and agreement activities in a systems life cycle are captured in life cycle models. Many big organizations, such as the Department of Energy, Department of Defense, and NASA, have structured their own predetermined set of processes throughout the life cycle stages. These organizations employ steps differently to satisfy contrasting business and risk-mitigation strategies [15]. The most generic form of the life cycle model as shown in Figure 1.3.

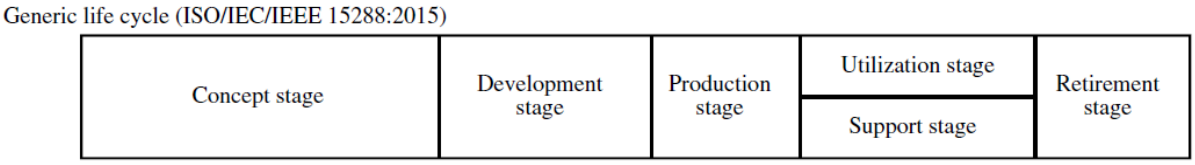


Figure 1.3. Generic Life Cycle Model [15]

All these stages have a purpose, which are shown in Table 1.1.

ISO 15288 has defined processes and activities to meet the purpose of each life cycle stage. The processes are applied at each step of the life cycle as per the system's complexity [17].

Concept

This stage begins with the realization of upgrading the existing or developing a new system of interest with enhanced capabilities and new technologies. Industries use this stage to explore and study the latest technologies to be enabled to the system of interest. They also investigate the potential risk of enabling these technologies to threaten the fundamental pillar of the industries. Threats such as product recalls have been impacting big industries for decades. For instance, the Firestone tire recall that affected Ford vehicles took more than seven years from the first report until a final decision was reached [18]. Companies can avoid recalls/failure by performing thorough work at the conceptual phase.

Table 1.1. Generic life cycle stages, their purposes, and decision gate options [17]

Life Cycle stages	**Purpose**	**Decision Gates**
Concept	Define Problem Space 1. Exploratory Research 2. Concept Selection Characterize solution space Identify stakeholder's needs Explore idea and technologies Refine stakeholder's needs Explore feasible concepts Propose viable solutions	Decision Options • Proceed with next stage • Proceed and respond to action items • Continue this stage • Return to preceding stage • Put a hold on project activity • Terminate Project
Development	Define/refine - system requirements Create solution description - architecture and design Implement, verify,- and validate system	
Production	Produce system Inspect and verify	
Utilization	Operate system to - satisfy users' needs	
Support	Provide sustained - system capability	
Retirement	Store, Archive, - or dispose of the system	

Development

The development stage specifies, defines, and analyzes the system of interest. It receives the output from the concept phase to produce the output, including a prototype, enabling design, and documentation, and cost estimates for other phases [16]. At this stage, the critical activity is to clarify that the elements and their interfaces are specified to be developed and later tested and evaluated. The ultimate goal is to refine the system, develop at the concept phase, and produced to satisfy all system requirements and stakeholder needs.

Production

The production phase is dedicated to analyzing the product capabilities required to produce the system. The analysis may bring some product modifications. Assessment of these modifications is essential before being adapted, as they would influence system requirements [17].

Utilization

The product/system is operated in this stage to deliver the stakeholder needs. Management of these upgrades is carried out to satisfy additional stakeholder requirements.

Support

This phase provides an effort to ensure that the system provides uninterrupted service throughout its operation.

Retirement

This phase begins when the system possesses no capabilities to satisfy the system requirements during its operation. Current development in system engineering emphasizes proper disposal of the system is planned at the concept phase.

ISO 15288

The international standard has defined a set of processes to facilitate communication among acquirers, suppliers, and other stakeholders in the life cycle of a system [15]. These standards are applied to the full life cycle of systems, including conception, development, production, utilization, support, the retirement of systems, and the acquisition and supply of systems, whether performed internally or externally to an organization [15].

1.1.4 Requirements Engineering

As per ISO/IEC/IEEE 29148:2011, requirements engineering: The interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system, software, or service of interest [19].

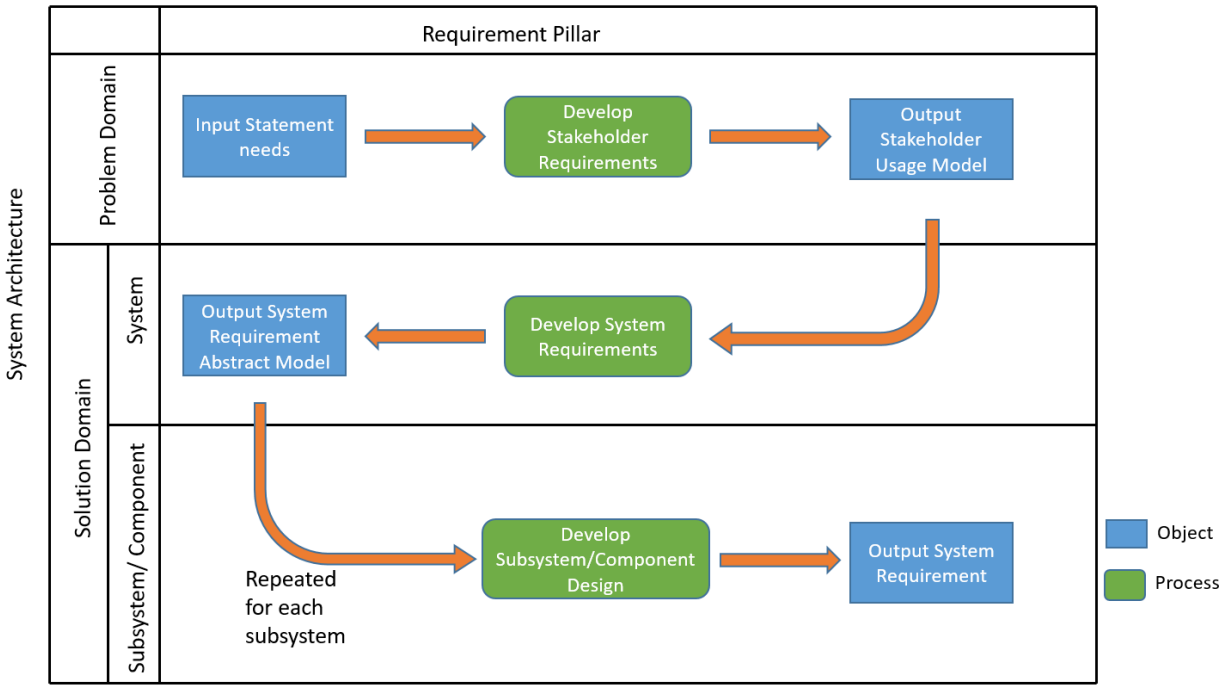


Figure 1.4. The System Development Process from Requirements Perspective [20]

The requirements pillar in the framework follows the ISO/IEC/IEEE 29148:2018 systems and software engineering - life cycle processes - requirements engineering standard [19], aligned with the ISO/IEC/IEEE 15288:2015 systems and software engineering - system life cycle processes [15]. This standard is also applicable to ISO/IEC/IEEE 15289:2019 Systems and software engineering —the content of life cycle information items used for our software and hardware intensive human-made system [21]. Definitions based on ISO/IEC/IEEE 29148:2018 are below.

Requirement

Requirements are statements that translate or express a need and its associated constraints and conditions.

Requirements Elicitation

The use of systematic techniques, such as prototyping and structured surveys, proactively identifies and documents customer and end-user needs [20]. In this framework, the requirements exist at every level of system development. Its relationship is not just limited to the system but also software or other items of interest. Requirement engineering is involved in every stage of product development.

Significance of requirements engineering throughout the development of the system

After the system is well-defined, the system's development is performed when the system is well-defined. This definition of the system is established when the need for the course is realized. There have been many instances where less emphasis on the requirements engineering has led to the whole system's failure. The Standish's Chaos report published in 1995 shows that the combined effect of "incomplete requirements & specification" and "changing requirements & specification" was responsible for a 24.1% increase in system development complexity [22]. Whereas a clear statement of requirement contributed to 13% success of the projects studied. These impacts are shown in Tables 1.2 and 1.3.

1.1.5 System Architecture and Architecture Description

One of the technical processes in ISO/IEC/IEEE 15288:2015 defines the architecture definition process concerned with architecture design [15]. Architecture description uses architecture definition process and other technical processes; hence, it becomes necessary to distinguish between architecture and architecture description. The explanation of the terms of system architecture is below.

Table 1.2. Impact of requirements elicitation in project success [22]

Project Success Factor	%of Responses
1. User Involvement	15.9%
2. Executive Management Support	13.9%
3. Clear Statement of Requirements	13.0%
4. Proper Planning	9.6%
5. Realistic Expectations	8.2%
6. Smaller Project Milestones	7.7%
7. Competent Staff	7.2%
8. Ownership	5.3%
9. Clear Vision & Objectives	2.9%
10. Hard-Working, Focused Staff	2.4%
Other	13.9%

Table 1.3. Impact of requirements elicitation in project failure [22]

Project Success Factor	%of Responses
1. Lack of User Input	12.8%
2. Incomplete Requirements & Specifications	12.3%
3. Changing Requirements & Specifications	11.8%
4. Lack of Executive Support	7.5%
5. Technology Incompetence	7.0%
6. Lack of Resources	6.4%
7. Unrealistic Expectations	5.9%
8. Unclear Objectives	5.3%
9. Unrealistic Time Frame	4.3%
10. New Technology	3.7%
Other	23.0%

Architecture

System fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution [23].

Architecture Descriptions

Architecture descriptions are the work product of system architecting and express architectures for the system of interest [23].

Figure 1.5 below shows that architecture descriptions are a work product, and architecture is an abstract consisting of concepts and properties. Stakeholders of a system have concerns for the system-of-interest considered in relation to its environment. An architecture description includes one or more architectural views. An architecture view (or view) addresses one or more of the system's stakeholder's concerns.

1.1.6 MBSE Methodologies

Estefan has provided a brief overview of MBSE methodologies. It states that MBSE methodologies are a collection of related processes, methods, and tools used to support systems engineering discipline in a model-based context [24]. Different industries are currently making various efforts to implement MBSE, and they adopt and tailor methodologies based on their need. A comparitive study is needed before adapting a particular methodology. Alai has evaluated OOSEM/SysML and ARCADIA/Capella for system architecture development [25]. Discussed below are two of the leading model-based systems engineering methodologies.

INCOSE Object-Oriented Engineering Method (OOSEM)

OOSEM is an integrated top-down, model-based approach that uses SysML (System Modeling Language) to capture and analyze the multi-domain system's requirements and design specifications [12].

OOSEM includes activities such as [12]:

- Analyzing stakeholder needs
- Defining system requirements
- Defining a logical architecture
- Synthesizing candidate allocated architectures

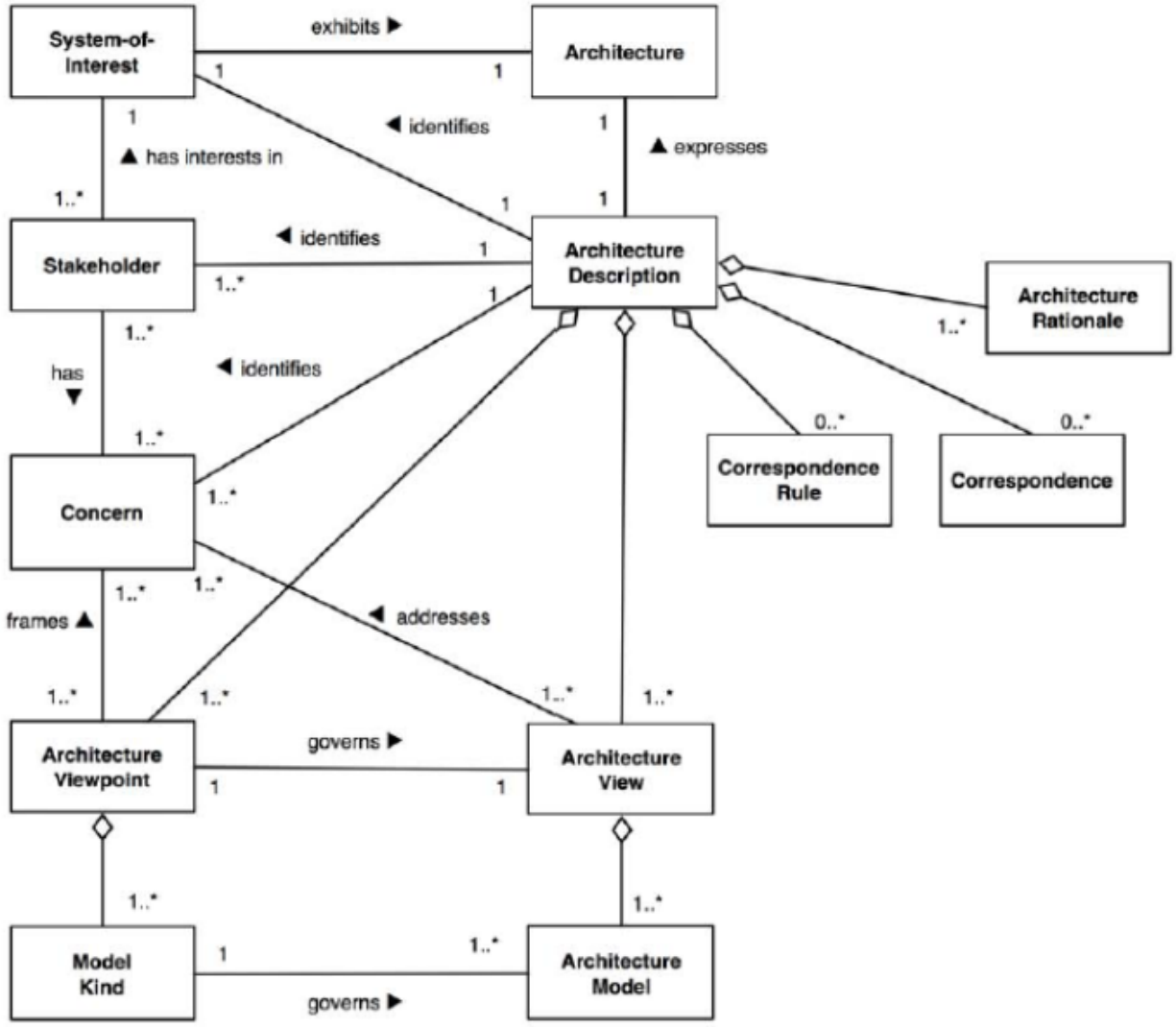


Figure 1.5. Architecture Description [23]

- Optimizing & evaluating alternatives
- Validating & verifying system

OOSEM allows providing views that are captured by systems engineers in the form of SysML based artifacts generated by the activities mentioned above [2]. Figure 1.6 shows the activities along with artifacts generated during the system development process.

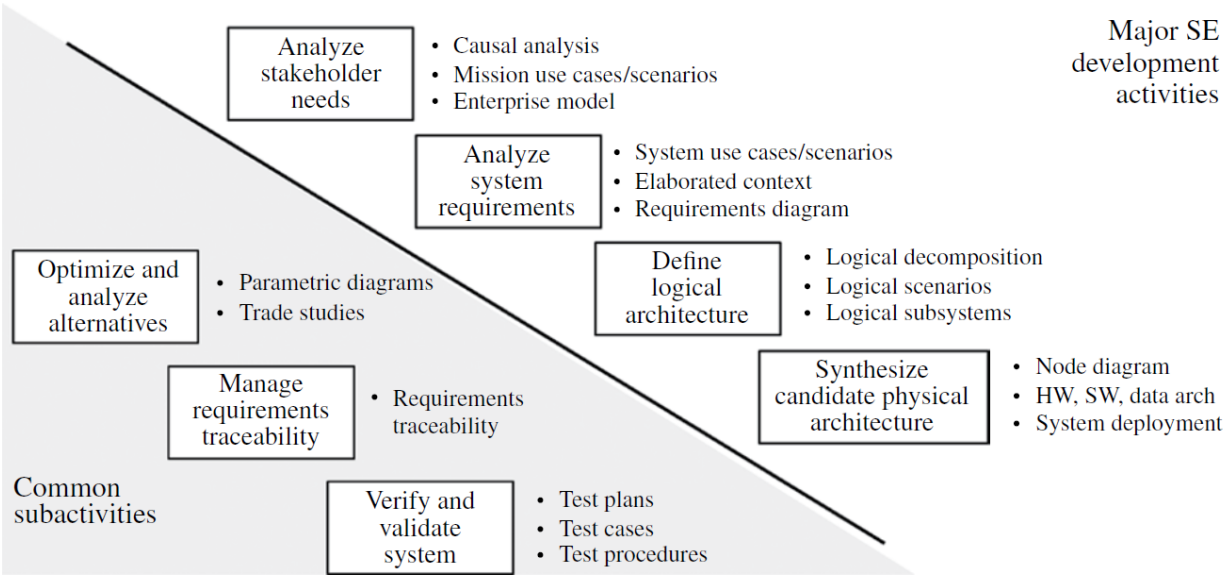


Figure 1.6. OOSEM Activities and Modeling Artifacts [17]

Object Process Method (OPM)

Dori describes Object-Process Method (OPM) using Object-Process Language (OPL), views produced through Object-Process Diagrams (OPDs) [26]. OPL is a language for humans and machines to express the system's functions, structure, and behavior in a single integrated model [26]. The underline principle of OPM states that everything in the universe can be expressed in the form of an object (a physical or imaginary thing that exists or has the potential of existence) and process (the way/pattern in which the object changes).

1.1.7 Dynamic Modeling and Simulation

Modeling and simulation are based on three fundamental concepts; system, model, and experiment. The system specifies the model, and the experiment specifies the simulation criteria. Peter Fritzson shows computer-supported mathematical modeling and simulation using object-oriented, component-based approaches through the powerful Modelica language [13]. The need for simulation is to fill the gap for extracting information from an operational system through experiments. Experiments come with potential risks while performing and costs for building them. We create models to replace the operating system used to perform

experiments, reducing the risk and cost associated with physical systems experiments. An experiment conducted on the virtual computer-based models is called simulation. The models described here are mathematical models. Mathematical models are behavioral descriptions of the system through the relationship between the independent and dependent variables.

Readers should note that models have their limitations and cannot replace the existing system. A perfect model will require a massive amount of information, making it almost impossible because of the associated information buildup needed to build the model itself. An entirely well-defined perfect system will have zero entropy.

All systems are dynamic because their state changes with time. The systems in equilibrium do not change with time; hence, they are time-independent—such systems are a static system, a subset of the dynamic system. Dynamic models depend on the continuous or discrete-time variable.

The modeling and simulation play a vital role in product design and development as it reduces product development time and increases product desired quality. Modeling and simulations fit at every phase of product development. During the requirement analysis, stakeholders provide needs that can include input parameters to design the model. Specification of logical components in the architecture happens in the system design phase. These components are used from the simulation component library or created as new as per the design specification. At the implementation phase, the simulation model saves the enormous cost of building a physical prototype. During the system verification and validation phase, the virtual prototype can verify the subsystem and request change if there are any issues. Similarly, product development employs modeling and simulation in the later stages.

1.1.8 Modeling Languages

Two modeling languages are used in this thesis. SysML 1.6 for modeling system architecture and Modelica 2.0 for mathematical/dynamic modeling.

SysML - System Modeling Language

UML language was adapted to implement MBSE by Object Management Group in 2007 [27]. Morkevicius and Gudas showed that it was too complicated to solve the system engineering domain-specific problems [28]. OMG released the first version of a domain-specific language, SysML, in 2007 to tackle UML issues [29]. SysML is a modeling language that provides objects for describing problems and solutions. It is also a mode of communication between stakeholders, system engineers, and specialist engineers. As stated by Silingas and Butleris, SysML alone cannot successfully implement MBSE in the organization. Hence, it needs to be amalgamated with the appropriate methodology and framework to become useful for the organization [30].

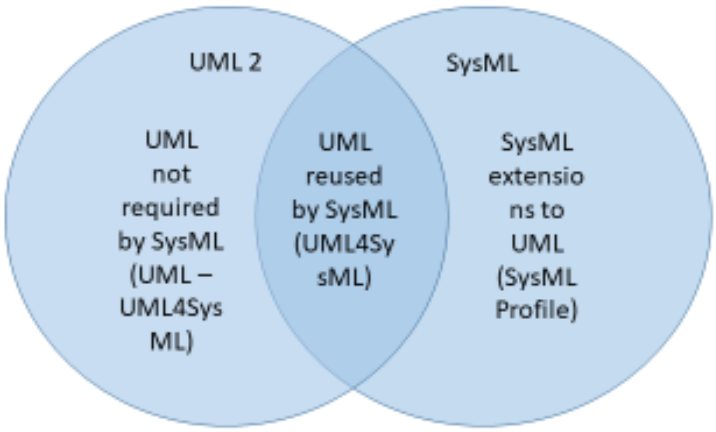


Figure 1.7. Overview of SysML/UML Interrelationship [29]

SysML consists of nine diagrams types of diagram are shown in Figure 1.8.

Package diagram (PKG) is implemented to organize the system architecture model. Package diagrams convey information about the structure of the model itself [31]. Package, model, model library, profile, and view are the element types the package diagram can represent.

Block Definition Diagram (BDD) represents the structural aspect of the architecture of a system. BDD's have capabilities to depict hierarchy, interrelationship, and quantitative information across the system. BDD's can define the model elements such as blocks, actors, value types, constraint blocks, flow specifications, and interfaces.

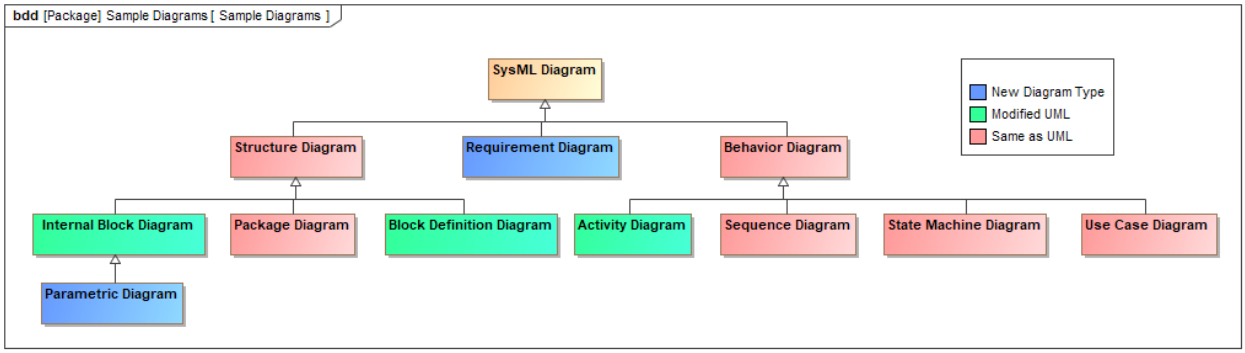


Figure 1.8. SysML Diagrams [1]

Internal Block Diagram (IBD) is used to represent the internal structure of a block element. It provides the static view in which a block's internal structure and connections among part properties. The inner elements for the block class as parts, ports, and connectors. Whereas, for a package element, it can be notes, shapes, and comments.

Requirements Diagram (RED) is used to establish requirement traceability throughout the system development process. There are seven different types of requirement relationships: containment, trace, derive, refine, satisfy, verify, and copy. These relationships can leverage auto-generating requirement traceability and verification matrices (RTVMs) and perform automated downstream impact analysis when requirements change [31].

Use Case Diagram (UC)) represents the use cases (declared kind of behaviors) of the system from the external perspective (actors). Usually, requirements analysts create use case diagrams to capture the system's functional stakeholder needs from the actor's perspective.

Activity Diagram (ACT) represents a sequence of activities (behavior/function) performed by a system of interest or subsystem. It provides a dynamic view of the system in contrast to the static view of the structural diagrams. Usually, an activity diagram is the first of its kind to represent the behavior in the form of use case scenarios of the system.

Sequence Diagram (SQ) serves the same purpose as the activity diagram, but implementation is when the purpose is to represent the interactions between the system's part properties. These blocks interact with one another as operational calls and asynchronous signals to produce emergent behavior.

State Machine Diagram (ST) represents the state's change (behavior) of the system's internal structure. State machine diagrams can represent states of blocks at all system hierarchy levels (such as the system of interest, subsystem, or component). Every block can have one state machine diagram associated with it.

Parametric Diagram (PAR) represents relationships between constraint parameters to create a composite mathematical model. Like IBD, the parametric diagram displays the internal structure but connected value properties and constraint parameters.

Modelica - Dynamic Modeling Language

As discussed earlier, we need dynamic modeling and system simulation to reduce risk, cost, and time associated with performing physical experiments. Modelica is an object-oriented equation-based programming modeling that allows the specification of mathematical models of complex natural or man-mad systems, e.g., for computer simulation of dynamic systems where behavior evolves as a function of time [13]. The fundamental building blocks of Modelica programs are classes, also called models. These classes are like blueprints whose instances are used by the Modelica compiler to create objects. Figure 1.9 shows a sample equation with one variable and two constants (parameters) using Dymola text editor.

```
class SampleEquation
  Real x(start=1);
  parameter Real a=5;
  parameter Real b=3;
equation
  der(x) = a + b*x;
end SampleEquation;
```

Figure 1.9. Modelica Program in Dymola Text Editor

Modelica has adapted the acausal modeling style, which declares equations without specifying the input and output variables. The causality of the input and output variable is fixed only after the equation is solved. This type of modeling style is well suited for physical modeling.

Figure 1.10 shows the process involved in the translation and execution of a Modelica model. Firstly, the Modelica source code is parsed and converted into an internal representation. The classes are inherited and expanded in the translation process, giving a flat set of equations, variables, constants, and function definitions. The flattened equations are sorted topologically according to the data-flow dependencies between the equations. Sorting and converting the Differential-Algebraic Equations (DAEs) to a block lower triangular form (BLT-transformation) is performed. This optimized module uses algebraic simplification algorithms, a symbolic index reduction method, to keep only minimal equations that are solved numerically. Eventually, it creates a link between the generated C code for the reduced equation and a numeric equation solver. C compiler (numeric solver for DAEs) executes this C code for specified simulation time interval [t1, t2]. The results are a set of functions of time, and each variable's plots are with the respective time interval of simulation.

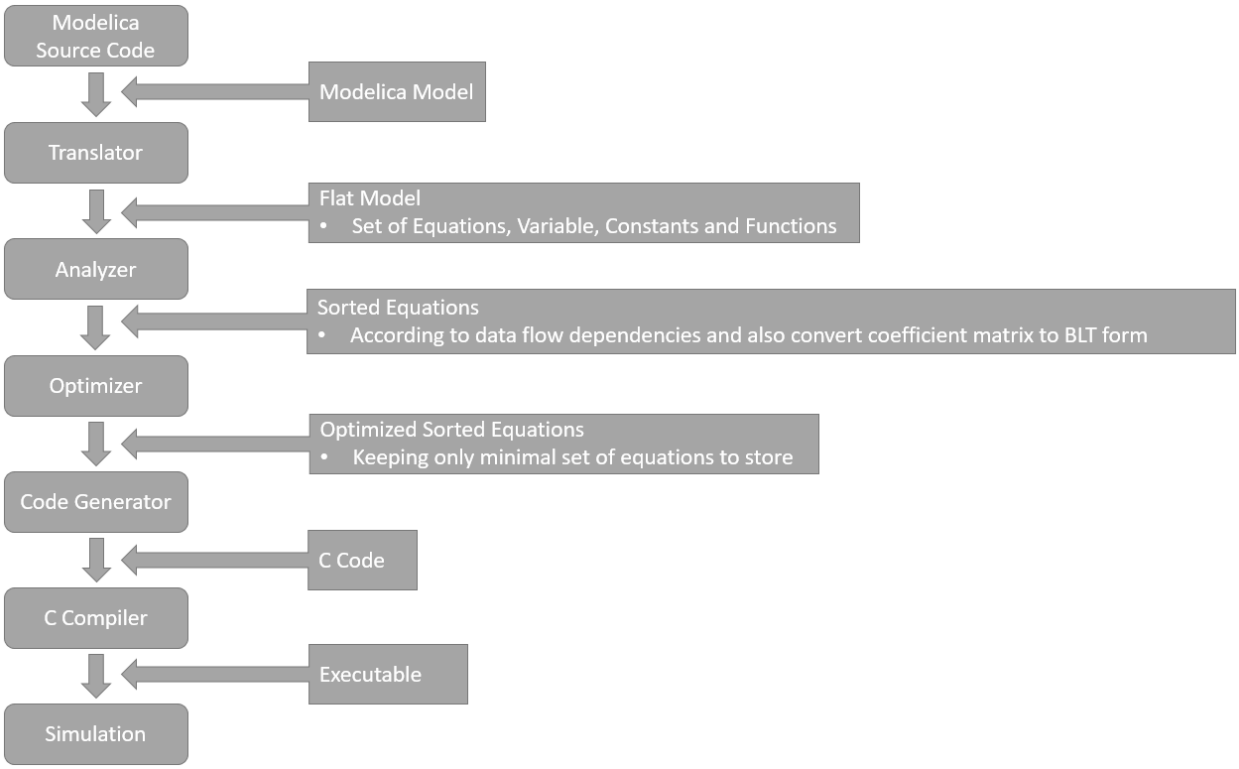


Figure 1.10. Stages of Translating and Executing a Modelica Model

1.1.9 Modeling Tools

Cameo Enterprise Architecture and Dymola are used to develop the system and dynamic model, respectively.

Cameo Enterprise Architecture - SysML Environment

Cameo Enterprise Architecture is No Magic's commercial product, based on core product MagicDraw that offers the most robust standards-compliant DoDAF 2.0, MODAF, NAF 3, NAF 4, and UAF 1.0 via a UAF standardized solution [32]. Cameo Enterprise Architecture supports system architecture development using SysML language. The purpose of choosing this environment is that it is widely used in leading industries and readily available for academic research [32]. For this thesis, SP3 is used to support all nine SysML diagrams and implements based on the Object Management Group's SysML Standards. The tool also supports model integration with dynamic environment tools such as Dymola, using Functional Mock-up Interface standards. Cameo Enterprise Architecture can export detailed views in reports, and using "report wizard" adds to resource ex-changeability across the organization.

Dymola - Modelica Environment

Dymola is a Modelica language-based commercial modeling and simulation tool suitable for various kinds of physical systems. Dymola is a component-based multi-domain environment capable of modeling electrical, mechanical, thermal systems, etc. This environment provides better modeling of actual physical systems; hence, we use it to model the district cooling system. Dymola has unique multi-engineering capabilities, which means that models can consist of components from many engineering domains and allow for complete systems that better represent the real world. Libraries in many different domains are available that contain components for mechanical, electrical, control, thermal, pneumatic, hydraulic, power train, thermodynamics, vehicle dynamics, air-conditioning, etc. [33].

The motorsport and energy systems industries use the Dymola environment. Dymola is chosen for this thesis purpose as the academic version supports applying industrial strength

model libraries and several favorably-priced packages for educational use. Dymola also supports FMU export through FMI 2.0 standards [34] for model exchange and co-simulation.

1.1.10 Functional Mockup Interface

The Functional Mock-up Interface (FMI) is a free standard that defines a container and an interface to exchange dynamic models using a combination of XML files, binaries, and C code zipped into a single file [35]. The FMI (Functional Mock-up Interface) defines an interface to be implemented by an executable called an FMU (Functional Mock-up Unit) [36]. This thesis makes use of FMI 2.0 standard [34]. Users can perform simulations using a simulation solver of the simulation environment (FMI for Model Exchange) or through the FMU solvers (FMI for Co-simulation allows the simulations).

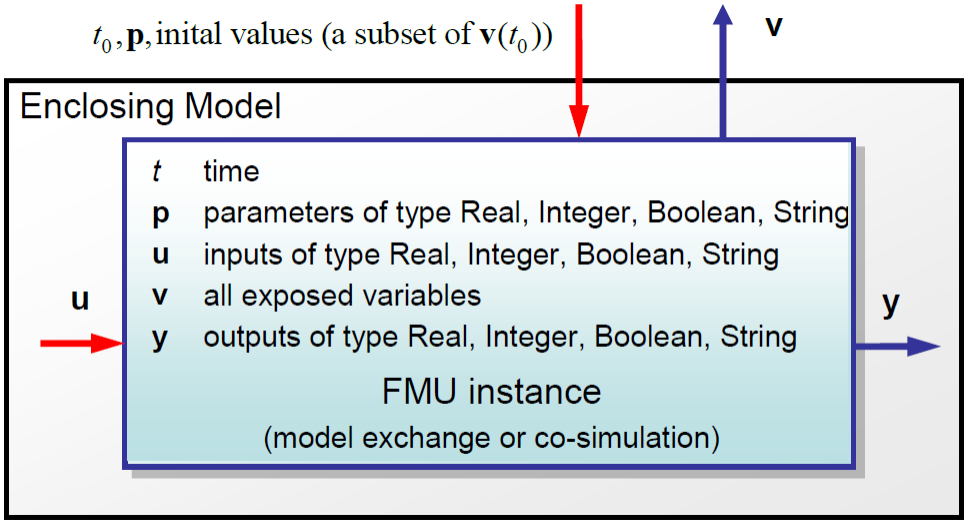


Figure 1.11. A Schematic View of a FMU with Data Flow between the Environment and an FMU (Red is Information Provided to FMU and Blue is Information Provided by FMU) [34]

The FMI for Model Exchange interface defines an interface to the dynamic system model described by differential, algebraic and discrete-time equations [36]. It provides an interface to evaluate these equations as needed in different simulation environments and embedded control systems, with explicit or implicit integrators and fixed or variable step-size [36]. The interface design allows the description of large models.

The FMI for Co-Simulation interface is designed both for the coupling of simulation tools (simulator coupling, tool coupling) and coupling with subsystem models, which have been exported by their simulators together with its solvers as runnable code. The goal is to compute time-dependent coupled systems consisting of subsystems that are continuous in time (model components described by differential-algebraic equations) or are time-discrete (model components described by difference equations, for example, discrete controllers).

1.2 Literature Review

Balchanos et al. [37] presented a parametric study that yields a map used by an operator to choose the best chiller plant operational settings based on weather conditions. They developed a district cooling network model in Modelica and analyzed through simulations for different configuration settings and loading conditions. Such a modeling baseline provides a starting point to improve chilled water performance by adding new modeling techniques. The author develops a high-fidelity dynamic simulation model for selected buildings in Modelica by considering the chiller plant's core dynamics for controls evaluation [38]. Li et al. [39] show how a transient water-cooled centrifugal chiller model is developed in Modelica with Dymola by interconnecting thermal expansion valve, centrifugal compressor, and shell-and-tube heat exchangers. The authors propose a developed chiller model to validate control performance at the simulation phase before the experimental phase. A consistent initialization of system differential-algebraic equations (DAE) for the centrifugal chiller is challenging to obtain because of multiple connections between components [40]. Li et al. [41] proposed a direct initialization method for consistent initialization for centrifugal chiller.

Bandapudi et al. [42] demonstrated the mechanism to meet the requirements of a dynamic model of a vapor compressor centrifugal chiller based on the first principles that run at speed close to real-time. Matei et al. [43] presented the primary modeling methodologies used to simulate dynamic systems. For instance, the author creates the physical-interaction modeling method based on bi-directional information flow between components and modeled using physical conservational laws [43]. Wang et al. [44] propose a systems architecture paradigm for modeling and simulation by integrating architecting tools, executable modeling tools, and analytical tools. The authors perform co-simulation of SysML and MATLAB/Simulink

using an extension of SysML that contains the description of continuous-time behavior [44]. A sample model created was verified for system specification by running simulations with test cases [35]. SysML4Modelica is an extension of SysML, which represents the common Modelica language constructs [36]. SysML4Modelica's specification's objective is to leverage the benefits of both languages and provide bi-directional mapping between SysML and Modelica. SysML4Modelica describes performance requirements and uses external tools to perform requirement verification [45].

An exponential increase in an inter-related system's complexity has generated a need for a systematic approach to modeling and simulating a DCS. The above literature shows the concepts/techniques of systems engineering activities (using SysML) and system analysis (using Modelica) of complex energy systems as two separate entities. This thesis addresses the gap between these two concepts to integrate the system requirements and system analysis using SysML and Modelica.

1.3 Literature Gap

The MagicGrid framework for system architecture development has four columns called pillars that capture the system's different aspects, such as requirement, structure, behavior, and parametric. This framework also has a row that depicts the system's abstraction level from the problem definition to the solution domain.

There has been considerable work done around developing a framework for defining modeling processes. This framework gives us an overview of the kids of artifacts generated at each step of system specification and design. Also, it explains how to manage traceability relationships in the framework [46]. Morkevicius et al. [46] show that the MagicGrid framework currently orients to a system model creation. It needs additional work to include support of system variants, engineering analysis, and verification & validation. Mazeika et al. stated that the current framework does not support full model management and needs a few more pillars to support system variants to perform trade-off analysis and verification and validation [46]. Kalpak Kalvit's findings show that system architects can carry out simulation & analysis in behavioral modeling rather than parametric modeling [47]. It indicates a need

for a row or additional cells to be added to the MagicGrid Framework. This will leverage in capturing the engineering and trade-off analysis activities performed at the solution domain.

As the system becomes more complex, the product's implementation and design become more expensive and risky. There is significantly less work done in understanding the complexity of the system architecture model. However, measuring and understanding the complexity of proposed systems architecture (models) is very important for the whole product development enterprise [48]. Kinnunen also states that there is no widely used systems architecture (model) complexity measures, and large systems development projects are rare and not repeatable, making empirical (comparative) studies hard to perform [48]. It shows a need for an additional pillar that could support an additional aspect of the system development: the complexity aspect.

Kaslow et al. have shown that SysML can model different aspects of a system either directly or through an interface with another model [49]. There is a need for such a reference model for using the updated framework in developing a system architecture model for the energy systems. Firstly, there is no literature found on developing a system model of energy systems using a framework. Similarly, no literature has a specific elaborate framework that captures different aspects of the system, such as workflow for complexity measurement and engineering analysis. This thesis contributes to filling all these gaps in system development.

1.4 Thesis Objective

This thesis aims to provide an integrated framework for system architecture development and engineering analysis. This framework is applied to a district cooling system, which is a complex energy system. This study will help system architects in system specification and design. It will provide them elaborate activity workflow and abstraction, such as problem definition and solution domain. It also provides a workflow that extends beyond the requirement, structure, behavior, and parametric aspects to the system's complexity aspects. MagicGrid framework and Zachman style matrices are a basis for this framework. The cells in the framework represent the different activities carried out during the definitiondevelopment process. Inputs to these cells are defined, and the generated artifacts are listed. The system modeling language is used to develop the system model in Cameo Enterprise Architecture.

No literature exists on the system architecture model for a district cooling system. The activities defined in the framework are applied to a district cooling system. As per the activities' specified cells, identification of DCS's stakeholders elicits stakeholder needs. Subsequently, the problem domain of the DCS is specified. Calculation of the system complexity index at each abstraction level, such as a problem and solution domain, provides various solutions based on the solution domain's configuration. These configurations are tested and evaluated in a dynamic modeling environment. Dynamic modeling of a DCS, including specifications of each component in the system architecture model's solution domain, is modeled and tested in Dymola. Finally, different assemblies are modeled and tested based on the configuration. The simulation results provided by Dymola verifies the requirements specified in system architecture models.

2. SYSTEM ARCHITECTURE DEVELOPMENT AND ANALYSIS FRAMEWORK

This chapter describes the approach (methodology) used in this thesis. The development of an integrated framework is developed using the following steps:

1. The system architecture development activities are described to capture the system's requirements, structure, and behavioral aspects.

2. Adding an extra row to the MagicGrid capture the engineering analysis and verification of requirements.

3. Added a complexity pillar (column) to the MagicGrid framework to measure the complexity of the system architecture in terms of complexity index.

4. Merging architecture development framework activities with the activities associated with the new column and row.

5. Defining workflow for the newly evolved framework.

6. Demonstrating implementation of the enhanced framework for a district cooling case study.

This chapter shows how the ISO 15288 standard's technical processes are mapped to the framework's rows. It also describes the complexity pillar and its development. Later the workflow for the framework is defined. After implementing the case study using the developed, the quality of the system architecture model was evaluated by peer reviewers. Peer reviewers also evaluated the dynamic model.

2.1 Technical Processes

"A process is an integrated set of activities that transform inputs (for example a set of data such as requirements) into desired outputs (for example a set of data describing a desired solution [15]". ISO/IEC/IEEE 15288:2015 has established processes in four groups:

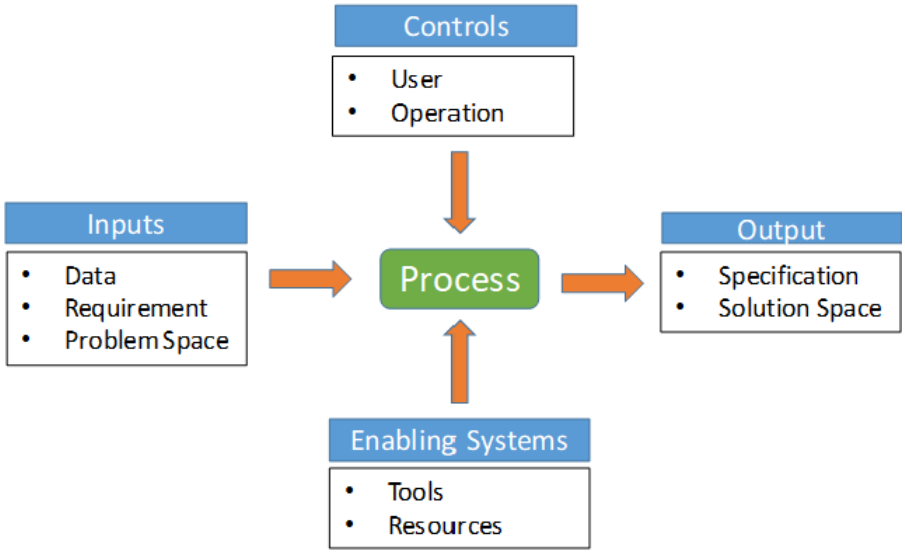


Figure 2.1. Process Flow Inputs and Outputs

agreement processes, organizational project-enabling processes, technical management processes, and technical processes. Activities that support the system's execution at each phase system life cycle effectively and consistently are defined using 14 technical processes shown in the figure below. Technical processes help define the requirements for a system, to transform the requirements into an effective product, to permit consistent reproduction of the product where necessary, to use the product to provide the required services, to sustain the provision of those services, and to dispose of the product when it is retired from service [15].

At each life cycle phase of the system, technical processes are used, such as the early phase (concept), middle phase (development), and latter phase (production, utilization, support, and retirement) of the system life cycle. During the early stage, cross-functional studies using the technical processes define the initial system concept, capture technical feasibility challenges, and provide estimates on cost (development and variable) and risk of adopting the new technology. It also offers viable solution options; these solutions are categorized based on user experience, technological feasibility, cost, and risk. The processes used in the middle phase defines, specifies, and realizes the system. "During later system life cycle stages, they may be used on legacy systems to make technology refreshments or technology insertions, as well as to correct variations from expected performance during production, utilization, support, and retirement [17]".

The scope of the thesis involves only five technical processes. Figure 2.2 below shows the categorization of these five technical processes into two groups based on their product life cycle roles.

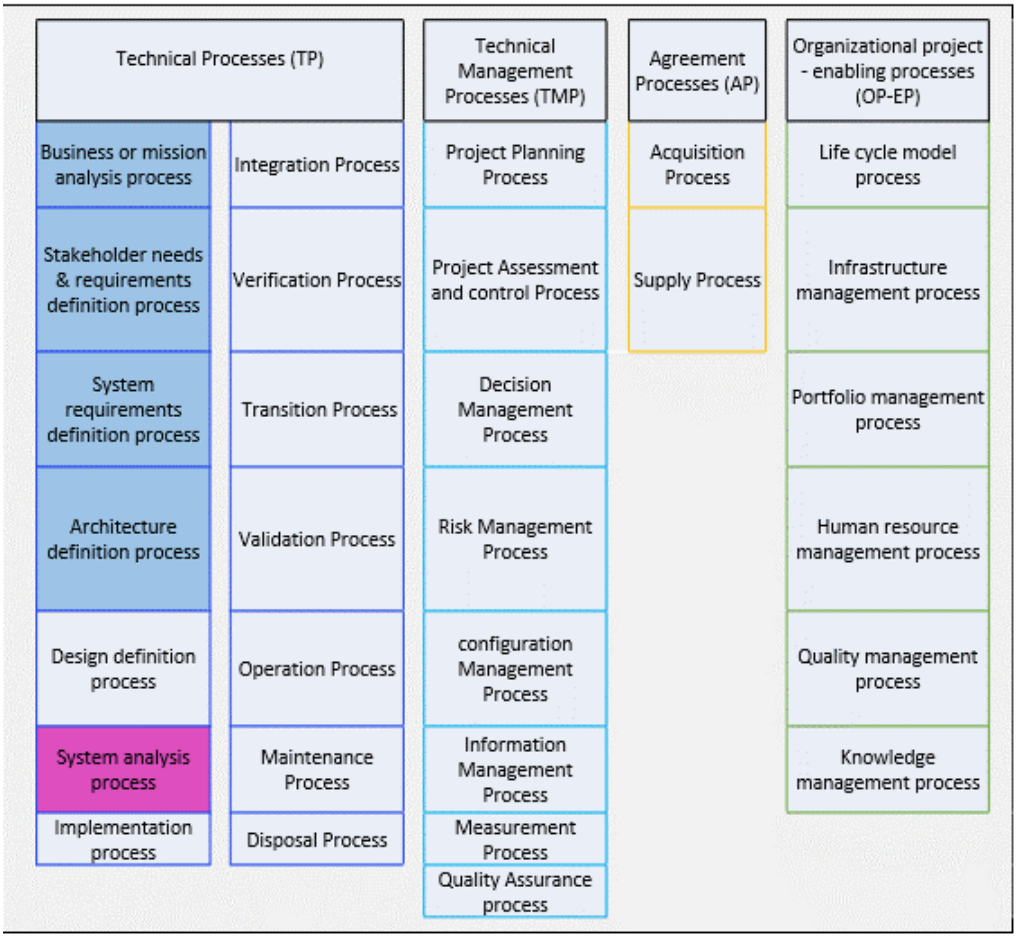


Figure 2.2. System Life Cycle Processes [17]

2.1.1 Technical Processes Concerning System Architecture Development

These activities are majorly concerned with system architecture development.

Business or mission analysis process - 6.4.1: This process defines the problem space and identifies and assesses all alternative solution classes across the solution space.

Stakeholder needs and requirement definition process - 6.4.2: This process is to identify the stakeholders who have a direct or indirect concern with the existence of the system at a certain point of the system life cycle. Subsequently, stakeholder needs are defined and

transformed into stakeholder requirements. Enabling systems (tools, resources, etc.) are identified to support activities that are based on stakeholder needs and system requirements.

System requirements definition process 6.4.3: This process involves activities that specify system requirements (such as functions, non-functions, interface, performance, and physical) derived from the stakeholder requirements. These system requirements tell what the system needs to satisfy to meet the stakeholder needs.

Architecture definition process 6.4.4: Stakeholder has concerns, and the system exhibits architecture that describes and identifies these concerns. The process involving the generation of alternate system architecture and viewpoints to understand the problem space and determine the best solution is called the architecture definition process.

2.1.2 Technical Process Concerning System Analysis

System Analysis process 6.4.6: This process is a basis for providing definite solution criteria for tradeoff analysis. Part of the activity involves generating a virtual environment (of system-of-interest and its environment) that can simulate the system behavior based on the design definition process's definite constraints. The criteria identified earlier are utilized to analyze the simulation's data, which provides the backbone for deciding on the alternate solutions provided in the architecture definition.

2.2 Integrated System Architecture Framework

It has become necessary for systems like the district cooling system to provide an integrated approach that could handle system architecture development and engineering analysis. With its increasing complexities, it becomes necessary to have a mechanism to predict the system's behavior during the early development phase. In this thesis, the MBSE approach facilitates activities like requirements, design, analysis, and verification of complex systems using computerized models. Different domain models produce a large amount of data, and MBSE supports the inter-dependencies of these data. The system-level model maintains consistency, clear communication, and supports multiple views addressing stakeholders' different needs. In this thesis, a system architecture model is created using SysML to develop

a system's multi-domain architecture to satisfy stakeholder needs. This model is capable of executing the behavior of the system. The physical model of the dynamic case study is created using Modelica. The Zachman matrix-style framework inspires the framework. Here there are four pillars of which the first three comes from MagicGrid [6], representing different aspects of the system architecture. A new pillar representing the complexity of the system architecture is added.

The row represents the abstraction level. First, four belongs to the MagicGrid framework. Naas et al. have shown that MagicGrid can be implemented well without having a parametric pillar [50]. A new row is added to capture the engineering analysis and requirement verification of the solution architecture. The cells shown in Figure 2.3 are the result of the intersection between these rows and columns. The cell represents system development activities pertaining to an aspect of the system at a particular abstraction level. Activities performed at each cell is defined and explained in the implementation chapter. The artifacts generated at each cell are described while implementing this framework to the case study. As per the MagicGrid structure, the rows are grouped into domains, such as the problem and solution domain. An enterprise architecture framework provides two main viewpoints, problem and solution; operational and system viewpoints [51], logical and physical in NAF, business, and engineering in Zachman framework [8]. The activities specified in these cells map the technical processes specified in ISO 15288 standards. Such as activities in the first and second row maps to the technical processes 6.4.1 and 6.4.2. Whereas, activities in row three are maps to the 6.4.3 technical process. The solution domain specified in row 3rd and 4th maps the 6.4.4 technical processes. The last row maps that focus on tradeoff analysis based on the simulation results are consistent with the 6.4.6 system analysis process. The description above shows that the framework's activities are consistent with system cycle processes ISO standards.

Initially, a system model was generated to capture stakeholder and systems requirements and subsequently provide a logical and physical architecture view. Later, a generic Modelica model of the system of interest was developed in Dymola. The simulation results were analyzed for performance evaluation in terms of chilled water output temperature, COP, and energy consumption. The values of the results were traced back to the system require-

Figure 2.3. Integrated System Architecture and Analysis Framework

ments. The Modelica model was configured so that the chiller model satisfies the new needs and requirements. The modified chiller model was again simulated and analyzed for further performance improvement. This iterative framework provides quick development time and accurate mapping of the physical model in real-time with simultaneous requirement verification. Alternatively, the model can also be simulated in cameo systems architecture using the cameo simulation toolkit. The analytical and high-fidelity simulation model is integrated using FMI (functional mock-up interface) standards. FMU (functional mock-up units) is a zipped file, a combination of XML description files and binaries files that supports implementation by FMI standards for model exchange and co-simulation [34]. The FMU file generated from the Modelica model is dragged in the block definition diagram. This FMU block containing blocks, value properties, and behavior was executed using the cameo simulation toolkit. The framework was developed to demonstrate the capabilities of MBSE tools and methods to help meet the inter-dependability of system engineering and engineering analysis of a complex energy system like the district cooling system.

2.2.1 Complexity Pillar

System complexity can only be understood when the system itself is well defined. A system is a set of elements working together to produce the desired output. When a particular system doesn't have a desirable output, the collection of elements is still a system but not a system of interest. Researchers perform many efforts to understand the system's complexity. Sillito defined complexity at the problem and solution domain as subjective and objective [52]. Sillito describes subjective complexity as the observer's incapability to understand the system and objective complexity as the technical or system characteristic [52]. Some systems have high variability in parameters. Such systems show non-linear, hierarchical, and emerging, and self-organizing behaviors [53]. Such systems develop unpredictable behavior as they may emerge some variable at any system time.

Here, we try to understand the complexity of developing system architecture at a different abstraction level. Complexity can be categorized as structural, behavioral, etc. The complexity for each category of the system can increase with time but can never remain isolated. The structural complexity is a discrete inherent property of the system. It has a

one-on-one relationship between the sizes of the system. We introduce an additional pillar or aspect in the existing MagicGrid framework: a complexity number C and complexity index $\hat{C}$ to quantize the system's complexity from the system architecture's perspective. The relation between the complexity index and the complexity number is,

$$\hat{C} = F(C) \tag{2.1}$$

F in equation (2.1) represents the relationship which categories complexity index and complexity number,

$$\hat{C} = ln(C) \tag{2.2}$$

Possible factors contributing towards system complexity:

- Size of system (number of components)
- Behavioral complexity of each component (Number of variables, inputs and outputs)
- Complexity arising due to interactions between different components

The complexity number is a sum of complexity arising due to structural and behavior characteristics. The equation 2.3 and 2.4 shows this relation,

$$\hat{C} = \hat{C}_s + \hat{C}_b \tag{2.3}$$

2.2.2 Mathematical Modeling of Complexity Index

The identification of the complexity of a system is a very broad concept. The system's complexity, as we know, can be structural, behavioral, etc., and are these complexities are interdependent. Using complexity theory, we develop a framework to analyze the system's behavior and generate a predictive model system.

The number of artifacts that are generated at a different level of abstraction gives rise to structural complexity. Artifacts are associated with an increased number of elements. This elements can be blocks (system/subsystem/component) or functions. The functions are allocated to the subsystems and components, which gives rise to interconnection and

relation between two types of elements. The complexity increases with an increase in the number of elements and interconnections. Figure 2.4 shows the relationship between elements at different development stages. Here elements are the subsystems which act as the nodes for various connections between stage 2 and stage 3.

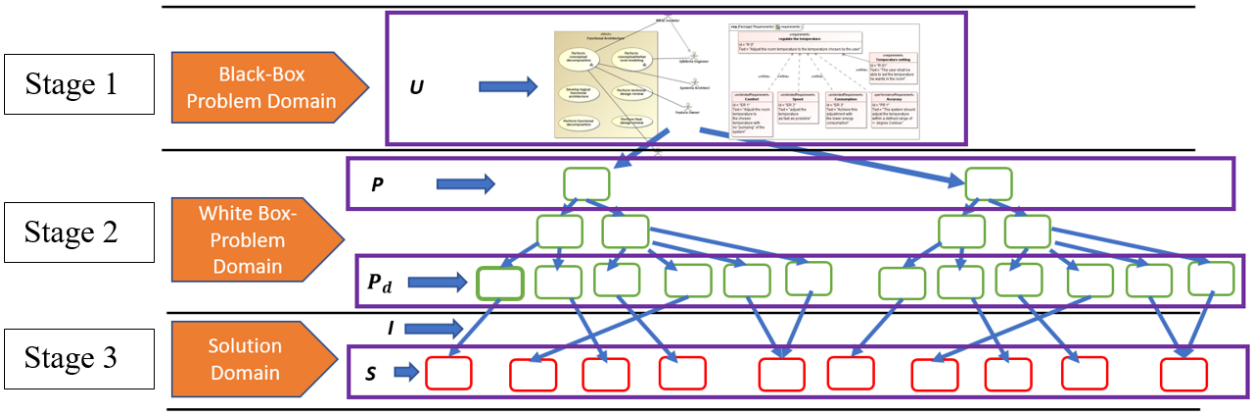


Figure 2.4. Set of Architectural Elements Generated at Different Abstraction Levels

Complexity arises after the interaction of the two levels of system development. The functional level interacts with the implementation level giving rise to relationship and behavior. Here the complexity is calculated on the number of interactions at these two levels, which occurs after allocation.

From a black-box perspective,

$\hat{C}_s = f(S, I)$, structural complexity is a function of the number of components (S), and interfaces(I), $\hat{C}_b = f(U, P)$ behavioral complexity is a function of number of use case (U) and functions associated with each use case scenarios. The complexity index of the system architecture at the black box level is given below,

$$\hat{C} = ln\left(\sum_{u=1}^{u} P^U\right) + ln\left(\sum_{s=1}^{s} I^S\right) \tag{2.4}$$

From the white-box perspective, $\hat{C}_s = f(S, I)$, structural structural complexity is a function of the number of components (S), and interfaces (I), $\hat{C}_b = f(P, P_d)$, behavioral complexity is a function of the number of decomposed functions at functional architecture (P_d) and

functions (P) associated with each use case scenarios. The complexity index of the system architecture at the white-box level is given below,

$$\hat{C} = ln\left(\sum_{p=1}^{p} {P_d}^P\right) + ln\left(\sum_{s=1}^{s} I^S\right) \tag{2.5}$$

The time involved in developing the architecture depends on the complexity index of the system architecture and the complexity involved in developing individual components. This can be shown as,

$$S = \sum \mathrm{i}_n, P = \sum \mathrm{j}_p \tag{2.6}$$

Where i and j are the index of complexity for the n^{th} component and p^{th} function, respectively. We can determine the values of i and j using test data. This thesis does not focus on developing the method to write procedures to determine i and j. From above, it can be inferred that program time (discrete duration in terms of days) for the particular feature can be written as shown in equation 2.7.

$$T = \theta(\hat{C}) \tag{2.7}$$

For θ, the challenge lies in finding the relationship between the complexity index and the program's time. It can be evaluated using test data.

In this thesis, measuring the complexity index is a proposed method to calculate the complexity index. A detailed study needs to be done to start implementing complexity index measurement at the organizational level.

2.2.3 Workflow

This section focuses on stating the established workflow used to develop the system architecture using this framework. Figure 2.6 shows the suggested sequence of activities in the framework.

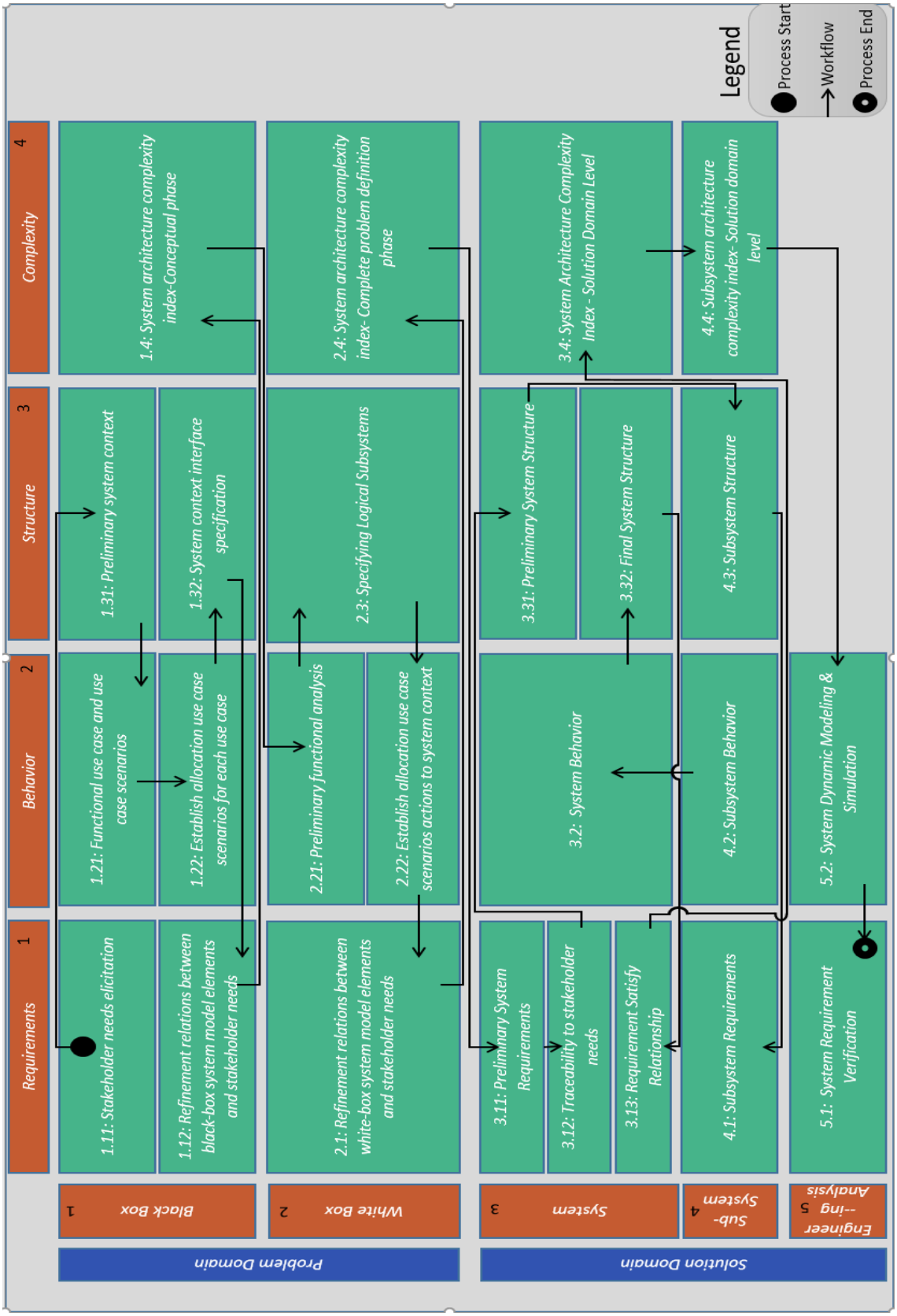


Figure 2.5. Integrated System Architecture Development and Analysis Framework Workflow

Sequence of activities

1. Cell 1.11: Stakeholder needs elicitation
    (a) Step 1: Statement of need
    (b) Step 2: Identify Stakeholders
    (c) Step 3: Generating Stakeholder needs
2. Cell 1.31: Preliminary system context
3. Cell 1.21: Functional use case and use case scenarios
    (a) Step 1: Identify use case
    (b) Step 2: Specifying use case scenarios for each use case
4. Cell 1.22: Establish allocation use case scenarios actions to system context
5. Cell 1.32: System context interface specification
6. Cell 1.12: Refinement relations between black-box system model elements and stakeholder needs
7. Cell 1.4: System architecture complexity index-Conceptual phase
8. Cell 2.21: Preliminary functional analysis
9. Cell 2.3: Specifying Logical Subsystems
    (a) Step 1: Identifying the interface to the system
    (b) Step 2: Identifying the logical subsystems
    (c) Step 3: Specifying the interfaces between the logical subsystems
10. Cell 2.22: Final Functional Analysis and Allocation
11. Cell 2.21: Refinement relations between white-box system model elements and stakeholder needs

12. Cell 2.4: System architecture complexity index- Complete problem definition phase
13. Cell 3.11: Preliminary System Requirements
14. Cell 4.3: Subsystem Structure
15. Cell 4.1: Subsystem Requirement
16. Cell 4.2: Subsystem Behavior
17. Cell 3.2: System Behavior
18. Cell 3.32: Final System Structure
19. Cell 3.13: Requirements Satisfy Relationship
20. Cell 3.4: System Architecture Complexity Index - Solution Domain Level
21. Cell 4.4: Subsystem Architecture Complexity Index - Solution Domain Level
22. Cell 5.2: System Dynamic Modeling & Simulation
23. Cell 5.1: System Requirements Verification

3. IMPLEMENTATION OF CASE STUDY

3.1 Case Study Description

In this thesis, an integrated system architecture development and engineering analysis framework are applied to a district cooling system. Below is the description of the district cooling system (system of interest).

3.1.1 District cooling system

A district cooling system (DCS) distributes thermal energy in the form of chilled water from a central plant to commercial, residential, and industrial buildings for space cooling and dehumidification [54]. The generation of the cooling effect is at the plant rather than at each facility. DCS consists of a chilled water production plant called a district cooling plant (DCP), distribution piping system, and customer interconnection called energy transfer stations (ETSs) [54].

Liquid Chilling Systems

Liquid chilling systems are machines, which removes heat from the secondary liquid (such as brine, water, or coolant) through a refrigeration cycle [55]. The refrigeration cycle categorizes into vapor absorption, vapor compression, and gas cycle. Provided below is a further description of the operational principle of the vapor compression cycle.

3.1.2 Vapor Compression Cycle

Refrigerant goes through various states during the refrigeration cycle. Figure 3.1 shows that the chilled water enters the evaporator (generically, chiller) at 12 degrees Celsius and leaves 6.6 degrees Celsius. The condenser water enters the condenser at 30 degrees Celsius and goes to a cooling tower at 35 degrees Celsius. Figure 3.2 shows property diagrams of the vapor compression cycle.

Ideal operations for vapor compression cycle are listed below:

Compression 1-2: In the reversible adiabatic process, the saturated vapor (state 1) or wet vapor (state 1') compresses into high-temperature vapor. It is recommended to start the process with a dry saturated refrigerant state rather than wet vapor as the liquid refrigerant locks in the cylinder's head, which may eventually damage the compressor components.

Condensing 2-3: In the reversible constant pressure process, the superheated vapor is de-superheated and condensed in saturated liquid (state 3).

Expansion 3-4: In the adiabatic throttling process, saturated liquid (state 3) expands into low-temperature wet vapor. It is preferred to have no flash gas at state 4, as it reduces the net refrigeration of the chiller per unit compressor power consumption. Sub-coolers further cool the condensed refrigerant below its saturated condensing temperature, reduce flashing, and subsequently increase the refrigeration effect.

Evaporation 4-1: In constant pressure reversible cycle, the low-temperature vapor absorbs the heat from the surroundings to cool the chilled water.

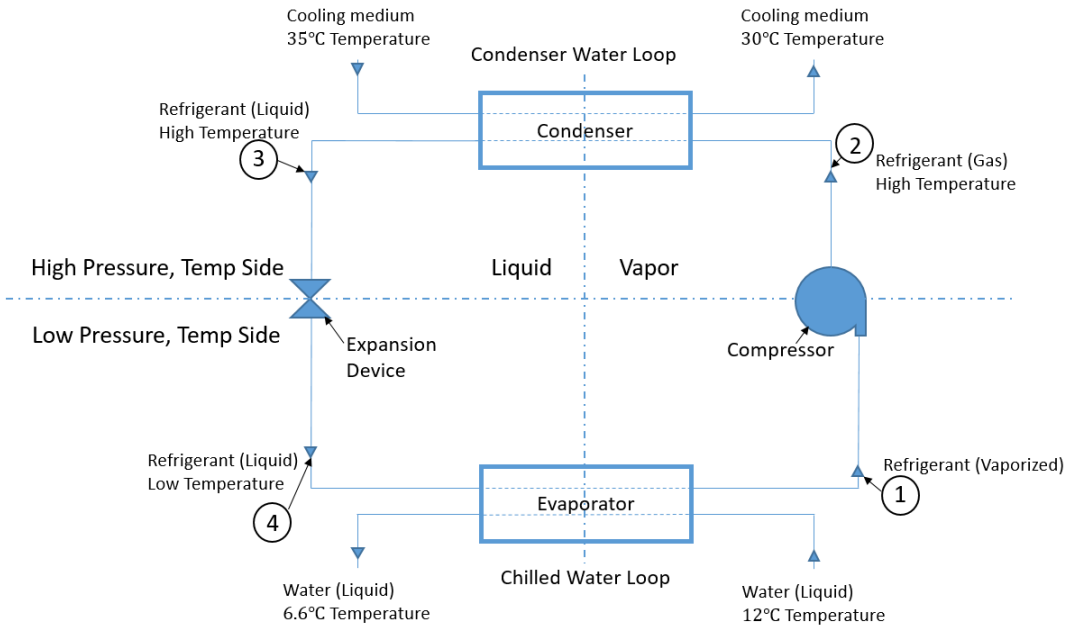


Figure 3.1. System Life Cycle Processes

Basic Operation: The basic chiller system consists of three fluid loops.

- Refrigeration loop: This loop produces the refrigeration effect.

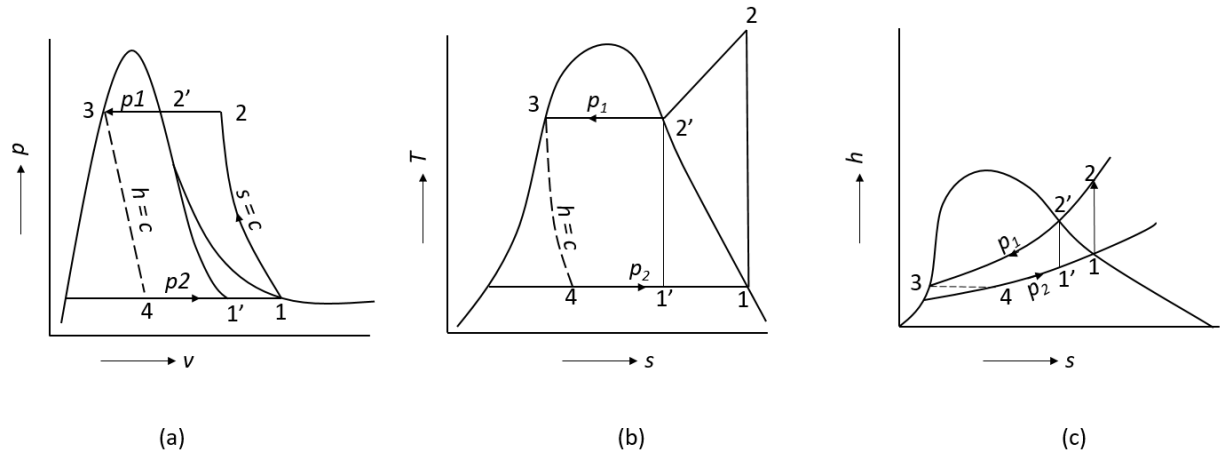


Figure 3.2. Vapor Compression Cycle- Property Diagrams [56]

- Condenser loop: This loop extracts heat from the refrigeration loop and dissipates it to the environment through the cooling tower.
- Chilled water loop: this loop dissipates heat to the refrigeration cycle and extracts heat from the home or substation.

Centrifugal Chiller

Unlike a reciprocating compressor, the centrifugal compressor doesn't have constant displacement. Therefore, the centrifugal compressor provides a wide range of capacities, modulated over a limited range of pressure ratios. The power consumption of the centrifugal compressor is almost proportional to the load capacity. Hence, it provides energy conservation. The compressor can continuously vary the capacity, which provides close temperature control.

Chilled-Water System

Multiple chiller systems: Multiple chiller configurations are used where standby capacity, operational flexibility, and undisruptive maintenance are extended use cases. In this scenario, multiple chiller connections can either be in parallel or series configurations.

Some of the stakeholder needs:

- The system shall provide standby capability in case of maintenance. Solution provided by multiple chillers: During scheduled maintenance for one chiller, the rest of the chillers can operate to satisfy the cooling needs.
- The system should support baseload.
- The system should support the variable load to operate at the most efficient point.

As per the ASHRAE handbook, for chilled water temperature above 7.2 degrees Celsius, all units should be controlled by the combined exit water temperature [55]. And for chilled water temperature below 7.2 degrees Celsius, all units should be controlled by its chilled water temperature.

3.2 System Architecture in SysML

3.2.1 Black Box - Problem Domian

Cell 1.11: Stakeholder Needs Elicitation

Asking questions to stakeholders is the most appropriate approach for this cell. The answers to these questions are the essential components for the existence and development of the system. If these requirements are not captured correctly, then the system's mere presence might be at risk, and the developed system will be different from what's expected.

Step 1: Statement of need

It is a simple statement that addresses the purpose or need of the system. The statement of need for this context is "Need of an efficient district cooling (DC) system that could provide uninterrupted chilled water supply to the customers in the residential and commercial buildings of the Barwa district on a 24*7 basis". The simple statement of need is accompanied by the preliminary analysis report, emphasizing the benefits of having a centralized cooling system [57], instantiating the stakeholder requirement process.

Step 2: Identifying stakeholders

Whom should we ask questions for extracting information? Below is the list of a potential group of stakeholders:

- Users – One of the most important groups of stakeholders is those people who will use the system. Those include people of the 6,000 apartments in 128 buildings, such as international schools, nurseries, retail outlets, banks, health clubs, mosques, restaurants, and various recreational facilities.

- Maintenance and operating staff: The maintenance staff maintains the system's health, and operators handle the system's working

- Investors: Qatar Government.

- Government: Stated in Qatar's government laws and regulations on design, installation, and operations of a district cooling system are the DC design & water management code. It also includes the human health and environment safety code that the DC system needs to abide by. It states the laws, standards, regulations, principles, and requirements (such as key performance index) that apply to DC retailer's systems and business activities [58].

- Managers: Marafeq will operate the plant and provide full maintenance services. Whereas Green Technologies works as a specialist consultant agency on the project

Step 3: Generating Stakeholder needs

Eliciting Stakeholder needs (SNs) are by either asking questions to the stakeholders identified in step 2. The information provided by these stakeholders is refined and specified in the SysML requirement diagram with a specific identifier as SN [46]. The stakeholders provided information related to the system in the form of documents and lists. This list is imported in the form of a spreadsheet in the requirements diagram. The model organization is shown in the Figure 3.3 below. Figure 3.3 also shows a part of SNs for the DC system. System architects realize the following requirements after interviewing the stakeholders. The SN1 Supply Chilled Water was derived from the statement of need. SN2 Set Desired Temperature was provided by the customer and operator when asked, *"what should you be able to do with the system?"* SN3 Desired KPI and SN4 Cooling Load are provided through the government rules and regulation document [58]. Shown below are the identified complex stakeholder needs. The system architects perform a detailed analysis of each stakeholder's needs.

#	Name	Text
1	85 Stakeholder Requirements	
2	DC 300 Certification Requirements	Product Certification Requirements
3	DC 303 Certification Basis	Wherever the certification mark appears, it shall be accompanied by a description of the basis for certification.
4	DC 302 Other Claim Conflict	The certification Mark shall not appear in conjunction with any human health or environmental claims, unless verified and approved in writing by the standardizing authority.
5	DC 301 Standard Certification	A standard certification mark may appear on the product, packaging, secondary documents, and promotional materials, only in conjunction with the certified product.
6	DC 200 Information Requirements	Product must be labeled in accordance with industry standard practice to identify the model numbers, unit serial numbers, and other pertinent information.
7	DC 203 Lubrication	Manufacturers must indicate, either with the chiller operating instructions or on the label, the correct lubricant for the type of refrigerant used.
8	DC 202 Refrigerant Evacuation and Charging	Manufacturers must provide, either with the chiller operating instructions or on the label, the correct procedure for refrigerant evacuation and charging.
9	DC 201 Labeling	Chillers must be labeled as to the types of refrigerant (hydrofluorocarbon or hydrochlorofluorocarbon) they contain.
10	DC 100 Environmental and Performance `Requirements	Product-specific environmental and performance requirements
11	DC 105 Noise Requirements	Manufacturers must make available chiller operating noise characteristics, as evaluated in accordance with ARI Standard 575-94.
12	DC 104 Energy Efficiency	The full-load and integrated part-load value (IPLV) efficiencies, evaluated
13	DC 103 EPA Compliance	The refrigerant used must have an ozone depleting potential less than or equal to 0.02 as determined by the EPA.
14	DC 102 Refrigerant Release	Manufacturer must demonstrate that the testing of chillers for refrigerant leaks prior to shipment must not result in the release of refrigerants into the atmosphere.
15	DC 101 Leak Testing	Manufacturer must test the system for refrigerant leaks prior to shipment.
16	DC 100.1 Chilled water temp	The chilled water temperature shall be 4.4 degree Celsius

Figure 3.3. Detailed Stakeholder Needs

#	Name	Text
1	SN1 Statement of Need	Need of an efficient district cooling system that could provide uninterrupted chilled water supply to the customers in the residential and commercial buildings of Barwa district on 24*7 basis.
2	SN1.1 Supply Chilled Water	The system shall be able to provide chilled water.
3	SN1.2 Set desired temperature	The operator shall be able to operate at desired.
4	SN1.3 Desired KPI	The system shall operating within the KPIs determined by the Regulator.
5	SN1.4 Cooling load	The system shall be able to deliver the design Cooling Load.
6	SN1.5 Uninterrupted water supply	The system shall be able to provide uninterrupted water supply.
7	SN1.6 Sound Level	The system noise level shall be as per building noise regulation.
8	SN1.7 Mass	The total mass of the system shall be as per building mass regulation.
9	SN1.8 Cost	The Operation and maintenance cost should be low

Figure 3.4. Stakeholder Needs

Cell 1.31: Preliminary System Context

The system context is the external view of the operational system. It determines the environment in which the system operates and the actors who interact with the system. This exterior view does not specify the internal structures of the system. For example, a commercial vehicle uses in an environment that includes roads, climate, dust, other cars, etc. The primary actors are drivers, passengers, and to the far end pedestrians, mechanics, etc.

In the district cooling system, the cooling system interacts with the domestic water supply, electricity provider, etc. The system's primary actors are the customers and categorized as residential and commercial occupants, maintenance, and operating staff. There can be more than one system context for the system of interest. For the case study purpose, we are using the system context shown below in figure 3.5.

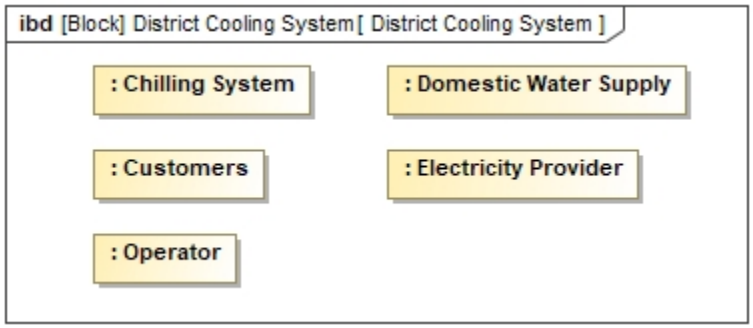


Figure 3.5. District Cooling Plant Context

Cell 1.21: Functional Use Cases and Use Case Scenarios

Functional use cases are use cases that provide users the value of the system. The use case scenario depicts the user's experience while interacting with the system in context to the specific use case. The functional stakeholder needs are analyzed to convert them into functional use cases and scenarios. The use cases should address each stakeholder's needs shown above, and a system context is needed to specify the use cases. Therefore, the system architects perform preliminary development of the system context.

Step 1: Identify use cases

After generating the system context, the system architects specify the use case for the specific system context. The use case that addresses supply chilled water need is defined as providing chilled water supply. The functional stakeholder needs are refined using use cases and use case scenarios. In this context, focus is on the high-level use case "Provide chilled water supply" for the district cooling plant, refined from functional stakeholder needs SN1 and SN2. Figure 3.6 shows the use case diagram for the district cooling system context. The use case diagram describes the high-level functionality that is achieved by the

system. It also specifies its interaction with the actors and the environment. The operational analysis is performed while keeping in mind the operational context as a black-box. The functionality is from the black box perspective, and it doesn't account for the system's internal sub-functionalities. The external systems, such as electricity providers and water supply providers, are also concerned with the district cooling system's use case. With this system, architects achieve the high-level objective of our course and interactions with the external systems and actors.

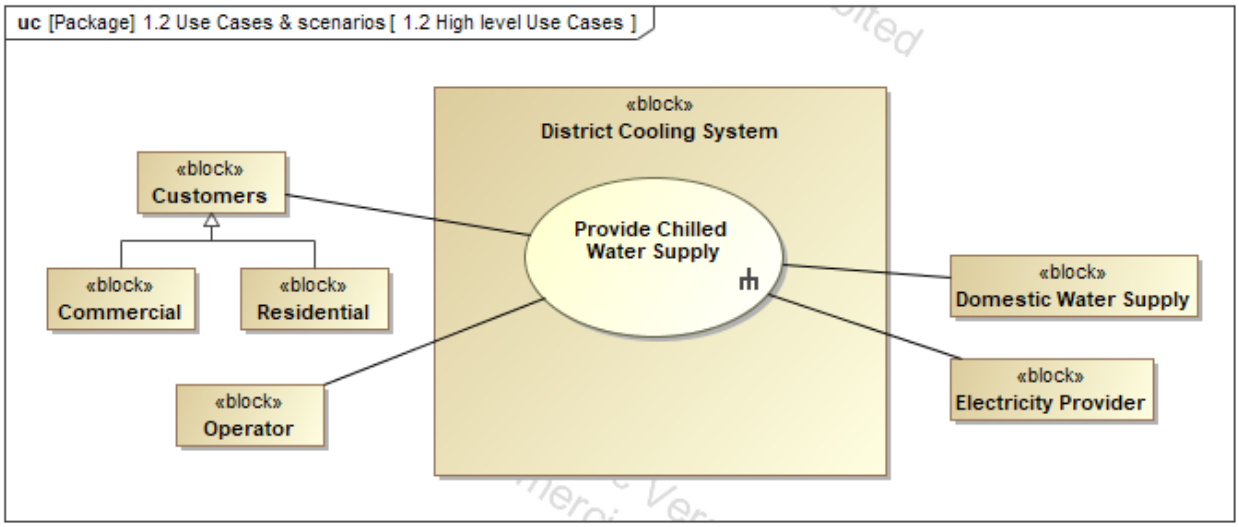


Figure 3.6. Use Case Diagram for the District Cooling System Context

Step 2: Specifying the use case scenario for each use case

Every use case has a use case scenario that describes the set of activities and its flow. These activities are required to be performed by the subsystem to deliver the use case. System Architects categorize the information in scenarios into primary and alternative flow/s. It tells us about the preconditions, the sequence of actions, and alternative flows during conditional checkpoints. This activity is shown in Figure 3.7 in the SysML activity or sequence diagram. The best approach is to list down the actions needed to be performed by the system to deliver the use case. Below is the list of activities for providing chilled water supply.

1. Activate chilling system

2. Check system status

3. Evaluate System Status
4. Monitor pre-condition settings
5. Modify pre-condition settings
6. Display system information
7. Start chilling system
8. Set chilled water temperature
9. Check temperature
10. Attain the Desired Temperature
11. Sustain temperature
12. Stop chilling system
13. Monitor post-conditions settings
14. Modify post-condition settings
15. Deactivate chilling system

Cell 1.22: Establish Allocation of Use Case Scenarios Actions to System Context

The activities specified in the use case scenario of the "Provide chilled water supply" use case are allocated to the system of interest, actors, and environment. The allocation is established considering the system context. Generic allocation is also shown in Figure 3.7, which is not related to any system context. This generic allocation can be used to specify multiple system context.

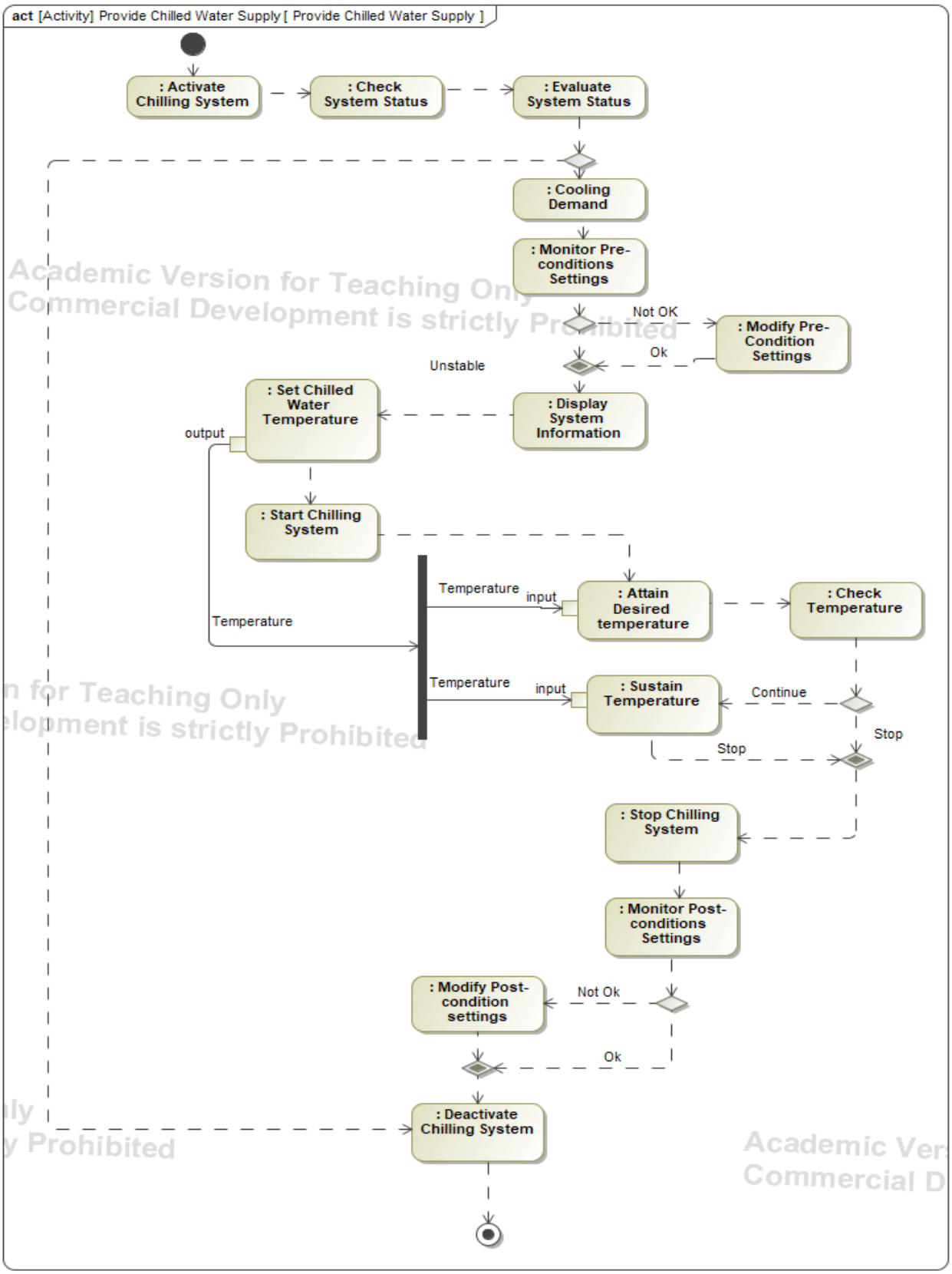


Figure 3.7. Use Sase Scenario for "Provide Chilled Water Supply" Use Case

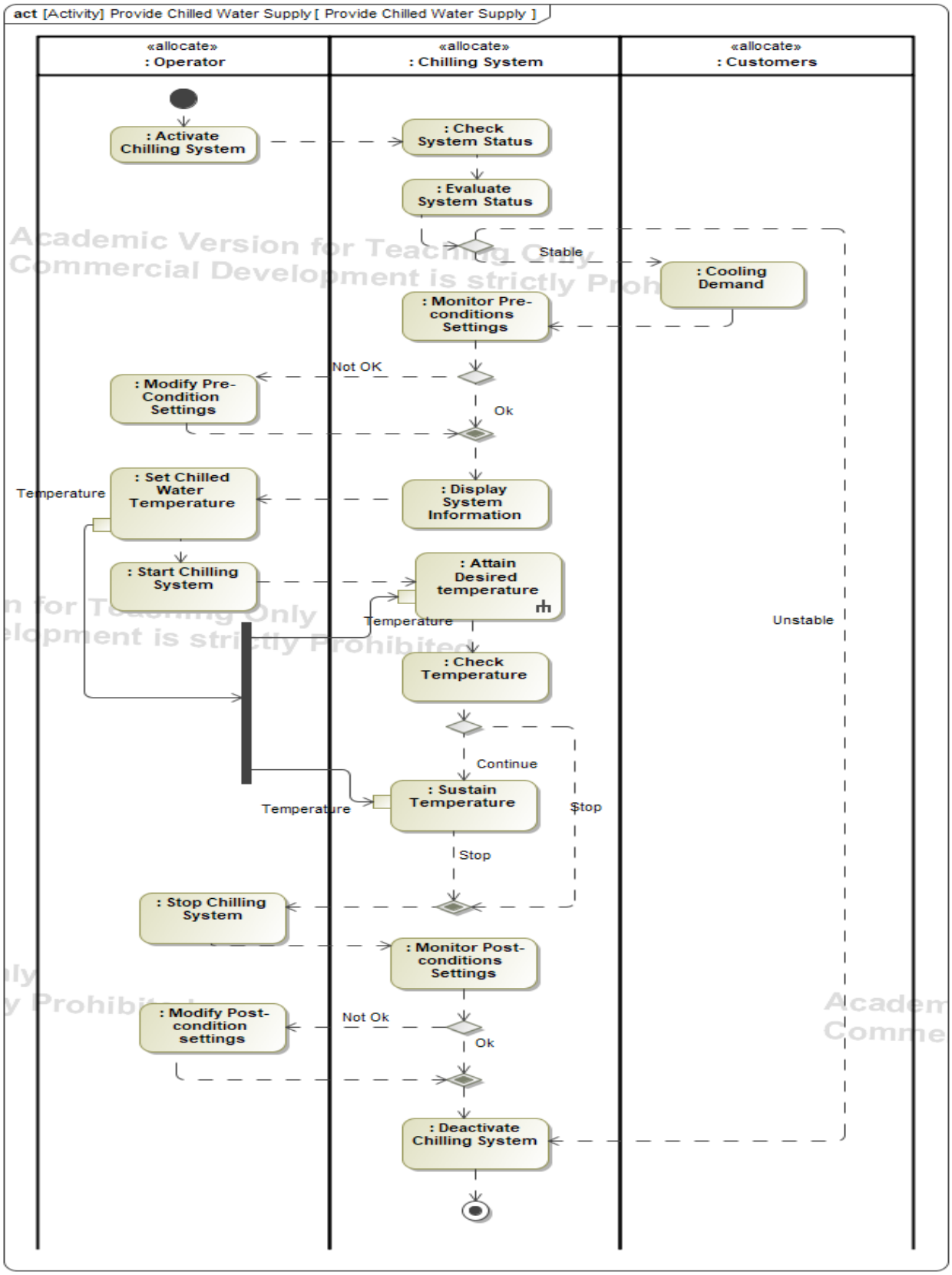


Figure 3.8. Use Case-Scenario with Allocation to the Participants of DCS Context

Cell 1.32: System Context Interface Specification

The system context identifies different ways of utilizing the system. Figure 3.9 shows the interaction of the system with the environmental entities and actors. It also specifies the information that is shared between these entities. This information can be in the form of force, energy, data, signal, or material. The chilling district system receives cooling demand from the customers and delivers chilled water in return. The operator controls the working of the system, and the system provides its operational status in real-time. The environmental entities such as domestic water supply and electricity provider provide water and electricity, respectively.

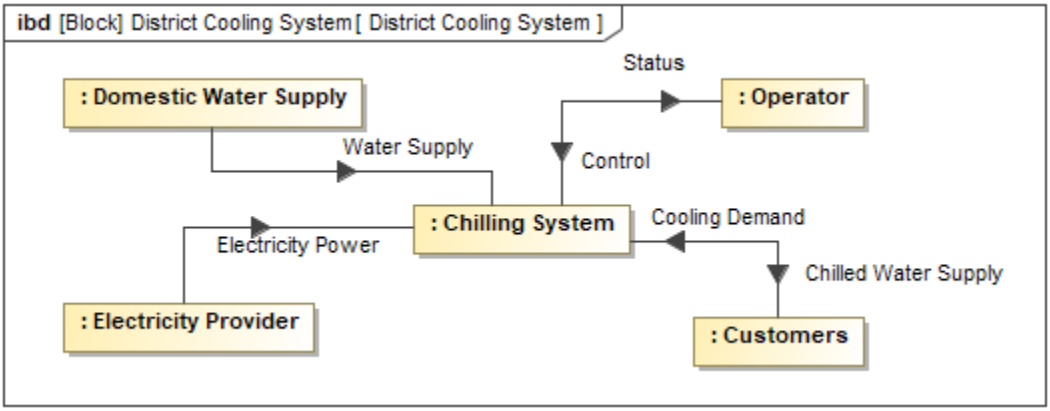


Figure 3.9. System Context Interface Specification

Cell 1.4: System architecture complexity index-Conceptual phase

This activity is the initial evaluation and analysis of the system's architectural complexity from the black-box perspective. The system context diagram provides the values of the number of structures and interfaces. The use case diagram and use case scenario functions associated with each use case are extracted from the use case diagram.

Number of systems S = 1;

Number of interfaces I = 6;

Number of use case U = 1;

Number of functions/activities P = 15.

When these values are computed in equation 2.4, we get,

$$\hat{C} = ln(\sum_{u=1}^{u} 15^{U}) + l_n(\sum_{s=1}^{s} 6^{S}) = 4.9980 \tag{3.1}$$

The structural complexity $\hat{C}_s = 2.7080$, is greater than behavioral complexity $\hat{C}_b = 1.7915$ of the system architecture.

3.2.2 White Box – Problem Domain

Cell 2.2: Functional Analysis and Allocation (Cell 2.21 and Cell 2.22)

Functional Analysis is performed on the functional use case specified earlier during the problem definition using a black-box perspective. Now, the black-box is opened, and action flow pertaining to the system's internal structure is analyzed. The difference between the action flow specified in the 1.21 use case and scenarios and allocation and this activity is that action flows, and allocation specified in the use case scenarios are actions flows performed explicitly by the system, users, and environment. Whereas the system's internal structure performs the action flow specified in functional analysis, and it doesn't show the flow outside the system. The functions abstraction is called system function, subsystem functions, and components functions based on abstraction. Domain functions are those performed by the user or environment to support the system's functionalities. The most elegant part of these functional decompositions is the traceability achieved at each stage. Use cases are refined by the stakeholder needs. Every use case is refined by specifying the use case scenario. This use case scenario provides us a high-level picture of what the system will perform and what functionalities are needed to be performed by the users and environment to support the system functionality.

Further, each system's functional decomposition functions into a subsystem function can be called a refinement of system functions. This function decomposition is carried out further on the subsystem functions to specify component functions based on system development. In a nutshell, one can infer that component functions refine subsystem functions; subsystem

functions refine system functions; system functions refine use case; use case refines stakeholder needs (SNs). Figure 3.10 provides the detailed traceability map in the traceability section.

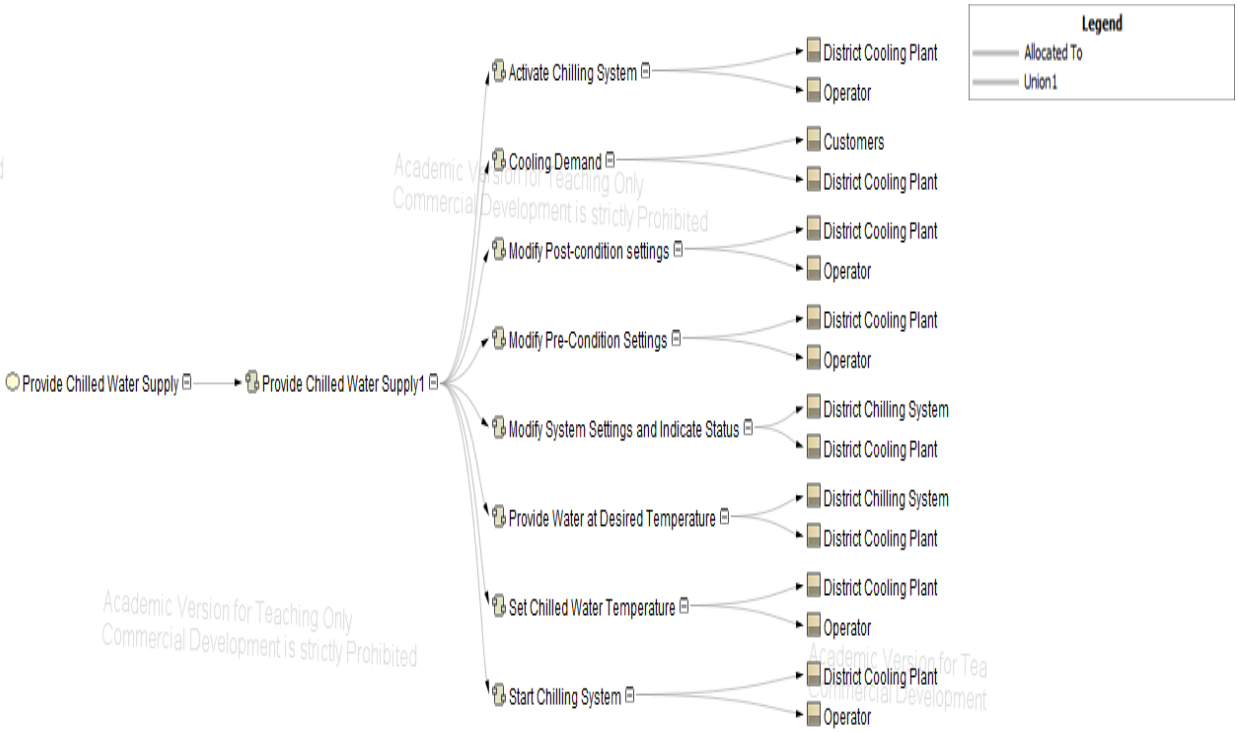


Figure 3.10. System Functions Refined from Use Case and Functional Decomposition

Cell 2.21: Preliminary Functional Analysis

The preliminary functional analysis involves the functions decomposition of each activity or function performed by the system. Decomposing every function in the use case scenario helps understand what subsystem or component of the system will perform activities to deliver the function at the system level. To illustrate further, providing a stereotype of the functions at a different abstraction level. The functions/activities specified in the use case scenarios are a by one or more subsystems/components for the system; hence it is called system functions. For instance, the "attain desire temperature" function/activity is a system function that collects functions performed by the subsystems/components. The functions specified after decomposing the system function are called subsystem functions.

It is essential to note that these subsystems are logical. Figure 3.11 shows the functional decomposition of the "attain the desired temperature" system function. The decomposition at this level produces scenarios that are also called white-box scenarios. Every function in the use case scenario (also referred to as a black-box scenario) require decomposition into a white-box scenario. This means that we have to perform the activity in this cell multiple times, which is equal to the number of functions in all white-box scenarios.

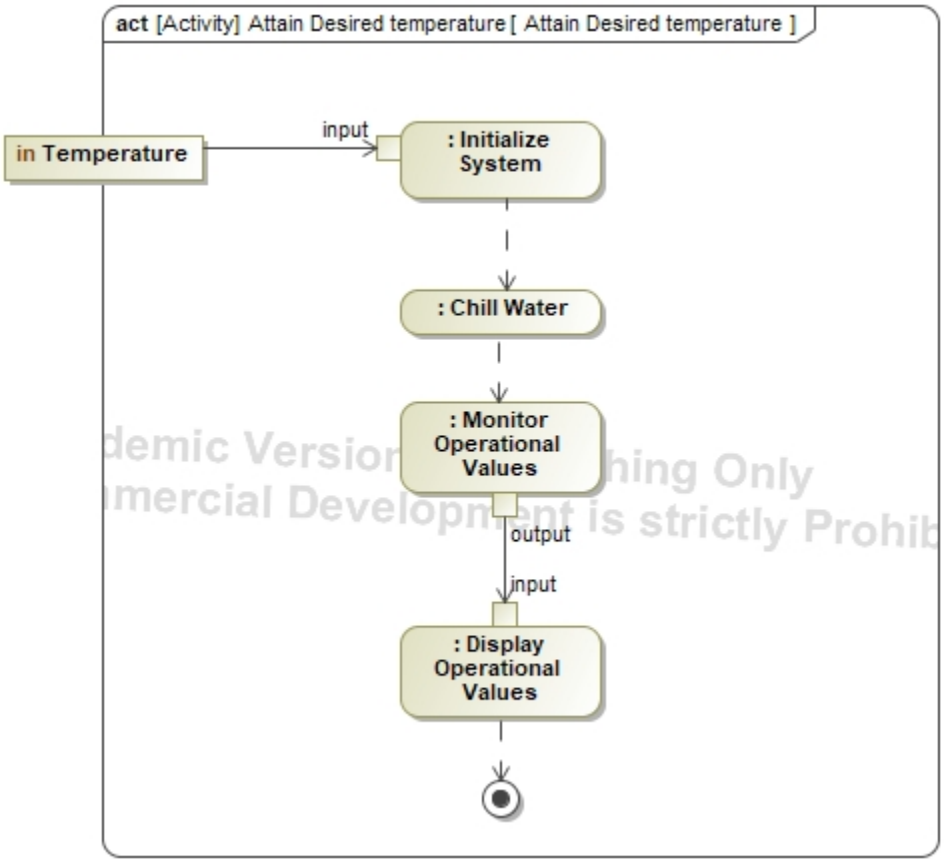


Figure 3.11. White-Box Scenario for "Attain Desired Temperature" System Function

Cell 2.3: Specifying Logical Subsystems

The activity in this cell needs to be performed for every white case scenario. This activity starts with identifying the interfaces to the system, followed by identifying logical subsystems based on the preliminary functional analysis, and finally ends with specifying the interconnections between the subsystems.

Step 1: Identifying the interfaces to the system

The system's inputs and outputs are defined, which eventually assists in the definition of the logical subsystems. The process of identifying logical subsystems is brainstorming. Every information that comes 'in' needs an input function and an output function for one's that goes 'out' of the system. For instance, the district chilling system receives "chilled water temperature" from the operator. Hence we need an input function, "attain the desired temperature," and the transformation function is the function that transforms the input function into the output function. The "chill water" is the transformation function producing operational values through an output function named "display operational values." This produces output for the operator, which is the operational values. The system context analysis shows that the district chilling system takes water, electricity, cooling load, and control from domestic water suppliers, electricity providers, customers, and operators. Status and chilled water supply are provided to the operator and customer, respectively. Figure 3.12 shows these interfaces to the *district chilling system*.

Step 2: Identifying the logical subsystems

The logical subsystems that perform the functions specified in the white-box scenarios are identified. Preliminary functional analysis at Cell 2.21 gives insight into the internal blocks or logical subsystems needed to execute the subsystem functions identified. In this instance, a cooling system is required to achieve the chill water function. Similar analysis reveals that the controller and HMI (human-machine interface) systems are needed to carry out other subsystem functions.

Step 3: Specifying interfaces between logical subsystems

The logical subsystems identified in the previous step are interconnected, and they share information with each other. These subsystems receive information from outside the system of interest, and they transform these inputs into desirable outputs to be delivered to the users or systems outside the system boundary. In this step, we specify these interconnections between subsystems and inputs/outputs outside the 'system of interest.' We also specify the interconnections between the subsystems and link them with connectors. SysML proxy ports are used to specify the ports on the part properties. The best practice to initiate this process is to connect the part properties with the ports on the boundary of the system of interest.

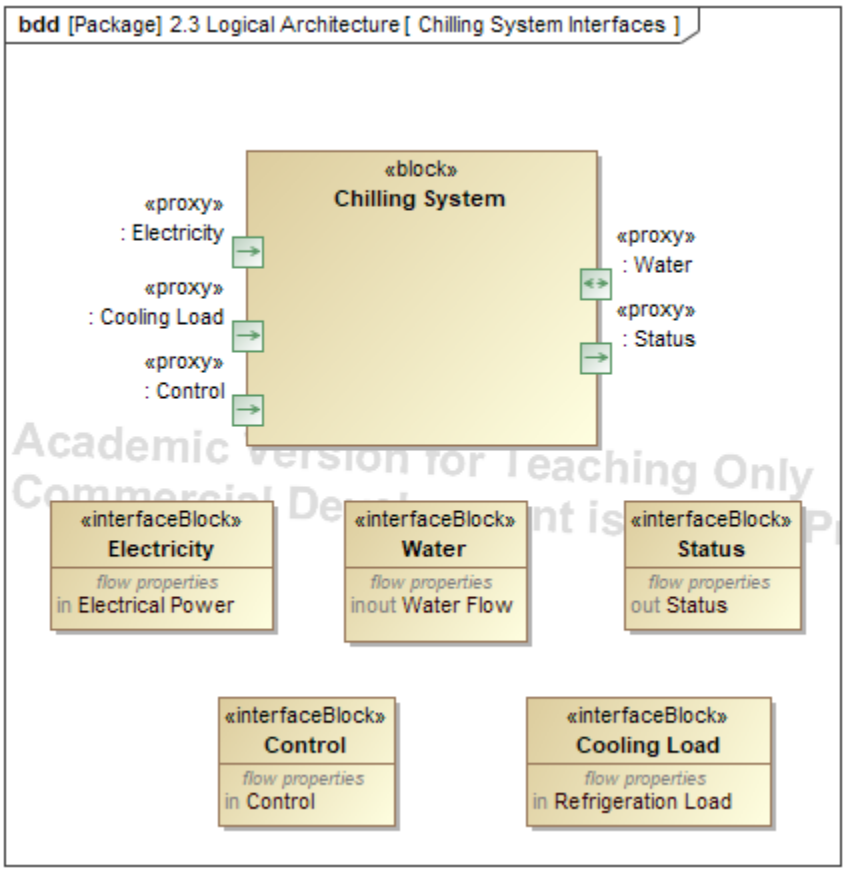


Figure 3.12. Logical Interfaces to the District Chilling System

For instance, electricity coming from outside the SoI is required by all their subsystems, so we connect those proxy ports with connectors. The process is followed by specifying the internal connections between the part properties. This can be represented through the SysML internal block definition diagram. Figure 3.13 shows that the cooling system receives water from the domestic water supply, and the controller system receives control from the operator through the HMI system.

Cell 2.22: Final Functional Analysis and Allocation

The precious activity defined and specified the structural aspect of the system from the white-box perspective. Functional analysis is completed by allocating subsystem functions to logical subsystems. White-box perspective functional analysis can have multiple layers of functional decomposition. Every system function specified in black-box functional analysis

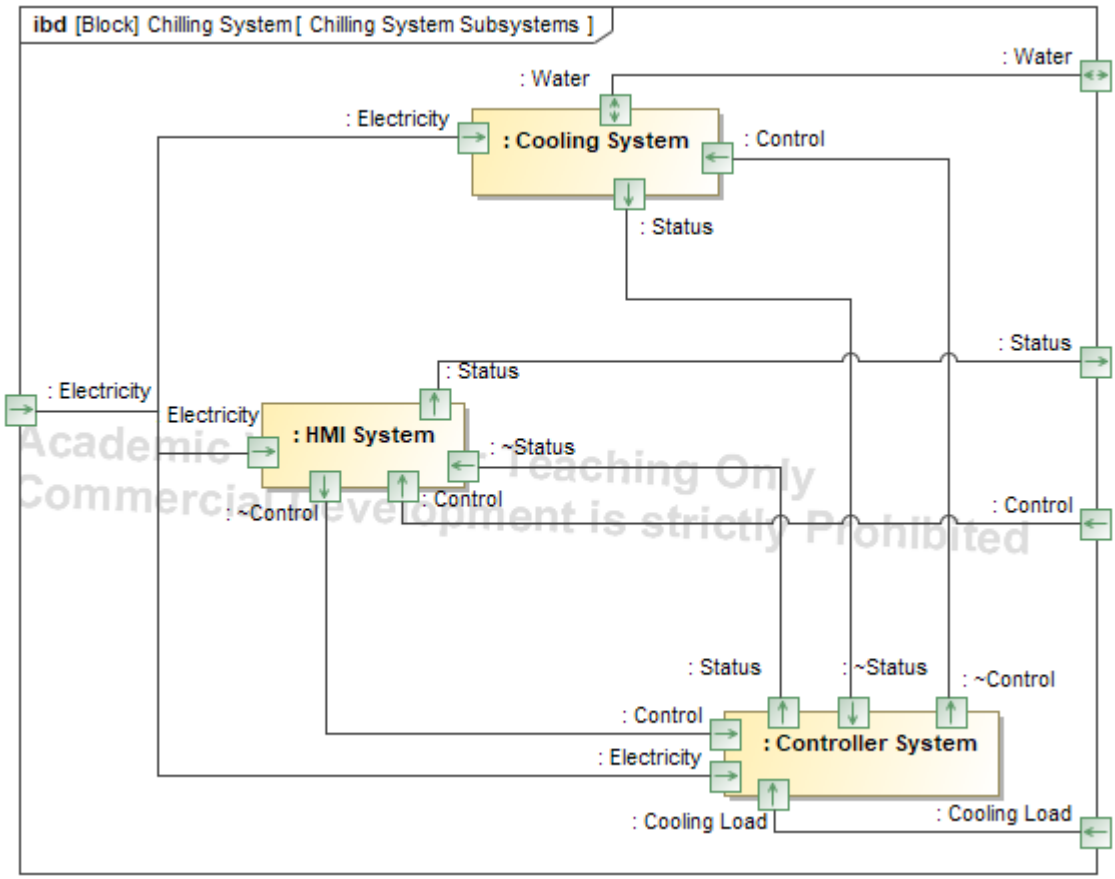


Figure 3.13. Subsystems of District Chilling System and its Interconnections

can be decomposed into subsystem functional analysis. Similarly, every subsystem function can be decomposed into sub-subsystem or component level functions that are allocated to sub-subsystem or components. It should be noted that structural analysis (Cell 2.3 specifying logical subsystems) should be performed prior to each level of depth. We require logical structure to be specified, because we cannot complete the functional analysis without allocating these different tier functions to respective tier logical subsystems. The depth of functional analysis depends on the need for abstraction level. Some organizations can stop at the subsystem functional analysis, and some might go deeper due to the depth required in developing software functions. Figure 3.14 shows tier-one functional decomposition where subsystem functions such as display operational values are allocated to the *HMI logical subsystem*.

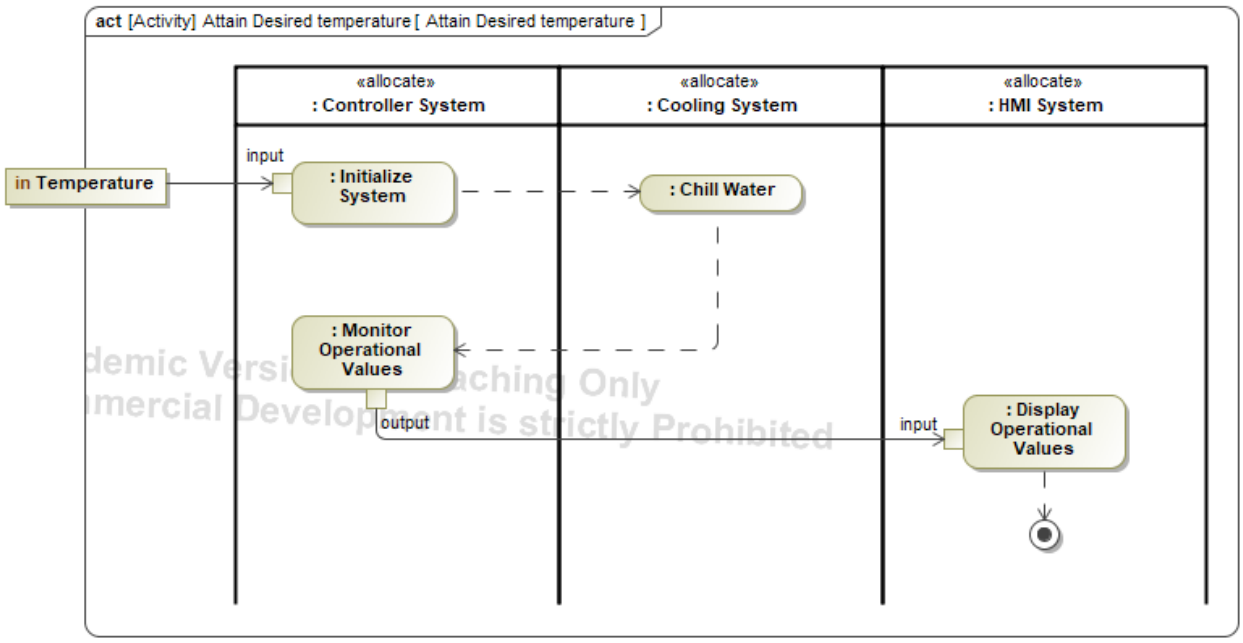


Figure 3.14. White-Box Scenario Functional Allocation to Logical Subsystems

Cell 2.1: Refinement Relations between White-Box System Model Elements and Stakeholder Needs

The elements specified in the white-box analysis are refined from the stakeholder needs. Traceability from among these elements and stakeholder needs is required. Stakeholder needs is raw data provided to initiate the system development process. The black-box elements refine stakeholder needs. The white-box elements are the product of a deeper analysis of the black-box elements. Eventually, a traceability relationship is established that goes from white-box elements to the black-box elements and finally to stakeholder needs. The refine relation is used to display this relationship. The SysML refine requirement matrix is used to display this relationship. The subsystem functions such as *chill water* refine the *supply chilled water* functional stakeholder needs. MoEs elements such as *sound level* value property refine the sound level non-functional stakeholder needs. The logical structures also refine some stakeholder needs. Logical structure such as cooling system refines *supply chilled water* stakeholder need.

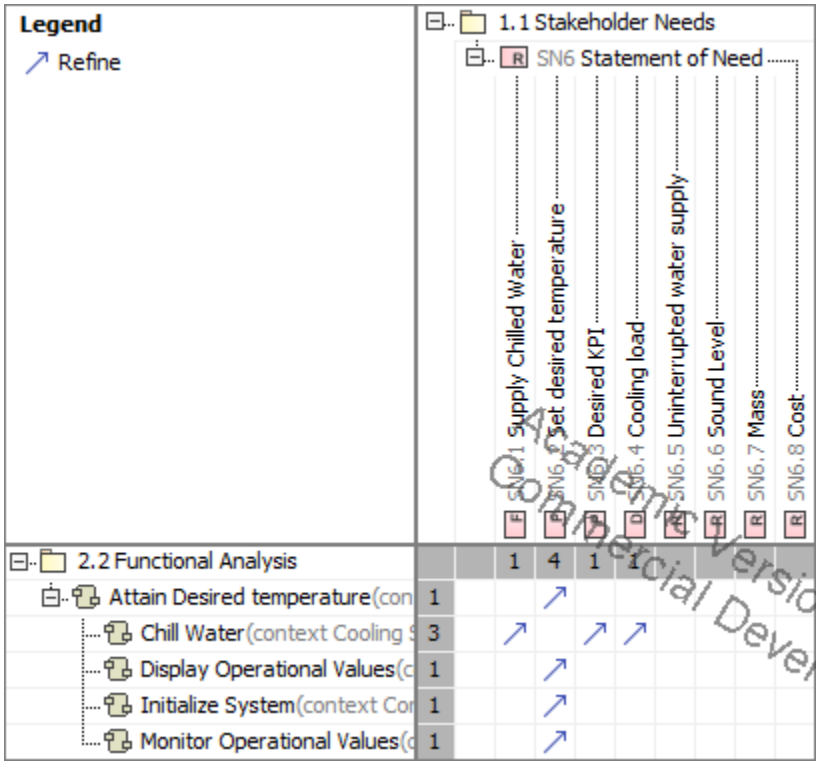


Figure 3.15. Refine Stakeholder Needs by White-Box Elements using Req. Matrix

Cell 2.4: System architecture complexity index- Complete problem definition phase

This activity is the final evaluation and analysis of the system's architectural complexity from the problem domain's white-box perspective. The internal block diagram of each subsystem provides the values of the number of structures and interfaces. The functional decomposition at the white-box level provides values for P and Pd.

Number of systems S = 3,

Number of interfaces I = 4+5+6 = 15,

Number of functions in each use cases P = 15,

Number of functions decomposed for each use case functions $P_d = 52$

When these values are computed in equation 2.5, we get,

$$\hat{C} = ln(\sum_{p=1}^{p} Pd^P) + l_n(\sum_{s=1}^{3} I^S) = 67.3927 \tag{3.2}$$

The structural complexity $\hat{C}_s = 8.1241$, is considerably smaller than behavioral complexity $\hat{C}_b = 59.2686$ of the system architecture.

3.2.3 Solution Domain

Cell 3.11: Preliminary System Requirement

System requirements are the first step to design system architecture derived from stakeholder needs. The elements specified in the problem domain refine these requirements. Further, after designing potential solutions for the system, the subsystem requirements are derived from the system requirements. Before proceeding further to the component level requirements, engineering modeling and analysis are performed at each system hierarchy level. Figure 3.16 shows how system requirements refine stakeholder needs. SN Noise Requirement is specified, and specific requirements are provided for two different instances.

#	Name	△ Text
1	R SN5 Chiller System Requirements	
2	R SN5.1 System Efficiency	The efficiency of the operational system shall not be less than KW/TR.
3	P SN5.2 O & M Cost	The O & M cost of the system shall not be more than.. $/TR
4	F SN5.3 Temperature Control	The operator shall be able to control the pre and post operational conditions
5	R SN5.3.1 Monitoring System	The system shall be able to monitor its operational conditions.
6	D SN5.6 Cooling Load	The system shall be able to produce TR at peak hours.
7	I SN5.8 Display Screen	The system shall have touchscreen display to show setting and receive inputs.
8	D SN5.4 Design Flow Rate	The system shall maintain design flow rate.
9	P SN5.5 Noise Requirement	The system shall not produce noise more than recommended value at day and night tme.
10	P SN5.5.1 Noise Requirement at Night-time	The maximum noise of the system at night-time background shall not exceed 58 dB, time-weighted over an 8 hour period
11	P SN5.5.2 Noise Requirement Difference	The noise of the system shall not exceed existing background noise levels by more than 5 dB, time-weighted over 5 minutes period.
12	Ph SN5.7 Total Mass	The Total Mass of the system shall not exceed 3,000 kgs.

Figure 3.16. System Requirements of District Cooling Systems

Cell 3.12: Traceability to Stakeholder Needs

It is essential to establish a traceability relationship between system requirements and stakeholder needs. System requirements cannot exist isolated with be associate with at least one stakeholder's needs. It can be written that every system requirement is derived from the

stakeholder needs. And all stakeholder needs are refined by one or two system requirements. For instance, of the chilling district system, system requirement *Design Flow Rate* is derived from *SN Supply Chilled Water*. Figure 3.17 shows traceability using the derive requirement matrix, and the relationship is established using SysML «derivReqt» relation.

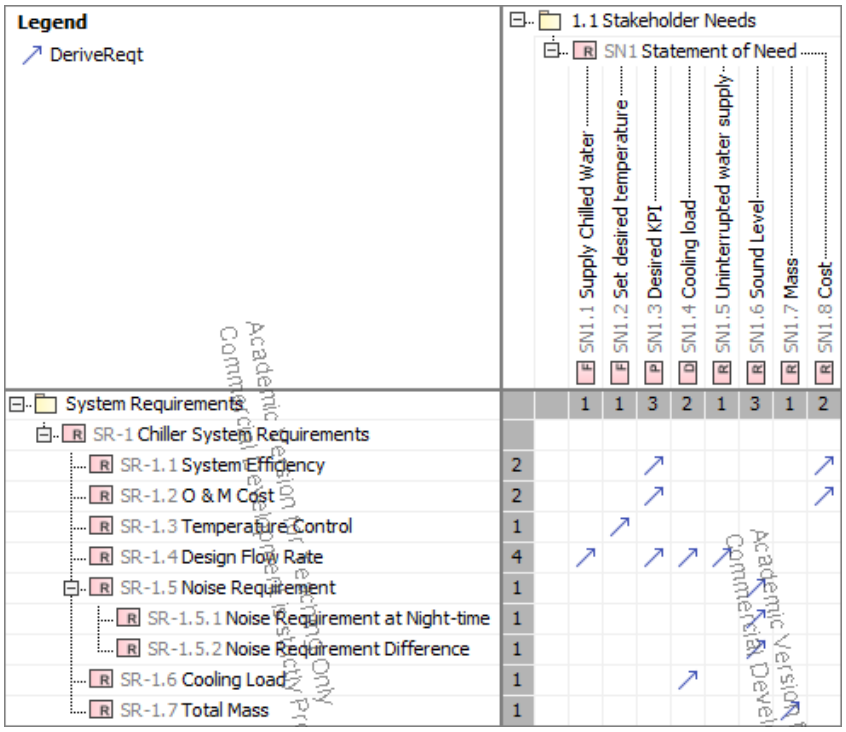


Figure 3.17. Traceability between System Requirement and Stakeholder Needs

The system requirement refinement matrix in Figure 3.18 shows system requirements refine elements identified in the white-box perspective.

Cell 3.31: Preliminary System Structure

After specifying the system requirements, building the system structure is the next step. The structure of high-level solution architecture is defined; the internal parts of the system of interests are specified. Controller, HMI, monitoring, cooling are the four subsystems identified as parts of the district cooling system (the system of interest and system under design). The internal parts specified in these cells are physical parts, and they differ from parts in the logical subsystem cell as they are logical subsystems. The interfaces to these

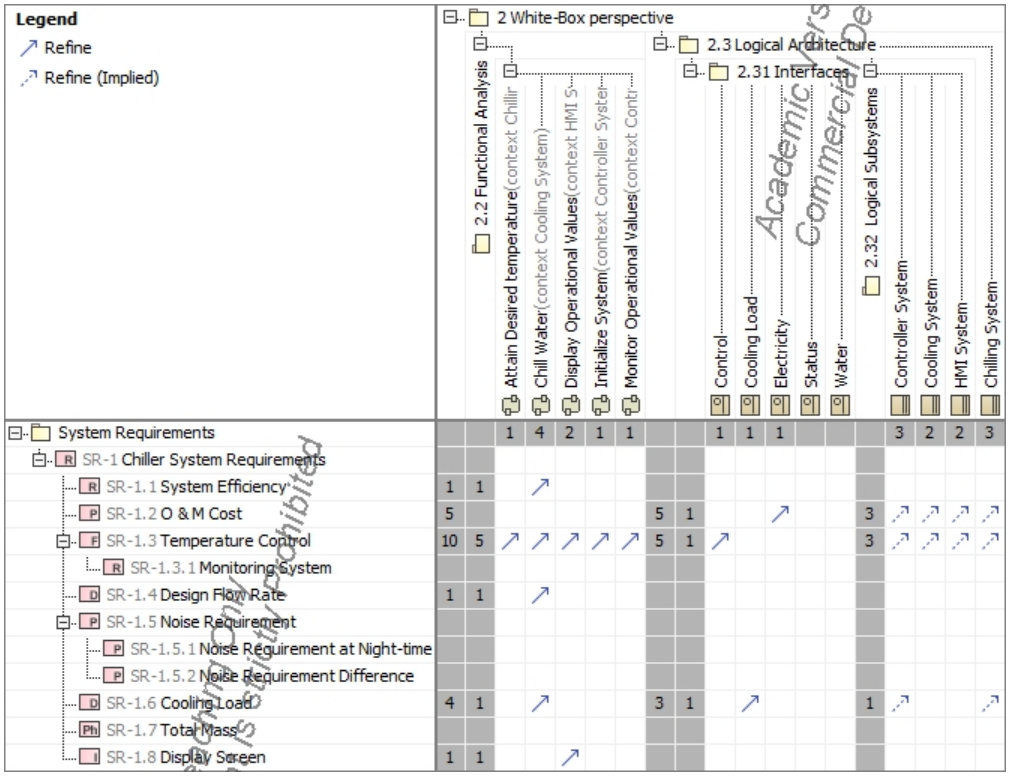


Figure 3.18. System Requirements Refine Elements Identified in White-Box Perspective

subsystems are identified as physical interfaces. Figure 3.20 shows the BDD diagram showing the physical parts and interfaces of the system of interest. The parts are typed as part properties of the district chilling system. The interfaces of the subsystem "cooling system" are typed using the interface block. Interface block is typed with flow property to specifying the type and direction of the flow-through ports. Figure 3.19 shows the abstraction between high-level solution architecture and white-box logical subsystems.

Cell 4.3: Subsystem Structure

After identifying and specifying the subsystem of the system of interest in the previous steps and interfaces to each subsystem, respectively, we can start building the solution architecture for each subsystem. There are four subsystems, but considering the demonstration solution architecture for the cooling system (a subsystem of interest) is first. The cooling structure's internal structure consists of an evaporator, compressor, expansion valve, con-

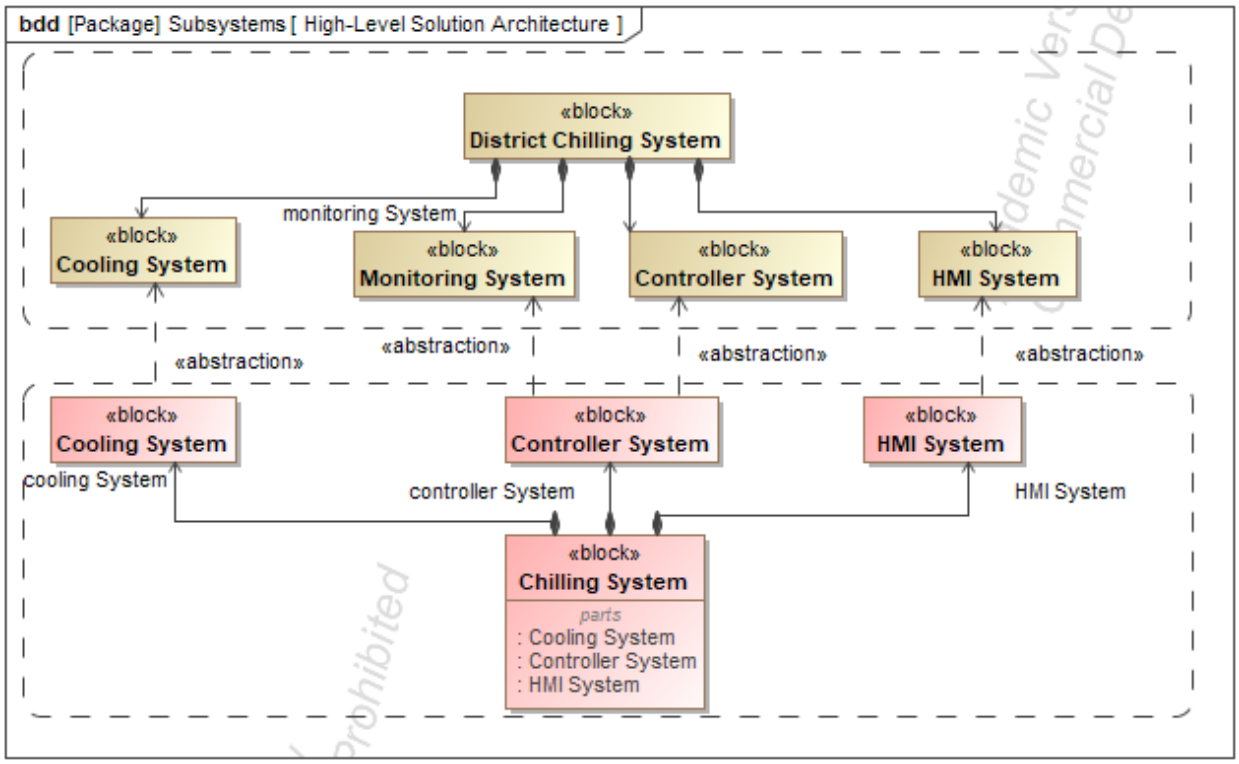


Figure 3.19. High-Level Solution Architecture

denser, motor, pump, cooling tower, and condenser water valve. These parts are represented as blocks and shown in the SysML block definition diagram shown in Figure 3.21.

Once the subsystem's internal structure is specified, then the interfaces of the parts of the subsystem are specified. SysML internal block diagram shown in Figure 3.22 is used to represent the interfaces and connections between the subsystem components.

Cell 4.1: Subsystem Requirements

The system comprises of subsystems; therefore, subsystems need to be specified. Designing subsystems is dependent on the derivation of the requirements for these subsystems from the system requirements. Subsystem requirements for each subsystem are derived. These requirements refine the system requirements and elements identified in row 3 and stored in the requirement diagram. The traceability relationship between the subsystem requirements and other elements is represented using refine requirement, refine matrix, and derive requirement matrix.

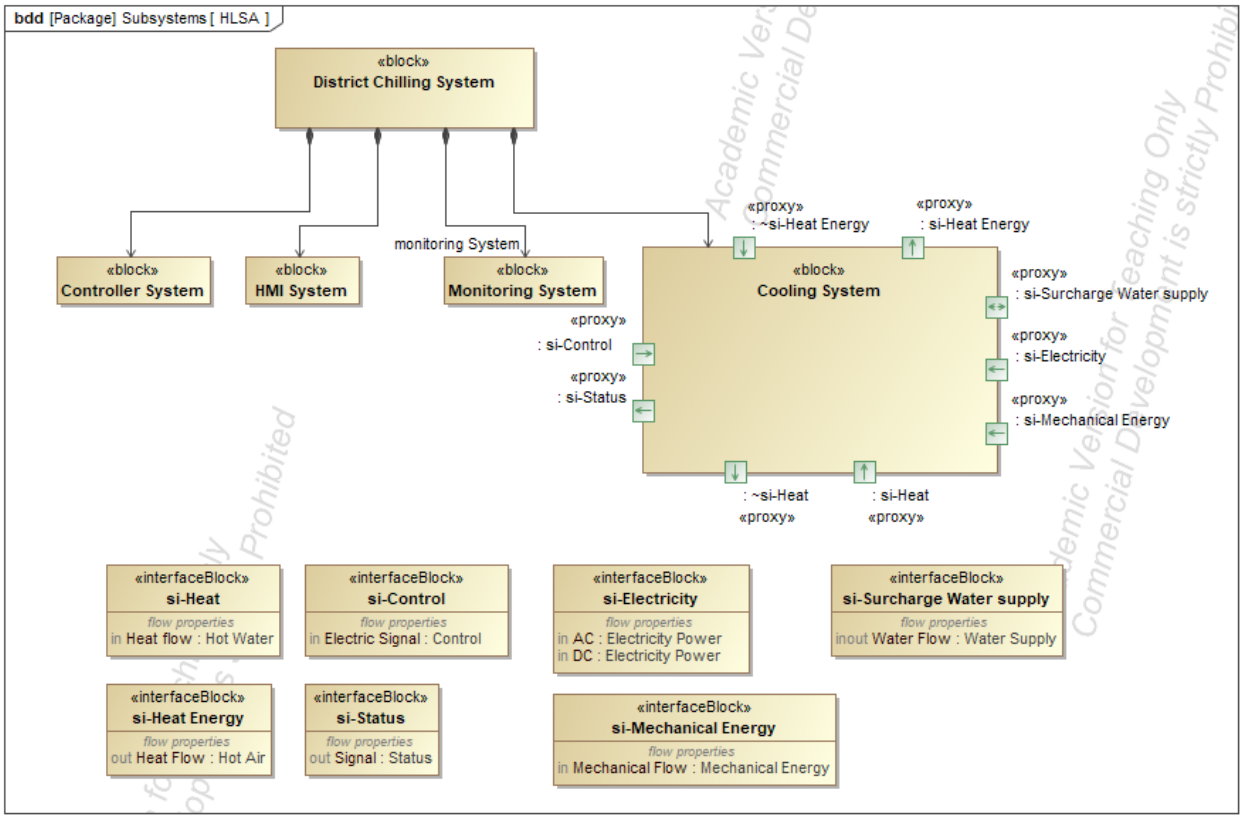


Figure 3.20. Physical Parts of the District Chilling System and Interfaces Defined Identified for the Subsystems

Cell 4.2: Subsystem Behavior

System behavior is an emergent behavior that arises after integrating behaviors of all subsystems. Therefore, subsystem behavior is modeled first and then integrated into system behavior. Every subsystem has a state that changes over time, and this state change is represented using the state machine diagram created for every subsystem. Figure 3.23 below shows the state machine diagram for the cooling system.

Cell 3.2: System Behavior

After defining all subsystem's behavior in the previous activity, creating the system behavior by integrating all subsystem behaviors follows. The activity in this cell also provides us information on how subsystems interact with each other. The activities from Cell 3.31 to

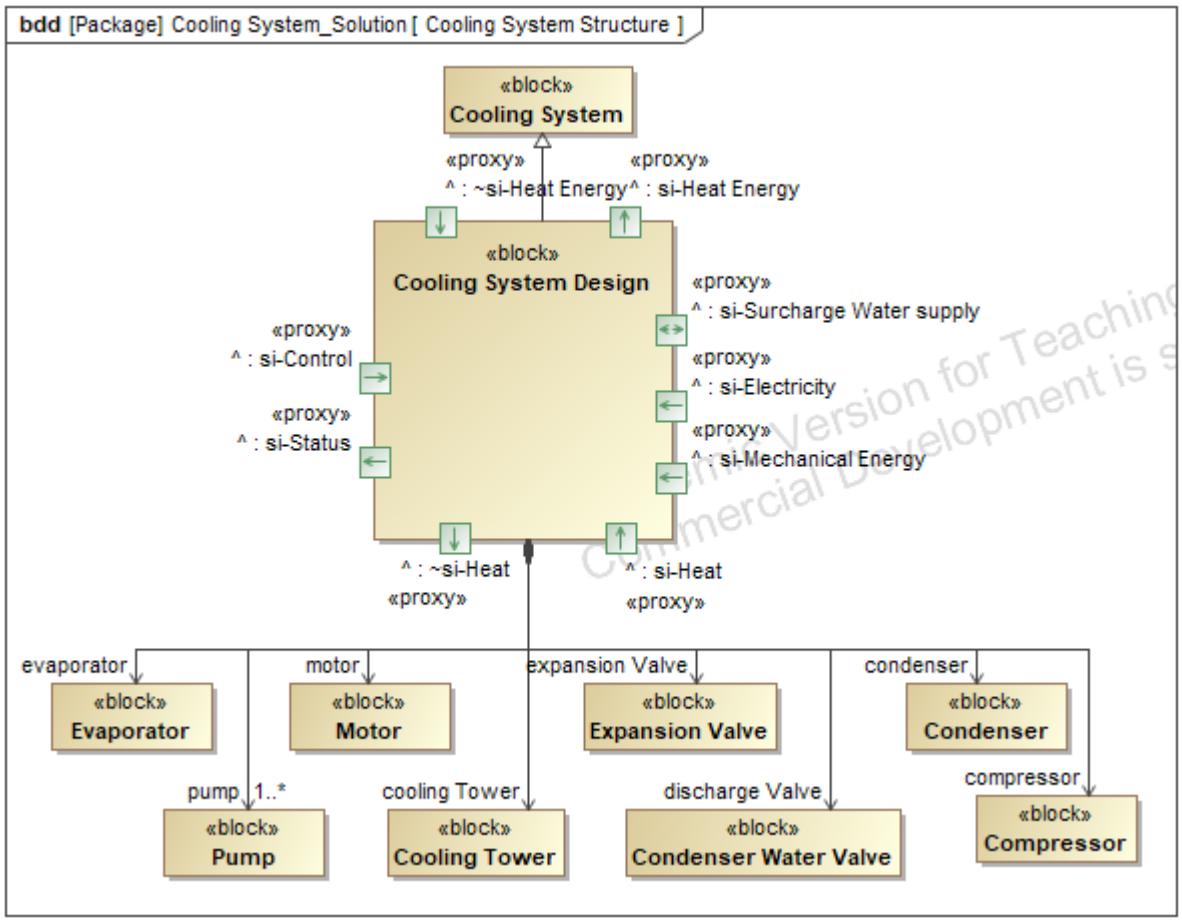


Figure 3.21. Block Definition Diagram of the Physical Structural Decomposition of Subsystem of the SoI

Cell 3.2 are repeated, iteratively, multiple times for each subsystem. Therefore, it provides multiple solution architects for each subsystem, thereby giving us multiple system behavior for the complete system.

Cell 3.3: Final System Structure

The previous activity provides multiple system behavior that integrates multiple subsystem solutions resulting in a tradeoff analysis to choose a subsystem solution. Then, the system is integrated, and the interfaces between these subsystems are checked. The subsystem interfaces are verified by checking if the cooling system can communicate with the

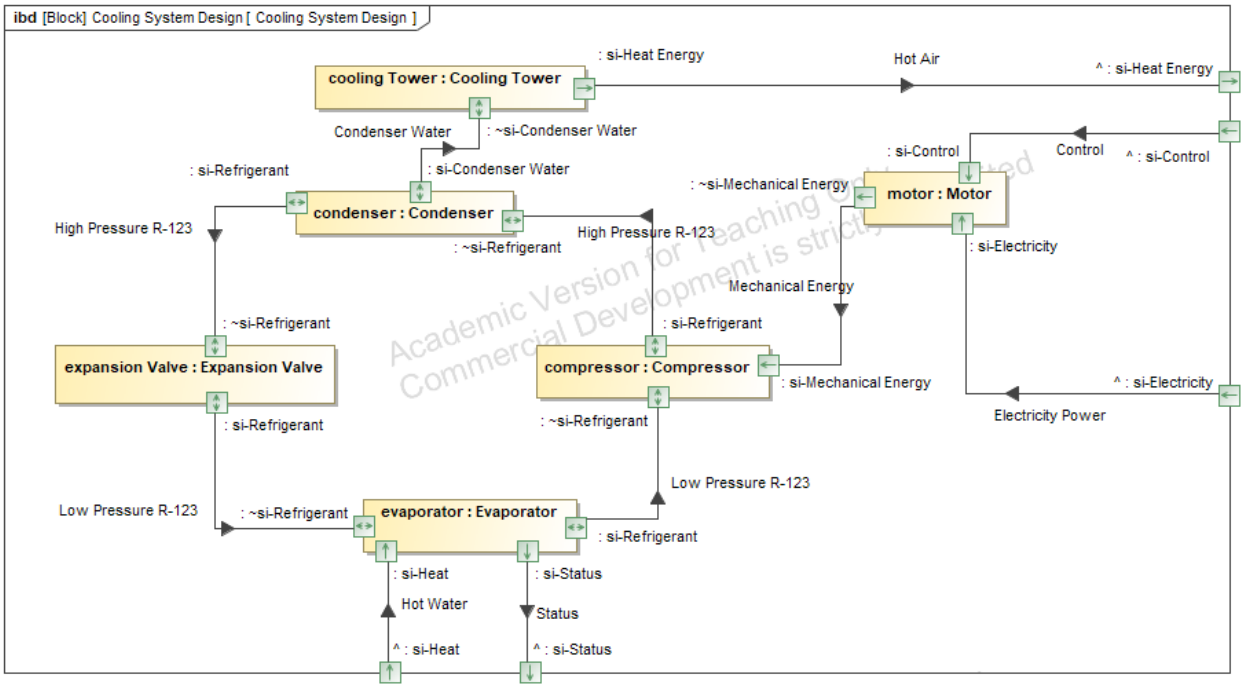


Figure 3.22. Internal Block Diagram Showing Interfaces and Connections between Parts of Subsystem

external entities that interact with the district chilling system, and the interfaces between all subsystems are checked before finalizing the system structure.

Cell 3.13: Requirement Satisfy Relationship

After having the solution architecture for the system and subsystem, it is crucial to represent the relationship between the structure and requirements at the system and subsystem level. The subsystem components perform functions to satisfy system requirements, and component meets subsystem requirements. This relationship is represented in the SysML satisfy requirement matrix with requirement satisfy relationship. In this fashion, traceability from component level function to stakeholder needs can be established. Every subsystem should at least meet one system requirement.

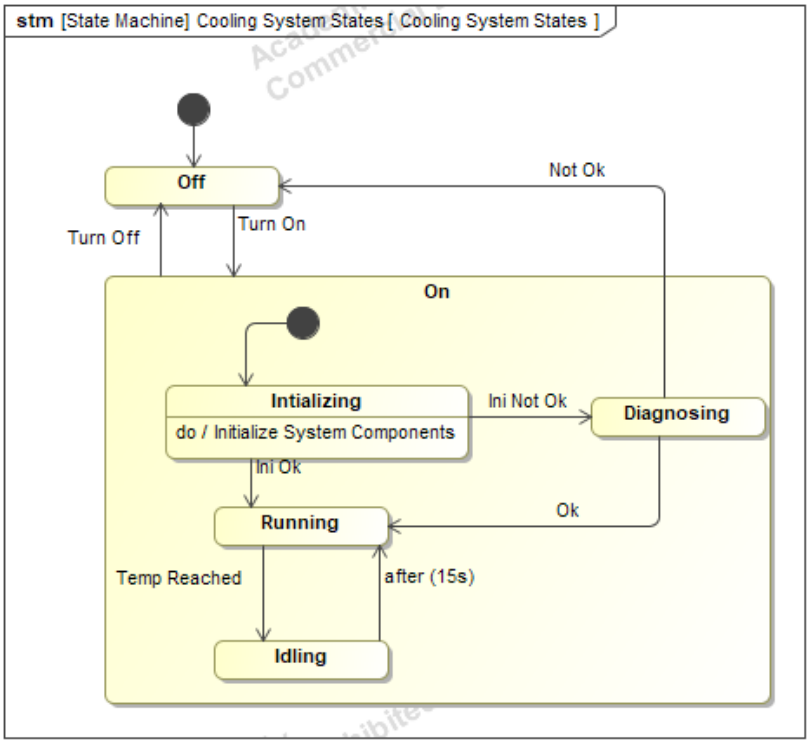


Figure 3.23. State Machine Diagram Representing the State Transitions

3.2.4 Engineering Analysis

Cell 5.2: Dynamic System Modeling & Simulation

The activities performed till now specify requirements, structure, behavior, and complexity index for the system, subsystems, and components. Creating the prototype is done to implement the solutions identified and performed after defining subsystem behavior or establishing an integrated system behavior. The mathematical model created depicts the system/subsystem behavior. And this behavior is simulated in any suitable simulation environment. In this thesis, Modelica language-based Dymola software replicates the system/subsystem's behavior. All subsystems are modeled and simulated based on the solution architecture at the subsystem level. After testing all subsystems, the subsystems are assembled and simulated, which provides a complete behavioral view of the system. A brief description of the modeling and simulation of the subsystem and system in Dymola is provided in section 3.3.

Cell 5.1: System Requirement Verification

Simulation results of subsystem/system models performed in the previous activity verify the system requirements.

3.3 Dynamic Modeling and Simulation in Modelica

3.3.1 Model Organization

The dynamic model is organized in the form of packages, as shown in Figure 3.24. The *District_Cooling_System* package has three main packages. *Components* package for storing all parametrized components needed for the district cooling system. It includes components like a chiller, cooling tower, pipe, pump, etc. Assembly package has different district cooling system configurations, and *Test_Library* package has test models of different components and assemblies.

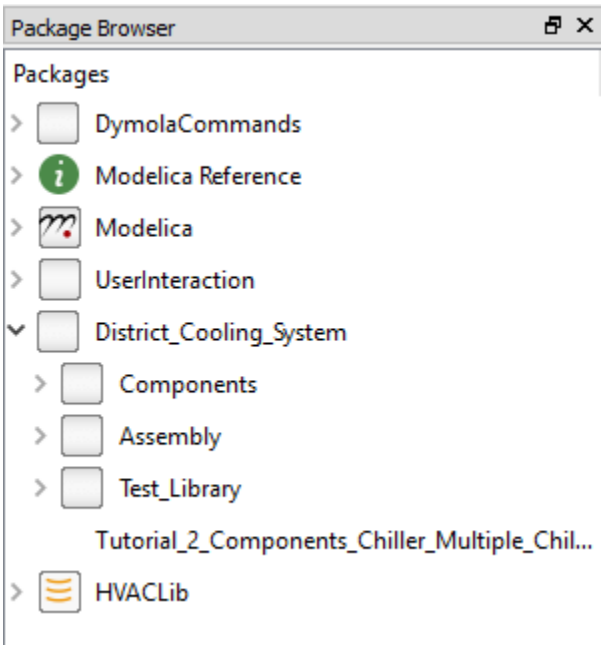


Figure 3.24. Dymola Model Organization in the Package Browser

The purpose of this organization is to bring synchronization between components used in assemblies and tests. The components in the components package are extended from the *HVACLib*. For instance, Figure 3.25 below shows that the *Chiller_Actual* component is

extended from *HVACLib.Components.Cooler.ChillerVaporComp_table* (Vapor compression chiller based on interpolation of manufacturer data), which uses *HVACLib* and Standard Modelica libraries. These components are used in assembly and test models. The parametrized components are synchronously updated in the assemblies and test models, thus reducing errors. The components library acts as the building block.

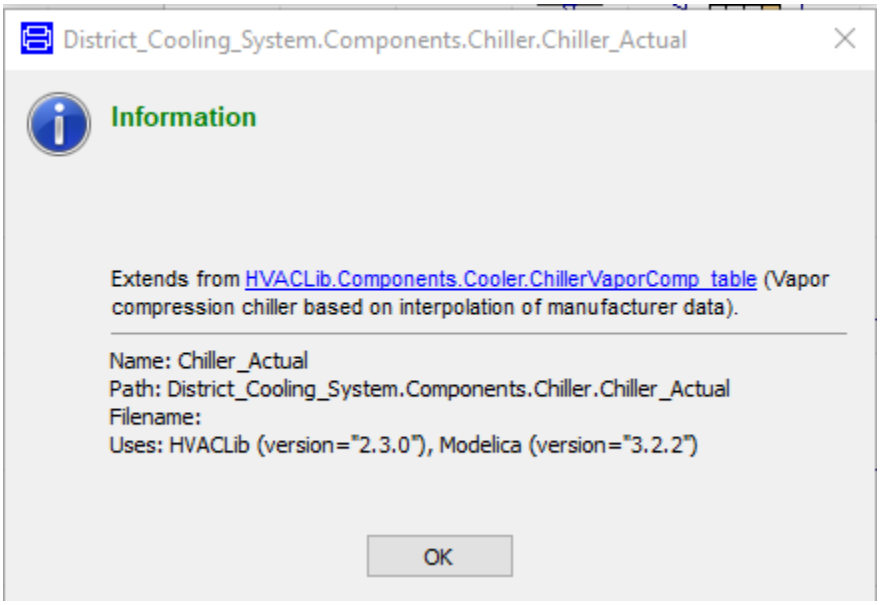


Figure 3.25. Information of the Chiller_Actual Component in the Component Package

3.3.2 Component modeling

Pipe

Model Description: The pipe model used for this case study has a mass transportation and heat transfer behavior. This model component describes the mass balance as stationary and is computed by equation 3.3 [59] given below.

$$\frac{dm_{in}}{dt} - \frac{dm_{out}}{dt} = 0 \tag{3.3}$$

Whereas equation 3.4 [59] below shows that the energy balance computes the change of the inner energy considering the convectional energy transport and potential inner heat sources and energy inputs via the heat ports.

$$M_{\mathrm{i}} c_p \frac{dT}{dt} = \dot{H} - \dot{H}_{\mathrm{i+1}} + \dot{Q}_{source} + \dot{Q}_{heatports} \tag{3.4}$$

The pipe model is extended from *HVACLib.Components.Pipes.NPipe*. To test the model pipe for regular operation, we need to provide the source and sink for the liquid to the pipe. The pipe also loses heat; hence a heat sink is connected to the pipe. There is a steady increase in the mass flow rate for 60 seconds provided by the block shown in Figure 3.26.

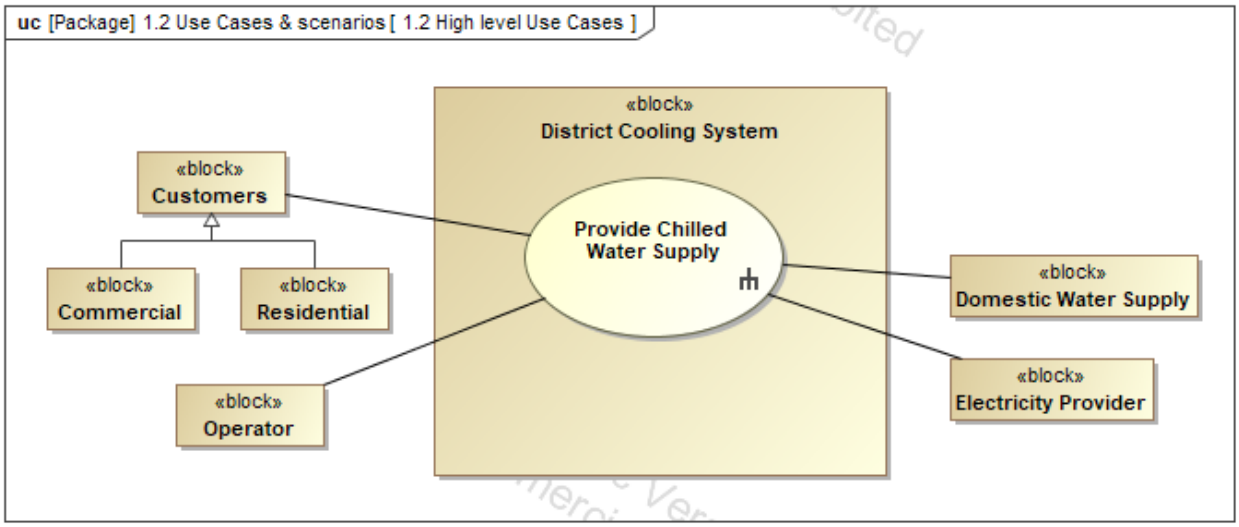


Figure 3.26. Dymola Layout to test_pipe Model

Model setup: The mass flow rate is set up to increase from 0.1kg/s to 7kg/s in 60 seconds steadily. The medium flowing through the pipe is liquid water taken from standard Modelica library, *(Modelica.Media.Water.ConstantPropertyLiquidWater)*.

Simulation Setup: Simulation time to test the model is 100 seconds with a 1-second interval. The dassl (differential/algebraic system solver) is an efficient solver and used since it

Table 3.1. Pipe parameters

Type	Names	Values	Description
Geometry			
Volume	V[n]	d-hyd^2/4*$Modelica.Constants$	Volume [m^3]
Integer	Name	1	Discretization
Length	L	0.5	Length of Pipe [m]
Length	delta_x[n]	ones(n)∗L/n	Length of pipe [m]
Diameter	d-hyd	0.025	Hydraulic diameter [m]
Pressure	p[n]	ones(n)∗le5	Pressure Pa
Heat Flow Rate	Q_flow_s[n]	fill(0,n)	Heat Source [W]
Temperature	T-start	293.15	Temperature [K]

Table 3.2. Pipe Connector Notations

Type	Names
FlowPort_in	flowPort_in
FlowPort_out	flowPort_out
HeatPort	heatPort[n]

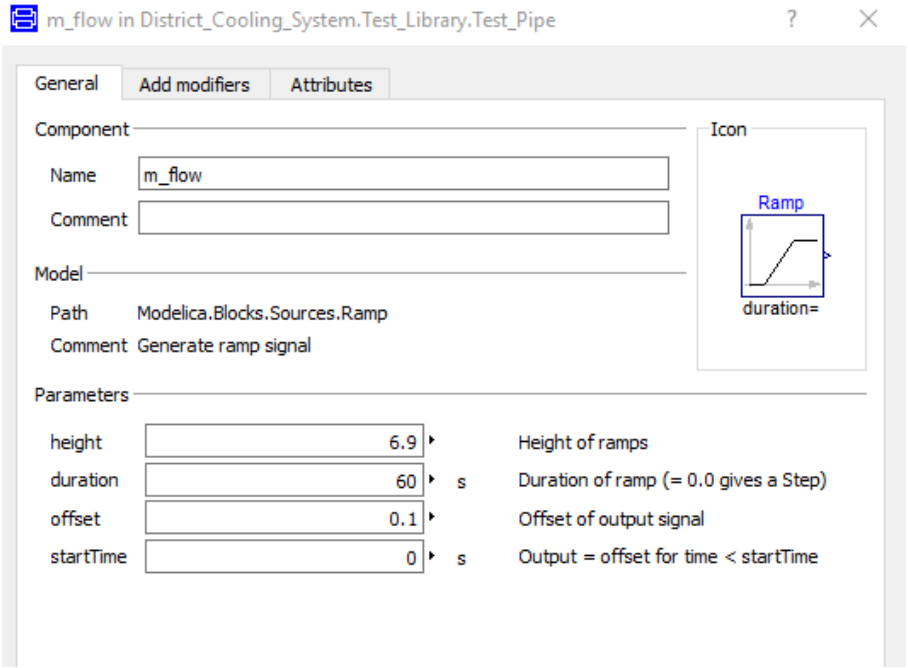


Figure 3.27. Mass Flow Rate Profile Setup

is an implicit, higher-order, multi-step solver with a step-size control. As such, it is relatively stable for a wide range of models [60]. This solver is based on a backward differential formula and has a mature source code. The simulation setup window is shown in Figure 3.28.

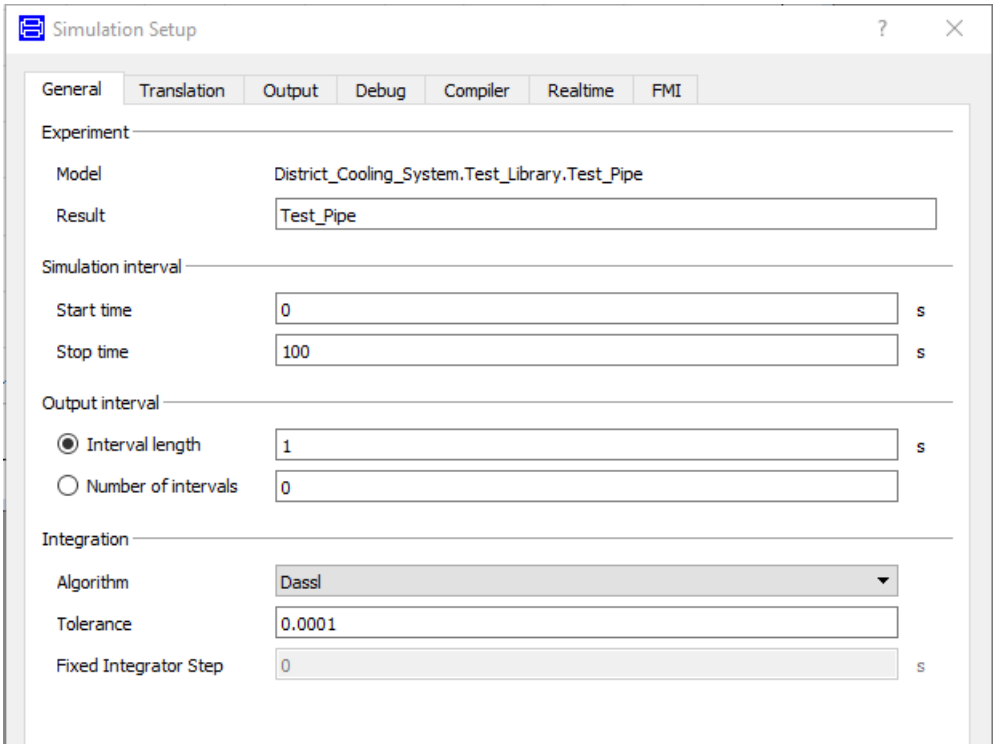


Figure 3.28. Simulation Setup for the Pipe Model.

Simulation result: The test results in Figure 3.29 show that the pipe's mass flow rate is output is identical to the source mass flow rate. Note that the pipe's mass flow rate is negative because it is leaving the pipe.

Pump

A simple pump test model is used for testing. This model is extended from HVAC library whose classified name is written as *HVACLib.Components.PumpsAndBlower.Pump*. The pump model generates a constant mass flow rate based on the specified parameter or a variable mass flow if there is the use of mass flow rate input. The pump's electrical power

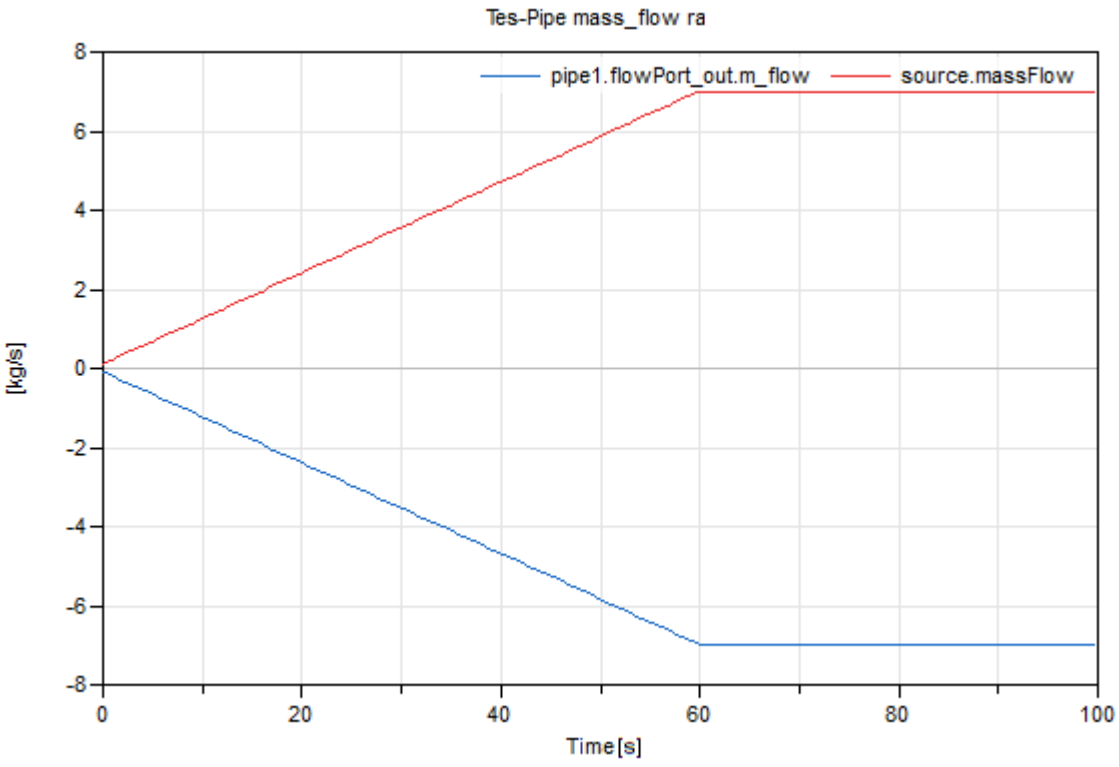


Figure 3.29. Mass Flow Rate in the Pipe

$\mathbf{P_{el}}$ consumption is equal to the nominal power consumption [Watt] for a constant mass flow rate [kg/s] is shown in equation 3.5 [59],

$$P_{el} = P_{el,nom} \tag{3.5}$$

For variable mass flow rate the P_{el} is given as below in equation 3.6 [59],

$$P_{\text{el}} = \left(\frac{\dot{m}}{\dot{m}_{nom}}\right)^{f_{el}} P_{el,nom} \tag{3.6}$$

f_{el} the efficiency correction factor for part-load operation is taken as 0.8.

The energy balance of the pump is given below in equation 3.7 [59],

$$M_{\text{i}} c_p \frac{dT}{dt} = \dot{H}_{\text{in}} - \dot{H}_{out} + \dot{Q}_s \tag{3.7}$$

Here, Q_s is the heat loss rate. During initialization, if the heat transfer dissipation parameter is turned true, heat flow from the pump to fluid is considered zero.

The pump model is attached to a variable flow rate source, and hence it produces a variable flow rate at the sink. The test pump's simulation results show that the pump's power consumption increases proportionally for 60 seconds. The power consumption becomes constant after 60 seconds as the mass flow rate becomes constant.

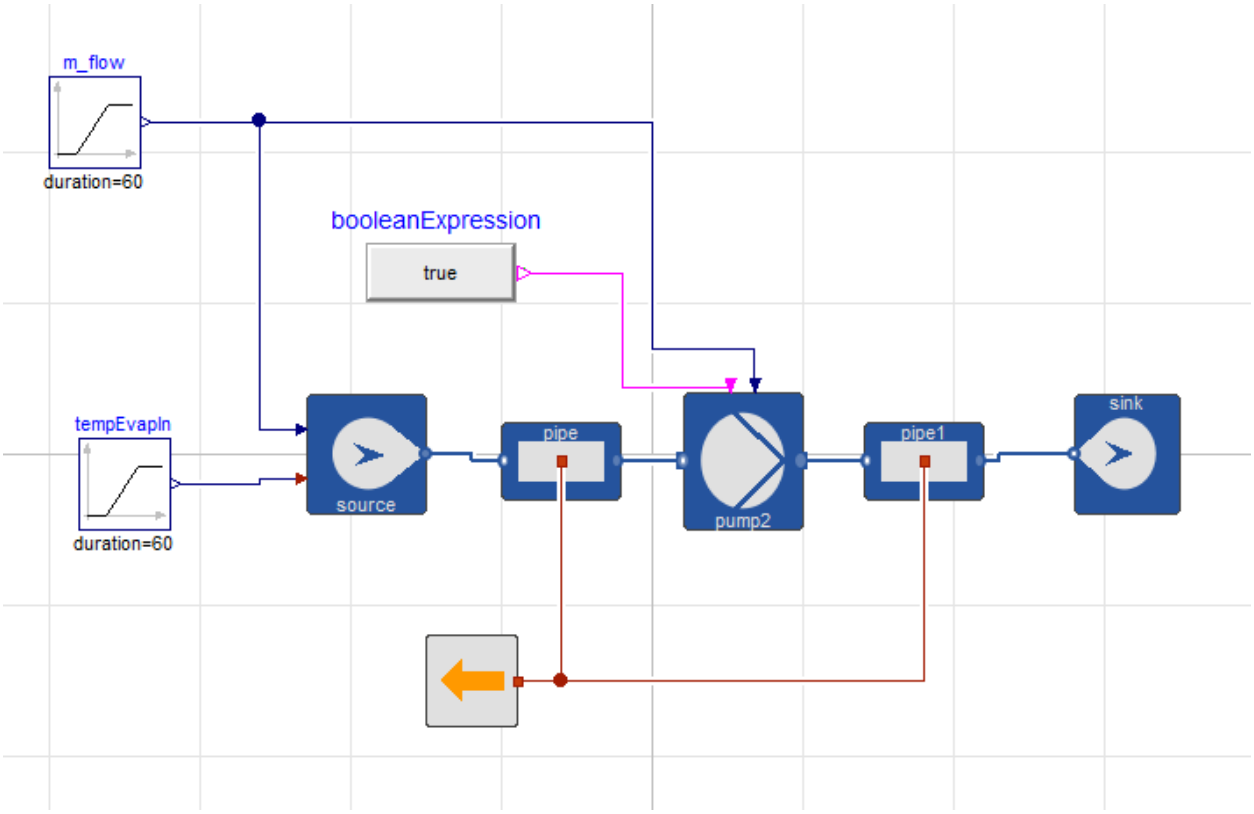


Figure 3.30. Dymola Layout of the Pump Model

Cooling Tower

The cooling tower chosen for this application is dry cooler as it uses environmental air (cross-flow) to extract heat from the condenser fluid. The blower controls the airflow, and the m_flow_nom parameter defines the nominal mass flow rate of air. Parameter

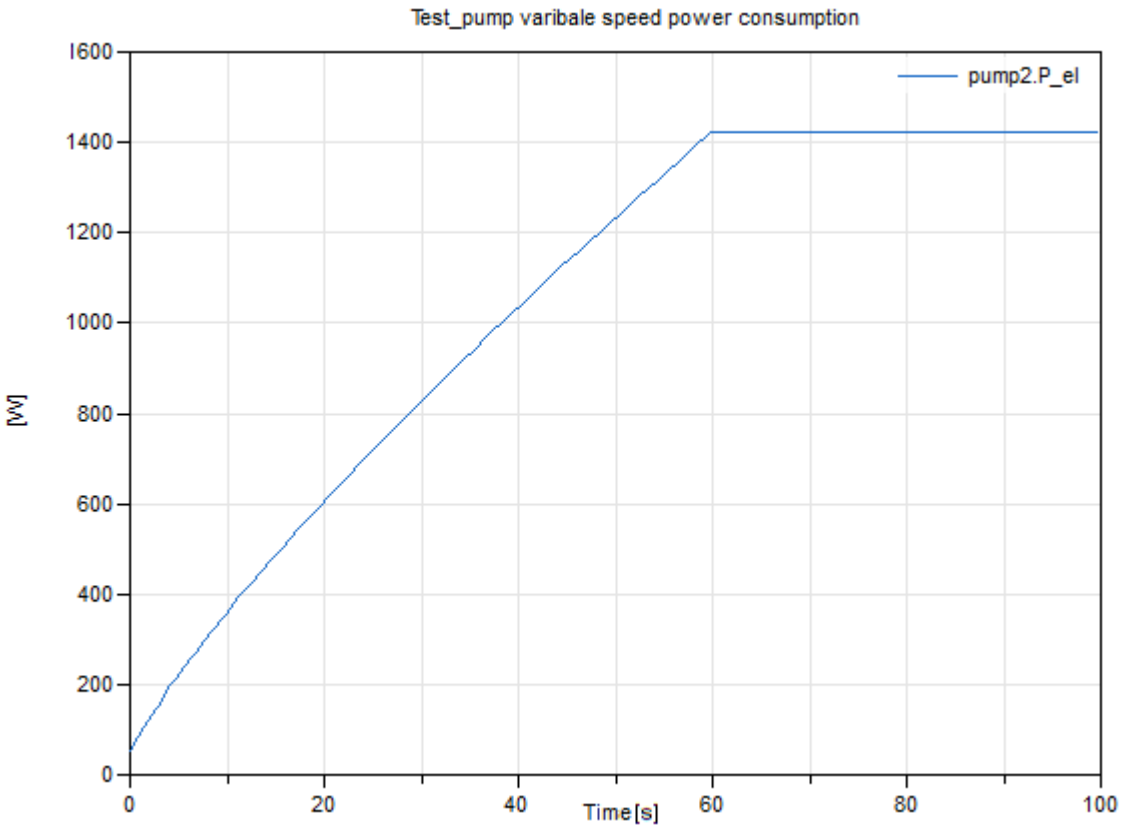


Figure 3.31. Power Consumption of the Pump Model with Variable Flow Rate for first 60 Seconds

m_flow_off is used when the blower is turned off and is solely dependent on the natural convection. Equation 3.8 [59] below shows the heat transfer flow.

$$\dot{Q} = \in C_{min} \left(T_{in,h} - T_{in,c}\right) \tag{3.8}$$

Here, $(T_{in,h} - T_{in,c})$ is the temperature difference between inlet hot condenser fluid and inlet cold atmospheric air.

This model's heat transfer is modeled using the epsilon-NTU (Number of transfer units) as it is the most appropriate approach for cross-flow heat exchangers.

Here, epsilon is shown below in equation 3.9 [59],

$$\in = \frac{c_{max}}{c_{min}} \left(1 - exp\left(-\frac{c_{min}}{c_{max}}\right)\left(1 - exp(-NTU)\right)\right) \tag{3.9}$$

Where, NTU is calculated using,

$$NTU = \frac{AU}{cmin} \tag{3.10}$$

Here A is the heat transfer area, and U is the overall heat transfer coefficient.

Equations 3.11 [59] and 3.12 [59] below show C_{min} and C_{max} are the heat capacity range, which are calculated using mass flow rate and specific heat $c_{p,h}$ and $c_{p,c}$ for the hot and cold fluids, respectively.

$$C_{max} = max\left(\dot{m}_h c_{p,h}, \dot{m}_c c_{p,c}\right) \tag{3.11}$$

$$C_{min} = min\left(\dot{m}_h c_{p,h}, \dot{m}_c c_{p,c}\right) \tag{3.12}$$

Figure 3.32 shows the Dymola layout for testing of the cooling tower. The condenser fluid mass flow rate is kept variable, and the airflow rate at a constant 15 kg/s. Simulation results in Figure 3.33 show the temperature profile of the mixing fluids.

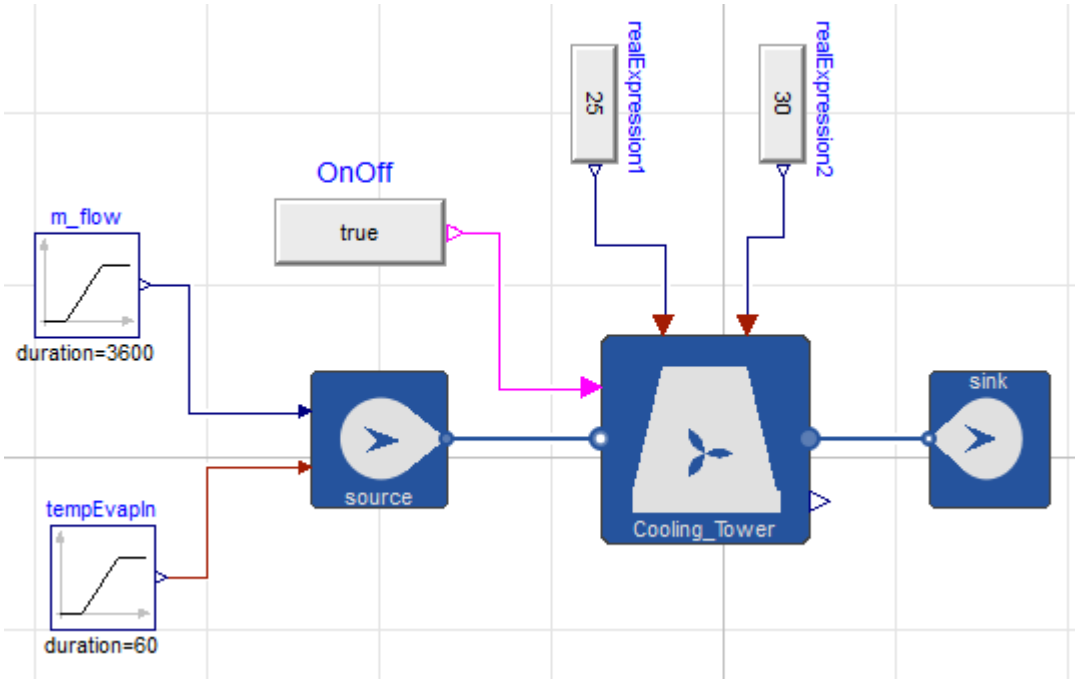


Figure 3.32. Dymola Layout of Cooling Tower Test Model

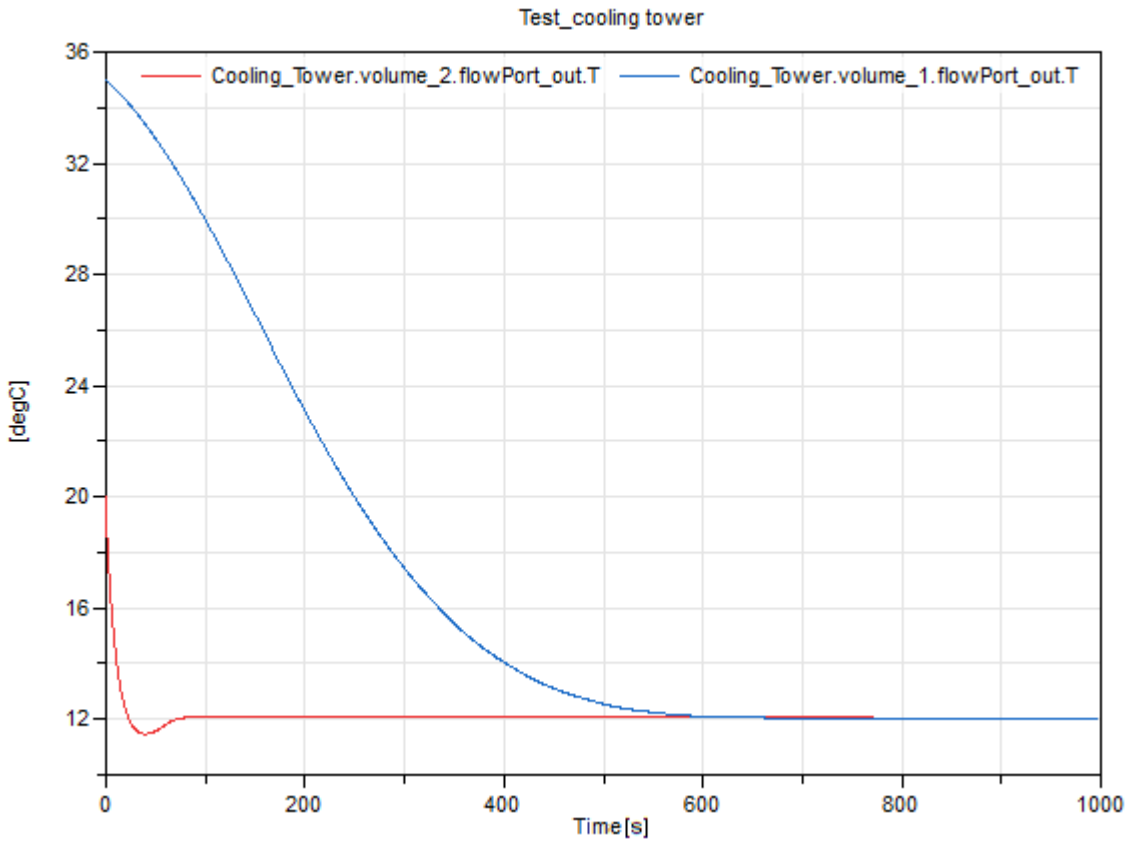


Figure 3.33. Water Temperature Profile for Cooling Tower Model

Chiller

The vapor compression chiller model is based on the interpolation of the chiller manufacturer's data. The mode of the model can be used as a chiller or a heat pump. Equation 3.13 [59] below shows that the cooling power $\dot{Q}_{cool}$ (heat entering chiller at evaporator) of the chiller is proportional to the difference between evaporator inlet temperature and target temperature.

$$\dot{Q}_{cool} = \dot{m}_{ev}\dot{c}_p \left(T_{ev,in} - T_{target}\right) \tag{3.13}$$

Heat $\dot{Q}_{heat}$ leaving the chiller at the condenser is equal to the sum of heat entered in the chiller at the evaporator and heat added to the compressor's refrigerant.

$$\dot{Q}_{heat} = \dot{Q}_{cool} + f_p P_{el} \tag{3.14}$$

f_p, is the fraction of electrical compressor power added to the refrigerant. The Dymola layout shown in Figure 3.34 is a test model of a chiller which will be used for multi-chiller assembly.

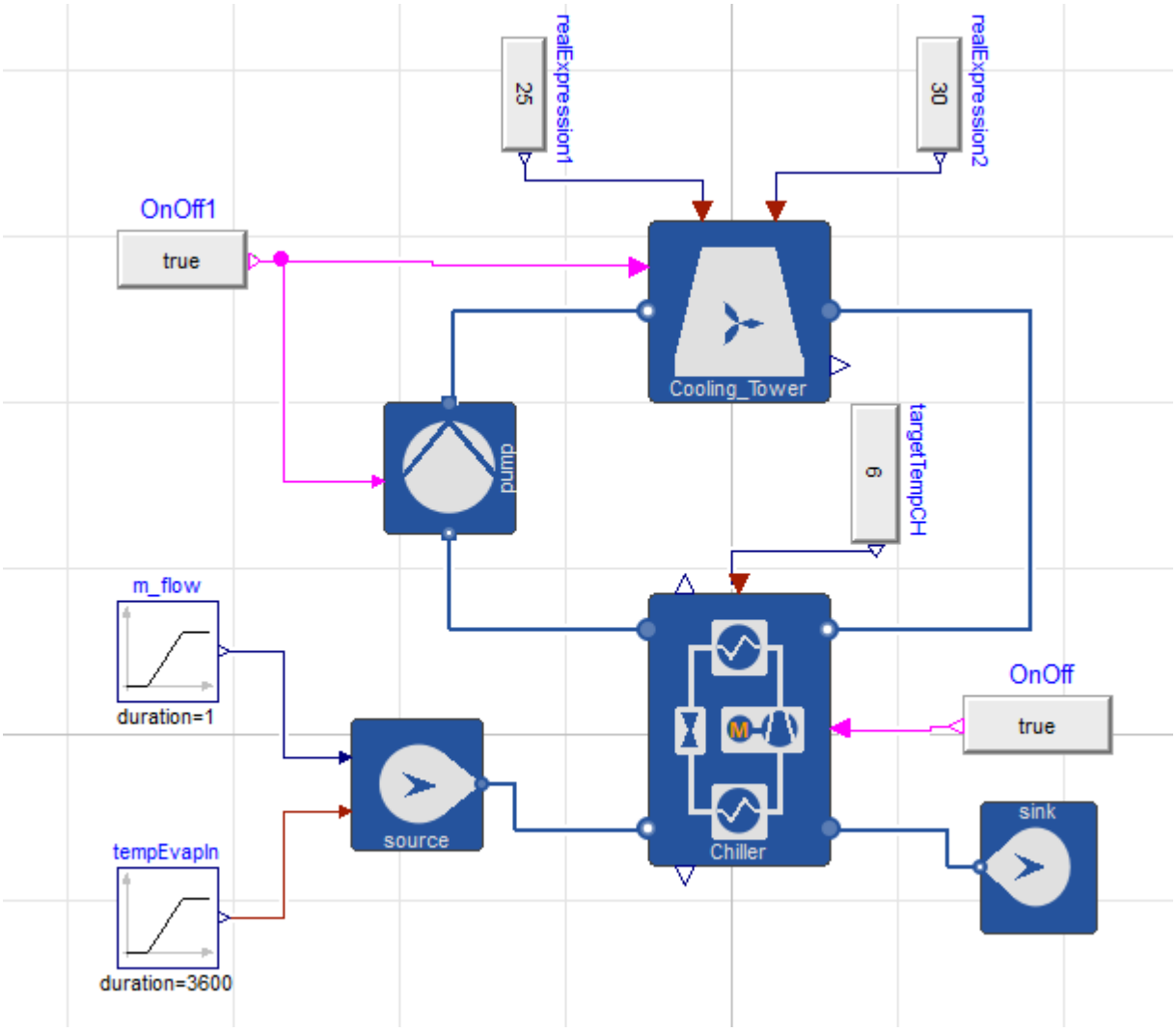


Figure 3.34. Dymola Layout of Cooling Tower Test Model

3.3.3 Modeling District Cool System

The district cooling system is the system of interest when dynamic behavior is considered for modeling and simulation. The component-oriented, physical-interaction modeling methodology is used because, at every instance, one component's behavior is interrelated with the action of other components. This method uses physical conversation laws for this bidirectional interaction between components. The mathematical equation used to build the

components of the system determines the system's behavior. Modeling in simulation tools is based on basic constructs such as components, ports, and links. Components contain the structure and behavior of the system element. The ports act as a gateway for the inlet/outlet of the information/physical flow into the components, and links allow information flow between the components. All these constructs come together and map the structure/behavior of the actual system.

The generic chiller model was developed in Modelica with Dymola using HVACLib (heating, ventilation, and air-conditioning) library. Using a predefined library provides a platform for an optimum trade-off between simulation time and accuracy. The chiller plant model has dynamic models of the water-cooled centrifugal chiller, pumps, cooling towers, etc. The plant has fourteen chillers, which are arranged in a parallel configuration with seven chiller modules as shown in Figure 3.35.

Each chiller module consists of two cooling towers and two chillers connected in parallel. We have also studied the chiller module, which has two chillers connected in series. The plant consists of three loops. The driving loop (refrigerant loop) produces the cooling effect, and the power of the centrifugal compressor controls it. The other loop is the chilled water loop, which carries chilled water to the buildings. The third loop is the condenser water loop used to extract the thermal energy from the refrigerant loop and dissipate into the atmosphere through a cooling tower.

model, and every chiller is controlled through a Boolean input to turn it ON and OFF. The cooling water is circulated using a constant flow pump. Target temperature T_{target} is the desired evaporator outlet temperature, and the range of T_{target} can be from 4.4 to 8℃. The chiller's cooling power is calculated using the target temperature, as shown in equation 3.15 [59] below.

$$Q_{cool} = m_{ev} c_p \left(T_{ev,in} - T_{target}\right) \tag{3.15}$$

For the COP calculation, the Dymola model is simulated at different part-load conditions as per the AHRI standard 551/591-2018 [61]. The water chiller ratings are at 100%, 75%, 50%, and 25% load relative to the full load for part-load, as per the AHRI standard 551-591 (SI)-

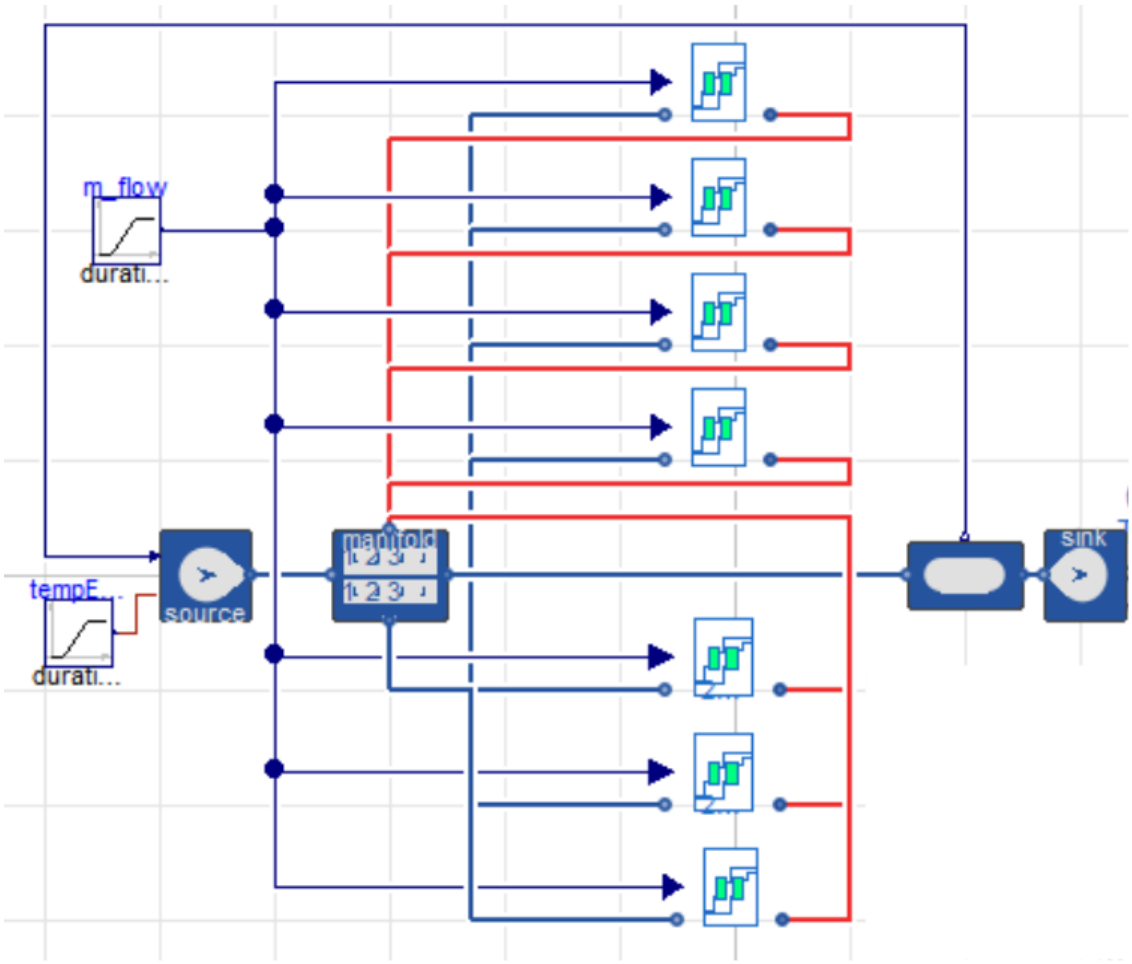


Figure 3.35. Dymola Layout of Cooling Tower Test Model

2018, IPLV.SI (integrated part-load value) is the COP of the chiller that can be calculated using the equation stated in equation 3.16 below.

$$IPLV.SI = 0.01A + 0.42B + 0.45C + 0.12D \tag{3.16}$$

IPLV.SI is calculated in terms of COP. Here, A, B, C, D are the COP calculated by simulating the model at 100%, 75%, 50%, and 25% of the full load, respectively.

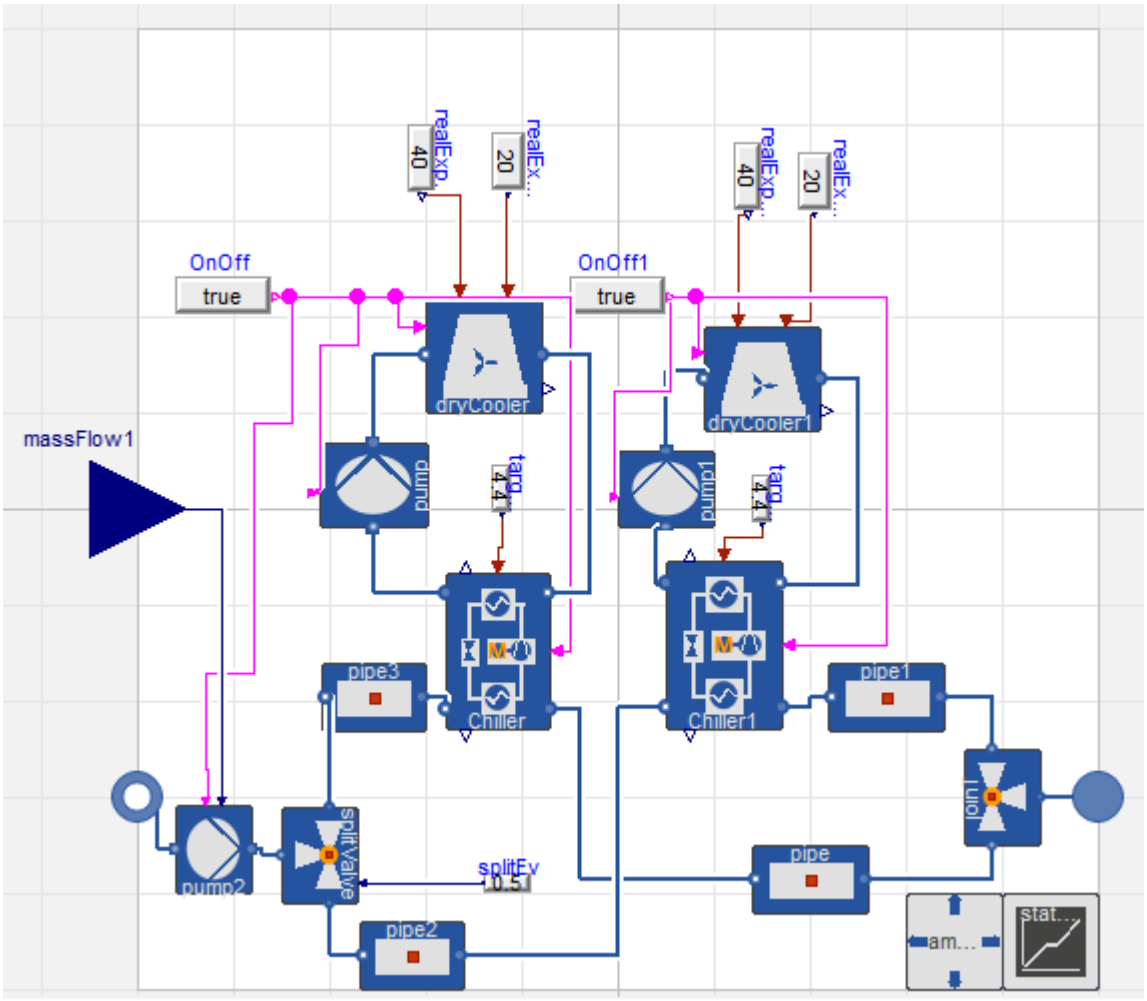


Figure 3.36. Modelica Model of the Chiller Module (2 Chillers in Parallel) in Dymola

3.4 SysML-Modelica-FMI Integration

At this stage, the system architecture model in Cameo Enterprise Architecture and the dynamic model in Dymola is ready. The simulation results can be used to verify the system requirements in Cameo Enterprise Architecture. For this, the variable and the behavioral equations in Dymola are imported in Cameo Enterprise Architecture. Functional Mock-up Unit (FMU) is used for model exchange using FMI standards. The FMU for all component test models such as $Test\_Vapor\_Comp\_Chiller, Test\_Pipe, Test\_Cooling\_Tower$, and $Test\_Pump$ was generated. The FMU for the Multiple Chiller plant (Dymola model of District Cooling System) was generated. Figure 3.37 shows the FMU export window specifying

the version and type of FMU file to be generated. For translating the FMU, the simulation tab in Dymola was used. Model exchange and Co-simulation using Cvode mode are used.

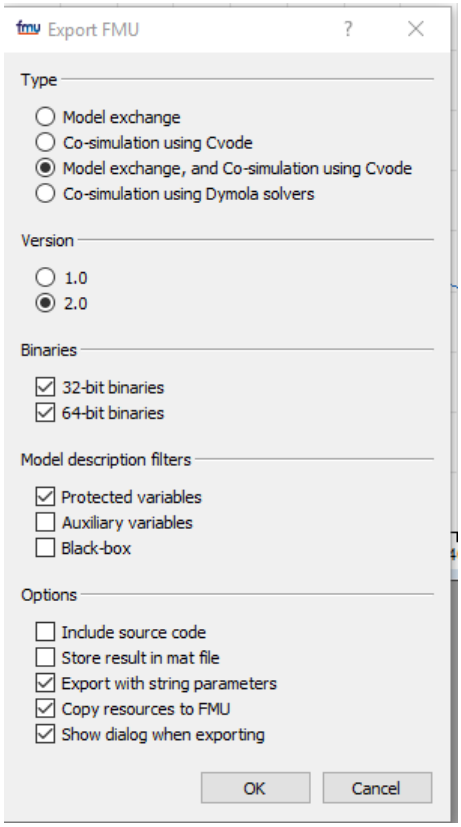


Figure 3.37. Export FMU Setting Window

A new simulation configuration diagram is created, and the FMU file is dragged and dropped in the Package diagram shown below. The FMU file that was drag and dropped in the simulation environment was typed with «fmu» stereotype. The execution target in the Simulation Configuration was set to *District_Cooling_System.Test_Library.Test_Vapour_Comp_Chiller* block by dragging the «fmu» block onto simulation configuration element. The simulation configuration element was simulated by setting the timing properties in the specification of the simulation configuration setting window.

4. RESULTS AND DISCUSSIONS

4.1 SysML - Modelica - FMI Integration

One of the most critical gaps in this thesis was integrating engineering analysis and system architecture modeling. Although some work integrates those aspects of system design, the level of depth achieved in each field collectively is significantly less. This section targets the results achieved in integrating Cells 5.2 and 5.1 to other Cells of the framework. The solution architecture developed in the cameo enterprise architecture derived the development of the dynamic model in Dymola. The Dymola model of chiller created is based on the requirements and specifications specified in the solution architecture. After the simulation, the chiller model produced the profile of chilled water temperature, which is specified by variable name $Chiller.flowPortCold\_out.T$ is shown in Figure 4.1.

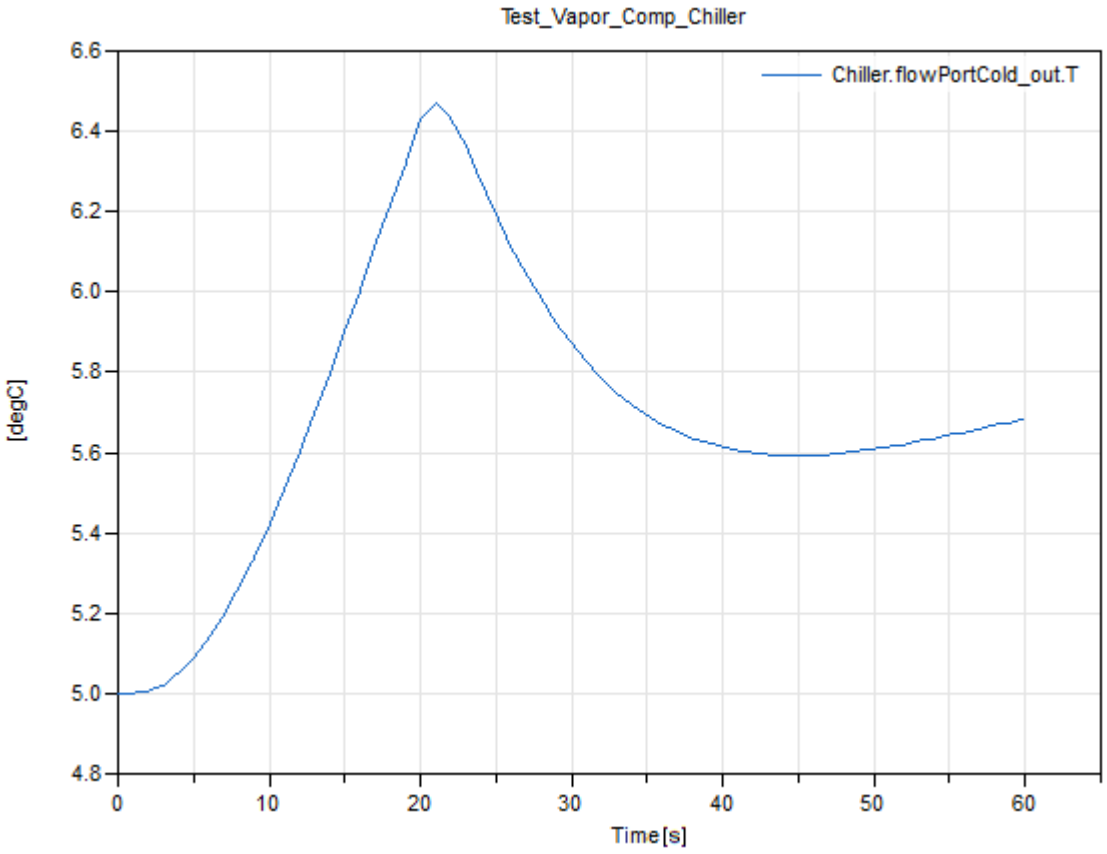


Figure 4.1. Chilled Water Temperature Achieved by Vapor Compression Chiller Model Simulated in Dymola Environment

The FMU for the chiller model was generated and imported in the chiller design verification package in Cameo Enterprise Architecture. The imported «fmu» block of the chiller model is simulated with the same initial conditions as simulated in Dymola. Value property for parameters and variables are created during the simulation. The graph below shows the $Chiller.flowPortCold\_out.T$ variable generated after simulating «fmu» block in Cameo Enterprise Architecture. Figure 4.2 shows the profile of chilled water temperature when simulated in the Cameo Enterprise Architecture. Both of the profile indicates that the model in both environments was consistent and produce the same results.

The system requirement chilled water temperature states that the "Chiller temperature shall not exceed 6.6 degrees Celsius" this requirement is verified by the value property associated with chiller «fmu» block. This relationship is shown in the SysML verify requirement matrix with «verify» relation from value property to the system requirement.

Similarly, various results are obtained for other subsystems.

4.2 Chilled Water Temperature of all Different Modules

As mentioned earlier, the district cooling plant consists of 7 chiller modules. Each module's inlet temperature increases from 10 degrees Celsius to 12 degrees Celsius within 3600 seconds time span. Target chilled water temperature is 4.4 degrees Celsius. Figure 4.4 shows the chilled water temperature profile across all chiller modules connected in a parallel configuration.

4.3 Test Chiller Model Results

The dynamic Modelica model of chiller developed in Dymola provides output chilled water temperature and IPLV.SI. The input parameters for the simulation are uploaded from the sample weather report and the hourly loading condition. The target temperature (EWT) is set to 4.4 degrees Celsius, and the inlet water temperature to the module is variable. It increases from 10 degrees Celsius to 12 degrees Celsius within 3600 seconds time span. The model is simulated for 1000 seconds to reach a steady-state. Simulation results in Figure 4.5 show the temperature profile and COP of a chiller.

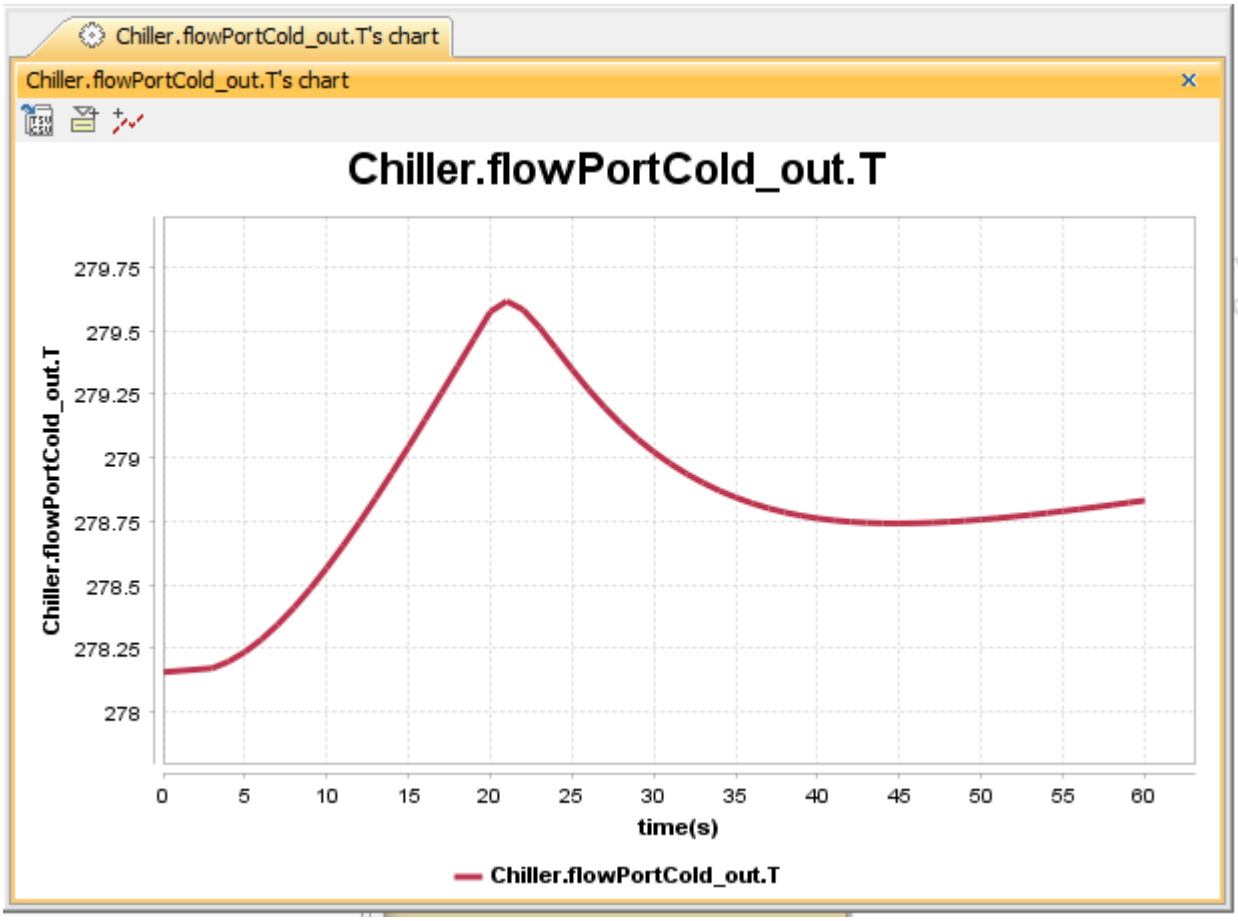


Figure 4.2. Chilled Water Temperature Achieved by Vapor Compression Chiller Model Simulated in Cameo Enterprise Architecture Environment

4.4 Energy Consumption v/s Mass Flow Rate

The energy costs are computed by the consumption of the different energy carriers and the corresponding prices. For the electricity consumption and generation, two prices have to be defined: the consumption of electricity provided by the public grid and one for the fed in electricity into the public grid. The mass flow rate of fluid increased from 7kg/s to 14kg/s; the chiller module's power consumption does not increase significantly. Hence, the recommendation is to increase the mass flow rate to satisfy the increase in cooling load.

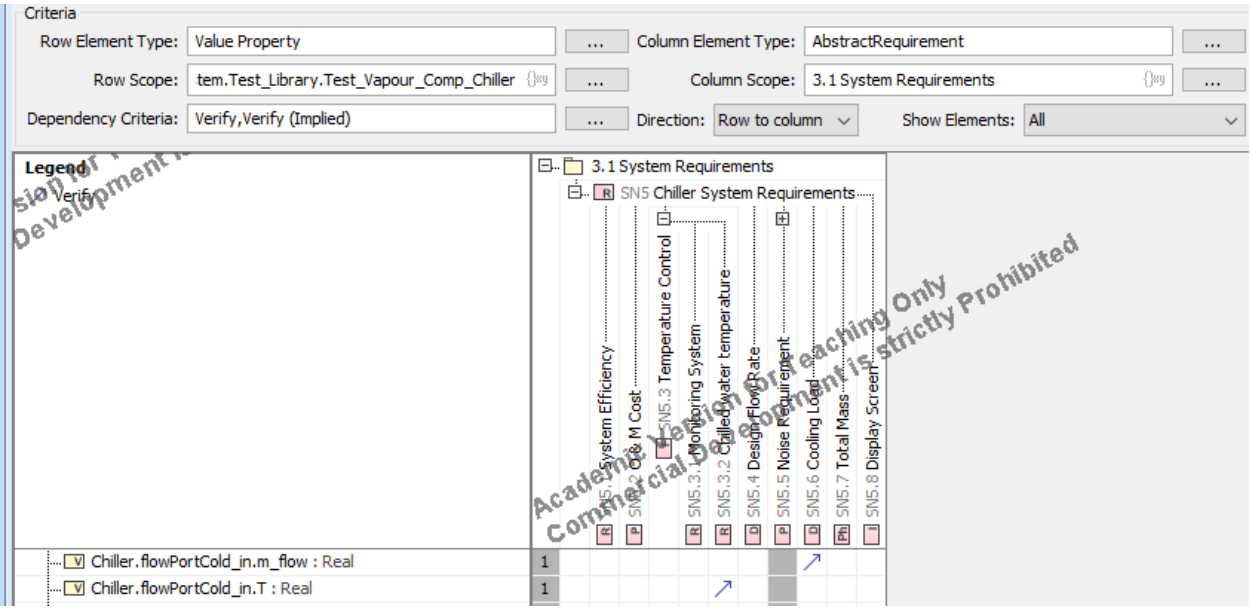


Figure 4.3. Verify Requirement Matrix Showing Value Property

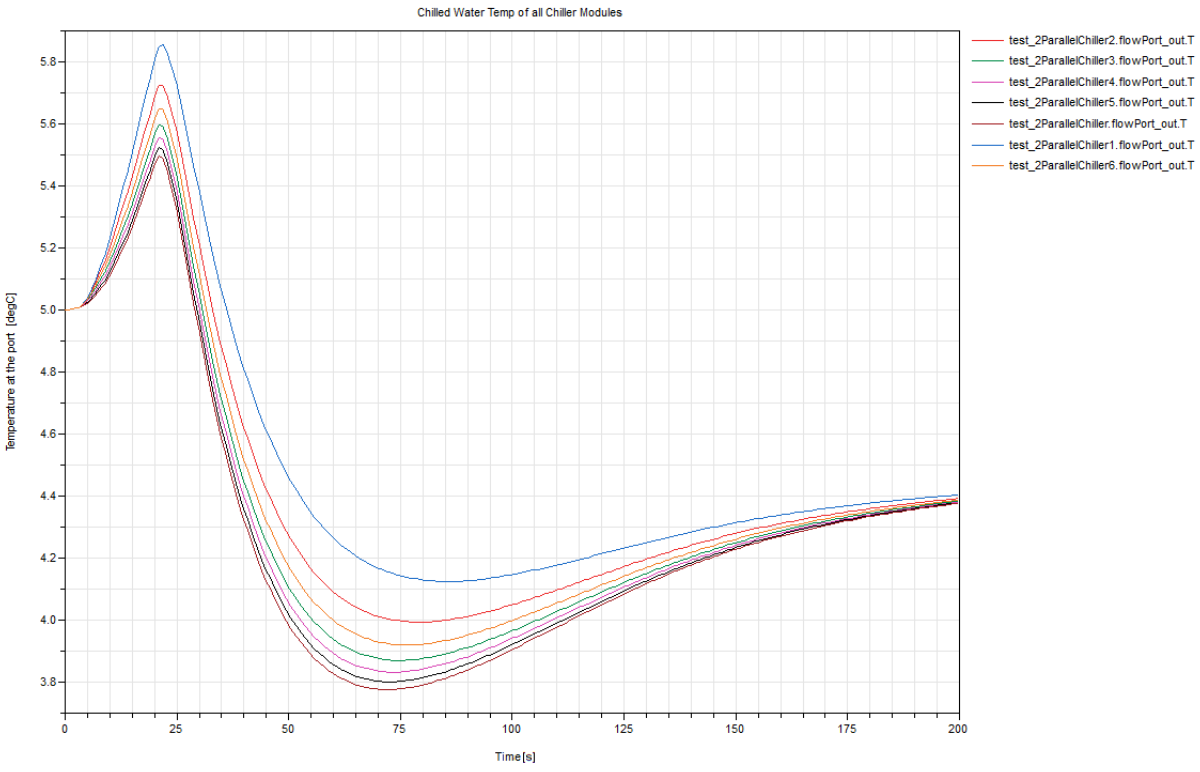


Figure 4.4. Chilled Water Temperature across all Chiller Modules Connected in Parallel

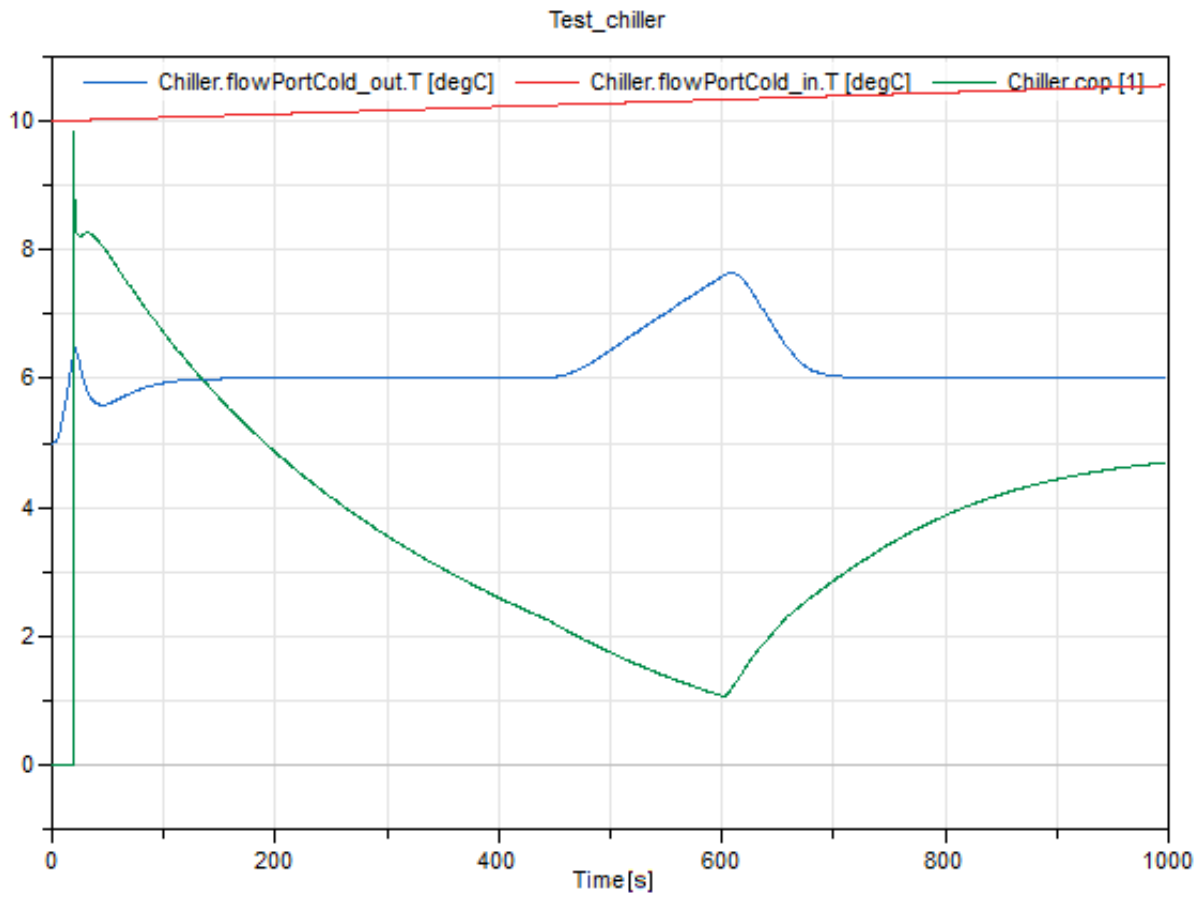


Figure 4.5. Chilled Water Output Temperature of Single Chiller

4.5 Scenario 1

Result shown in Figure 4.7 shows the temperature profile of outlet water from complete chiller plant.

Initial conditions:

- Target temperature = 4.4 degree Celsius
- Inlet water temperature to chiller plant = source.flow.Port.T = variable 10 degree Celsius

The output temperature from chiller plant = sink.flow.Port.T = converges to 4.4 degree Celsius.

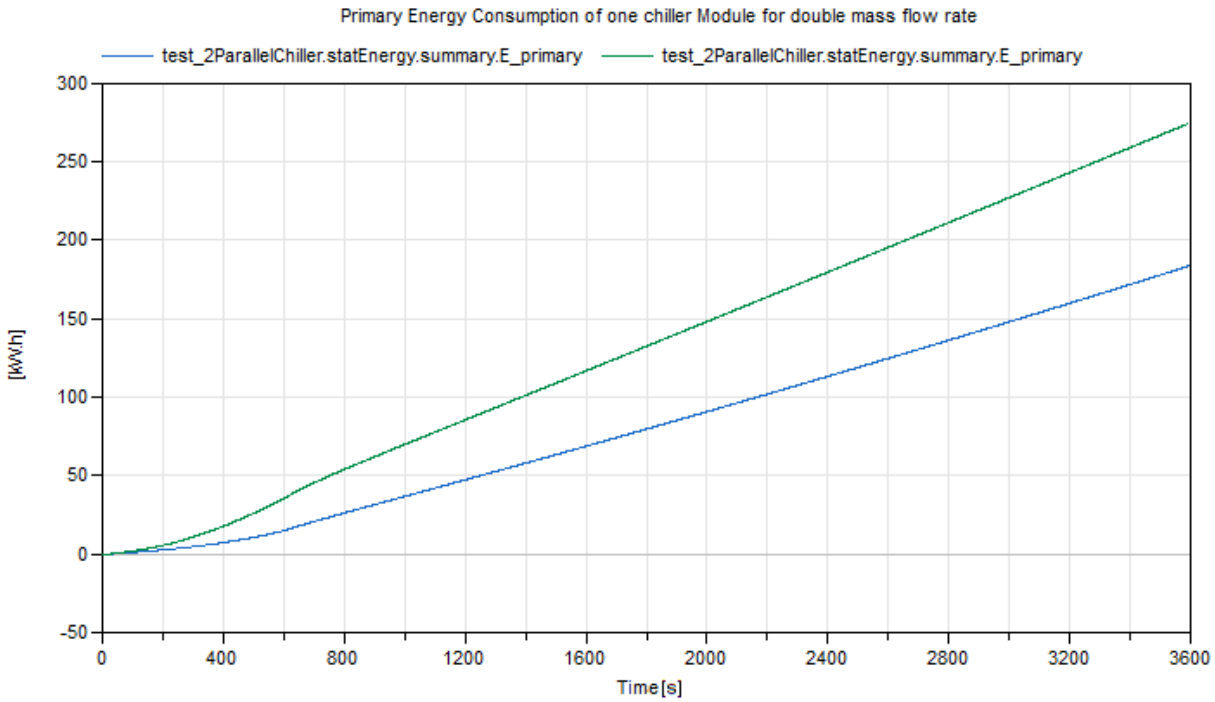


Figure 4.6. Energy Consumption Comparison for Increased in Mass Flow Rate of Fluid

4.6 Scenario 2

Result shown in Figure 4.8 shows the temperature profile of outlet water from two different module.

Initial conditions:

- Target temperature = 4.8 degree Celsius for both chiller
- Inlet water temperature to chiller plant = source.flow.Port.T = constant 12 degree Celsius
- Electric power: $P_{module1} = 1.5 * P_{module2}$

The difference in chilled water output temperature is negligible for 1.5 times the power.

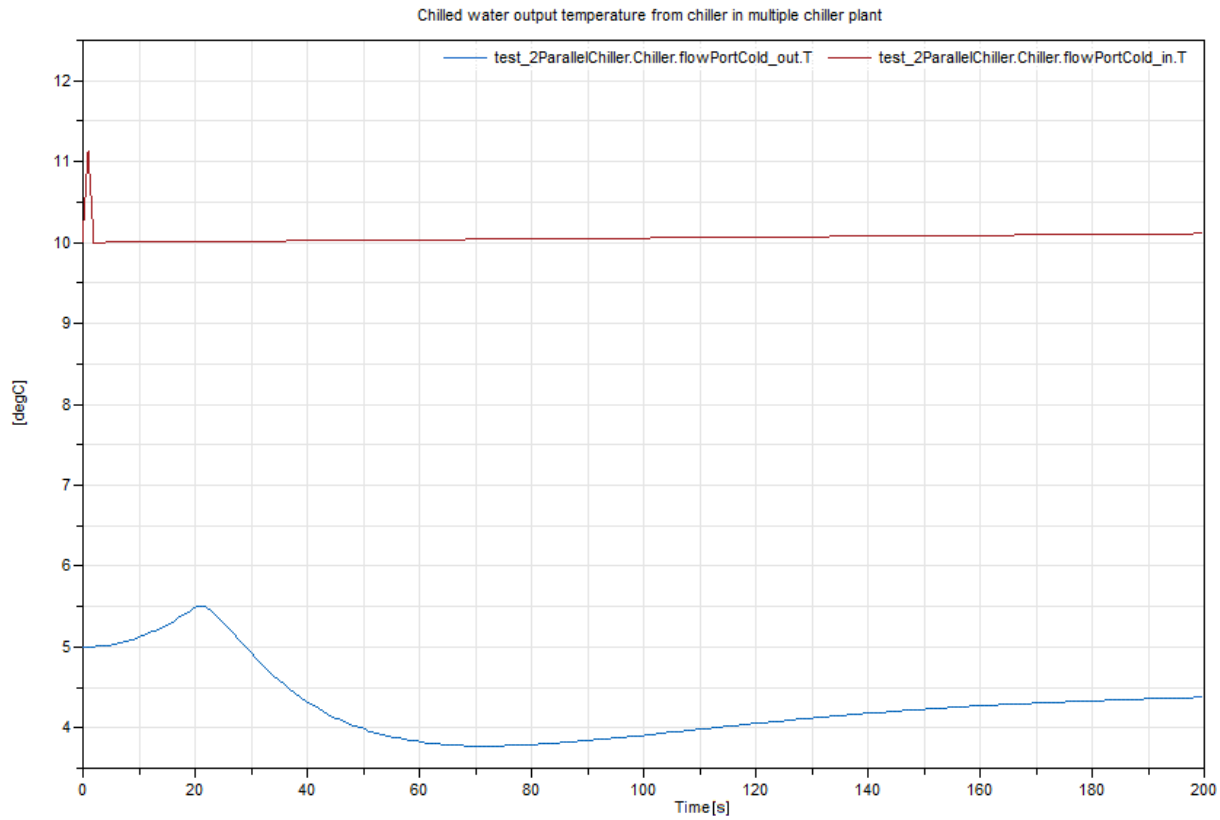


Figure 4.7. Chilled Water Output Temperature from Chiller in Multiple Chiller Plant (7 module)

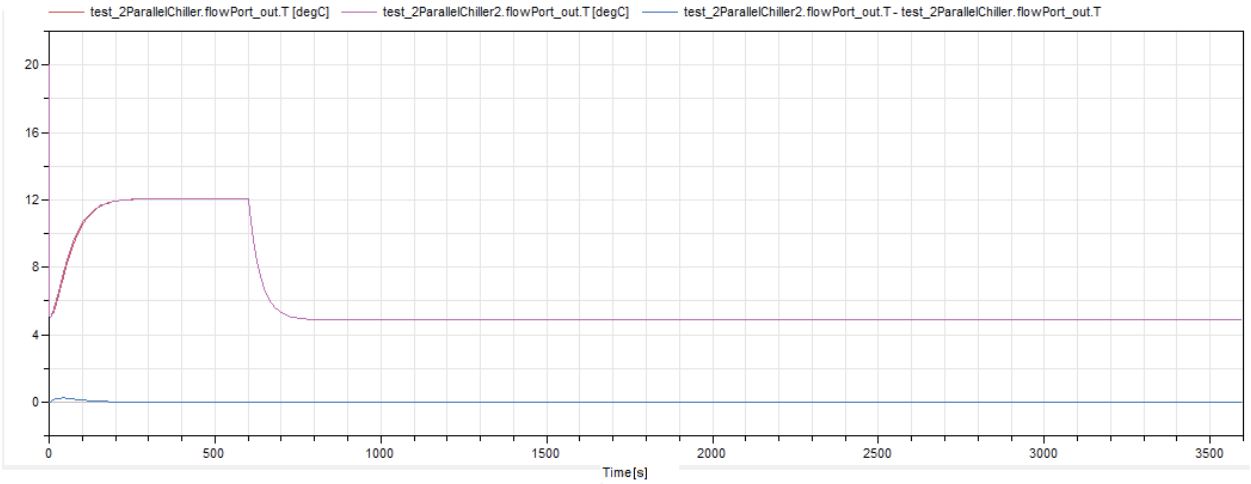


Figure 4.8. Comparison of Chilled Water Output Temperature from Two Modules with Different Set Power

4.7 Scenario 3

A sample case study was used to calculate the COP in terms of IPLV.SI. A test scenario was executed where the chiller model was set to full chiller capacity of 500 tons of refrigeration. The model was instantiated as per the capacity requirements. The Table 4.1 shows the input values of parameters for part loading was extracted from the AHRI standard 551/591-2018.

Table 4.1. AHRI Standard 551/591-2018 Input Variables for Part Loading Conditions

Evaporator(All Types)	
All Loads	7.0
Flow Rate (L/s per kW)	0.0478
Water-Cooled Condenser	
100% load EWT, °C	30.00
75% load EWT, °C	24.50
50% load EWT, °C	19.00
25% load EWT, °C	19.00

After simulating the chiller model, results show that the COP for four different rated part loads conditions is 6.05, 6.42, 6.32, and 4.56, respectively.
Calculation of IPLV.SI is based on equation 3.16. We get,

$$IPLI.SI = 0.01 * 6.05 + 0.42 * 6.42 + 0.45 * 6.32 + 0.12 * 4.56 = 6.1481 \tag{4.1}$$

5. CONCLUSION AND FUTURE WORK

Although many existing MBSE methodologies and frameworks target the development of system architecture models, very few provide a detailed workflow that starts from stakeholder needs and extends to engineering analysis and requirement verification. The framework was developed to bridge the gap between the systems engineering models and systems engineering analysis. The framework also filled in the gap to measure the system complexity based on the complexity index. The Modelica simulation of a single chiller module has successfully predicted the IPLVI.SI is 6.1481, which is within the expected range of 5-7. System requirements refined the stakeholder needs, and system requirements eventually derived the solution domain. The solution domain acted as an input to the dynamic model developed in the Dymola software. The structure pillar provided the components aspects, and requirements provided specification aspects. The Dymola model of district cooling components and their assemblies were simulated based on the scenarios specified in the subsystems/components behavior pillar. Different simulation scenarios provided different simulation results that were used to verify the system/subsystem/component requirements. The model was finely tuned and configured to satisfy the high-level and system-level objectives. The Dymola model was successfully imported in the Cameo Enterprise Architecture tool through the FMI standards of model exchange. The Functional Mock-up (FMU) simulation results resembled the simulation results performed in the Dymola environment. The FMU simulation results provided graphs of the variables typed by value properties with respect to time. These value properties were tied up to verify the relationship to the system requirements. Some of the value properties, such as $Chiller.flowCold\_out.T$, verified the chilled water temperature system requirement. Various such value properties were tied up with the system requirements for verification. These system requirements are derived from stakeholder needs. Therefore their relation was represented using the derived requirement matrix. This provides us with traceability from variables used in the dynamic model to value properties in «fmu» block to system requirements and the stakeholder requirements. The integration of SysML-Modelica with FMI standards provided a platform for the verification process. For instance, the performance requirement captured in the SysML model was verified with the

Modelica simulation results. The simulation results supports the framework based on closed-loop information flow between the system architecture model and the simulation model. The proposed framework bridged the gap between system-level requirements and domain-specific engineering analysis. The complexity index equation represented that a complexity index can be extracted at each level of abstraction. This index depends on the depth of modeling, which proportionally increases with the complexity of the system. The results shows that structural complexity of the system increases steadily from 2.7080 to 8.1241. But the behavioral complexity increases drastically from 1.7915 to 59.2686 at white-box level. This concludes that the mathematical model developed for measuring complexity index predicts the structural and behavioral complexity of system architecture.

The future works target developing test cases for different types of systems that can initiate specifying the complexity index at problem and solution domain. This will help predict the time of development, effort, cost, and risk associated with developing a particular solution. The Dymola model export into the cameo enterprise architecture was manual and can be automated, which would provide real-time verification.

REFERENCES

[1] M. Hause *et al.*, "The sysml modelling language," in *Fifteenth European Systems Engineering Conference*, vol. 9, 2006, pp. 1–12.

[2] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[3] H. Kim, D. Fried, P. Menegay, G. Soremekun, and C. Oster, "Application of integrated modeling and analysis to development of complex systems," *Procedia Computer Science*, vol. 16, pp. 98–107, 2013.

[4] J. S. Topper and N. C. Horner, "Model-based systems engineering in support of complex systems development," *Johns Hopkins APL technical digest*, vol. 32, no. 1, 2013.

[5] H. Kim, D. Fried, and P. Menegay, "Connecting sysml models with engineering analyses to support multidisciplinary system development," in *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2012, p. 5632.

[6] A. Morkevicius, L. Bisikirskiene, and N. Jankevicius, "We choose mbse: What's next?" In *Complex Systems Design & Management*, Springer, 2016, pp. 313–313.

[7] P. Pearce and M. Hause, "Iso-15288, oosem and model-based submarine design," 2012.

[8] J. A. Zachman, "A framework for information systems architecture," *IBM systems journal*, vol. 26, no. 3, pp. 276–292, 1987.

[9] A. Dalvi, "Integrated system model of district cooling for energy consumption optimization," Nov. 2020.

[10] S. Werner, "District heating and cooling," 2013. [Online]. Available: https://doi.org/10.1016/B978-0-12-409548-9.01094-0.

[11] K. M. Powell, W. J. Cole, U. F. Ekarika, and T. F. Edgar, "Optimal chiller loading in a district cooling system with thermal energy storage," *Energy*, vol. 50, pp. 445–453, 2013.

[12] S. Friedenthal, R. Griego, and M. Sampson, "Incose model based systems engineering (mbse) initiative," in *INCOSE 2007 symposium*, vol. 11, 2007.

[13] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[14] D. Brück, H. Elmqvist, S. E. Mattsson, and H. Olsson, "Dymola for multi-engineering modeling and simulation," in *Proceedings of modelica*, Citeseer, vol. 2002, 2002.

[15] ISO, "Iso/iec/ieee 15288: 2015, systems and software engineering–system life cycle processes," 2015.

[16] D. Dutil, J. Rose, W. Suryn, and B. Thimot, "Software quality engineering in the new iso standard: Iso/iec 24748-systems and software engineering—guide for life cycle management," in *Proceedings of the Third C* Conference on Computer Science and Software Engineering*, 2010, pp. 89–96.

[17] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, and T. M. Shortell, *Systems engineering handbook: A guide for system life cycle processes and activities*. John Wiley & Sons, 2015.

[18] K. Ahsan, "Trend analysis of car recalls: Evidence from the us market," *International Journal of Managing Value and Supply Chains*, vol. 4, no. 4, p. 1, 2013.

[19] I. ISO, "Ieee: Iso/iec/ieee 29148, systems and software engineering, life cycle processes," *Requirements engineering*, 2011.

[20] J. Dick, E. Hull, and K. Jackson, *Requirements engineering*. Springer, 2017.

[21] N. Tremblay, C. Y. Laporte, D. Poliquin, and J. Menaceur, "Implementing systems engineering and project management processes in a canadian company–overview and results achieved," in *INCOSE International Symposium*, Wiley Online Library, vol. 29, 2019, pp. 105–129.

[22] T. Clancy, "The chaos report," *The Standish Group*, 1995. [Online]. Available: www.csus.edu/indiv/r/rengstorffj/obe152-spring02/articles/standishchaos.pdf.

[23] A. Zalewski, "Modelling and evaluation of software architectures," *Prace Naukowe Politechniki Warszawskiej. Elektronika*, 2013.

[24] J. A. Estefan *et al.*, "Survey of model-based systems engineering (mbse) methodologies," *Incose MBSE Focus Group*, vol. 25, no. 8, pp. 1–12, 2007.

[25] S. P. Alai, "Evaluating arcadia/capella vs. oosem/sysml for system architecture development," Ph.D. dissertation, Purdue University Graduate School, 2019.

[26] D. Dori *et al.*, *Model-based systems engineering with OPM and SysML*. Springer, 2016.

[27] O. OMG, "Unified modeling language (omg uml)," *Superstructure*, 2007.

[28] A. Morkevičius and S. Gudas, "Enterprise knowledge based software requirements elicitation," *Information technology and Control*, vol. 40, no. 3, pp. 181–190, 2011.

[29] O. OMG, *Systems modeling language (omg sysml), version 1.3*, 2012.

[30] D. Silingas and R. Butleris, "Towards customizing uml tools for enterprise architecture modeling," *Information Systems*, pp. 25–27, 2009.

[31] L. Delligatti, *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[32] *Cameo enterprise architecture software*. [Online]. Available: https://www.nomagic.com/products/cameo-enterprise-architecture.

[33] *Dassault systemes*. [Online]. Available: https://www.3ds.com/products-services/catia/products/dymola.

[34] *Modelica association project, "functional mock-up interface"*. [Online]. Available: https://fmi-standard.org.

[35] R. Kawahara, D. Dotan, T. Sakairi, K. Ono, H. Nakamura, A. Kirshin, S. Hirose, and H. Ishikawa, "Verification of embedded system's specification using collaborative simulation of sysml and simulink models," in *2009 International Conference on Model-Based Systems Engineering*, IEEE, 2009, pp. 21–28.

[36] C. J. Paredis, Y. Bernard, R. M. Burkhart, H.-P. de Koning, S. Friedenthal, P. Fritzson, N. F. Rouquette, and W. Schamai, "5.5. 1 an overview of the sysml-modelica transformation specification," in *INCOSE International Symposium*, Wiley Online Library, vol. 20, 2010, pp. 709–722.

[37] M. G. Balchanos, J. Kim, S. J. Duncan, and D. N. Mavris, "Modeling and simulation-based analysis for large scale campus chilled water networks," in *13th International Energy Conversion Engineering Conference*, 2015, p. 3998.

[38] P. Li, S. Yuan, K. Kang, and H. Reeve, "Evaluation of optimal chiller plant control algorithms in model-based design platform with hardware-in-the-loop," 2016.

[39] P. Li, Y. Li, and J. E. Seem, "Modelica based dynamic modeling of water-cooled centrifugal chillers," 2010.

[40] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*. Siam, 1998, vol. 61.

[41] P. Li, Y. Li, and J. E. Seem, "Consistent initialization of system of differential-algebraic equations for dynamic simulation of centrifugal chillers," *Journal of Building Performance Simulation*, vol. 5, no. 2, pp. 115–139, 2012.

[42] S. Bendapudi, J. E. Braun, and E. A. Groll, "Dynamic model of a centrifugal chiller system–model development, numerical study, and validation.," *ASHRAE Transactions*, vol. 111, no. 1, 2005.

[43] I. Matei, C. Bock, and I. Matei, *SysML extension for dynamical system simulation tools*. US Department of Commerce, National Institute of Standards and Technology, 2012.

[44] R. Wang and C. H. Dagli, "An executable system architecture approach to discrete events system modeling using sysml in conjunction with colored petri net," in *2008 2nd annual IEEE systems conference*, IEEE, 2008, pp. 1–8.

[45] T. Johnson, A. Kerzhner, C. J. Paredis, and R. Burkhart, "Integrating models and simulations of continuous dynamics into sysml," *Journal of Computing and Information Science in Engineering*, vol. 12, no. 1, 2012.

[46] A. Morkevicius, A. Aleksandraviciene, D. Mazeika, L. Bisikirskiene, and Z. Strolia, "Mbse grid: A simplified sysml-based approach for modeling complex systems," in *INCOSE International Symposium*, Wiley Online Library, vol. 27, 2017, pp. 136–150.

[47] K. K, "Application of an innovative mbse (sysml-1d) co-simulation in healthcare," *Doctoral Thesis*, 2018.

[48] M. J. Kinnunen, "Complexity measures for system architecture models," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[49] D. Kaslow, L. Anderson, S. Asundi, B. Ayres, C. Iwata, B. Shiotani, and R. Thompson, "Developing a cubesat model-based system engineering (mbse) reference model-interim status," in *2015 IEEE Aerospace Conference*, IEEE, 2015, pp. 1–16.

[50] M. Chami, P. Oggier, O. Naas, and M. Heinz, "Real world application of mbse at bombardier transportation," *The Swiss Systems Engineering Day (SWISSED 2015), Kongresshaus Zurich, 8th September*, 2015.

[51] U. D. of Defense, *Dod architecture framework, version 2.0, volume 2: Architectural data and models architect's guide*, 2009.

[52] H. G. Sillitto, "6.2. 4 on systems architects and systems architecting: Some thoughts on explaining and improving the art and science of systems architecting," in *INCOSE International Symposium*, Wiley Online Library, vol. 19, 2009, pp. 970–985.

[53] J. H. Holland, *Complexity: A very short introduction*. OUP Oxford, 2014.

[54] S. Tredinnick and G. Phetteplace, "District cooling, current status and future trends," in *Advanced District Heating and Cooling (DHC) Systems*, Elsevier, 2016, pp. 167–188.

[55] A. Handbook, *HVAC systems and equipment*. Chapter, 1996, vol. 39.

[56] P. Nag, *Engineering thermodynamics*. Tata McGraw-Hill Education, 2013.

[57] (2017). "Disctrit cooling in qatar," [Online]. Available: https://www.euroheat.org/wp-content/uploads/2017/09/1.1-171024_1400-1530-1-POL-Ibrahim-Mohammed-A-Al-Sada-IDCHC-Presentation-Final.pdf.

[58] Design and W. M. C. 2016. (2016). "Dc design and water management code 2016," [Online]. Available: www.km.com.qa/CustomerService/ServiceRegulations/DISTRICT%20COOLING%20Design%20and%20Water%20Managementcode%202016.pdf.

[59] *Dymola, hvac library*. [Online]. Available: https://www.claytex.com/products/dymola/model-libraries/hvac-library.

[60] M. Association. (2020). "Openmodelica user's guide release v1.17.0," [Online]. Available: https://www.openmodelica.org/useresresources/userdocumentation.

[61] P. Date, "Approved american national standards," *ANSI: Washington, DC, USA*, 2015.