Creating and validating a Domain Specific Language for Arrowhead Framework in the Internet of Things domain

Gunasekaran, Raghavendran

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

MASTER IN EMBEDDED SYSTEMS

MASTER THESIS REPORT

Creating and validating a Domain Specific Language for Arrowhead Framework in the Internet of Things domain

Author:
Raghavendran Gunasekaran
Student ID: 1328484

Supervisors:
TU/e
Ion Barosan (Asst. Professor)
Önder Babur (Post Doct.)
Mahdi Saeedi Nikoo (Doct. Candidate)

November 3, 2020

This page is intentionally left blank.

Abstract

Arrowhead Framework is an IoT automation Framework which helps in inter-operability and integration of Internet of Things (IoTs) abstracted into systems and services. The Arrowhead Framework architecture is composed of local automation clouds which comprises of essential services used by a SoS application [**4**]. It aims towards collaborative automation and industrial IoT by building System of Systems (SoS) from IoT components and enabling inter-operability among them [**2**]. This Framework further helps in building complex interoperable IoT systems. There is a need for these complex critical systems in Arrowhead Framework to be modeled and validated before the actual development of these systems in order to avoid faults or vulnerabilities. A Domain Specific Language (DSL) is a type of formal language comprising of elements from a specific domain. A DSL makes modeling process easier by making use of domain specific elements rather than general purpose modelling language elements. A DSL for Arrowhead Framework encompassing its complete structure and elements in its entirety needs to be established. This established profile comprising of Arrowhead Framework constructs would serve as a basis for Model Based System Engineering (MBSE), migrating from document based information exchange to effective model based information exchange. Effectively, any complex IOT automation system in Arrowhead Framework can be modelled with multiple views encompassing the requirements, behaviour, structure and parametrics of the system. The tool for creating the DSL is selected based on the requirements for DSL creation and validation process. At the end of the selection process where several tools were evaluated, Cameo Systems Modeler was selected as the suitable tool for this assignment. The DSL for Arrowhead Framework is then created by the method of UML extension using profiles which comprises of stereotypes and tagged values. A complete DSL comprising of the primary elements of the Framework and the different systems and services in the Framework are created. This DSL is validated to check for well-formedness using Object Constraint Language (OCL) constraints. A model is built using this Framework and code is generated for this specific model for validation. The generated code is then validated in the Arrowhead Framework environment which in turn validates the created DSL. Effectively, this created DSL enhances communication and understanding which in turn saves time and improves productivity.

Acknowledgements

This thesis work was possible only because of the support of some people that I would want to mention here. I would first want to thank my main supervisor, professor dr.Ion Barosan for allowing and continuously inspiring me to work on this project. He was a pillar of support all through this assignment providing regular feedback, valuable insights and guidance. I would also want to extend my gratitude towards my supervisor dr.Önder Babur for his constant feedback, advice and suggestions whenever required. I would further like to thank Mahdi Saeedi Nikoo for sharing his expertise and knowledge in Arrowhead Framework throughout the assignment. I would like to thank all three of my supervisors for being available at all times for meetings, welcoming me with enthusiasm whenever I needed guidance.

Géza Kulcsár from IncQuery Labs deserves a special mention for helping me with the questions I had while working on this assignment. I am also grateful to the team from IncQuery Labs for sharing their work on the Arrowhead Framework profile. I would also like to thank the entire Arrowhead Framework team for building this Framework which eventually provided me with an opportunity to work on this assignment.

I want to express deep and sincere gratitude to my parents and sister for their continuous and unparalleled support. I would like to further thank Samruddhi, who was always there for me.

None of this would have been possible without the blessings of the Almighty.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

CPS	**C**yber **P**hysical **S**ystems
IBM	**I**nternational **B**usiness **M**achines
IoT	**I**nternet **o**f **T**hings
OCL	**O**bject **C**onstraint **L**anguage
SoS	**S**ystem **o**f **S**ystems
VTL	**V**elocity **T**emplate **L**anguage
DSL	**D**omain **S**pecific **L**anguage
SOA	**S**ervice **O**riented **A**rchitecture
REST	**Re**presentational **S**tate **T**ransfer
QoS	**Q**uality **o**f **S**ervice
SysML	**S**ystems **M**odeling **L**anguage
UML	**U**nified **M**odeling **L**anguage
BDD	**B**lock **D**efinition **D**iagram
IBD	**I**nternal **B**lock **D**iagram
MOF	**M**eta **O**bject **F**acility

Dedicated to my Mother…

Chapter 1

Introduction

1.1 Background

Industry 4.0 is the currently evolving industrial shift which focuses on automating traditional industrial systems and establishing interaction among devices at lower levels in the plant with complex higher level enterprise systems. Technologies such as IoT and CPS are ubiquitously used in Industry 4.0 to improve flexibility. Standard protocols used for communication in Industry 4.0, paves way for establishing interaction between, possibly, any systems or devices on the factory floor. Although this provides several advantages, it does create concerns in terms of security and reliability. To address these concerns and to improve scalability, production flexibility and automation engineering, the notion of Arrowhead Framework was proposed.

The Arrowhead Framework is a collaborative research project worked upon by several European companies, universities and research institutes. This Framework is a currently developing and evolving standardization that aims towards inter-operability and creation of Industry 4.0 industrial automation systems. It is an IoT automation framework which helps in inter-operability and integration of IoTs abstracted into systems and services [**2**]. The main objective of the Arrowhead Framework is to provide a way for all users to work in a unified manner as a whole, which makes high levels of interoperability possible. In addition, it aims towards collaborative automation and industrial IoT by building SoS from IoT components and enabling inter-operability among them [**2**].

1.2 Problem Context

While the Arrowhead Framework provides a way to build complex inter-operable automation systems, these systems and their corresponding interactions should be secure and reliable. To ensure reliability and inter-operability, these systems have to be modelled and validated before the actual implementation. In order to model these complex automation systems in the Arrowhead Framework with the Framework specific constructs, it dictates the need for a DSL. Use of a DSL would help in concisely expressing the models, by using domain specific concepts, which in turn leads to better understanding and improved communication [**42**]. Furthermore, it improves the productivity, quality, and reliability of models [**42**]. IncQuery Labs had created a basic profile for the Arrowhead Framework to address these issues [**17**]. However, it doesn't cover every aspect of this continuously evolving Framework. Furthermore, the models created using specific domain elements should be validated using tools specific to that domain. Thus, a DSL for the Arrowhead Framework covering all concepts and elements of this Framework is needed. In addition, the models created from this DSL should also be validated in the Arrowhead Framework environment.

1.3 Problem Definition and Research Questions

While a basic SysML profile or DSL for the Arrowhead Framework exists, it doesn't cover every aspect of this continuously evolving framework. A complete SysML profile or a DSL should be established, encompassing the structure of the Arrowhead Framework systems and services, and the interactions among them. The profile would provide a basis for modelling complex IoT systems, large automated systems and other systems based on the Arrowhead Framework. Furthermore, the Arrowhead Framework in itself needs fair amount of time and effort to be comprehended. It would be easier for those with limited knowledge on modelling languages, to seamlessly model complex IoT systems in this framework with these domain specific constructs. In addition, at the beginning of this assignment, there wasn't any existing methods for validating these models in the Arrowhead Framework. This further creates a need for generating Arrowhead Framework specific code from these models, which can be validated in the Arrowhead Framework environment. Thus, essentially, a complete workflow of creating a DSL for Arrowhead Framework, building a model using elements from this DSL, generating Arrowhead Framework specific code from this built model and validating this model by validating the code generated from it, in the Arrowhead Framework environment, needs to be completed.

Identifying the gaps in the current setup is the first step of holistic realisation of the setup which would help to reach the goal. This involves formulating the research questions and finding the different approaches for addressing them.

- ***Research Question 1*** *: Does SysML have enough semantics for creating a DSL for Arrowhead Framework?*
- ***Research Question 2*** *: How to build the SysML profile for Arrowhead Framework? Which elements of the Framework can be profiled and which elements cannot be?*

1.4 Project Plan

The first step of this research assignment should be to perform an extensive literature study to understand the Arrowhead Framework concepts completely, the importance of this framework, the value it offers in terms of inter-operability, integrating complex automation systems, ease of implementation and others. Further literature study would provide understanding of the need for a DSL and the applications and usefulness of this DSL. In addition, it would help in understanding the existing DSL created for this framework and other work completed with this framework.

The second stage of literature study should be done to understand about the different modelling languages for systems modelling, different methods for creating DSLs, processes for creating a DSL and methods for validating a DSL.

The next step should be performed to collate the features required in the modelling tool. Assessment and evaluation of the different modelling tools should be completed to find the best suited tool that supports the selected modelling language and provides the option for building a DSL using the selected method from the previous stage. Furthermore, it should provide the option for building a model using the elements in the DSL and the option to generate custom code from the model. In addition, it should provide the option for exporting the generated code.

The following step should be performed to identify the Arrowhead Framework concepts. On identification, a database should possibly be created based on the collated information on the Framework elements. These elements should be used for building the profile and model libraries.

These elements should include concepts, physical and logical entities, relations, attributes and processes in the Framework.

The subsequent step will be performed to build the DSL for the framework, with the elements from the created database. The following step should identify the application for which the model needs to be built. On identification, the model should be built using the profile elements.

The next step will be to identify the template language used for the template/code generation and then, to understand the concepts and syntax of that language. Furthermore, the model-specific information which needs to be used in the code should be identified. On identification, the template code for generating the code from the model should be developed.

The last step should be performed to validate this assignment. The created DSL should be validated for well-formedness using OCL constraints. The model should be validated in the Arrowhead Framework environment through the generated code. This would eventually validate both the model and the DSL using which the model was created.

1.5 Risk Assessment

There are several risks at different levels of severity in this assignment which were identified. These risks are listed below.

1. Unavailability of fully functional features in the software tool.
2. Constantly evolving Framework with new versions comprising of additions, deletions and modifications of systems, services and operations.
3. It is also possible that this DSL might not cover all the aspects of this Framework.

1.6 Structure of the report

This report is organized into seven different chapters and the description of each of these chapters is listed below.

Chapter 1: The first chapter is the Introduction chapter which covers the background of this assignment, problem context, problem definition and research questions, project plan, risk assessment and the structure of the report.

Chapter 2: The second chapter encompasses the entire architecture of Arrowhead Framework. It further discusses on the primary concepts of SOA, SoS and the Local Cloud approach used in the Framework. Effectively, all the information needed to have a fundamental understanding of the Arrowhead Framework is included in this chapter.

Chapter 3: This chapter discusses about the SysML, its inception, its relation with UML and the different diagrams in SysML. Furthermore, it discusses on the SysMOD methodology and the V-model system development lifecycle. In addition, it covers information on DSL and its need and the different methods for creating one.

Chapter 4: This chapter covers the DSL created for Arrowhead Framework, elucidating on the methodology used for creating this DSL. It further provides a detailed information on every element part of the created DSL including the semantics and rationale. Furthermore, it covers the model libraries created for the core systems from the profile elements.

Chapter 5: This chapter encompasses the detailed information on the model built using the created DSL elements. In addition, it discusses about the code generation method in the tool and the language used for generating.

Chapter 6: This chapter elucidates on the validation of the created DSL. It covers the different parts of validation, namely, validating using OCL, validating the system and service aspects and validating the code generated from the model.

Chapter 7: This chapter concludes this assignment. It encompasses the contributions and learnings from this assignment. Furthermore, it includes the recommendations for future research work.

1.7 Summary

The intent of this project is to improve the existing Arrowhead Framework profile by extending it into a new complete DSL for the Framework. The created DSL would provide a way for modeling complex IoT systems in the Framework which in turn would help to bridge the gap in communication, knowledge sharing and understanding. A model has been built using this DSL to understand the intuitiveness of modeling systems in the Framework using this DSL. Several validation methods have been suggested which includes validating the DSL using OCL constraints and by generating code for the specific model and validating the code in the Arrowhead Framework environment. Thus, this project creates an entire modeling workflow where elements used for modeling are first created, the created elements are then used for building a model, OCL constraints are used to validate the DSL, code is generated for the specific created model and this generated code is then validated in the Arrowhead Framework environment.

Chapter 2

Architecture of the Arrowhead Framework

2.1 Introduction

The first step for creating a Domain Specific Language for a Framework is to understand its architecture completely. This chapter covers the architecture of Arrowhead Framework in its entirety. It provides detailed information on different concepts based on which the Framework was created. This would help in better understanding the Framework and its elements.

2.2 Introduction on the Arrowhead Framework

The Arrowhead Framework architecture comprises of IoT-based automation SoS which emphasizes on:

- Interoperability of a wide range of IoT and legacy systems.
- Handling data in real time for faster communication and control computations.
- Providing better security to systems and data.
- Engineering of automation systems.
- Scalability of automation systems paving way to large integrated automation systems. [**2**]

This Framework is based on the Spring-Boots Framework in Java and REST web services [**16**]. Furthermore, the Arrowhead Framework project offers client skeleton code and examples in various programming languages such as Java, Python, C++ and NodeJs [**14**]. A recent addition to this list is Kalix, which is a Java 11 library for developing eclipse arrowhead systems [**24**].

2.3 Service Oriented Architecture

The architecture of Arrowhead Framework is based on the SOA paradigm. SOA has been defined by James Bean [**25**] as:

"A service-oriented architecture (SOA) is a combination of consumers and services that collaborate, is supported by a managed set of capabilities, is guided by principles, and is governed by supporting standards."

Information exchange between a provider system and a consumer system is represented by an entity called service [**6**]. Services are the primary building blocks of SOA [**6**]. This service exchange between a provider system and consumer system is depicted in Figure 2.1. A service

provider registers its service with a service registry system. The registered service is discovered by the service consumer using the service discovery process. This discovered service is then consumed by the service consumer.

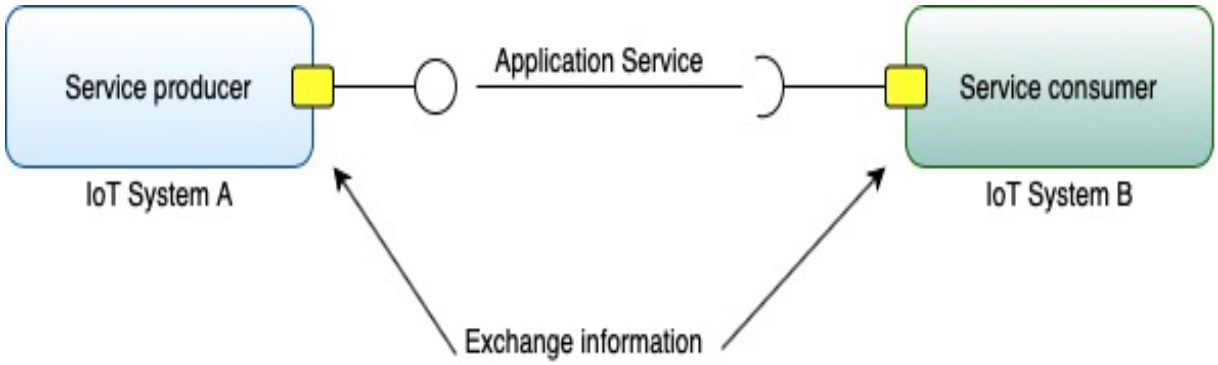

FIGURE 2.1: Service exchange between a producer and a consumer [**4**]

The Arrowhead Framework architecture adheres to the primary properties of SOA such as:

1. Loose Coupling-

   Services are distributed over several systems and devices. In order for a run-time data exchange to occur, there isn't a necessity for a system to know the other system during design time. Service registry and discovery mechanisms are supported, where a new service is registered at the service registry by its own. Upon registration, this service is made discoverable for other systems. A system needing to consume this service can use the service discovery mechanism to consume it [**26**]. This loose coupling between systems is represented in Figure 2.2 [**4**].

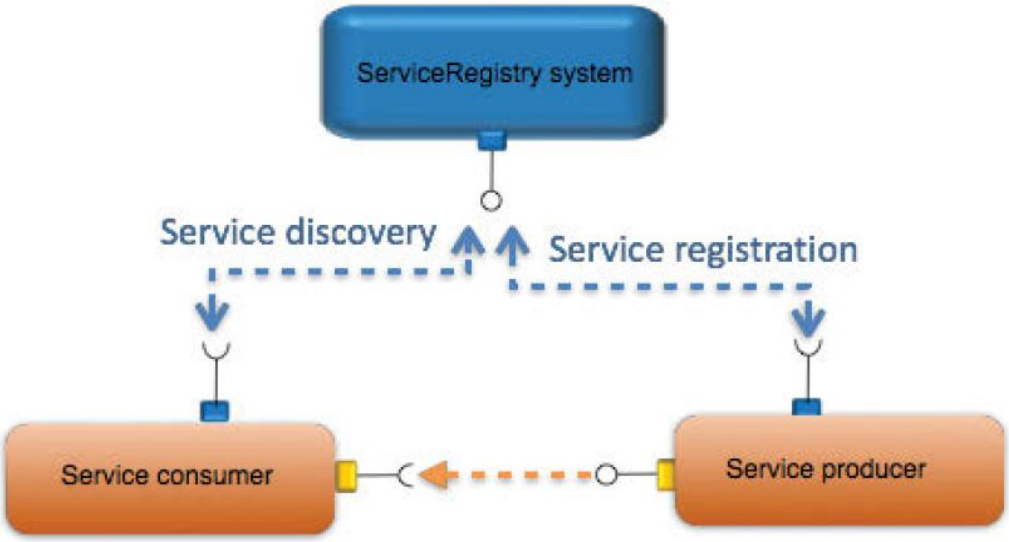

FIGURE 2.2: Loose Coupling in SOA [**4**]

2. Late Binding-

   A system can connect with other systems at anytime when it requires data, that is, the exchange of data between two systems happens during run-time. An orchestration method of interaction could be put to use for establishing this late binding between systems [**4**].

3. Autonomy-

   Systems own their data and they are responsible for making decisions on which systems they can share their data. Every system can act on its own, independent of the other systems. Moreover, in the event of a service exchange setup between two systems, this exchange would be unsupervised, that is, no other supporting services/functionalities would need to intervene [**4**].

4. Push and Pull behaviour-

   A service exchange can be initiated either by a system consuming the service or by a system producing the service. While pull behaviour is when the service exchange is initiated by the service consumer, the push behaviour is when the service exchange is initiated by the service producer [**4**].

2.4 System of Systems

As mentioned earlier, the architecture of Arrowhead Framework comprises of IoT-based automation SoS. A SoS can be defined as a large network of complex and geographically distributed systems functioning together to achieve a common goal where these component systems are independent and significant in their own right [**27**]. An example SoS in the healthcare domain is shown in Figure 2.3 for further understanding. In this Figure 2.3, the different systems such as Patient Management System, Laboratory System, Imaging Management System, Telemetry System and Pharmacy System perform different individual functions of their own. These systems collaborate together to form the Health Care System which has a specific purpose of handling health care related functions.

There are five main characteristics of a SoS which helps in identifying one [**27**]:

1. Operational Independence of Components-

   In the case of a SoS comprising of component systems being disassembled, each of these component systems should have the ability to operate autonomously for a useful purpose. Effectively, this ensures that a SoS comprises of component systems which are completely autonomous and useful by virtue of itself [**27**].

2. Managerial Independence of the Elements-

   One of the necessary condition is that component systems should be operating independently. Although, a separate acquisition and integration of component systems is done, they still maintain a continued operational existence which is not dependent on the SoS [**27**].

3. Evolutionary Development-

   The SoS may not appear fully formed and its existence and development is progressive where addition, deletion and modification of goals and functions are done as time progresses [**27**].

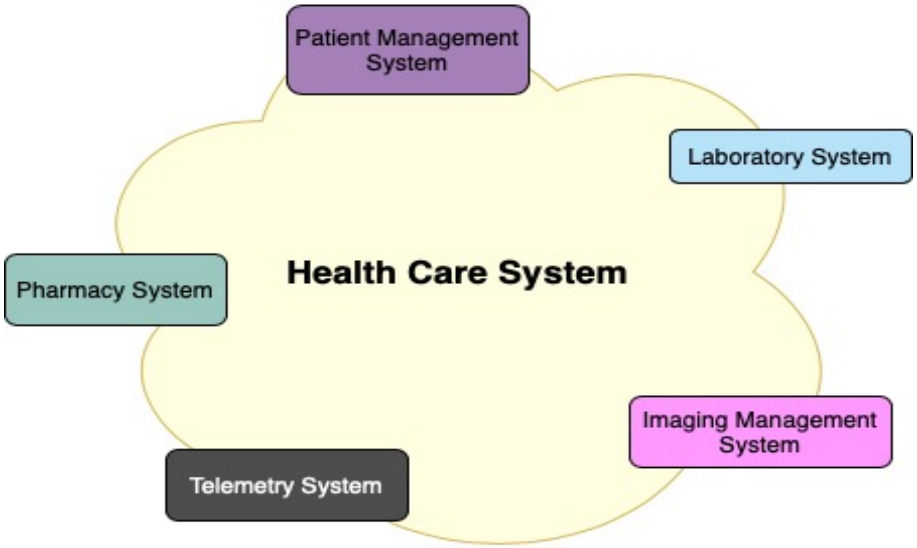

FIGURE 2.3: SoS in Health Care domain [**29**]

4. Emergent Behaviour-

   The SoS perform several functions and accomplish goals which aren't part of any of the component systems contained in it. These are emergent behaviours of the unified SoS and this cannot be localised to any specific component system. Effectively, the principal goals of the SoS are accomplished by these behaviours [**27**].

5. Geographic Distribution-

   The geographical extent of the component systems is large, where the bare minimum should be that, information exchange among components can occur readily and not vast quantities of energy or mass [**27**].

2.5 The Local Cloud approach in Arrowhead Framework

Several forms of cloud computing are becoming prominent in the industry. An open global internet cloud has several limitations and thus, not suitable for automation purposes. Automation tasks have tight requirements on security, operations, real-time communication and computation, maintenance and others. Consequentially, a notion of local clouds was devised which would adhere to the above automation task requirements. This local cloud is formed by including devices and systems with services, which are necessary to perform the required automation tasks. Effectively, this local cloud forms a boundary containing the necessary systems and this boundary safeguards the systems inside from any inbound or outbound communication through the outside open internet [**28**].

The local clouds in an Arrowhead Framework architecture is depicted in Figure 2.4. These local clouds comprise of systems providing and consuming essential services which collaborate as a SoS. Effectively, a local cloud acts as a SoS where a set of systems collaborate amongst themselves to build a much more complex automation system. Furthermore, the characteristics of a local cloud are similar to the aforementioned characteristics of a SoS. Thus, a local cloud acts by itself as a SoS. [**3**]

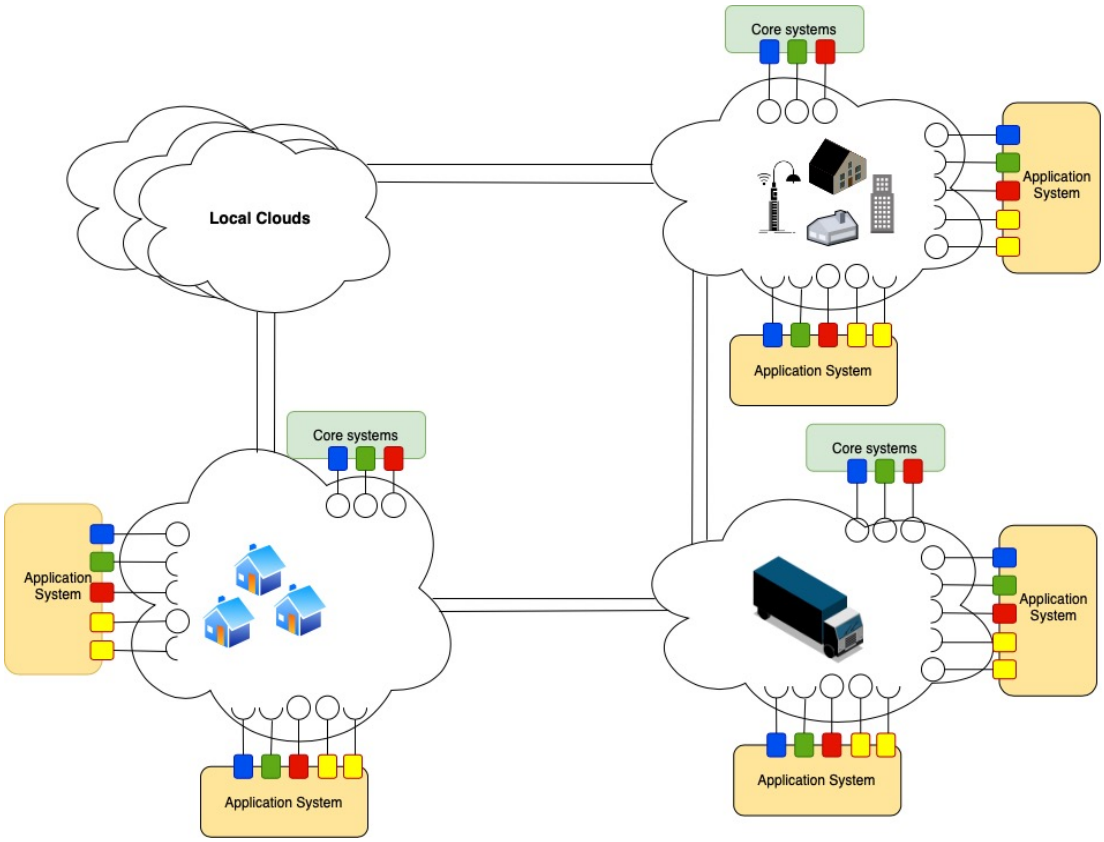

FIGURE 2.4: Local Clouds acting as a SoS [4]

The architecture of Arrowhead Framework is composed of local automation clouds which guarantee certain features correlated to a global cloud such as:

- Cloud service infrastructure.
- Services for supporting automation.
- Latency properties can be guaranteed or predicted very well.
- Essential security properties such as authorisation and authentication.
- Scalability of automation systems leading to vast integrated complex automation systems.
- Ease of Application engineering.
- Multi-stakeholder integration.

In some instances while building large automation systems, if the necessary information is not available within a particular local cloud, secure inter-cloud interaction could occur. This interaction is where one local cloud communicates with other local clouds that follow the guidelines of the Arrowhead Framework for information exchange. Inter-cloud interaction helps in building vast complex scalable automation systems [4].

2.6 Properties of a Local Cloud

The primary properties of a basic local automation cloud in the Arrowhead Framework are:

1. Self Contained

   A local cloud encompasses the resources which are needed to establish itself ensuring autonomous local operation. This eliminates the dependency on other external resources [**28**]. The primary external resources needed to establish a local cloud are:

   - Registries- Help in maintaining a record of the devices, systems and services deployed within the local cloud. Furthermore, this establishes the possibility for discovery of the aforementioned. The three main registries needed in the Arrowhead Framework local cloud are:
     - Service Registry.
     - System Registry.
     - Device Registry.
   - Service Orchestration where a central entity can coordinate the interaction among different services.
   - Service authentication and authorisation for secure validation of service consumer/entity and secure access control/service consumption respectively.

2. Support for Automation requirements

   - Support for automation system design, configuration, deployment, operation and maintenance.
   - Enablement of event based service exchange.
   - Enablement of audit of service exchange.
   - Support for communication QoS.
   - Storage of audit information and historical data [**28**].

3. Provide a security fence to external networks

   The availability of necessary resources contained within the local cloud eliminates the need for external communication with other resources outside. Consequentially, the physical communication layer is not connected with any other external networks [**28**].

4. Secure bootstrapping and software updates

   Deployment and update of devices, systems and services should be secure [**28**].

5. Support for device, systems and service metadata

6. Support for protocol and semantics transparency

   Interoperability of IoT systems requires protocol and semantic transparency which leads to scalability [**28**].

7. Support for secure administration and data exchange with external resources

   In case a need for inter-cloud service exchange arises, the aforementioned service exchange administration should be possible such as service discovery, service authentication, service authorisation and service orchestration. Furthermore, the exchanged service payload data should have end to end encryption [**28**].

When integrating the aforementioned SOA properties with these properties of a local cloud, it provides a necessity for certain essential aspects in the local cloud. These aspects are registering systems and services with the cloud, discovery of the registered services, authorising the systems before registering their services and orchestrating exchange of services between the service producer and service consumer. Eventually, this lead to forming the primary systems needed to form the Arrowhead Framework local cloud [**4**]. The three primary core systems in the Arrowhead Framework are Service Registry System, Authorisation System and Orchestration System. The Arrowhead Framework local cloud acting as a SoS with the three primary core systems is shown in Figure 2.5.

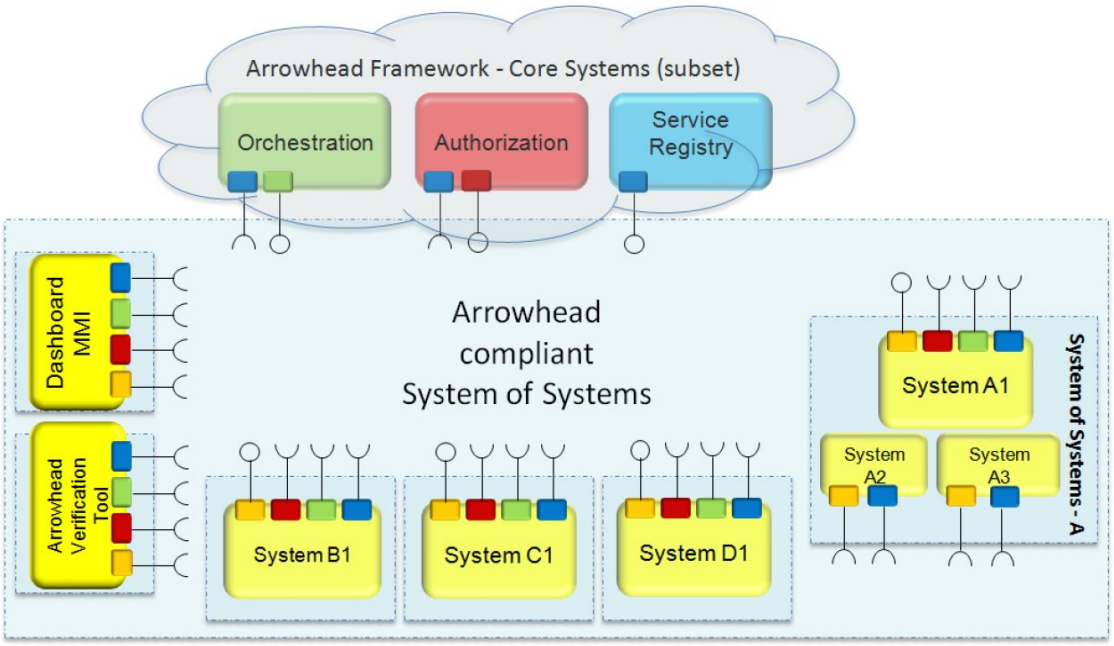

FIGURE 2.5: Arrowhead Framework compliant SoS with the three primary core systems [**31**]

2.7 Arrowhead Framework Design Principles

While the primary systems needed for establishing the local cloud is determined, a set of strict design principles for Arrowhead Framework is required. This would ensure that a clear demarcation between Arrowhead Framework compliant systems with other systems is established. Furthermore, it helps in having a better understanding of the Framework for designers and developers which would help them with developing, deploying, maintaining and managing these systems.

The design of the Arrowhead Framework is based on the following [**4**]:

1. The Arrowhead Framework adheres to the aforementioned properties of SOA.
2. The framework compliant SoS follow the aforementioned characteristics of a SoS.
3. A local cloud becomes Arrowhead Framework compliant only if it adheres to the aforementioned primary properties of a local automation cloud.
4. The framework provides the option for dynamic creation and of new services, the consumption of these newly created ones and removal of existing services.

5. In addition, it supports both the push and pull approaches.
6. Furthermore, it supports the publish-subscribe approach.
7. A system producing a service has the initial authority of its own service.
8. Only at the level of service exchange, information is guaranteed.
9. The networking in the framework is more information centric.

The above design requirements of the Arrowhead Framework are accommodated through a set of three core mandatory services which helps in forming a basic SoS and an aggregation of automation support services. In this way, the framework comprises of two sets of services, namely, the Core Services and the Application Services. The local clouds in this framework are governed by some core services. The core services are further divided into two, namely, the Mandatory Core Services and the Automation Support Core Services [**4**]. The three mandatory core services are Service Registry services, Authorisation services and Orchestration services.

2.8 Structure of Arrowhead Framework

The primary structure of Arrowhead Framework comprises of a local cloud acting as a SoS. This Arrowhead Framework local cloud comprises of three basic building blocks and they are:

1. Devices - Any entity which is a hardware or machine which has the capability to compute, communicate and store data, and also, host one or many Arrowhead Framework compliant systems along with their services is an Arrowhead Framework device [**4**].
2. Systems - A system in the Arrowhead Framework is an entity in a local cloud that registers with the mandatory System Registry and is capable of producing or consuming Arrowhead Framework compliant services while relinquishing its authority over the services provided by it [**4**].
3. Services - The exchange of information between a producer and consumer is represented by an entity called as Service [**31**]. It can be comprehended as a software unit which is a logical representation of a specific task or holds a specific information [**30**].

Any system in this local cloud could possibly be hosted on a hardware device. A system in the framework either provides or consumes one or more services or it could both provide and consume services [**4**].

2.9 Arrowhead Framework Core Systems

The Arrowhead Framework provides a set of core systems, where the mandatory core systems provide the indispensable services in the local cloud and the support core systems help in automation support. In order to build scalable complex automation systems with better security, inter-operability between incompatible systems, service exchange across different local clouds and better bootstrapping, these systems have been introduced in the Arrowhead Framework [**4**]. There is an increasing number of these support core systems which are being developed currently. The support core systems listed below are either currently being developed or have already been developed [**4**] [**32**]:

- Service Registry System (Mandatory Core System)

- Authorisation System (Mandatory Core System)
- Orchestration System (Mandatory Core System)
- System Registry System
- Device Registry System
- Event Handler System
- Gatekeeper System
- Gateway System
- QoS Manager System
- Choreographer System
- Onboarding Controller System
- Translation System

These core systems along with the application systems collaborate with each other to build a complex automation system, forming a SoS. The minimum requirement for functioning of a local automation cloud is the collaboration of the three mandatory core systems with the application systems as shown in Figure 2.6 **[4]**. These different core systems are elucidated in detail in the following sections.

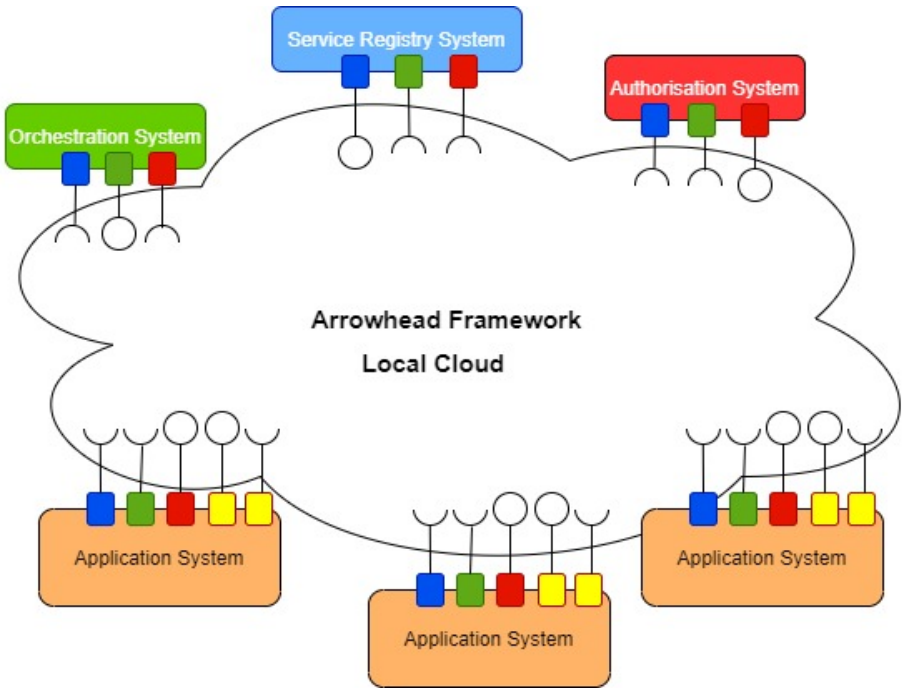

FIGURE 2.6: A local cloud functioning with the minimum requirement of the three mandatory core systems **[4]**

2.10 Arrowhead Framework Documentation Model

The Arrowhead Framework documentation model provides information on how the SoS, Systems and Services should be described. It comprises of the SoS level documentation, Systems level documentation and Service level documentation [**2**]. The service level documentation comprises of four documents namely Service Description (SD), Interface Design Description (IDD), Communication Profile (CP) and Semantic Profile (SP) [**4**]. The relationship among these Arrowhead Framework documents are depicted in Figure 2.7 [**4**]. In addition, the relationship between the documents part of the service level documentation and their dependency on technological information is depicted in Figure 2.8.

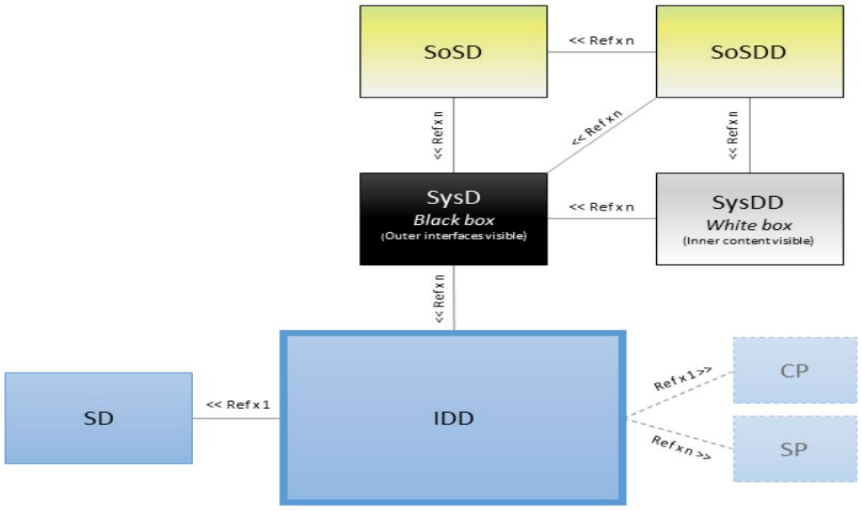

FIGURE 2.7: Relationship among the Arrowhead Framework Documents [**4**]

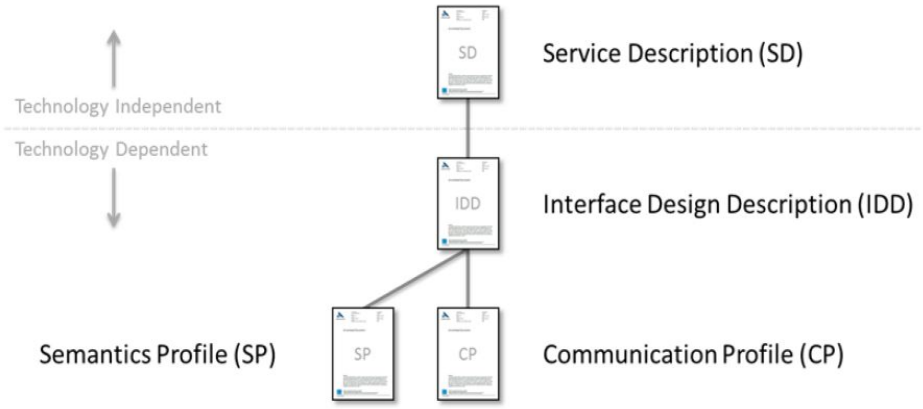

FIGURE 2.8: Relationship between the documents in the service level documentation [**33**]

2.11 Maturity of Arrowhead Framework

The Arrowhead Framework is a currently developing and evolving standardization and research work that aims towards interoperability and creation of Industry 4.0 industrial automation systems. According to the Arrowhead Framework development team, there is still some substantial work remaining for completely standardising this framework. However, the current version of this framework developed with these core systems with their corresponding services is stable [**34**] The current version of Arrowhead Framework is 4.2.0 [**32**].

2.12 Summary

This chapter covered the key concepts of the Arrowhead Framework. It elucidates on the different concepts based on which Arrowhead Framework was created such as SOA and SoS. It further discussed on the local cloud approach in this Framework and the advantages of a local cloud. In addition, it covered the design principles of this Framework for developing Arrowhead Framework Systems. The different mandatory and support core systems were also discussed in this chapter. In the following chapter, the modeling languages and methodologies are discussed in detail with focus on SysML. In addition, the concept of DSL and the need for one is elucidated.

This page is intentionally left blank.

Chapter 3

Systems Modeling Language

3.1 Introduction

Systems Modeling Language (SysML) is a graphical modeling language used for system engineering purposes [7]. It helps in design, specification, analysis, verification and validation of a wide range of systems and SoS [7]. The design of SysML has ensured that powerful constructs which are not cumbersome can be used for modeling a broad range of systems engineering applications [21]. This chapter describes the inception of SysML, its relationship with UML, the set of diagrams it provides and its usefulness. Furthermore, it discusses on the SYSMOD methodology. In addition, this chapter covers the concept of DSL and the need for a DSL for Arrowhead Framework.

3.2 Inception of SysML

The SysML was initially proposed by International Council on Systems Engineering(INCOSE) as a customisation of UML for systems engineering purposes in 2001 [23]. UML is a general purpose developmental modeling language used in software engineering which helps in specifying, visualizing and documenting models along with their structure and design, specifically for software systems. UML is a modeling language used predominantly in the field of software engineering [22]. The need for a general purpose modeling language in the field of systems engineering with specific constructs lead to the inception of SysML. Both UML and SysML have been adopted as a standard by Object Management Group(OMG) [23]. SysML was published as the ISO/IEC 19514:2017 standard (Information technology – Object Management Group Systems Modeling Language) [9]. SysML was adopted by OMG in 2006 and starting from this year, it has been published as OMG SysML [23].

3.3 Relationship between SysML and UML

As mentioned earlier, UML is a modeling language used predominantly in the field of software engineering. An important feature of this language is that it provides the options for extending it for domain specific purposes. Eventually, SysML was developed by extending UML for systems engineering application purposes. SysML uses a subset of UML with its own set of extensions [8].

This relationship between UML and SysML can visualised in the Venn Diagram in Figure 3.1 [21]. There are three parts in this Venn Diagram. The section highlighted by purple color with the text "UML not required by SysML" is the first part. This is a part of UML which is not required to implement SysML. The section created by the intersection of the two circles and highlighted by both the colours with the text "UML reused by SysML" is the second part. This is a part of UML which is required for implementing SysML and it illustrates the UML modeling constructs which are reused by SysML. The section highlighted by white colour with the text "SysML extensions to UML" is the third part. This is a part of SysML which represents new modeling constructs defined for SysML which have no equivalent constructs in UML or those that replaces UML constructs [21].

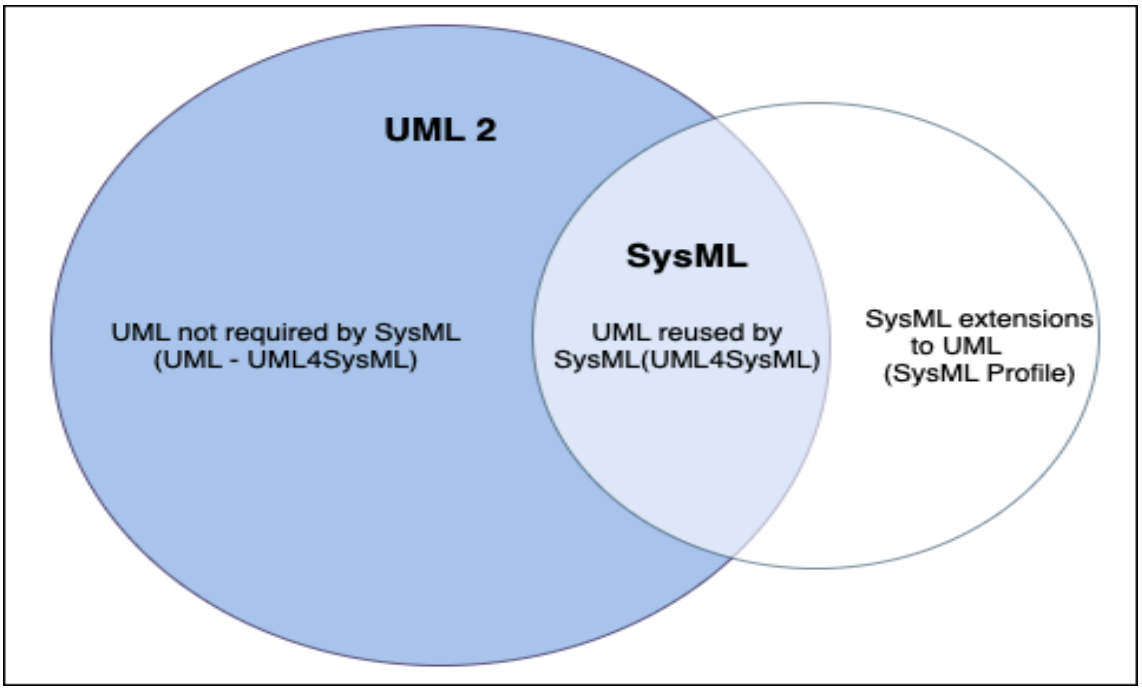

FIGURE 3.1: Relationship between SysML and UML [**21**]

In addition, the Figure 3.2 shows the relationship between SysML and UML alongside other modelling packages. As shown in this diagram, the UML metamodel is imported by SysML. The SysML uses stereotypes which extends the required elements from UML metamodel. Furthermore, SysML also uses other model libraries depicted in this diagram which provides some elements needed for units, dimensions, quantities, values, value types and others [**36**].

3.4 Diagrams provided by SysML

There are nine different types of SysML diagrams which are listed below. These nine diagrams represent the different aspects of a system, that is, the structure, requirement and behaviour [**7**].

1. Block Definition Diagram (BDD)

   The BDD is the primary diagram in SysML which helps to convey the structural information about a system. Furthermore, it helps to express both the possible types of structures that can exist, namely, the structure that exists internally within a system and the external structure in an environment of a system. In addition, it provides the options for expressing the types of constraints that each structure should conform to and types of values that could be used in the operational system as well. A block is the primary structural unit in SysML which is used in a BDD. Several relationship types can be used to represent the relation between different blocks. Furthermore, the exchange of service, data, energy, matter, etc. between blocks can be represented using ports [**7**].

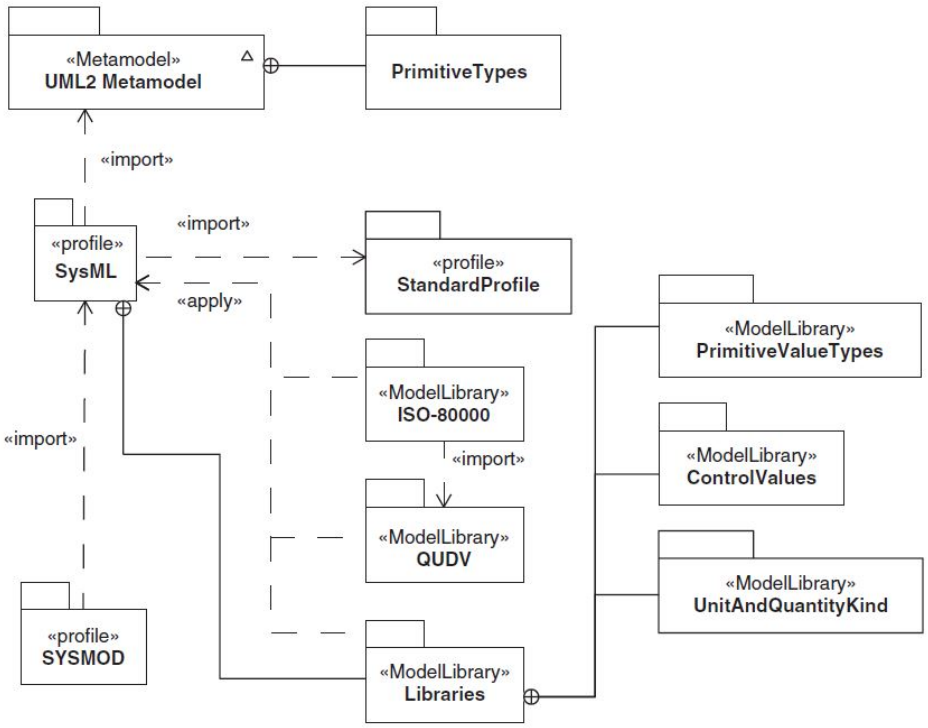

FIGURE 3.2: SysML Architecture Packages [**36**]

2. Internal Block Diagram (IBD)

   An IBD helps in representing the internal structure of a block in terms of the specific parts that exists in a built system, its properties and connectors among these parts. One of the significant aspects of this diagram is that, the information conveyed in an IBD complements that in a BDD and effectively, these two diagrams are created concurrently [**7**].

3. Use case Diagram

   A bird's eye view of a specific behaviour of a system which could be observed from outside the system as a unit of substantial work represents a use case. The Use Case Diagram helps in defining and viewing use cases and their relationship with those actors interacting with it and acquiring some value from the system. It provides a black-box view of the services provided by the system and the interaction with the actors deriving this service. Effectively, it acts as a system context diagram providing a view of the system needed for stakeholders in the early life cycle [**7**].

4. Activity Diagram

   A SysML activity diagram is an extension of the UML activity diagram. It helps in conveying the system's behaviour over time to the concerned stakeholders. It uses a sequence of actions to represent the behaviour of a block or any other structure. Furthermore, the emphasis of this diagram is conveying a continuous system behaviour and flow of items. The item flow that could be represented includes information, data, energy, service or any other physical entity that can be produced or consumed or even conveyed. Moreover, it helps in representing both sequential and concurrent flow [**7**].

5. Sequence Diagram

   Similar to the UML sequence diagram, the SysML sequence diagram also represents information about a system's behaviour over time focussing more on the communication messages between the different parts inside a system. It further helps in conveying a complete and unambiguous behaviour of the system, such as which part of the system performs or invokes each behaviour in the system and the order of occurrence of the behaviours. This effectively helps in using this diagram in the development stage of the system's life cycle [7].

6. State Machine Diagram

   A State Machine diagram helps in representing the complex behaviour of a system. It specifically helps in conveying information about the entire course of a system entity which has several possible life cycles which are difficult to comprehend. In addition, it helps to represent the different stages or states through which an entity could possibly pass during its entire course or life cycles. Furthermore, it helps in representing the conditions and guards only on which the entity transitions from one state to another [7].

7. Parametric Diagram

   A parametric diagram helps in creating the mathematical model of a system that comprises of a congregation of constraints, predominantly, a set of mathematical equations and inequalities. This helps to determine those parameter values which are valid in a minimal operating system. Furthermore, an analysis of critical system parameters from an engineering standpoint could be done. This includes evaluation of important metrics such as system reliability, performance and others. Holistically, this diagram serves as a bridge between requirements and system design, helping in combing these models by capturing constraints based on complex mathematical relations [7].

8. Package Diagram

   A package diagram provides provisions to illustrate the complex organisation of a model into simple comprehensible repositories. This helps the modeler to convey logical groupings of complex structures of the model which eventually aids the stakeholders to navigate through the model structure to locate elements at a granular level. This high level grouping could be based on several criteria such as ease of navigation, access control, configuration management or level of dependency [7].

9. Requirements Diagram

   A requirements diagram is used to convey the requirements for the design of the system and the relationship among them. In addition, it helps in traceability from requirements to the structure and behaviour of the system represented in the aforementioned diagrams. Effectively, it aids the stakeholders to understand the system requirements and trace them as well in the system model [7].

A diagram of a system's model represents only one of the views of the model but is not the model by itself. The classification of these diagrams based on representation of a particular aspect of a system is depicted in Figure 3.3 [7].

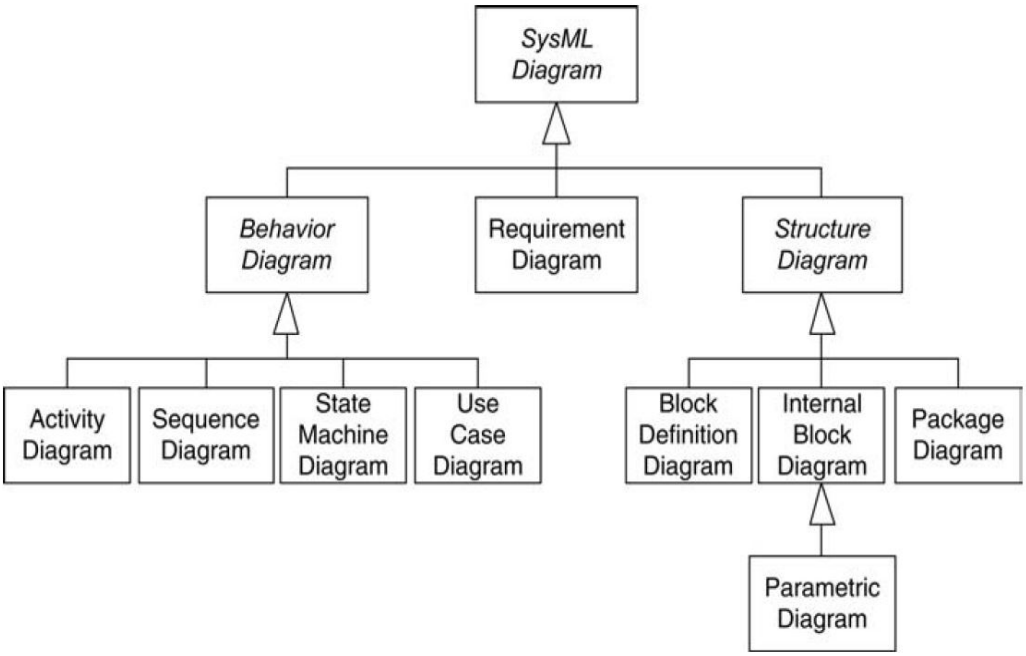

FIGURE 3.3: Classification of SysML diagrams [**7**]

3.5 SysML design specified by OMG

The fundamental design principles for SysML are:

1. Requirements-driven
    - The requirements of UML for System engineering purposes are satisfied by SysML [**21**].
2. UML reuse
    - UML can be reused in SysML wherever it is pragmatically needed for satisyfing the requirements. However, whenever a need for modification arises, only nominal changes can be done to the language. Effectively, it would make SysML easier to implement in those environments where UML is already supported [**21**].
3. UML extensions
    - UML provides extension methods using the profile mechanism which helps to extend the entities in UML for specific purposes. In this manner, in order to satisfy the system engineering requirements, SysML extends UML wherever needed [**21**].
4. Partitioning
    - The basic unit of partitioning in SysML would be a package. Furthermore, in order to limit circular dependencies among model elements, the packages segregate the model elements into logical groupings [**21**].
5. Layering
    - SysML packages are specified as an extension layer to the UML metamodel [**21**].

6. Interoperability

   - UML has the capability for XMI interchange. This capability has been inherited by SysML. Moreover, in order to support interoperability among other engineering tools, ISO 10303-233 data interchange standard is expected to support SysML [**21**].

3.6 Selection of SysML

There are several languages available for different modeling purposes. UML is a prominent modeling language which is predominantly used in the software engineering domain. However, this assignment requires a modeling language suitable for systems engineering which makes SysML a suitable option. SysML provides a requirements diagram which is not part of UML, which helps in capturing the requirements both at higher and granular levels. However, UML offers only a use case diagram which helps in predominantly capturing high-level requirements. Furthermore, SysML offers a Parametric diagram which helps in introducing equations and constraints which helps in building well-formed models. This diagram is not available in UML. In order to model a system, the first step is to collate the requirements, then create the structure of the system, followed by the behaviour of the system which are then well formed with parametric constraint specifications. SysML offers this capability to model a system by encompassing its requirements, structure, behaviour and parametrics.

3.7 The V-Model

The V-model is a visual representation of the system development life cycle. It represents the actions needed to be completed and the corresponding results needed to be achieved in the system development life cycle. This V-model is depicted in the Figure 3.4. It has different stages, starting from the left hand side with collating requirements and analysing these requirements, system analysis and design from these collated requirements, detailed design with logical and physical structures with information flow, implementation of the detailed design and then, unit testing. The right hand side of the V-model includes integration of the different modules and testing them, testing the integration of systems and sub-systems and system acceptance.

However, there are some drawbacks to this V-model. For instance, if there are any faults or vulnerabilities detected in the testing stage, then the entire cycle is followed again for re-designing the system to rectify the defects. This occurs due to the module and system integration testing being completed after the implementation phase. Eventually, there is a necessity created for early validation or continuous validation to be completed to avoid these issues. One of the proposed approaches is the "V in V-model" where virtual integration of system elements is completed at early stages. This allows for virtual integration testing and validation of each and every system component, information flow and related elements to detect faults even before the implementation stage. This "V in V-model" approach is depicted in the Figure 3.5. In the next section, the SYSMOD methodology is elucidated which is a set of guidelines that could be used for model based system development using the aforementioned V-model approach.

3.8 SYSMOD Methodology

SYSMOD is a term coined by Tim Weilkiens [**37**], which is derived from "Systems Modelling". It stands for Systems Modelling Toolbox which is predominantly used for system development, where it helps to bring a system idea to fruition with a system design. It is a logical approach which encompasses both a holistic and granular level of understanding of requirements for system development. Thus, it is a highly requirements driven approach. It provides with a set of

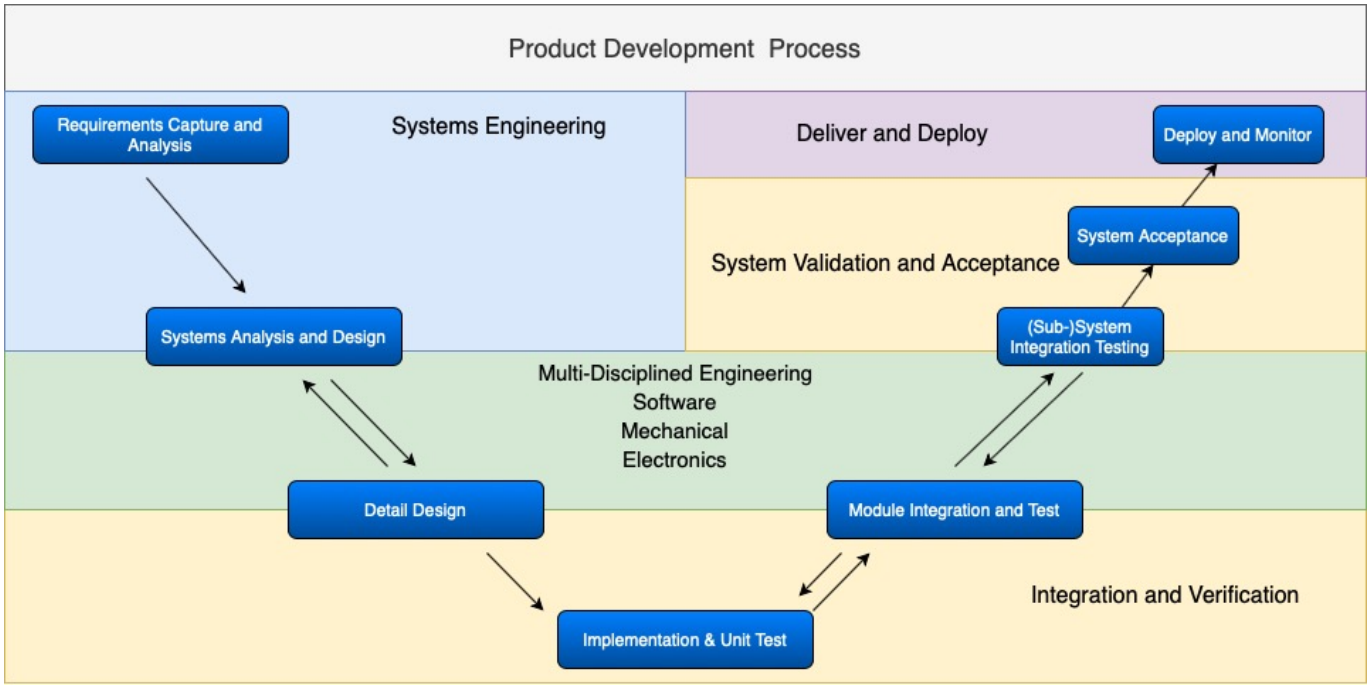

FIGURE 3.4: V-Model system development life cycle [**37**]

tasks, guidelines and best practices for systems modelling. The different tasks encompassed in this methodology are:

- Description of project context

  This is a textual description of the project goals that the team would work together on. It includes the first step of defining the system ideas and objectives. In addition, TRIZ(TIPS - Theory of Inventive Problem Solving) analysis methods, which is a systematic approach to understand and solve challenging problems can be employed for finding new creative notions.

- Identifying the stakeholders

  Individuals within or outside the organisation with interest towards the system, who could possibly influence or contribute to the requirements of the system are considered stakeholders. Identifying these individuals from a wider domain and prioritising these stakeholders would be the next step. This helps in gathering different perspectives on the system which would eventually lead to modelling a granular level of requirements of the system [**37**].

- Modelling the requirements.

  Different set of requirements in several categories need to be modelled. These requirements can be categorised on different basis for better clarity [**37**]. One of the example model would be categorising requirements based on quality characteristics, operations, security and policies [**39**]. Another example requirements model is the FURPS model [**38**] which comprises of:

  * Functional Requirements
  * Usability Requirements
  * Reliability Requirements

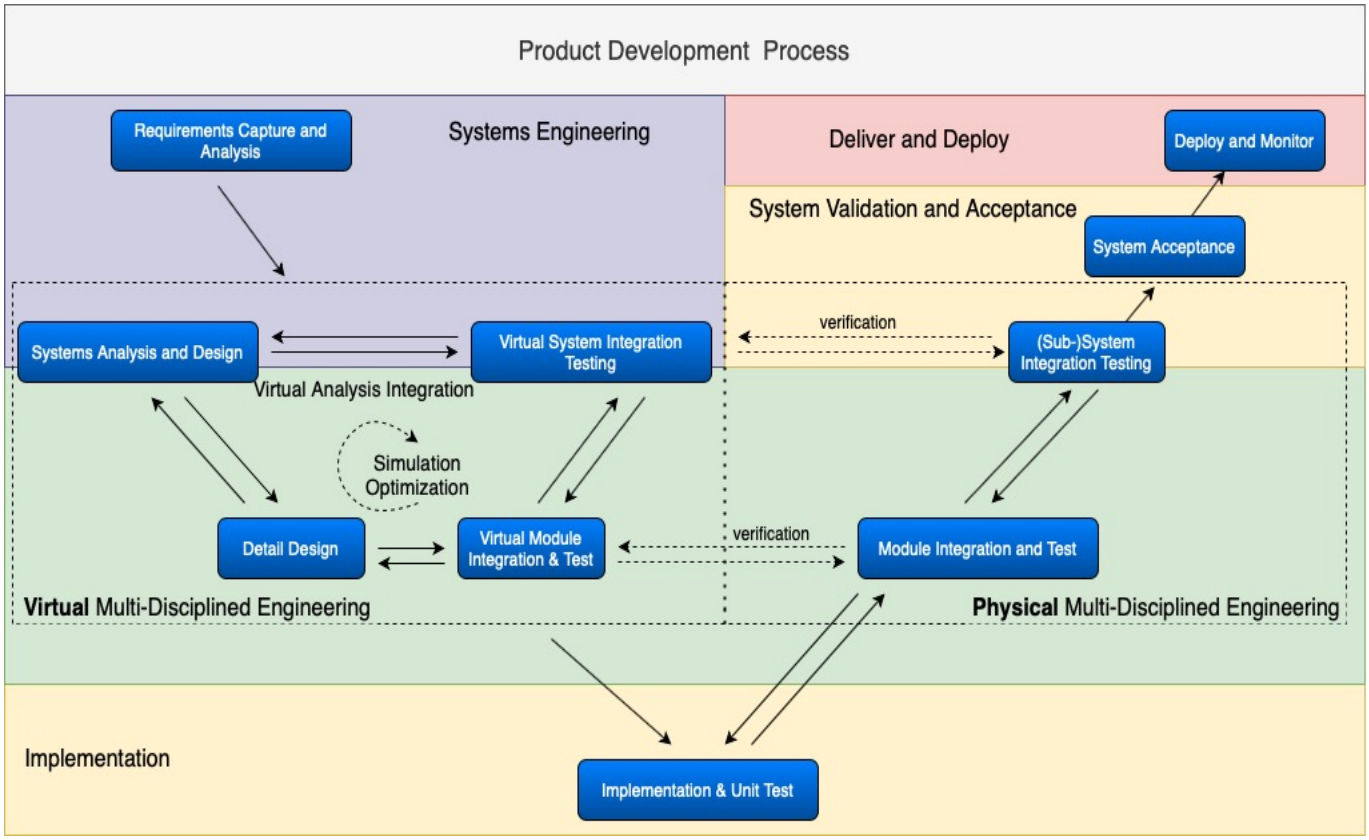

FIGURE 3.5: "V in V-Model" system development life cycle [37]

  - Performance Requirements
  - Supportability Requirements

- Modelling the system context

  For the benefit of the stakeholders, the collated requirements are translated into a pictorial description of the logical and physical structures of the system. This is the system context. It is modelled from a holistic view of the system as well as the individual components are modelled into sub-contexts to present a granular level of detail. Furthermore, context specific viewpoints could be created to demarcate the concerns of the different stakeholders based on their requirements [**40**]. In addition, in this stage of SYSMOD methodology, the process of identifying the system actors is completed [**37**]. Moreover, system context could be perceived as the representation of the highest level of abstraction of the system's structure.

- Modelling information flow

  The modelling of information flow among different components of the system makes it a comprehensible structure. Effectively, modelling the information flow becomes a part of the system context [**40**]. Furthermore, this modelling helps in a clear comprehension of the domain and it serves as the basis for understanding the possible use cases in the system. In addition, it helps in identifying the interaction points in the system [**37**].

- Modelling Use cases

Use cases which are an effect of the collated requirements, represent the different service functionalities provided by the system. The first step in this stage is to identify the use cases. On identification, a top-down approach is employed to model these use cases, where higher level use cases are first modelled, followed by the bottom level use cases [**37**]. Like system context represents the highest level of abstraction of system structure, use cases can be comprehended as the representation of the highest abstraction of system behaviour.

- Description of system processes

  Dependencies among use cases are identified through system processes. System processes represent the logical flow across the different use cases which effectively help in identifying the flow dependencies. In addition, they help in representing the aggregated behaviour of the system [**37**].

- Modelling domain knowledge

  A domain knowledge model is created to represent the domain structures in the project. This presents a coherent structural representation of domain specific ontologies in the project making it easier for the stakeholders to comprehend [**37**].

- Modelling system structure and behaviour

  From the collated requirements and with the modelled system context, a complete structure of the system along with its components, connections, ports and interfaces, operations, properties and others are modelled. Furthermore, the behaviour of the system alongside constraints and parameters, are modelled to realise the use cases modelled in the previous stages.

- Model Simulation

  Models are simulated in run time environments within the modelling tool. Simulation is done to detect flaws in design decisions which is useful at earlier stages since the cost of rectifications would be minimal. In addition, code generators can be employed to produce an executable model of the modelled system which can be used for both simulation and testing purposes [**40**].

- Model Testing

  Testing the model in a test environment, helps in validating the system model to check if the collated requirements are realised in terms of the system structure and behaviour. Furthermore, there can different basis for the tests to be conducted to validate the model for different purposes such as safety analysis, consistency analysis and others. Effectively, this helps in identifying the faults and vulnerabilities in the system model. On identification, these faults can be rectified at minimal cost, which would eventually pave way to the betterment of the system model.

Although, the SYSMOD methodology can be applied using the predominantly used modelling language for systems, which is, the OMG SysML(Object Management Group Systems Modelling Language), it can also be used for other modelling languages such as OMG UML(Object Management Group Unified Modeling Language) and others.

3.9 A need for Domain Specific Language

UML and SysML are generic purpose modelling languages and they lack specificity. The primary building blocks of UML and SysML are better suited to represent components in software engineering and systems engineering domains respectively, and not for a specific domain within them. A modelling language with arrowhead framework constructs would help in easier and better understanding of the models. Furthermore, it would help in improved communication with experts in the arrowhead framework domain. For instance, if a block or actor or generalization relationship in SysML is used to represent arrowhead framework elements and interactions, it might not be comprehendible by those who do not possess knowledge on SysML, but, are working with arrowhead framework. This creates a need to build a DSL for arrowhead framework.

3.10 Domain Specific Language

A DSL could be comprehended as a formal language which is a collation of expressions, symbols or sentences based on a specific formalism, where these elements represent entities specific to a domain. One of the definitions of DSL provided by Arie van Deursen, Paul Klint and Joost Visser [**41**] is

"A domain-specific language (DSL) is a small, usually declarative, language that offers expressive power focused on a particular problem domain."

There are several advantages to using a DSL over a general purpose language. There are:

1. Expressing using domain specific concepts which leads to better understanding and improved communication **[42]**
2. Improved productivity, quality, data longevity, reliability and others **[42]**
3. Productive tooling and highly concise **[42]**

The DSL used in this assignment is for modelling purposes and thus, it can be called as a Domain Specific Modeling Language **[35]**. There are several methods possible with UML for building a new DSL. To understand these methods better, knowledge on the architecture of UML is required. The next section elucidates about the architecture of UML.

3.11 Architecture of the Unified Modeling Language

UML has a four-layered architecture where each of these layers represent different levels of abstraction. These four distinct layers are:

1. M3: Meta-Meta-Model Layer-

   This topmost layer is where the language for specifying meta-models is defined. For instance, this layer contains the MOF language which can be used for creating the structure of meta data. The modelling formalisms of UML are defined in an uniform manner using this MOF **[20]**.

2. M2: Meta-Model Layer-

   This layer could be perceived as an instance of the meta-meta-model layer that encompasses definitions of the meta data structure. The meta-models in this layer are created from the M3 layer model. The UML meta-model created from the MOF belongs to this layer **[20]**.

3. M1: Model Layer-

   This layer could be perceived as an instance of the meta-model layer which encompasses definitions of objects or data in the M0 layer. Furthermore, it defines a language for specifying constructs which can be used for a particular domain. The M2 level meta-models describe the structure of M1 level entities. The models built using UML belong to this layer [**20**].

4. M0: Object or Data Model Layer-

   This layer is an instance of the M1: model layer. It comprises of data or objects [**20**].

3.12 Methods for creating a DSL

There are different options available for defining a DSL. Each of these methods have their own advantages and disadvantages. The method selected for the DSL implementation is based on cost and ease of implementation. There are three main methods available for defining a DSL [**35**] and they are:

1. Defining a new language
2. Extending and modifying an existing meta-model
3. Refining or extending an existing general purpose modelling language to include domain specific constructs

Although, the first method is a direct approach, it has several drawbacks and difficulties with implementation. This option would require a new modelling tool to support this new language with its constructs. Furthermore, it is expensive to accommodate the complex and intricate semantics behind these language constructs. The second method involves modifying the existing UML meta-model by adding new elements and concepts in the meta-model. This is a heavy weight extension method which has similar difficulties involved in its implementation like the first method. The third method is a light weight extension method which uses provision within UML that allows the extension of UML to add domain specific constructs. In addition, it allows the option of re-using the existing tools supporting UML. Furthermore, this method provides the option of using the existing UML model elements. Effectively, a DSL can be created with this UML extension method by meticulously re-using UML elements wherever possible and creating new domain specific elements, only where it is necessary [**35**]. The third method is the most effective one in terms of cost and ease of implementation and this method was selected for this assignment.

3.13 Summary

This chapter covered information on the Systems Modeling Language (SysML), its inception, relationship with UML and the different diagrams it provides. In addition, the V-model product development lifecycle along with SYSMOD methodology was elucidated in detail. The concept of DSL was discussed in detail along with the need for a DSL for Arrowhead Framework. While the need for a DSL has been established, this DSL should have the capability to be used along side an existing modeling language such as SysML. This would ensure that existing tools that support a general purpose modeling language, would also support modeling using the elements from this DSL as well. In addition, SysML elements can be reused to represent generic structures wherever needed along side the elements from the DSL. The UML profile extension method is elucidated in detail in the next chapter along with the different elements part of the created DSL.

This page is intentionally left blank.

Chapter 4

Arrowhead Framework - Domain Specific Language

4.1 Introduction

The need for a DSL for Arrowhead Framework has been established. While the need has been established, the tool needed for this assignment has to be selected based on the requirements of this assignment. One of the major tasks for this assignment was to select the suitable tool for DSL creation and validation. This chapter covers information on the tool selected for the assignment and the methodology for creating a DSL. This chapter further covers the elements created in the Arrowhead Framework profile and the semantics and rationale of the created elements. In addition, the elements in the profile used for creating the model libraries and the created model library elements have also been elucidated.

4.2 DSL Elements

To build a DSL encompassing the set of elements in the domain which would be widely used, there are some steps for DSL design implementation. Considering the Arrowhead Framework, the domain model should be defined with the following elements [**43**]:

- The elements in a DSL should be a core set of language constructs representing the primary concepts in the domain.
- Relationships between the core concepts or constructs.
- A set of attributes of these core concepts.
- A coherent graphical representation of the constructs in the model.
- Semantics of the representation of the model.

4.3 Methodology used for creating a DSL

The first step for creating a DSL is to identify the concepts in the domain. To identify the concepts, an extensive study on the domain should be performed. The next step is to identify the problem space in the domain. This involves interacting with the Arrowhead Framework team to understand their need for the profile. On identification of the problem space, the next step is the identify the language concepts. An extensive study on the different concepts in the UML meta-model should be done. This step would help in identifying the language concepts. At the end of this stage, relevant domain concepts can be defined by mapping the identified domain concepts with the identified language concepts. This step could possibly be performed for several iterations. At the end of this

step, the initial mapping between the domain concepts and language concepts would be completed [**46**]. This would serve as a basis for further mapping of domain concepts at different abstraction levels. At the end of this last stage, all the domain concepts at different abstraction levels would be mapped to different language concepts.

4.4 Tool Selection

Several versions of different modelling tools were considered and evaluated for this assignment. The enterprise or trial versions of these different tools were installed, integrated with external code generators and then, evaluated. Some of the tools considered were IBM Rational Software Architect Designer [**13**], IBM Rational Rhapsody Architect for Software [**5**], Cameo Systems Modeler [**45**] and MagicDraw [**15**]. These tools were evaluated based on the requirements of this assignment. These requirements for the modelling tool were providing support for UML profile extension method, support for modeling using SysML, support for model constraint languages such as OCL, support for in-built custom code or template generation, support for exporting data in terms of built models, created profiles and generated code, support for editing and updating models and time needed to get started with using the tool based on current licence availability. This comparison of the different tools based on the requirements for the Arrowhead Framework DSL assignment is shown in Table 4.1.

Serial No:	Requirements	IBM Rational Software Architect Designer	IBM Rational Rhapsody Architect for Software	Cameo Systems Modeler	MagicDraw
1	Support for UML profile extension method	Yes	Yes	Yes	Yes
2	Support for modeling using SysML	No	Yes	Yes	Yes
3	Support for model constraint languages such as OCL	Yes	No	Yes	Yes
4	Support for in-built custom Code/Template Generation	Yes	No	Yes	Yes
5	Support for exporting data(Models, Generated Code, Profile)	Yes	Yes	Yes	Yes
6	Editing and Updating Models	Yes	Yes	Yes	Yes
7	Ready to use in terms of license availability at the time of requirements analysis	Yes	Yes	Yes	No

TABLE 4.1: Comparison of the different tools based on the requirements for the Arrowhead Framework Domain Specific Language assignment [**45**] [**5**] [**13**] [**15**]

Initially, the decision was inclined towards IBM Rational Rhapsody Architect for Software since it satisfied several requirements. However, it didn't have the feature of custom code/template generator which helps with generating Arrowhead Framework compliant code. Code generator tools such as MDWorkbench that integrates with this tool was also installed and evaluated. However, since IBM Rational Rhapsody Architect for Software didn't support OCL, this tool wasn't the right option for this assignment. On the other hand, Cameo Systems Modeler and MagicDraw satisfied all the primary requirements. However, a license for Cameo Systems Modeler was already available and thus, it was selected as the right tool.

4.5 UML extension mechanism

UML provides the option of extending the existing UML elements. The UML profile component which is a subset of UML metamodel provides this option. This UML profile method does not create new elements in the metamodel. On the contrary, it contains provisions which would extend an existing UML element into a domain specific construct [**1**]. The UML profile comprises of three primary components for extensions and they are represented in Figure 4.1. These components are:

1. Stereotypes-

   Stereotypes are UML elements, that extend an existing UML element to create a new element in a particular domain representing a concept, physical or logical element, relationship or any other element in that domain. A stereotype is represented by the keyword «stereotype», a name and the corresponding meta-element it extends from [**1**].

2. Tagged Values-

   Tagged values are used to define new or additional information in a stereotype, essentially representing the attributes associated with that stereotype. Tagged values are represented with a name and a type [**1**].

3. Constraints-

   Constraints are rules that can be used to restrict the way these new stereotyped elements function or behave based on the domain context. Effectively, this helps in making a model well-formed. Constraints can be expressed in specific languages like OCL or even, natural language [**1**].

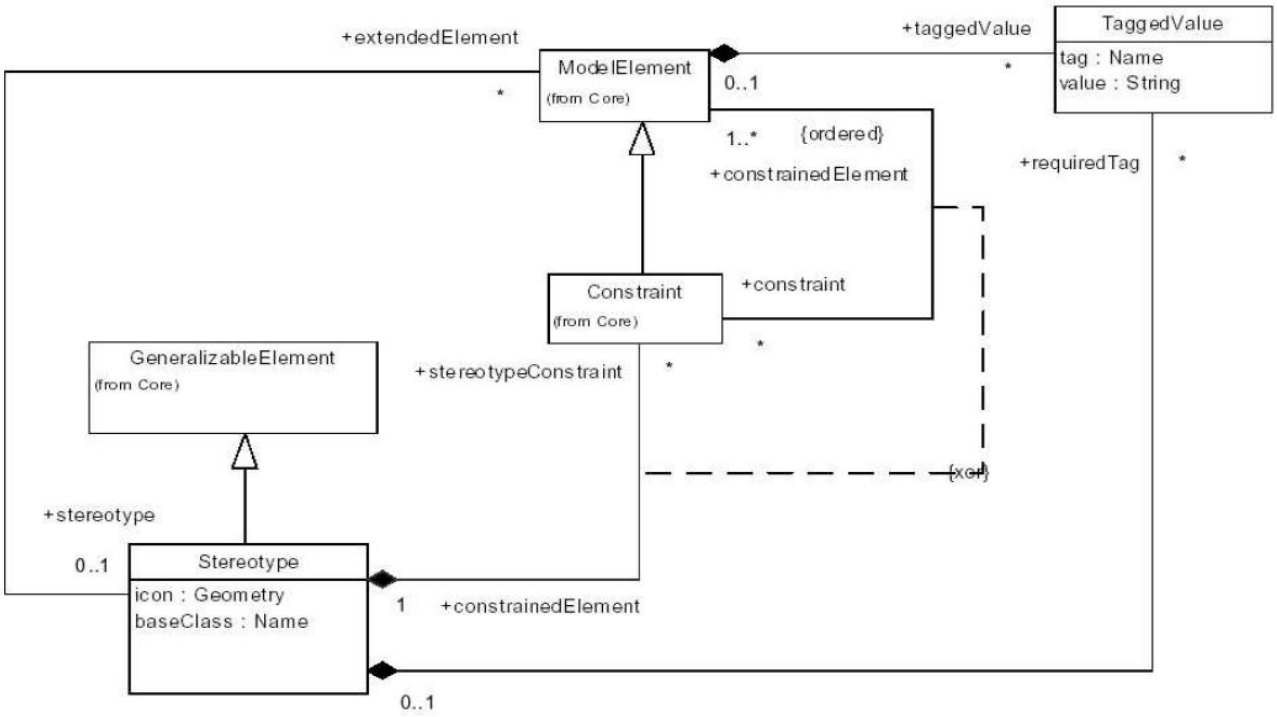

FIGURE 4.1: Extensibility Mechanisms in the UML Metamodel [**44**]

4.6 The Arrowhead Framework DSL

A profile is a type of package which comprises of stereotypes, which customize the UML for a domain specific purpose. A basic profile of the Arrowhead Framework has already been created by the IncQuery Labs team [**17**]. This profile covers quite some important concepts in the Framework. However, this profile doesn't cover every aspect of this framework and there are some concepts which need to be coherently defined. For example, their profile comprises of some elements which are being used to represent two different concepts. On one hand, these elements have been used to represent the different documents that are part of the documentation model in the Framework. On the other hand, these same elements represent the concepts in the Framework such as Systems, Services and others. Using the same element for representing two different concepts make it quite ambiguous for the modeler. Furthermore, there is a lack of clear representation of how the documentation model is related to the primary concepts in the profile.

In addition, on discussing with one of the members from the IncQuery Labs team, they had communicated that one of the primary goals for the Arrowhead Framework DSL was to migrate from document based communication approach to a model based communication approach. For this goal to be realized, they wanted to build a complete profile which would encompass all the concepts at different levels of abstraction. This was taken up as an additional goal in this assignment. Furthermore, some of the basic constructs defined in the profile by IncQuery Labs team have been reused in this assignment, in a way extending their profile [**17**].

The methodology defined in section 5.3 has been used to build this DSL. A conceptual model needs to be created as a starting point to ensure that the concepts are stable and unambiguous with explicit semantic relations [**12**]. Initially, the concepts in Arrowhead Framework were identified. Then, the concepts in the UML metamodel were identified. These identified concepts in Arrowhead Framework were mapped to the corresponding concepts in UML. This initial mapping between the two concepts has been depicted in Table 4.2. This initial mapping was used as a basis for creating the elements in the Arrowhead Framework DSL.

4.7 Elements in the the Arrowhead Framework profile

The complete Arrowhead Framework DSL comprises of three parts. The first part is the profile comprising of the domain elements in terms of stereotypes and tagged values. The second part is the mandatory core model library which comprises of the frequently used mandatory core systems and services. The third part is the support core model library which comprises of the frequently used support core systems and services. The initial mapping between the two concepts depicted in Table 4.2 was used as a basis for building the elements in the profile. The different elements part of the Arrowhead Framework concepts were identified and they are defined in the profile. This was an iterative process and all the elements of the profile were eventually defined. The set of all elements in the profile and their mapping with UML elements, relationship with SysML elements and other Arrowhead Framework elements are clearly depicted in the Table 4.3. This Table 4.3 can be comprehended as an extension of the Table 4.2 representing the initial mapping between the two concepts. The different elements that are part of the Arrowhead Framework profile are listed in the upcoming sub-sections.

4.7.1 Abstract SoS

Semantics and Rationale: The Abstract SoS stereotype extends the metaclass Class representing the concept of abstract SoS. In addition, it inherits from the Block stereotype in SysML and also, it is composed of the Abstract System stereotype. It is an abstraction of the SoS with its main functionalities and comprises of the abstract systems. Since its an abstract class, it doesn't contain

Arrowhead FrameworkConcepts	UML Concepts
System of Systems	Class
Local Cloud	Class
System	Class
Device	Class
Service(Includes the technological information)	Class
Deployed Elements (Deployed element specific information)	Property, Artifact
HTTP Method (POST, GET, PUT, DELETE, PATCH)	Operation
Data Semantics	Class
ServiceExchange	Connector, InformationFlow
Input and Output Parameters	Class
Arrowhead Framework Description Documents (Blackbox & Whitebox)	Artifact

TABLE 4.2: Initial mapping of Arrowhead Framework specific concepts to UML concepts

any specific technological definitions. This stereotype can be applied to any abstract Arrowhead Framework compliant SoS.

4.7.2 SoS

Semantics and Rationale: The SoS stereotype extends the metaclass Class representing the concept of concrete SoS. It inherits from the Abstract SoS stereotype and it is composed of the System stereotype. Since its a concrete class, it represents information on the technologies used for implementing the SoS and the different systems that are contained in it. Furthermore, similar to any other concrete class, the SoS can also be instantiated. This stereotype can be applied to any Arrowhead Framework compliant SoS.

4.7.3 LocalCloud

Semantics and Rationale: The LocalCloud stereotype is an extension of the metaclass Class. It inherits from the SoS stereotype and it is composed of device stereotype. Effectively, it inherits the composition relationship of SoS. Hence, it is composed of both devices and systems. It represents the concept of Local cloud demarcated by administrative boundary and the entities contained in it. This stereotype can be applied to any local cloud in the framework comprising of Arrowhead Framework compliant devices hosting systems that provide services.

4.7.4 Abstract System

Semantics and Rationale: The Abstract System stereotype extends the metaclass Class and inherits from the Block stereotype in SysML. This is an abstraction of a System which represents any abstract Arrowhead Framework compliant system. This stereotype can be applied to any abstract mandatory core system, support core system or application system.

Serial No	Arrowhead Framework Element Name	UML element to be extended	Relationship with SysML element	Relationship with Arrowhead Framework element	Other related information
1	Abstract SoS	Class	Generalization of Block		
2	SoS	Class		Generalization of Abstract SoS	
3	Local Cloud	Class		Generalization of SoS	
4	Abstract System	Class	Generalization of Block	Is a part of Abstract SoS	
5	System	Class		Is a part of SoS	System Type should be specified
6	Device	Class	Generalization of Block	Is a part of Local Cloud	
7	Abstract Service	Class	Generalization of InterfaceBlock		
8	Service	Class		Generalization of Abstract Service	Should include technological information such as security, protocol, semantics and others
9	DeployedEntity	Property, Artifact	Generalization of Part Property		
10	DeployedSystem	Property, Artifact		Generalization of DeployedEntity	Should include System specific attributes such as Address and port information
11	DeployedDevice	Property, Artifact, Device		Generalization of DeployedEntity	Should include Device specific attributes such as MacAddress and others
12	DeployedLocalCloud	Property, Artifact		Generalization of DeployedEntity	
13	HTTP Method	Operation			Should include the path attribute
14	POST	Operation		Generalization of HTTP Method	Should include the path attribute
15	GET	Operation		Generalization of HTTP Method	Should include the path attribute
16	PUT	Operation		Generalization of HTTP Method	Should include the path attribute
17	DELETE	Operation		Generalization of HTTP Method	Should include the path attribute
18	PATCH	Operation		Generalization of HTTP Method	Should include the path attribute
19	Data Semantics	Class			Should include semantic related attributes such as Ontology, Schema and others
20	MetaDataEntry	Class			
21	Security Type	Enumeration (UML element reused)			Should include the different security types
22	System Type	Enumeration (UML element reused)			Should include the different system types
23	Semantic Type	Enumeration (UML element reused)			Should include the different semantic types
24	Encoding Type	Enumeration (UML element reused)			Should include the different data encoding types
25	Compression Type	Enumeration (UML element reused)			Should include the different compression types
26	ServiceExchange	Connector, Information Flow			Should be used between a producer and consumer
27	InputParameter	Class			Should include the data encoding type
28	OutputParameter	Class			Should include the data encoding type
29	Document	Artifact			Should include the document name attribute
30	SoSD	Artifact		Generalization of Document	
31	SoSDD	Artifact		Generalization of Document	
32	SysD	Artifact		Generalization of Document	
33	SysDD	Artifact		Generalization of Document	
34	SD	Artifact		Generalization of Document	
35	IDD	Artifact		Generalization of Document	
36	CP	Artifact		Generalization of Document	
37	SP	Artifact		Generalization of Document	

TABLE 4.3: Final mapping of Arrowhead Framework elements to UML and SysML concepts

4.7.5 System

Semantics and Rationale: The System stereotype extends the metaclass Class and inherits from the Abstract System stereotype. It inherits the semantics of abstract system, representing any Arrowhead Framework compliant system which may or may not be hosted on a device. It can be applied to any mandatory core system, support core system and application system. It has a tagged value named Type which takes up the values from the defined enumeration System Type.

Concept of System in Arrowhead Framework: A system in the Arrowhead Framework is an entity in a local cloud that registers with the mandatory System Registry and is capable of producing or consuming Arrwohead Framework compliant services while relinquishing its authority over the services provided by it, to the orchestration system in that local cloud. Furthermore, a system should have the capacity to create new services on demand and be able to be configured dynamically. Moreover, a system can produce any number of services and can consume any number of services [**4**].

In addition, a system producing a service that can be consumed by other systems is called a service producer. On the contrary, a system which consumes services produced by other systems is called a service consumer. An exchange of service can happen between a service producer and a service consumer.

4.7.6 Device

Semantics and Rationale: The Device stereotype is an extension of the metaclass Class and also, it inherits from the Block stereotype in SysML. This stereotype represents the concept of Arrowhead Framework compliant devices contained within the local cloud.

Concept of Device in Arrowhead Framework: Any entity which is a hardware or machine which has the capability to compute, communicate and store data, and also, host one or many arrowhead framework compliant systems along with their services is an arrowhead framework device. Furthermore, a device would become arrowhead framework compliant only if [**4**]:

- It can be bootstrapped into an arrowhead framework local cloud
- It can have new systems and their services dynamically installed in them
- It can be registered to the DeviceRegistry System
- It can be dynamically configured

The profile diagram depicting the relationship between the aforementioned stereotypes is shown in Figure 4.2.

4.7.7 Abstract Service

Semantics and Rationale: The Abstract Service stereotype is an extension of the metaclass Class and it inherits from the Interface Block stereotype in SysML. It represents the concept of an abstract service which is Arrowhead Framework compliant.

4.7.8 Service

Semantics and Rationale: The Service stereotype is an extension of the metaclass class and it inherits from the abstract class service. It represents the concept of Arrowhead Framework compliant

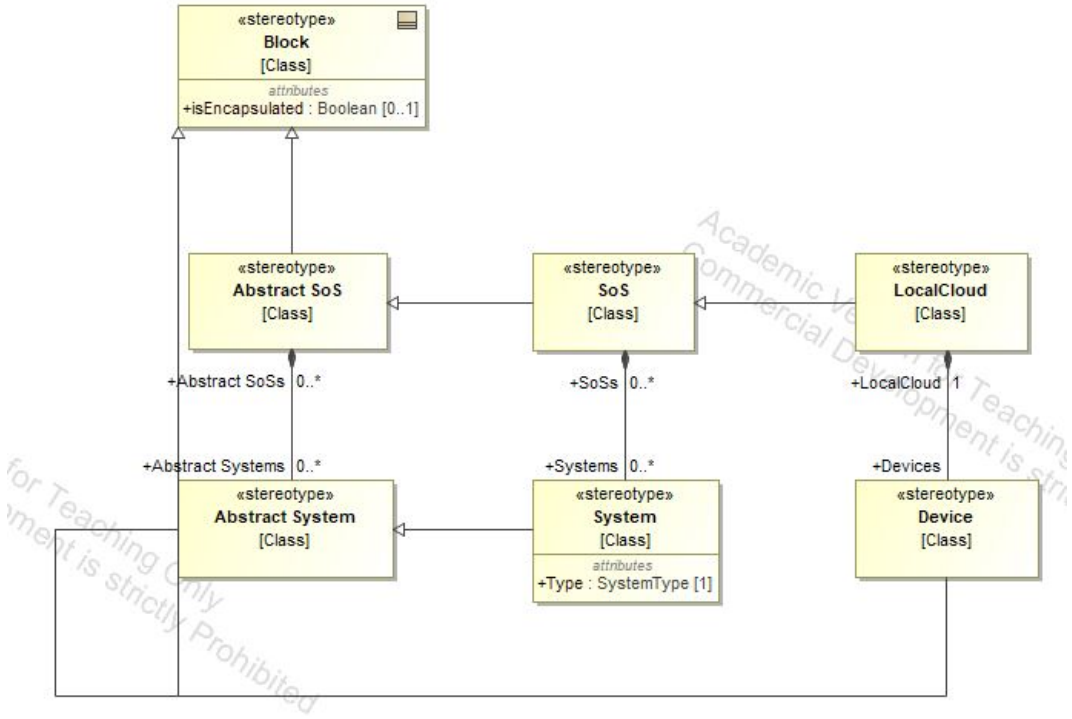

FIGURE 4.2: Profile Diagram comprising of SoS, Local cloud, System and Device stereotypes

service which could be provided or consumed by the system. It has several tagged values contained in it and they are listed below:

- Encoding - It represents the concept of type of encoding used for the specific service. The only type of values it can take is from the defined enumeration Encoding Type.
- Security - It represents the concept of security type for the specific service. The only type of values it can take is from the defined enumeration Security Type.
- Protocol - It represents the concept of type of protocol on which the specific service has been implemented. The only type of values it can take is from the defined enumeration Protocol Type.
- Compression Algorithm - Its a representation of the concept of the type of compression algorithm used for the specific service. The only type of values it can take is from the defined enumeration Compression Type.
- Semantics - It represents the concept of the type of semantics used for the specific service. The only type of values it can take is from the defined enumeration Semantics Type.
- EndOfValidity - It represents the concept of validity time for the specific service. It can take any string values.
- Path - It represents the concept of the path identifier for the specific service. It can also take any string values.
- Version - It represents the concept of the version identifier of the specific service. It can also take any string values.

Concept of Service in Arrowhead Framework: As mentioned earlier, the exchange of information between a producer and consumer is represented by an entity called as Service [**31**]. It can be comprehended as a software unit which is a logical representation of a specific task or holds a specific information [**30**]. Furthermore, a service is reusable, non-context specific and should be capable of being discovered by other systems. Moreover, a service in Arrowhead Framework could have associated meta data and it could support non-functional requirements such as different layers of reliability, security and real-time conditions [**31**].

In order for a service to be Arrowhead Framework compliant, it should be able to register with the mandatory core systems and be produced or/and consumed by an Arrowhead Framework compliant system. In addition, a particular service can be produced by a single system or even multiple different systems [**4**]. A service, for example, sensor data can be offered by multiple systems which can be consumed by other systems.

The profile diagram depicting the relationship between the service and abstract service stereotypes is shown in Figure 4.3.

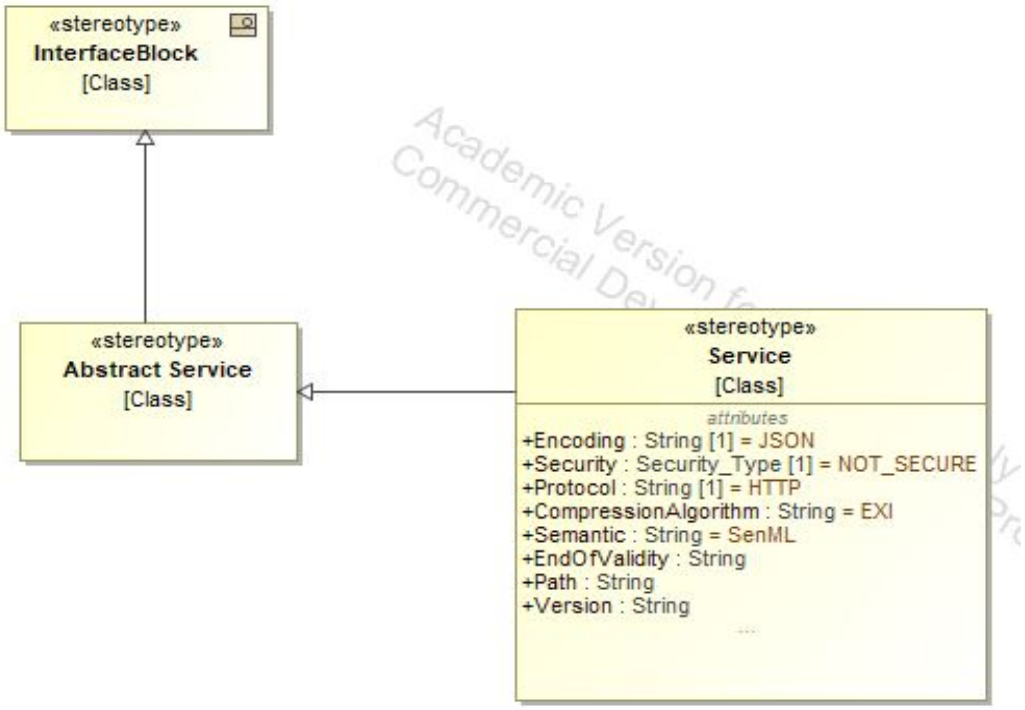

FIGURE 4.3: Profile Diagram comprising of abstract service and service stereotypes

4.7.9 DeployedEntity

Semantics and Rationale: The DeployedEntity is an extension of the metaclass Property and Artifact, and it inherits from the PartProperty stereotype in SysML. It represents any compliant entity deployed with the framework. It has a tagged value named identifier which represents the unique identifier of this entity and it can take any string values.

4.7.10 DeployedSystem

Semantics and Rationale: The DeployedSystem is an extension of the metaclass Property and Artifact, which inherits from the aforementioned DeployedEntity stereotype. It represents the concept of any Arrowhead Framework compliant system deployed within the local cloud. It has four tagged values contained in it and they are listed below.

- Address - It represents the concept of network address for the specific deployed system. It can take any string values in a specific format where numbers in the string are separated by three periods. (00.00.00.00) is an example of this format.
- Port - It represents the concept of port number of the specific Deployed System. It can take any integer values.
- AuthenticationInfo - It represents the authentication information used in the deployed system. It can take any string values.
- Metadata - It represents the concept of meta data associated with the deployed system. It can take values of the type MetaDataEntry which is a defined stereotype.

4.7.11 DeployedDevice

Semantics and Rationale: The DeployedDevice is an extension of the metaclass Property, Artifact and Device, and inherits from the DeployedEntity stereotype. It represents the concept of any Arrowhead Framework compliant device deployed within the local cloud.

- Metadata - It represents the concept of meta data associated with the deployed device. It can take values of the type MetaDataEntry which is a defined stereotype.
- MacAddress - It represents the concept of MacAddress for the specific device deployed within the local cloud. It can take any string values.
- NetworkInterface - It represents the concept of Network interface of the specific deployed device. It can take any string values.
- MacProtocol - It represents the concept of the Medium Access Control protocol associated with the deployed device. It can also take any string values.

4.7.12 DeployedLocalCloud

Semantics and Rationale: The DeployedLocalCloud is an extension of the metaclass Property and Artifact, and it inherits from the DeployedEntity stereotype. It represents the concept of a local cloud deployed with the framework. Thus, this stereotype can be applied to any local cloud deployed with the framework. It has a tagged value named GatekeeperSystemName which represents the name of the Gatekeeper system in that deployed local cloud for inter-cloud communication. This tagged value can take any string values.

The profile diagram depicting the relationship between the deployed entity, deployed local cloud, deployed device and deployed system stereotypes is shown in Figure 4.4.

4.7.13 HTTP Method

Semantics and Rationale: The HTTP Method is an extension of the metaclass Operation which represents the HTTP method type for any operation within a service. It has a tagged value named path which represents the Uniform Resource Locator (URL) sub-path used in the specific operation. This can take any string values.

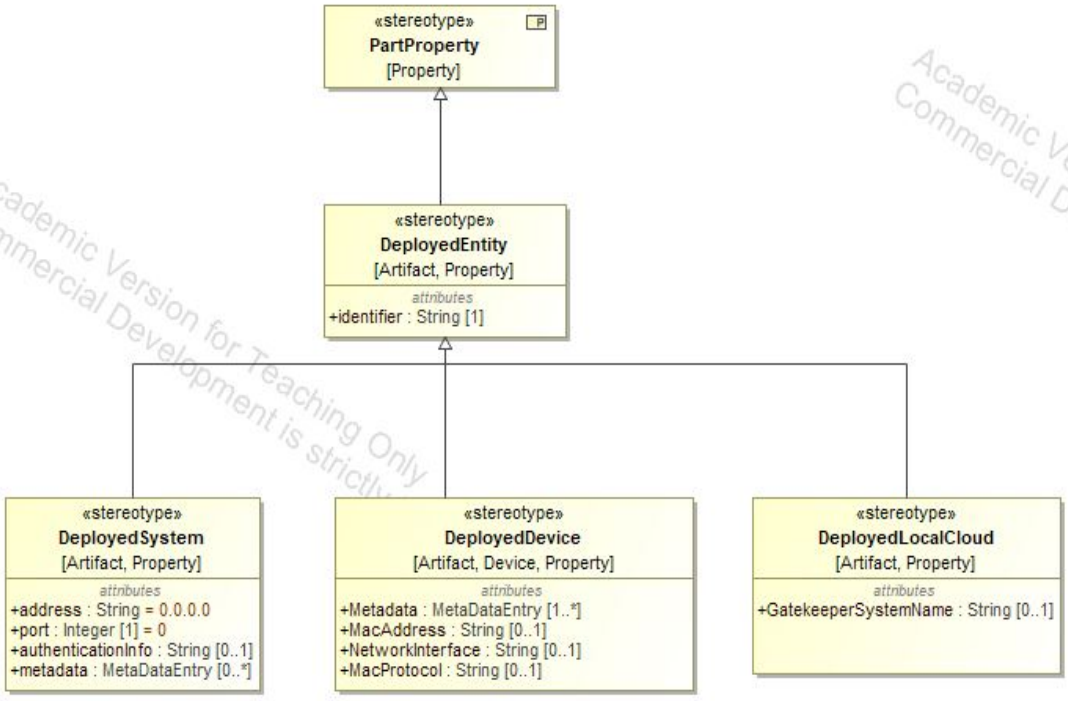

FIGURE 4.4: Profile Diagram comprising of Deployed Entity, Deployed Local cloud, Deployed System and Deployed Device stereotypes

4.7.14 POST

Semantics and Rationale: The POST stereotype is an extension of the metaclass Operation and inherits from the HTTP Method stereotype. It represents any HTTP POST method used in an operation.

4.7.15 GET

Semantics and Rationale: The GET stereotype is an extension of the metaclass Operation and inherits from the HTTP Method stereotype. It represents any HTTP GET method used in an operation.

4.7.16 PUT

Semantics and Rationale: The PUT stereotype is an extension of the metaclass Operation and inherits from the HTTP Method stereotype. It represents any HTTP PUT method used in an operation.

4.7.17 DELETE

Semantics and Rationale: The DELETE stereotype is an extension of the metaclass Operation and inherits from the HTTP Method stereotype. It represents any HTTP DELETE method used in an operation.

4.7.18 PATCH

Semantics and Rationale: The PATCH stereotype is an extension of the metaclass Operation and inherits from the HTTP Method stereotype. It represents any HTTP PATCH method used in an operation.

The profile diagram depicting the relationship between the different operation method stereotypes is shown in Figure 4.5.

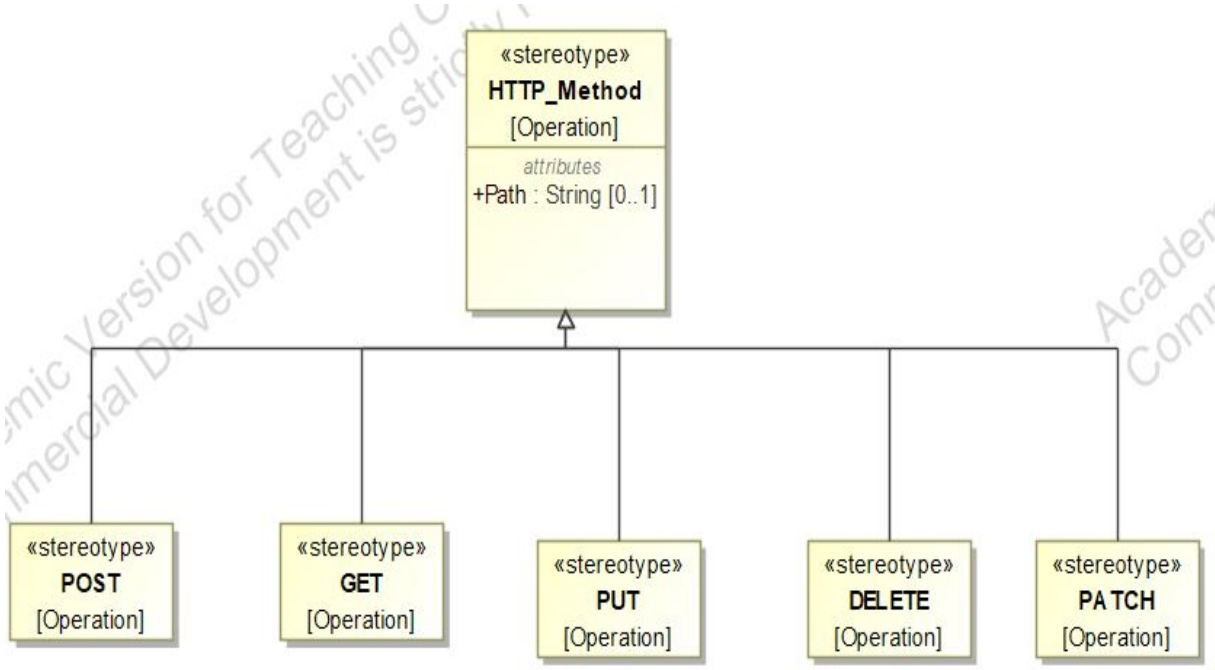

FIGURE 4.5: Profile Diagram comprising of the different operation method stereotypes

4.7.19 Data Semantics

Semantics and Rationale: The Data Semantics stereotype is an extension of the metaclass Class. It is a representation of the concept of semantics of the data involved in the framework to make the data understandable in the model. It could be used to define the type of semantics used for the data, which could be one or many, within the same local cloud. It has three tagged values defined in it and they are SemanticModel, Ontology and Schema. SemanticModel represents the semantic data model used for the specific data. Ontology represents the type of ontology used for the data such as entity, relation, role or any other type of ontologies. Schema represents the type of schema used in the data. All the aforementioned tagged values can take up any string values.

4.7.20 MetaDataEntry

Semantics and Rationale: The MetaDataEntry stereotype is an extension of the metaclass Class which represents the concept of metadata associated with a device or system. It has a tagged value named Data which can take up any string values.

4.7.21 Security Type

Semantics and Rationale: The Security Type is an enumeration with an applied stereotype of ValueType. It contains the enumeration literals which are Not_Secure, Certificate_Based and Token_Based.

4.7.22 System Type

Semantics and Rationale: The System Type is an enumeration with an applied stereotype of ValueType. It contains the enumeration literals which are Mandatory_Core_System, Support_Core_System, Application_System and Management_System.

4.7.23 Semantic Type

Semantics and Rationale: The Semantic Type is an enumeration with an applied stereotype of ValueType. It contains the enumeration literals which are SenML, SensML and others.

4.7.24 Encoding Type

Semantics and Rationale: The Encoding Type is an enumeration with an applied stereotype of ValueType. It contains the enumeration literals which are JSON, XML, CBOR and others.

4.7.25 Compression Type

Semantics and Rationale: The Compression Type is an enumeration with an applied stereotype of ValueType. It contains the enumeration literals which are EXI and others.

4.7.26 ServiceExchange

Semantics and Rationale: The ServiceExchange is a stereotype which extends the metaclass Connector and InformationFlow. It represents the concept of service exchange between a service provider and a service consumer in the framework and also, the inter-connection between them.

4.7.27 InputParameter

Semantics and Rationale: The InputParameter is a stereotype extending the metaclass Class. It represents the input parameter for an operation in a service. This stereotype should be applied to all parameters which are provided as an input to the operations in a service. It has one attribute named data encoding type which specifies the type of encoding used for the data. This attribute can only take up the values of the enumeration Encoding Type.

4.7.28 OutputParameter

Semantics and Rationale: The OutputParameter is a stereotype extending the metaclass Class. It represents the output parameter for an operation in a service. This stereotype should be applied to all parameters which are provided as an output to the operations in a service. It has one attribute named data encoding type which specifies the type of encoding used for the data. This attribute can only take up the values of the enumeration Encoding Type.

4.7.29 Document

Semantics and Rationale: The Document stereotype is an extension of the metaclass Artifact. It represents any document that is used in the Arrowhead Framework for design, implementation, deployment, maintenance or troubleshooting. This document further represents any higher level document that is generic or a lower level document that is specific. This stereotype can be applied to any document that is created in the model for any generic or specific purposes in the Framework. It contains an attribute named DocumentName which is used to specify the name of the document.

4.7.30 SoSD

Semantics and Rationale: The System of Systems Description (SoSD) stereotype is an extension of the metaclass Artifact which represents the SoS Description document. The System of Systems Description(SoSD) is a Black Box SoS level documentation which presents an abstract view of a SoS. Its description encompasses the main functionalities of the SoS and its generic architecture without specifying the precise technologies used in its instances. This document presents a high level overview of the SoS where every independent system is represented by main building blocks. Furthermore, some non-functional requirements are included in this document which are associated with reliability, resource usage and QoS. In addition, it includes security requirements which are generic and non-technical[**33**].

4.7.31 SoSDD

Semantics and Rationale: The SoSDD stereotype is an extension of the metaclass Artifact which represents the SoS Design Description document. The System of Systems Design Description(SoSDD) is the White Box SoS level documentation. It presents the description of the implementation for a specific scenario encompassing the complete description of the precise technologies used and the corresponding setup. Moreover, this document includes a description of the logical implementation of the primary functionalities in terms of use cases, structural and behavioral diagrams. Furthermore, this document covers the detailed description of the physical implementation including granular level of information such as the physical constraints and the physical location of devices and others. In addition, it covers description the realisation of the implemented non-functional requirements[**33**].

4.7.32 SysD

Semantics and Rationale: The SySD stereotype is an extension of the metaclass Artifact which represents the System Description document. The System Description document is the Black Box systems level documentation. This document encompasses information on the definition of the services provided and consumed, and the respective interfaces, without describing the exact technical implementation.Furthermore, this document coerces the use of formal or semi-formal models to accommodate descriptions encompassing semantic related information of systems which would pave way towards assessing the interoperability of different systems[**33**].

4.7.33 SysDD

Semantics and Rationale: The SySDD stereotype is an extension of the metaclass Artifact which represents the System Design Description document. The System Design Description document is the White Box systems level documentation. This document contains the description of the technological implementation of the arrowhead framework systems. This document has been made

optional and thus, in cases where the actual implementation cannot be exposed or when the implementing company's knowledge might be revealed, this document can be avoided. In these cases, only the use of SysD would suffice [33].

4.7.34 IDD

Semantics and Rationale: The IDD stereotype is an extension of the metaclass Artifact which represents the Interface Design Description document. The interface design description provides a detailed description of the actual implementation of the service. Service identifiers for specific service implementations are defined in this document. Furthermore, it makes reference to the Communication profile document for specifying the communication protocol information and encoding, encryption and compression techniques used. In addition, it makes reference to the semantic profile document for specifying the semantics of the information and data used [33].

4.7.35 SD

Semantics and Rationale: The SD stereotype is an extension of the metaclass Artifact which represents the Service Description document. A Service Description document provides an abstract description of the necessities of a system in order to provide or consume a service. It further provides the primary objectives and functionalities of the service alongside its abstract interfaces. Developers of an arrowhead framework compliant system compose this document which describes the application services. Each and every service has a set of non-functional requirements such as resource usage, response time, reliability, Quality of Service(QoS) and others which must be specified with detailed description as well [33].

4.7.36 CP

Semantics and Rationale: The CP stereotype is an extension of the metaclass Artifact which represents the Communication Profile document. The Communication Profile document is part of the service level documentation and it encompasses technical information pertaining to data encoding, data compression, data encryption and the communication protocol used in the actual implementation [33].

4.7.37 SP

Semantics and rationale: The SP stereotype is an extension of the metaclass Artifact which represents the Semantic Profile document. The Semantic Profile document is also part of the service level documentation and it encompasses information pertaining to the semantics of the information or data used in the actual implementation [33].

The profile diagram depicting the relationship between the different Arrowhead Framework document stereotypes is shown in Figures 4.6 and 4.7. In this diagram, two primary relationships have been used, namely, realise and refine. The realisation relationship in UML has been defined in OMG UML 2.5 specification [22] as

"Realization is a specialized Abstraction relationship between two sets of model Elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client)" [22]

SysDD is the system design description document which contains the implementation details of a particular system and the System is the actual implementation based on the details specified in SysDD. So, SysDD acts as the supplier and System acts as the client.

The other relationship used in the diagram is refine, which is a type of dependency relationship. This relationship represents that the element at the client end is concrete and the element at the supplier end is abstract [7]. This relationship is used between those documents which are concrete and those that are abstract. For example, SoSDD is a concrete white box document which encompasses every detail of the implementation and functionality whereas SoSD is only an abstract black box document which covers the main architecture details without specifying any technological information.

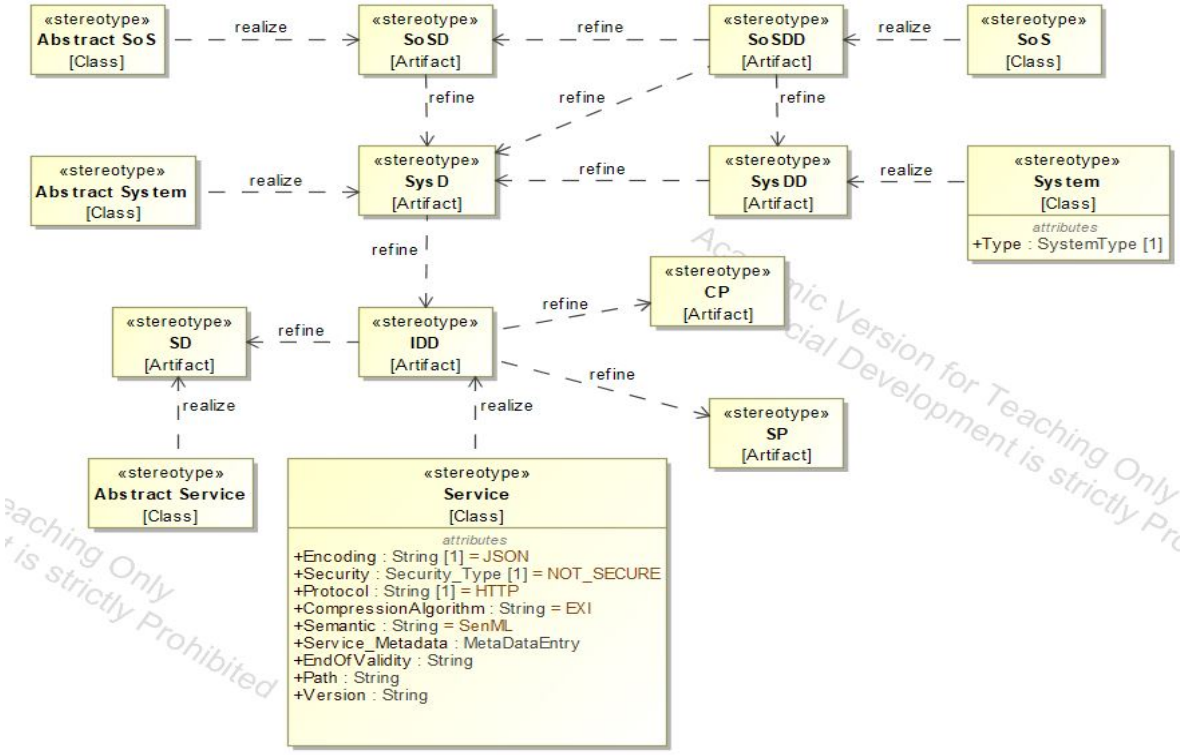

FIGURE 4.6: Profile Diagram comprising of the different Arrowhead Framework document stereotypes

This profile comprising of these different elements is converted into a shared package in Cameo Systems Modeler. This would check for normal dependencies and cyclic dependencies in the profile before converting it into a shared package. On conversion into a shared package, this profile can be imported into any project and its elements can be used for modeling.

4.8 Model Libraries for core systems

A model library is a type of package which comprises of elements, which are usually aimed to be re-used frequently in models [7]. The elements in the Arrowhead Framework profile are used for creating the elements in the model libraries. There are two primary model libraries in this assignment, namely, one for the Mandatory Core Systems and the other for the Support Core Systems.

A brief description of how the elements in the model libraries have been modelled follows. The different core systems and the services provided and consumed by them are first identified. The primary operations that are part of the services are identified along with their input and output parameters and operation method types. The abstract system and system stereotypes have been

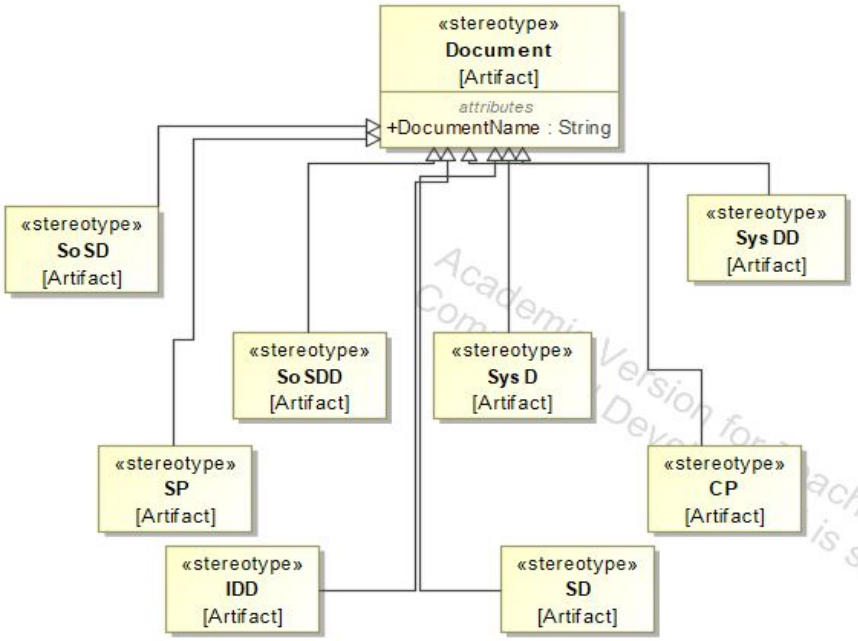

FIGURE 4.7: Profile Diagram comprising of the different Arrowhead Framework document stereotypes

used to represent the abstract and concrete system. The system names are specified in the tag definitions. Similarly the abstract service and service stereotypes are used to represent the services offered by the system. Service specific information is specified in the tagged values. The different operations that are part of the service are added into the abstract service stereotype without specifying any technological details. The technological details of the operation are specified only in the service stereotype. While specifying the operations that are part of the service, the method types are specified using the different HTTP Method stereotypes. While specifying the input and output parameters, the InputParameter and OutputParameter stereotypes have been applied to them based on their roles. In addition, the data semantics used in these parameters are specified using the data semantics stereotype. The provided and consumed services of the system are identified. These services are represented as ports in the system. If a particular service is consumed by the system, then the port is conjugated to represent service consumption by that system. If there are multiple services provided by the system, service stereotypes are created for each provided services. Furthermore, each of these services are added to the system as ports. The two core systems model libraries are elucidated in detail in the upcoming sections.

4.9 Mandatory Core Systems Model Library

The mandatory core systems model library comprises of the three mandatory core systems, namely, the service registry system, authorisation system and orchestration system. The services provided by each of these systems are also included in this library. Furthermore, the operations that are part of each of these services have been included in this library. The input and output parameters, method type and URL sub-path for every client operations have also been included in this library. These mandatory core systems are discussed in detail in the upcoming sections.

4.9.1 Service Registry System

The ServiceRegistry system provides the Service Discovery service which helps in storing information about the services registered by provider systems and then, enabling the discovery of these registered services [**4**]. The primary purposes of this system are [**32**]:

- Registering the services that application systems offer
- Publish information about these registered services to other application systems within the local cloud
- It can add, delete or modify service entries whenever necessary

The different operations part of the Service Registry system is listed in Table 4.4.

Function	**URL Subpath**	**Method**	**Input**	**Output**
Query	/query	POST	ServiceQueryForm	ServiceQueryList
Register	/register	POST	ServiceRegistryEntry	ServiceRegistryEntry
Unregister	/unregister	DELETE	Address, Port, Service Definition, System Name in query parameters	OK

TABLE 4.4: Service Registry Operations [**32**]

4.9.2 Authorisation System

The Authorisation system provides the AuthorisationControl service and TokenGeneration service which helps in managing the access rules for specific systems/services and for session control. Furthermore, it works on authentication of a system requesting a service for consumption and the authorisation to consume that service [**32**].

This system has two databases, one containing intra-cloud access rules and the other containing inter-cloud access rules. The former set of rules define which application system can consume which other application systems' services. On the other hand, the latter set of rules define which local clouds can consume what set of services from the local cloud which contains this specific authorisation system [**32**]. The different operations part of the Authorisation system is listed in Table 4.5.

Function	**URL Subpath**	**Method**	**Input**	**Output**
Get Public Key	/publickey	GET	-	Public Key
Check an Intercloud rule	/intercloud/check	POST	InterCloudRule	InterCloudResult
Check an Intracloud rule	/intracloud/check	POST	IntraCloudRule	IntraCloudResult
Generate Token	/token	POST	TokenRule	TokenData

TABLE 4.5: Authorisation System Operations [**32**]

4.9.3 Orchestration System

The Orchestration system is responsible for late binding between application systems, that is, providing orchestration information to application systems which helps them with information on

where they would have to connect to, in order to consume what services they need [4]. The orchestration rules contain information about [32]:

- Access information of the service provider in terms of network address and port number
- Information on the instance specification of the service provided by a provider system. This information includes details on the base URL, Interface Design Description(IDD) specification and other related metadata.
- Authorisation specific information in terms of access tokens and signatures
- Any other additional information required for establishing interaction

The Orchestration system helps in controlling the deployment of systems and the inter-connections between them. In terms of SOA, it is that system which helps in establishing other systems by providing direction, control and coordination [3]. It further allows the dynamic re-use of existing systems and services to create new services and functionality [10]. The different operations part of the Orchestration system is listed in Table 4.6.

Function	URL Subpath	Method	Input	Output
Orchestration	/orchestration	POST	ServiceRequestForm	Orchestration Response
Start store Orchestration by ID	/orchestration/id	GET	StoreEntryID	Orchestration Response

TABLE 4.6: Orchestration System Operations [32]

4.10 Support Core Systems Model Library

The support core systems model library comprises of the seven support core systems. The services provided by each of these systems are also included in this library. Like the mandatory core systems model library, the operations that are part of each of these services along with the respective input and output parameters, method type and URL sub-path for every client operations have been included in this library. These support core systems are discussed in detail in the upcoming sections.

4.10.1 System Registry System

This system holds information about which systems are registered in a local cloud and their unique identities, their meta-data and the services that they are bound to consume or provide [4]. On the whole, this system is responsible for providing a database to enable devices to register the information about the systems they are hosting and ensure that this information is available for other application systems in that local cloud. Furthermore, it provides the option for devices to add, remove or update this record stored in its database. In addition, it has the capability to generate a client certificate that can be used by the system whenever they want to provide services [32]. The different operations part of the System Registry system is listed in Table 4.7.

4.10.2 Device Registry System

This system stores several information about the devices deployed within the Arrowhead Framework local cloud such as their unique identities, meta-data and the systems deployed into the respective devices [4]. On the whole, this system holds the responsibility to provide the database

Function	URL subpath	Method	Input	Output
Onboard with Name	/onboarding/name	POST	SystemOnboardingWithNameRequest	SystemOnboardingWithNameResponse
Onboard with CSR	/onboarding/csr	POST	SystemOnboardingWithCsrRequest	SystemOnboardingWithCsrResponse

TABLE 4.7: System Registry System Operations [32]

for self-registration of devices and ensuring that this information is available to other systems in that local cloud. In addition, the records stored in the database can be added, deleted or updated in keeping with the arising necessities. Like the System registry system, this system should also be capable of generating a client certificate which helps in devices to register its systems [32]. The different operations part of the Device Registry system is listed in Table 4.8.

Function	URL subpath	Method	Input	Output
Onboard with Name	/onboarding/name	POST	DeviceOnboardingWithNameRequest	DeviceOnboardingWithNameResponse
Onboard with CSR	/onboarding/csr	POST	DeviceOnboardingWithCsrRequest	DeviceOnboardingWithCsrResponse

TABLE 4.8: Device Registry System Operations [32]

4.10.3 Event Handler System

This system helps in providing publish-subscribe functionality and a buffer for service consumption for resource limited application systems. It further helps in event logging and complex event filtering in these systems [4]. This system provides four services, namely, Publish, Subscribe, Unsubscribe and AuthUpdate. These services are responsible for publishing an event, registering and de-registering a subscription and updating the authorisation [32]. The different operations part of the Event Handler system is listed in Table 4.9.

Function	URL Subpath	Method	Input	Output
Subscribe	/subscribe	POST	-	OK
Unsubscribe	/unsubscribe	DELETE	-	OK
Publish	/publish	POST	-	OK

TABLE 4.9: Event Handler System Operations [32]

4.10.4 Gatekeeper System

The Gatekeeper system helps in secure inter-cloud interaction between two local clouds. This includes service discovery, orchestration, authentication and authorisation for inter-cloud service exchange [4]. It provides two services, namely, the Global Service Discovery (GSD) and the Inter-Cloud Negotiation (ICN) services. The former is responsible for discovering clouds containing provider systems providing a specific service. However, the latter is responsible for interacting with orchestrator systems from both the clouds in order to establish a service exchange. Furthermore, this system uses relay systems for discovering other local clouds [32]. The different operations part of the Gatekeeper system is listed in Table 4.10.

Function	URL subpath	Method	Input	Output
Init GSD	/gatekeeper/init_gsd	POST	GSDQueryForm	GSDQueryResult
Init ICN	/gatekeeper/init_icn	POST	ICNRequestForm	ICNResult

TABLE 4.10: Gatekeeper System Operations [32]

4.10.5 Gateway System

The Gateway system is responsible for establishing a secure path for service exchange between a producer and consumer in two different local clouds. It provides two services, namely, the Connect to Consumer and Connect to Provider services. Whenever the gatekeeper system in a service requesting local cloud initiates the Inter-Cloud Negotiation (ICN) service, these two services are invoked in the service consuming local cloud. During this process, whenever any of the two clouds require a gateway, then a secure path for data exchange is established through means of a relay system [32]. The different operations part of the Gateway system is listed in Table 4.11.

Function	URL subpath	Method	Input	Output
Connect to Consumer	/connect_consumer	POST	GatewayConsumerConnectionResponse	Server Port number
Connect to Provider	/connect_provider	POST	GatewayProviderConnectionResponse	GatewayProviderConnectionResponse
Get Public Key	/publickey	GET	-	Public Key string

TABLE 4.11: Gateway System Operations [32]

4.10.6 Choreographer System(Plant Description System)

This system paves way for executing workflows defined beforehand through orchestration and service consumption. These workflows are segmented into three parts and they are:

- Plans
- Actions
- Steps

The entire workflow is defined by the part plans. Plans contain the part actions. These actions seamlessly combine meaningful steps into a group. This ensures better transparency and eventually, leads to creating sequences of these groups. This system provides the Choreography service, where this service is invoked whenever a service provider wants to notify the choreographer that the executed step is completed [11]. The different operations part of the Choreographer system is listed in Table 4.12.

Function	URL subpath	Method	Input	Output
Notify that a step is done	/notifyStepDone	POST	SessionRunningStepData	OK

TABLE 4.12: Choreographer System Operations [32]

4.10.7 Onboarding Controller System

This system is responsible for onboarding in the local cloud. Whenever any new device with provisions for security and access control, wants to connect with the local cloud from outside, the onboarding controller system is invoked. This system ensures that the security of the local cloud is upheld at all times and no compromises are made in terms of security while onboarding these

new devices. Effectively, this system holds the responsibility for authenticating and authorising the device, systems hosted in the device and the services offered by the hosted systems [**32**]. The tasks handled by this system are [**32**]:

- Initializing the onboarding of devices
- Device registration at the device registry system
- System registration at the system registry system
- Service registration at the service registry system
- Initiating the normal operation of devices, systems hosted in the devices and services provided by the hosted systems

The different operations part of the Onboarding Controller system is listed in Table 4.13. Client, Private and Management endpoints for all services have been completed as well for all the core systems mentioned in this chapter.

Function	URL subpath	Method	Input	Output
Onboard with Name	/certificate/name	POST	OnboardingWithNameRequest	OnboardingWithNameResponse
Onboard with Name	/sharedsecret/name	POST	OnboardingWithNameRequest	OnboardingWithNameResponse
Onboard with CSR	/certificate/csr	POST	OnboardingWithCsrRequest	OnboardingWithCsrResponse
Onboard with CSR	/sharedsecret/csr	POST	OnboardingWithCsrRequest	OnboardingWithCsrResponse

TABLE 4.13: Onboarding Controller System Operations [**32**]

4.11 Summary

This chapter covered the methodology used for creating a DSL in detail. It further covered the elements created in the DSL, the logic behind creating the elements and the meaning of these elements and where they can be used in the model. In addition, the UML elements from which the DSL elements have been extended from have been discussed. Furthermore, the model libraries for the two main set of systems, namely, the mandatory core systems and support core systems, created from the profile elements have been elucidated as well. The next chapter discusses on how to use the profile for building a model and also, on generating code from the specific model.

Chapter 5

Modeling and Code Generation

5.1 Introduction

The Arrowhead Framework DSL has been created for the purpose of modelling IoT systems in Arrowhead Framework using the Framework elements. Thus, a model has to be built using the created DSL to understand the intuitiveness of modelling using the DSL. In addition, the created model should also be validated in the Arrowhead Framework environment. This dictates a need for generating code for the built model. Moreover, this generated code should be Arrowhead Framework specific. Furthermore, this generated code needs to be validated in the Arrowhead Framework. This chapter encompasses a car registration model built using the Arrowhead Framework DSL elements and the code generated for this model.

5.2 Car Registration Model

For understanding the ease of modelling using the created DSL, a simple car registration model was created. For this model, the Arrowhead Framework profile along with the model libraries were imported into the model project. On importing, the elements in the profile and model libraries were available for use.

A brief description of the car registration model follows. This model comprises of a local cloud comprising of two systems deployed in it. The first system is a car provider system which provides the service CarRegistration. The second system is a car consumer system which consumes the provided service. The CarRegistration service comprises of two operations, namely, the create-car and get-car operation. The create-car operation creates a set of given cars. The get-car operation fetches those cars in the list that matches the provided criteria.

5.2.1 Block Definition Diagram

A Block Definition Diagram is created for the car provider system. The Abstract System stereotype is added to the diagram and its name is set to Car Provider system. The System stereotype is then added into the diagram and its name is set to HTTP Car Provider System. The generalization relationship between the Car Provider System and HTTP Car Provider System is then established. In the same diagram, the Abstract Service stereotype is added and its name is set to CarRegistration service. In addition, the Service stereotype is added into the diagram with its name set to HTTPCarRegistration service. The create-car and get-car operations are then added into the HTTPCarRegistration Service. On creating the Service stereotype, ports are created in the system for every service provided by the system. The Car Provider System with the Abstract System stereotype would have one port corresponding to the one Abstract Service - CarRegistration. Similarly, the HTTP Car Provider System with the System stereotype would have one port corresponding to the Service - HTTPCarRegistration. The Block Definition Diagram of the Car Provider System is

depicted in Figure 5.1. Like the Car Provider System, another Block Definition Diagram is created for the Car Consumer system with the two stereotypes, namely, the Abstract System and System stereotypes. In addition, the consumed service and Abstract service is created as a conjugated port in System and Abstract System respectively. The Block Definition Diagram of the Car Consumer System is depicted in Figure 5.2. A third Block Definition Diagram is created for the LocalCloud.

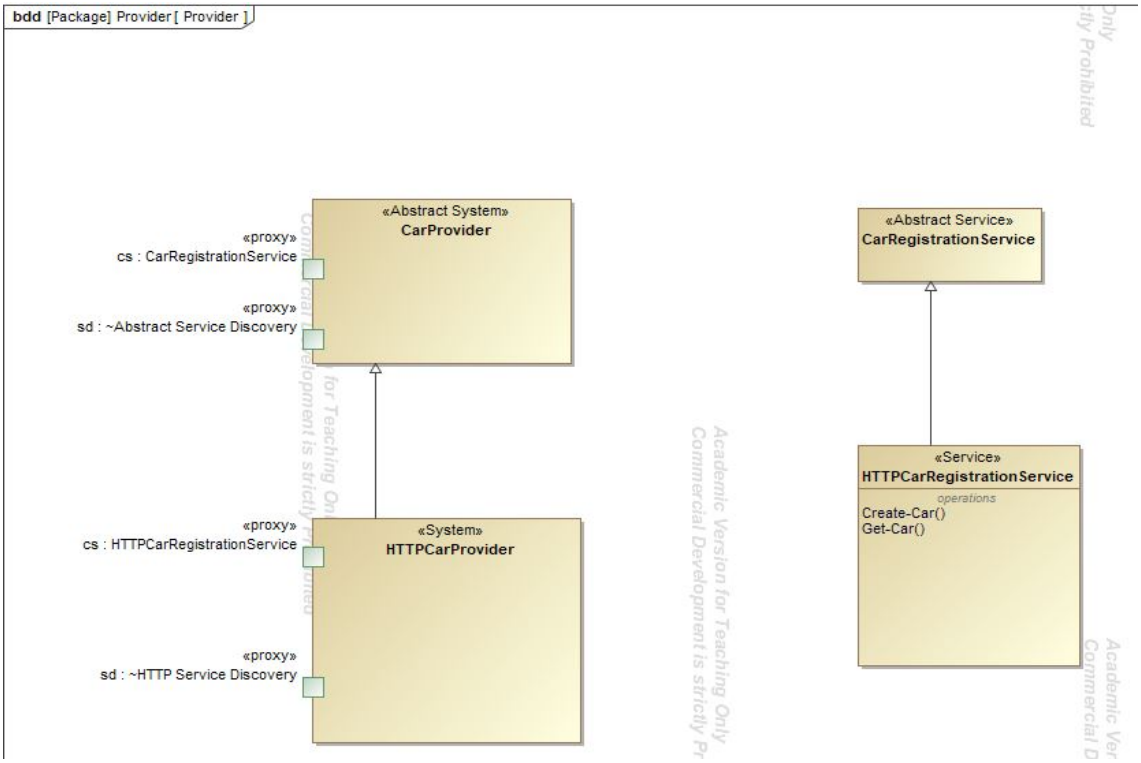

FIGURE 5.1: Block Definition Diagram of Car Provider System

5.2.2 Internal Block Diagram

An Internal Block Diagram is created for the Local Cloud. The internal parts of the Local Cloud comprises of the two systems. The DeployedSystem stereotype is added into the diagram with the name CarProviderSystem and the type set to HTTP Car Provider System. On the other hand, another DeployedSystem stereotype is added into the diagram with the name CarConsumerSystem and the type is set to HTTP Car Consumer System. The service ports in both the systems are then connected using the connector with the ServiceExchange stereotype to depict the exchange of services between the two systems. This Internal Block Diagram of the Local Cloud is depicted in Figure 5.3.

5.3 Velocity Template Language

Cameo Systems Modeler provides the option to generate templates from the model. These templates can be defined using template languages to extract a value from a particular element in the model and present it in the output template. This template generation feature in Cameo Systems Modeler has been used to generate Arrowhead Framework specific code from the model comprising of Arrowhead Framework DSL elements. Although, this is a very rudimentary method for generating code, it is the only built-in custom code generation option in Cameo Systems Modeler. The template generation module in this tool uses Apache VTL.

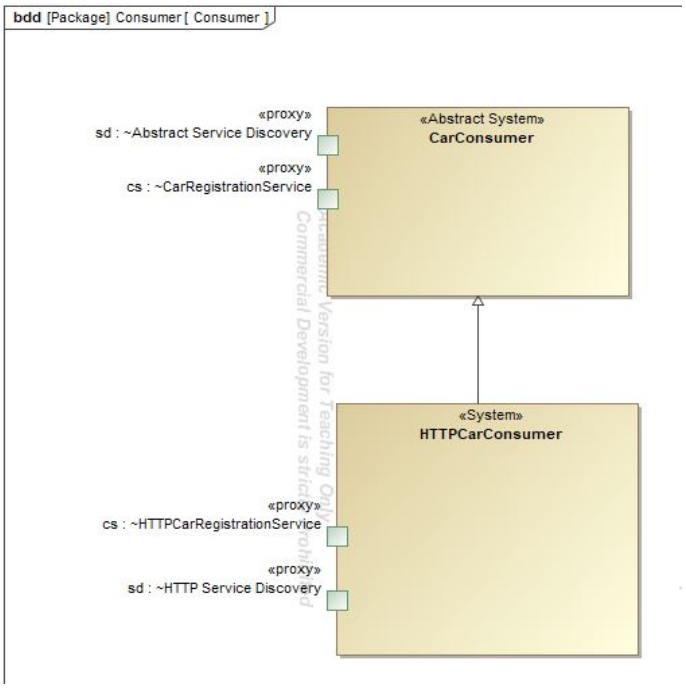

FIGURE 5.2: Block Definition Diagram of Car Consumer System

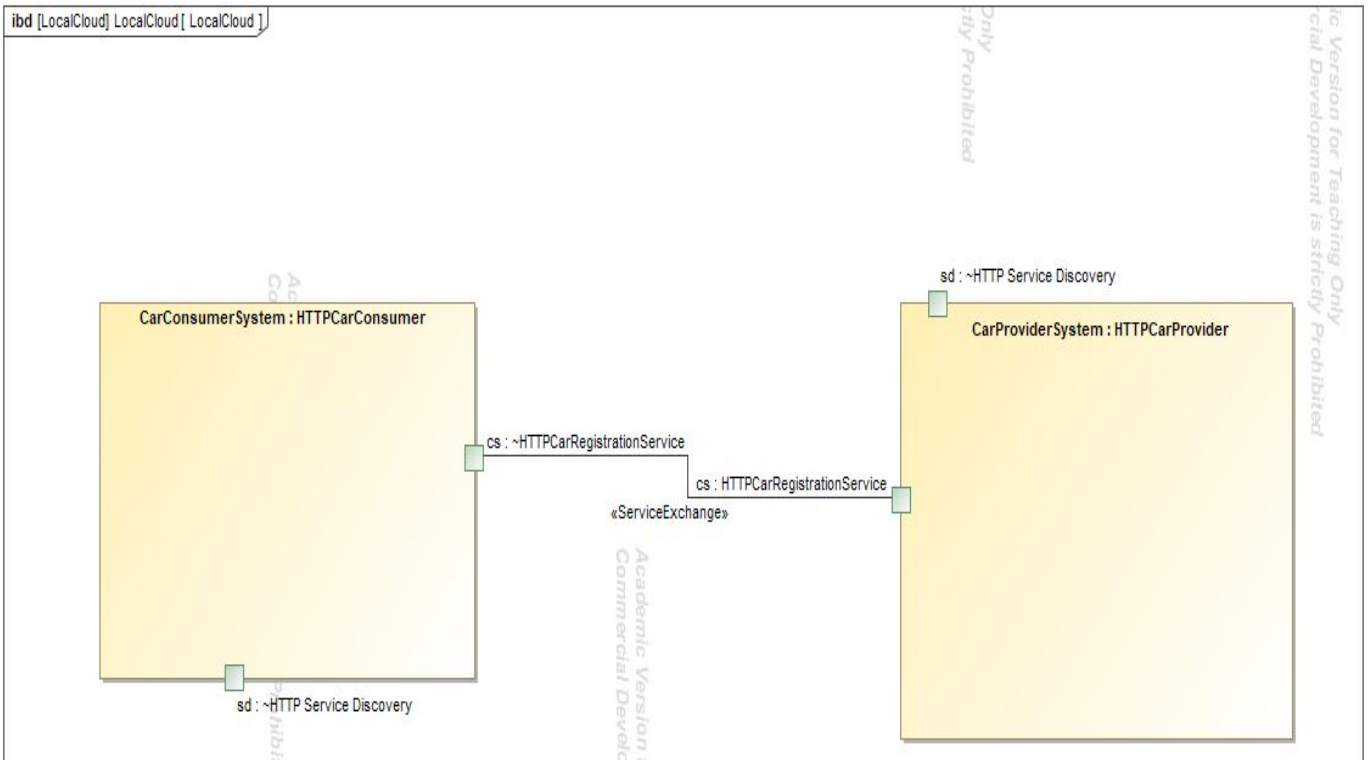

FIGURE 5.3: Internal Block Diagram of Local Cloud

Velocity Template Engine is Java based and it takes templates as input and references Java objects to produce template documents [**47**]. The combination of static content in the template file with variable data from the model, provides the required output file. The VTL directives provide the instructions for this combination to produce the output files [**48**]. Like Java, VTL supports methods

and annotations. Furthermore, it comprises of conditional statements, static text and placeholders referencing Java objects. As the name suggests, these placeholders would be replaced with values from those Java objects that they referenced while velocity processes a template [**47**].

5.3.1 VTL operators

This section elucidates on some of the primary operators used in VTL for template generation. The VTL operators used in Cameo Systems Modeler are:

1. References - References in VTL can be used to reference any specific Java objects in the model. There are three types of references which are listed below:

    - Variables - Any variable used in VTL begins with a dollar ($) symbol. A value can be set to this variable using the set directive. Furthermore, this variable can be used in conditional or loop statements [**49**].
    - Properties - Properties help in accessing or specifying specific properties of an object or identifier. Properties are referenced in VTL by using the period (.) symbol after an identifier that precedes with a dollar ($) symbol [**49**].
    - Methods - Like in Java, methods in VTL have the capability to perform a useful operation or function. For example, VTL uses getter and setter methods for getting a value from a variable or setting a value to a variable. The methods in VTL use the following syntax - [$velocityidentifier.method name(optional parameter)] [**49**].

2. Velocity Directives - The velocity directives are used to process the object values and manipulate them according to the needs for presenting relevant desired data in the output file. The velocity directives begin with a hash (#) symbol.

    - Set directive - The set directive assigns value to a variable or property. It uses the syntax - [#set($variable = "value")] [**49**].
    - Conditional directives - VTL supports conditional directives such as If, Elseif and Else statements. These directives begin with a hash (#) symbol and the conditions are specified within parentheses. The statements to be executed are placed in the next line before the final #end statement [**48**].
    - Loop directives - VTL provides the option for using loop statements. There is only one loop directive which is the #foreach directive. The conditions of the loop are specified within parentheses in the first line. The statements within the loop are placed in the next line before the #end statement in the last line [**49**].

3. Comments and unparsing code - VTL supports commenting in code to include annotations wherever necessary.

    - Single Line Comments - Single line comments are preceded with double hash symbols (##) [**47**].
    - Multi-Line Comments - Multi-line comments begin with the symbol (#) and end with the symbol (#). Multi-line Comments are placed between these two symbols [**47**].
    - Unparsed Code - Template engine wouldn't parse specific code in the template file. This is specified using certain symbols. It begins with the symbol (#[[) and ends with the symbol (]]#). Any text specified between these symbols aren't parsed by the template engine [**47**].

4. Macros - If there are segments in the template file which needs to be used in repetition, then macros are used in VTL. A macro can be defined with a name, to perform a specific function and then, they can be used wherever needed by calling them with the defined name.

5.4 Code Generation

To generate the Arrowhead Framework specific code from the models, the template generation feature in the tool is used. The first step of code generation is to study and understand the existing Arrowhead Framework specific code. The language in which the code should be generated, was selected as well. For this assignment, the Java implementation of the Arrowhead Framework specific code was selected.

As mentioned earlier in section 5.3, the template generation feature in the tool is rudimentary and it supports code generation only to a certain extent. In addition, for each code file that needs to be generated, a separate template file with VTL code should be provided as input to the tool. On providing the template file, the scope of the model should be selected where the model in its entirety or a specific package within the model can be selected. The VTL code written could be specific to the entire model or to a package within the model. In order to make this code generation process efficient, code files which are generic and those which are specific to the model were identified and separated. Only the sections in the code file which are specific to a model were selected for this code generation process. Code templates were used for the remainder of the code in the code file. In addition, the Arrowhead Framework code for the selected demo-car model already exists. Thus, the Arrowhead Framework elements and their corresponding properties and values in the model which should be used in the generated code were identified. The VTL code for extracting the properties and values of the different model elements in the model was developed. The remainder of the code as mentioned above would be the code templates from the provided demo-car code which was not parsed by the velocity engine. This was the best possible way for generating code using the built-in template generator tool in Cameo Systems Modeler.

One of the primary operators used in the VTL code for generating code files is the unparsed code operator which would include generic content in a specific code file. For example, in a Arrowhead Framework specific code file, most of the common library files added at the beginning of the code, would be generic and repetitive for other models as well. Effectively, in the template file given as input, these library files would be added as unparsed code, so that they would be made available in the generated code. An example unparsed code in VTL has been shown in the Listing 5.1. The output of this VTL code is shown in Listing 5.2. In this way, the client code templates from Arrowhead Framework were specified in the velocity template file for generating these code templates along with objects and values specific to the model.

Some of the values extracted from the model using VTL include System names and their attributes such as address and port number, service attributes such as encoding type, protocol type and others. A sample VTL code and the corresponding generated Arrowhead Framework specific code is shown below in Listing 5.3 and Listing C.1 respectively.

```
#[[
import java.util.List;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import eu.arrowhead.client.library.ArrowheadService;
]]#
```

LISTING 5.1: Input Velocity Template Language Code with unparsed section

```
import java.util.List;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import eu.arrowhead.client.library.ArrowheadService;
```

LISTING 5.2: Output Arrowhead Framework code

```
#literal()
final List<CarRequestDTO> carsToCreate = List.of(
#end
#foreach ($c in $System)
#foreach($vp in $report.filterElement($c.ownedAttribute, [Value Property]))
#if($vp.name)
#if($velocityCount == 1)
#literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 2)
#literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 3)
 #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 4)
 #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "));
#end
#end
#end
#end
```

LISTING 5.3: Input Velocity Template Language Code

```
final List<CarRequestDTO> carsToCreate = List.of(new CarRequestDTO("Nissan ", "Green "),
    new CarRequestDTO("Mazda ", "Blue "), new CarRequestDTO("Opel ", "Blue "), new
    CarRequestDTO("BMW ", "Gray "));
```

LISTING 5.4: Output Arrowhead Framework code

5.5 Summary

This chapter covered the details on how to use the created DSL for modeling an Arrowhead Framework application system. It further discussed on the different elements and diagrams part of the model. The template generation feature part of the Cameo Systems Modeler tool and the VTL language it uses for this generation process was discussed in this chapter. The VTL language and its operators have been descibed in this chapter for providing better understanding of this template language. In addition, the process for generating code using the template generation feature in the tool has been discussed in this chapter. Moreover, some examples of VTL code used in the code generation process along with the generated code has been discussed in this chapter.

This page is intentionally left blank.

Chapter 6

Validation

6.1 Introduction

The Arrowhead Framework DSL has been used to build a model. Furthermore, code has been generated for this specific model. There is a need to validate the created Arrowhead Framework DSL. This chapter covers the validation of the model and generated code. The validation completed in this assignment covers the correctness of the model built from the DSL, which in turn validates the DSL. IN addition, the DSL has been validated using OCL constraints which is described in this chapter as well. Moreover, the steps for validation have also been elucidated in this chapter.

6.2 Validating System and Service aspects

Some concepts of systems and services were specified in the chapter Arrowhead Framework - DSL. These are primary concepts which need to be validated for a functional Arrowhead Framework compliant system and services. This validation would in a way validate if the design decisions made have been right. Some of the concepts defined are that a system should have the capability to provide one or more services. This has been validated and it is shown in Figure 6.1. Furthermore, in the concept of service, it has been specified that a service can be provided by one or more systems. This has been validated as well and it is shown in Figure 6.2.

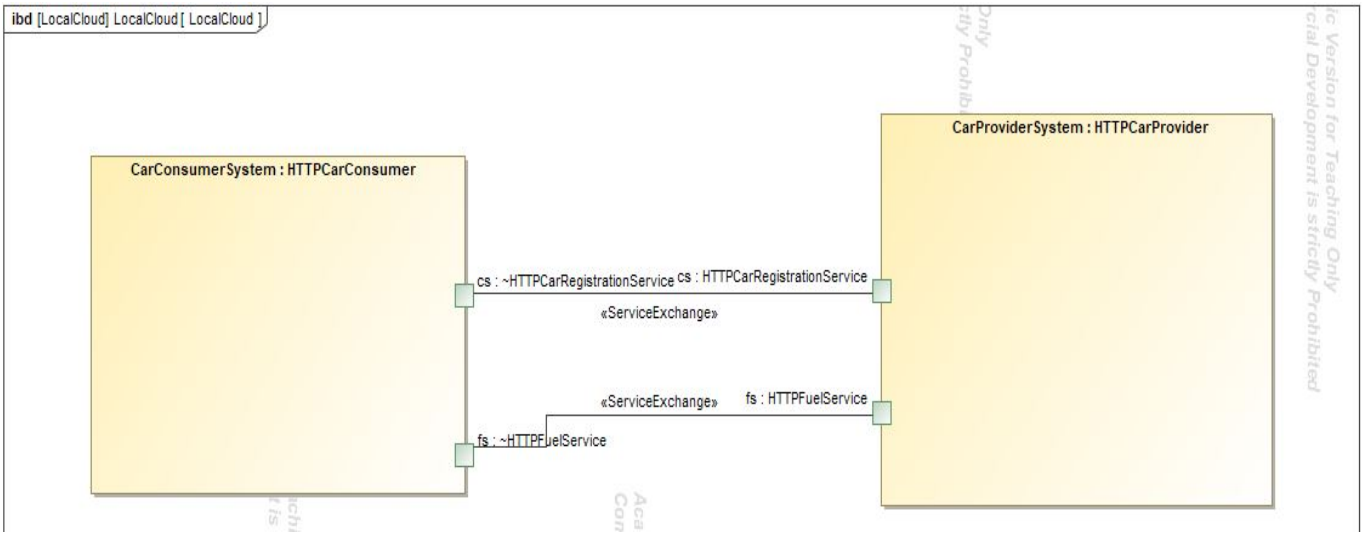

FIGURE 6.1: One system providing one or more services

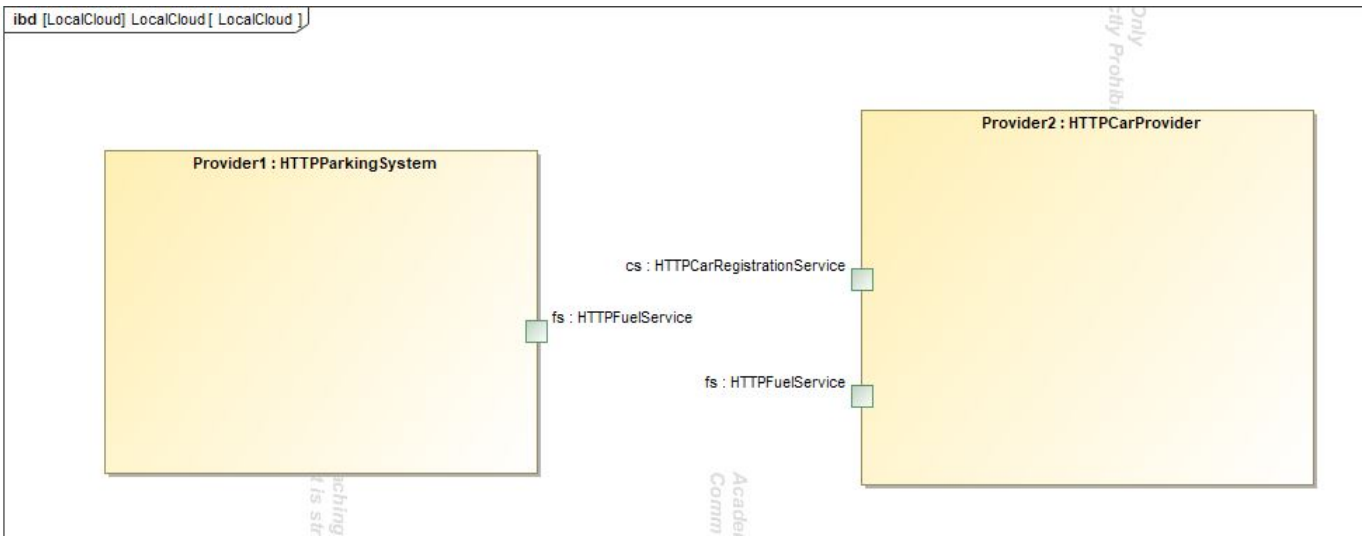

FIGURE 6.2: One service provided by more than one system

6.3 Validation using Object Constraint Language

Object Constraint Language (OCL) is a type of constraint language used for specifying rules in UML and SysML models [**50**]. Constraints are used in DSL to improve the well-formedness of models. OCL comprises of invariants, pre and post conditions, body expressions and other operators for specifying constraints in the model [**51**][**52**]. It is predominantly used for specifying invariant conditions for stereotypes in this assignment. The first step for using OCL in the model is to identify the stereotypes for which the constraints need to be applied. On identification, the constraints needed to be specified for the stereotypes is developed. These constraints are then developed in the specific constraint language. The constraints are then validated in the modeling tool.

The steps for OCL validation in Cameo Systems Modeler is described below in detail.

1. A profile which is a type of package, is first created within the Arrowhead Framework profile. The applied stereotype of this profile is set to ValidationSuite.

2. The constraint elements are then added into this package. The applied stereotype of the constraint elements are set to validationRule.

3. The constraint elements would have a new section called Validation Rule available, on changing the stereotype. In this section, the error message to be displayed, severity level of the constraint and abbreviation of the constraint can be specified. This validation rule section is depicted in Figure 6.3.

4. The constrained element is then selected and the constraint specification option is selected which opens up a new window. This specification window is depicted in Figure 6.4.

5. In the constraint specification window, the language is first selected as OCL 2.0 and then, the constraint is specified in OCL.

6. The check OCL syntax checkbox is selected to verify the syntax. The constraint is then evaluated and this shows an initial evaluation result for the constraint.

7. Once the constraints are specified in the profile, the created model is then opened in the tool. In the model, the validation option under the Analyze tab is selected.

8. The created validation profile is then selected from the list of validation suites for validation.
9. The validation results are then displayed in the notification window below.

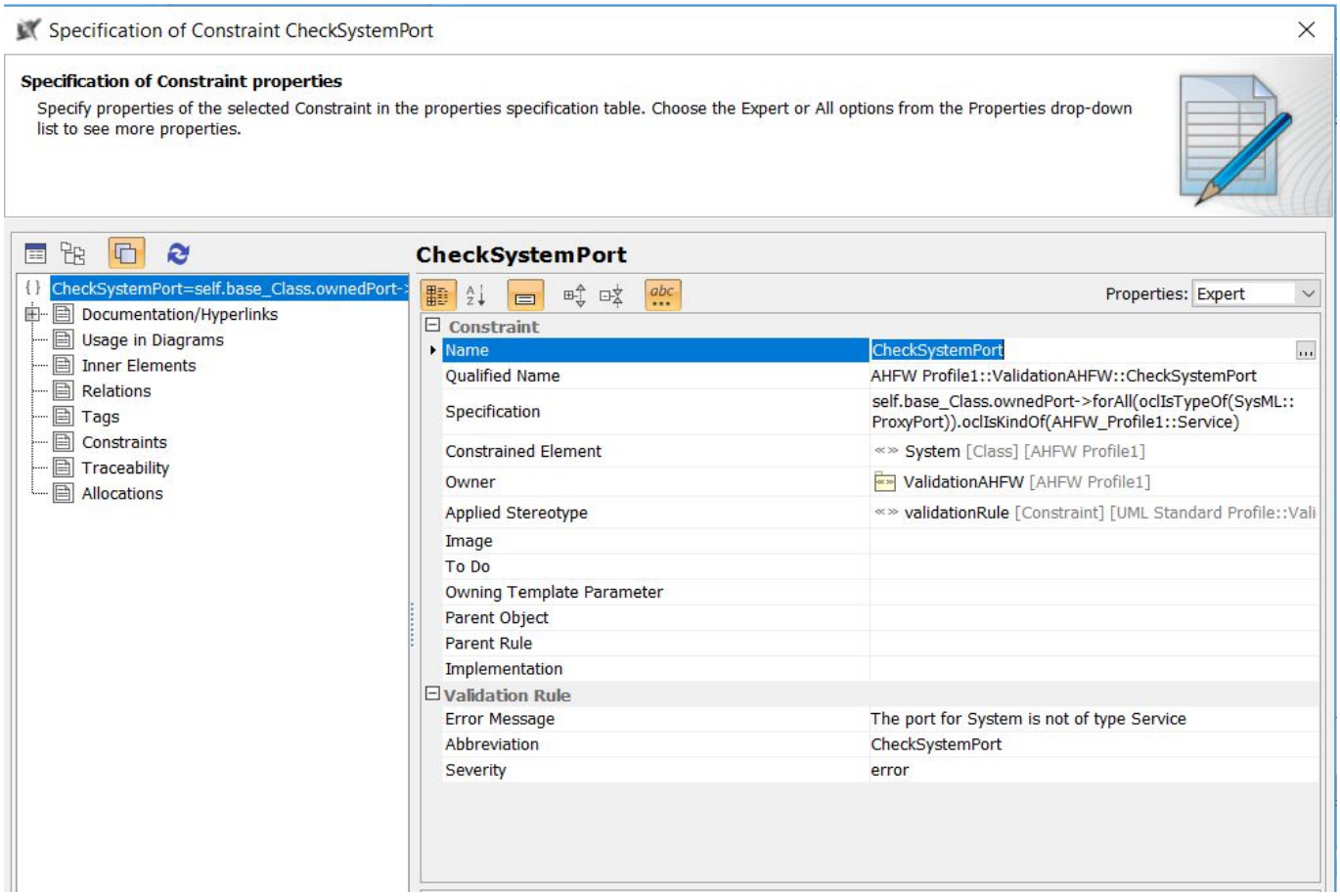

FIGURE 6.3: Constraint specification with the validationRule stereotype

The list of constraints validated in OCL are specified below:

1. In a model, the proxy port in a Abstract System stereotype should be of the type Abstract Service. This constraint is shown in OCL in Listing 6.1

```
context Abstract System inv CheckAbstractSystemPort:
self.base_Class.ownedPort->forAll(oclIsTypeOf(SysML::ProxyPort)).oclIsKindOf
(AHFW_Profile1::Abstract_Service)
```

LISTING 6.1: OCL Constraint 1

2. In a model, the proxy port in a System stereotype should be of the type Service. This constraint is shown in OCL in Listing 6.2

```
context System inv CheckSystemPort:
self.base_Class.ownedPort->forAll(oclIsTypeOf(SysML::ProxyPort)).oclIsKindOf
(AHFW_Profile1::Service)
```

LISTING 6.2: OCL Constraint 2

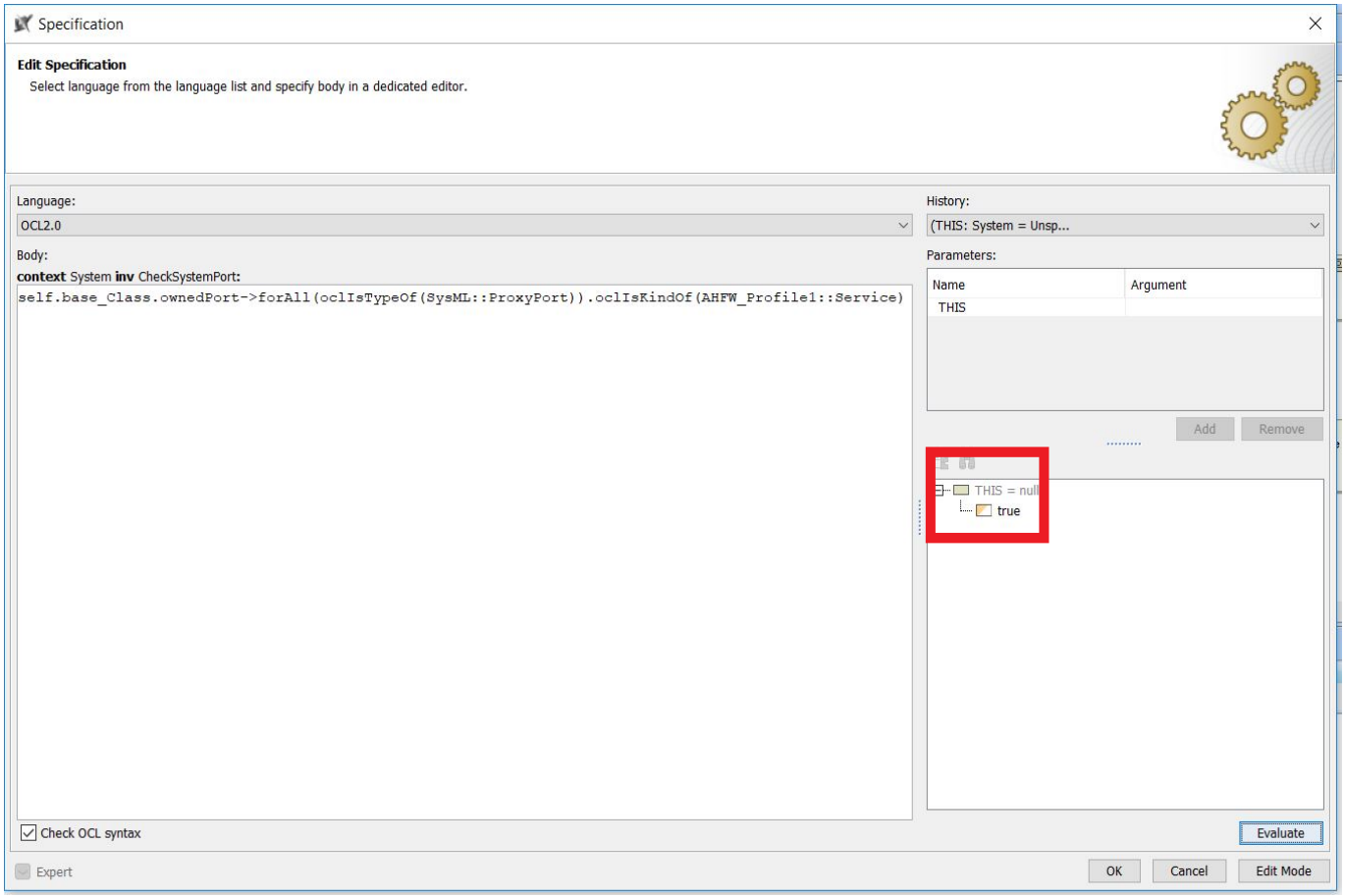

FIGURE 6.4: Evaluated OCL constraint

3. In a model, the DeployedSystem stereotype in the local cloud should be of the type System. This constraint is shown in OCL in Listing 6.3

```
context DeployedSystem inv CheckDeployedSystem:
self.base_Property.type.oclIsKindOf(AHFW_Profile1::System)
```

LISTING 6.3: OCL Constraint 3

The constraints in the Arrowhead Framework DSL, mentioned in listings 6.1, 6.2 and 6.3 have been validated as depicted in Figure 6.4. Furthermore, these constraints were validated in the car registration model as well and this validation was successful as shown in Figure 6.5.

6.4 Validating the generated code

The next step of validation is validating the generated code from the report template module in the tool. The template file is given as input to the tool to generate a code file. This step was repeated until all the code files were generated. This Java project comprising of the generated code files were executed in the Arrowhead Framework environment. The steps performed to execute the code files and complete validation are listed below. The tool used for these steps is IntelliJ IDEA.

1. The first step is to load the core-java-spring folder in IntelliJ IDEA by cloning from the Github Core-Java-Spring repository. The Arrowhead Framework version 4.1.3 has been used for this validation.

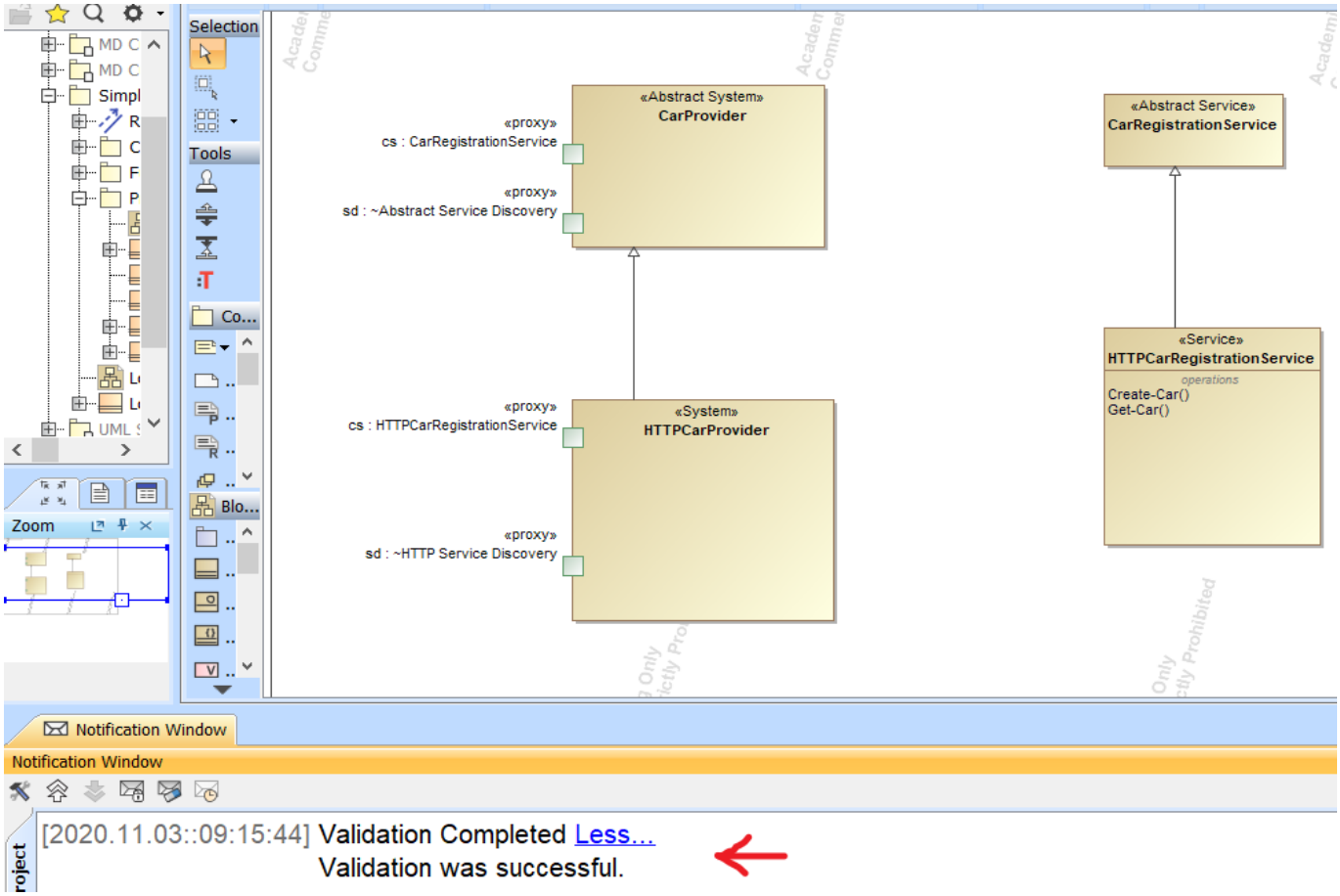

FIGURE 6.5: OCL validation of the car registration model

2. After verifying that the right versions of Java, Maven and MySQL are installed, directory commands are used to navigate to the main core-java-spring project folder.

3. At the root directory, *mvn install* command is executed to build all the projects available.

4. Once the build is complete, each of the core systems are started manually by following the next three steps.

5. The Service Registry System is started first by accessing the target directory within Service Registry System folder and executing the *arrowhead-serviceregistry-4.1.3.jar* file.

6. The Authorisation System is then started by accessing the target directory within Authorisation System folder and executing the *arrowhead-authorization-4.1.3.jar* file.

7. Similarly, the Orchestration System is then started by accessing the target directory within Orchestration System folder and executing the *arrowhead-orchestrator-4.1.3.jar* file.

8. The demo car Java project comprising of the generated code files is then loaded into IntelliJ IDEA.

9. Subsequently, the Service Provider system is then started by accessing the accessing the target directory within Service Provider System folder and executing the *demo-car-provider-4.1.3.jar* file.

10. Before starting the Service Consumer system, it has to be registered manually for the first time and intra-cloud authorisation rules have to be created.

11. The intra-cloud authorisation rules are created in MySQL database, where initially, car consumer demo system is registered. Post registration, the intra-cloud authorisation rules and authorisation intra-cloud interface connection rules are created.

12. Finally, the Service Consumer system is started by accessing the accessing the target directory within Service Consumer System folder and executing the *demo-car-consumer-4.1.3.jar* file.

On executing the Car consumer system, the create-car and get-car operations are verified. The create-car operation creates the set of car brands and colours specified in the model. Furthermore, the get-car operation fetches the set of car brands with the specified blue colour. This validation of the create-car and get-car operations in the Arrowhead Framework environment is depicted in Figure 6.6.

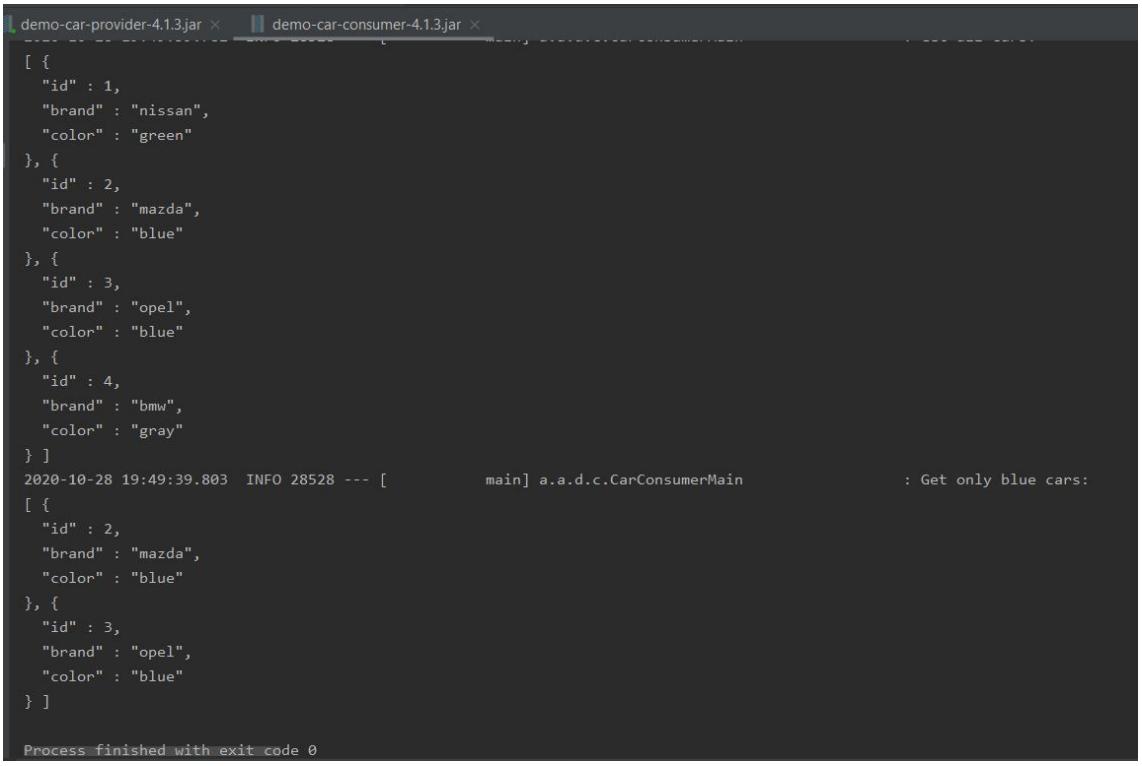

FIGURE 6.6: Validation of the create-car and get-car operations in the Arrowhead Framework Environment

6.5 Summary

This chapter covered the validation of the Arrowhead Framework DSL. The system and service aspects have been validated in this chapter. The created DSL was validated using OCL constraints and the steps for validation in Cameo Systems Modeler tool have been described in detail. Furthermore, the steps for validating the generated code has been elucidated. Finally, the generated code has been validated in this chapter as well. Effectively, validating the different system and service aspects, and the generated code validates the Arrowhead Framework DSL created in this assignment.

Chapter 7

Conclusions and Recommendations

This chapter concludes this thesis assignment by describing the contributions made and the recommendations for future work are elucidated. In addition, the results for the research questions discussed in Chapter 1 are addressed in this chapter as well.

7.1 Contribution

The research questions for this thesis assignment are listed below.

- ***Research Question 1*** *: Does SysML have enough semantics for creating a Domain Specific Language for Arrowhead Framework?*
  - This question is answered by completing the Domain Specific Language for Arrowhead Framework. This Domain Specific Language comprises of Arrowhead Framework specific stereotypes extending the existing UML elements and inheriting the features of existing SysML elements. The semantics of these SysML elements are coherently defined and thus, they have been used wherever needed by the stereotype element for inheriting its features. These SysML elements do have enough semantics for creating Framework elements in the DSL. The validation of the DSL further corroborates that SysML does have enough semantics for creating the DSL for Arrowhead Framework.
- ***Research Question 2*** *: How to build the SysML profile for Arrowhead Framework? Which elements of the Framework can be profiled and which elements cannot be?*
  - This question is answered by completing the SysML profile for Arrowhead Framework. Identification of concepts in Arrowhead Framework and UML would provide a basis for mapping. An initial mapping of these concepts are completed for the UML extension method. This mapping is then extended further by mapping all the elements in Arrowhead Framework with UML and SysML elements which completes the SysML profile for Arrowhead Framework.
- ***Research Sub - Question 2.1****: Which elements of the Framework can be profiled and which elements cannot be?*
  - All the primary elements contributing to the structure of Arrowhead Framework could be profiled. The profile includes the basic building blocks of the Framework and their relevant description documents as well.

Thus, this thesis assignment covers all the research questions specified in Chapter 1. It can be concluded that the Domain Specific Language created for Arrowhead Framework can be used for modelling any complex IOT automation systems. The different SysML diagrams and the SYSMOD

methodology discussed in chapter 3 can be used for modeling these systems in Arrowhead Framework. This created DSL can be validated using OCL constraints for checking well-formedness of the model. In addition, the created model can be validated by generating Arrowhead Framework code specific to the model. This can be done using VTL in Cameo Systems Modeler tool. Moreover, there are several reasons why users can use this DSL for Arrowhead Framework:

- Decreasing their time and increasing the intuitiveness with modelling complex automation systems with Arrowhead Framework.
- Complex automation systems in Arrowhead Framework can be modelled for better visual representation and thereby, increasing the ease of comprehension.
- It paves way for migrating from a document based communication approach to a model based communication approach which effectively saves time for users and effectively, increases their productivity.

7.2 Recommendations for future work

This section elucidates on the recommendations provided for future work on this thesis assignment.

1. The code generation task could be made generic rather than specific. It could be improved upon by generating Arrowhead Framework specific code for each and every model built using this DSL.
2. The code generation task used in this assignment requires constant user interventions. Thus, a code generation task could be completed, making it more intuitive and seamless, involving minimal user intervention.
3. An external code generator could be integrated with the modelling tool for seamless code generation and the differences with the current code generator can be compared.
4. Arrowhead Framework is an evolving Framework where several new systems and services are being developed as time progresses. The DSL created in this assignment could be modified or extended with these occurring changes.
5. Evaluate the inter-operability capabilities of this Framework, namely, semantic, communication protocol and data encoding inter-operability in these models.
6. Cameo Systems Modeler integrates with other tools such as Matlab/Simulink. These tools can be used for building models like kinematic models, complex mathematical models and others. These models could be integrated with Arrowhead Framework models in Cameo Systems Modeler and evaluated.
7. Evaluate the real time capabilities of complex IoT systems modeled in Arrowhead Framework.

Appendix A

Other Profile Diagrams

A.1 Profile Diagram comprising of elements and enumerations

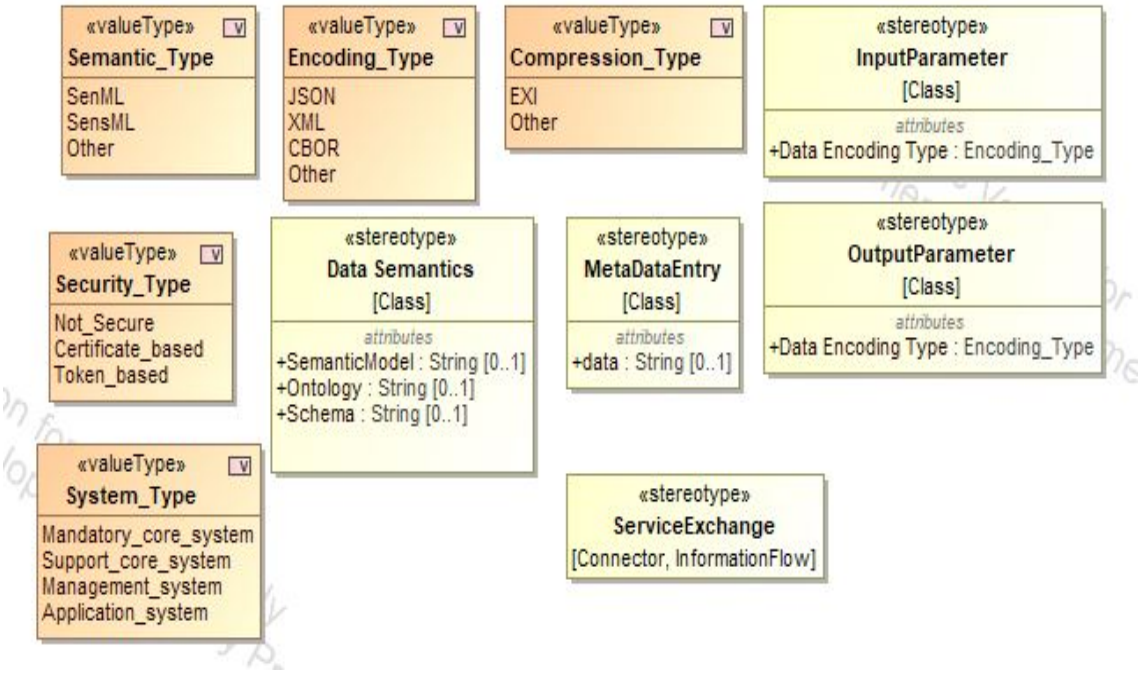

FIGURE A.1: Profile Diagram comprising of other stereotype elements and enumerations

Appendix B

Example Core System from Model Library

B.1 Service Registry System

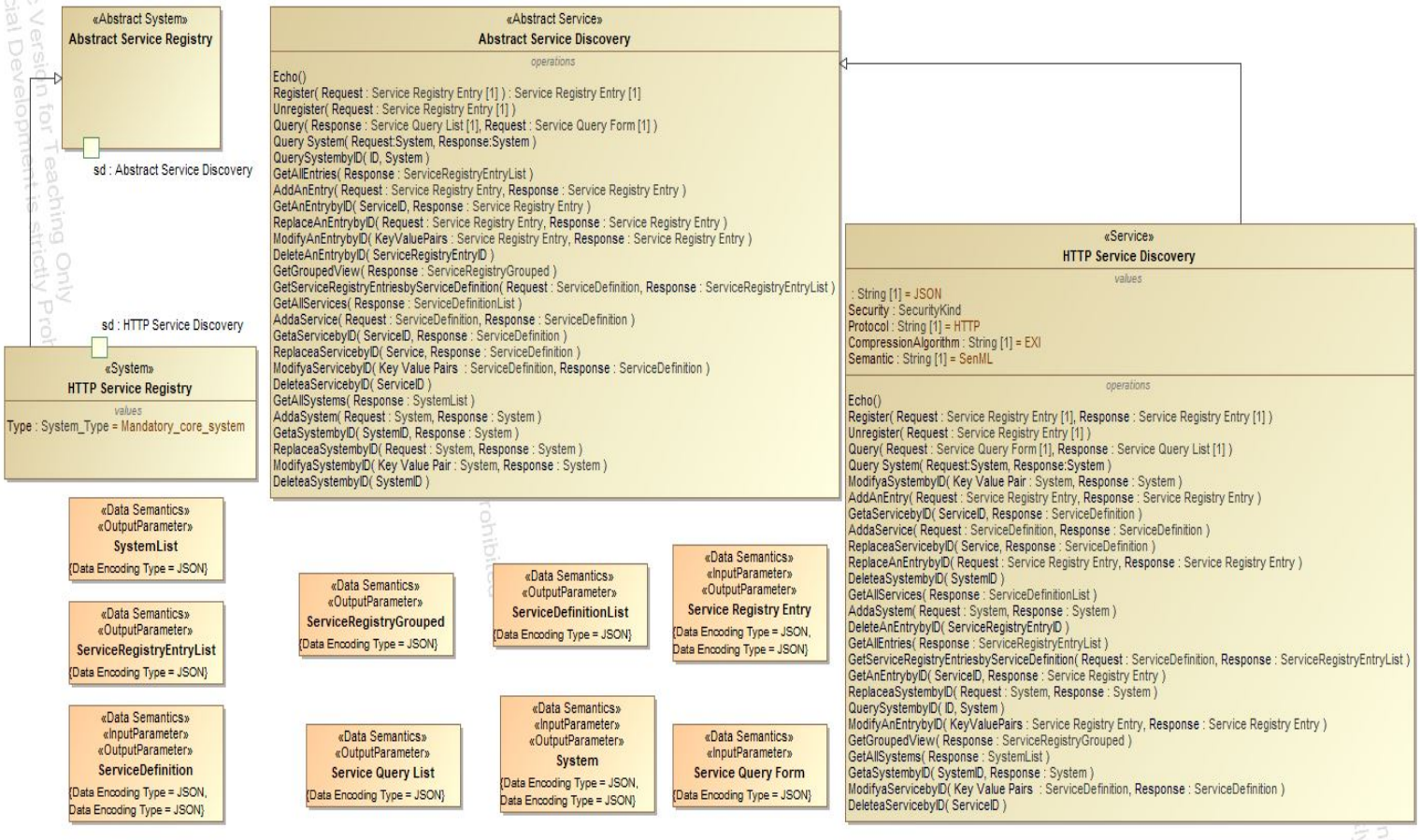

FIGURE B.1: Service Registry System

Appendix C

Example Velocity Template Language Code

C.1 CarConsumerMain

```
#[[package ai.aitia.demo.car_consumer;

import java.util.List;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.http.]]# #foreach ($c in $Service)
#set($x = $$c.tags.get(2))
#foreach ($t in $x)
#if($velocityCount == 9)
$t
#end
#end
#end
#[[;
import com.aitia.demo.car_common.dto.CarRequestDTO;
import com.aitia.demo.car_common.dto.CarResponseDTO;

import eu.arrowhead.client.library.ArrowheadService;
import eu.arrowhead.common.CommonConstants;
import eu.arrowhead.common.SSLProperties;
import eu.arrowhead.common.Utilities;
import eu.arrowhead.common.dto.shared.OrchestrationFlags.Flag;
import eu.arrowhead.common.dto.shared.OrchestrationFormRequestDTO;
import eu.arrowhead.common.dto.shared.OrchestrationFormRequestDTO.Builder;
import eu.arrowhead.common.dto.shared.OrchestrationResponseDTO;
import eu.arrowhead.common.dto.shared.OrchestrationResultDTO;
import eu.arrowhead.common.dto.shared.ServiceInterfaceResponseDTO;
import eu.arrowhead.common.dto.shared.ServiceQueryFormDTO;
import eu.arrowhead.common.exception.InvalidParameterException;

@SpringBootApplication
@ComponentScan(basePackages = {CommonConstants.BASE_PACKAGE,
    CarConsumerConstants.BASE_PACKAGE})
```

```
public class CarConsumerMain implements ApplicationRunner {

    @Autowired
   private ArrowheadService arrowheadService;

    @Autowired
   protected SSLProperties sslProperties;

    private final Logger logger = LogManager.getLogger(CarConsumerMain.class);

    public static void main( final String[] args ) {
      SpringApplication.run(CarConsumerMain.class, args);
    }

    @Override
   public void run(final ApplicationArguments args) throws Exception {
      createCarServiceOrchestrationAndConsumption();
      getCarServiceOrchestrationAndConsumption();
   }

    public void createCarServiceOrchestrationAndConsumption() {
      logger.info("Orchestration request for " +
          CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION + " service:");
      final ServiceQueryFormDTO serviceQueryForm = new
          ServiceQueryFormDTO.Builder(CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION)
                                                    .interfaces(getInterface())
                                                    .build();

      final Builder orchestrationFormBuilder =
          arrowheadService.getOrchestrationFormBuilder();
      final OrchestrationFormRequestDTO orchestrationFormRequest =
          orchestrationFormBuilder.requestedService(serviceQueryForm)
                                                          .flag(Flag.MATCHMAKING, true)
                                                          .flag(Flag.OVERRIDE_STORE,
                                                              true)
                                                          .build();

      printOut(orchestrationFormRequest);

      final OrchestrationResponseDTO orchestrationResponse =
          arrowheadService.proceedOrchestration(orchestrationFormRequest);

      logger.info("Orchestration response:");
      printOut(orchestrationResponse);

      if (orchestrationResponse == null) {
         logger.info("No orchestration response received");
      } else if (orchestrationResponse.getResponse().isEmpty()) {
         logger.info("No provider found during the orchestration");
      } else {
         final OrchestrationResultDTO orchestrationResult =
             orchestrationResponse.getResponse().get(0);
         validateOrchestrationResult(orchestrationResult,
             CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION);

         final List<CarRequestDTO> carsToCreate = List.of(]]#
#foreach ($c in $System)
```

```
#  if($c.name ==  CarConsumerSystem )
#foreach($vp in $report.filterElement($c.ownedAttribute, [Value Property]))
#if($vp.name)
#if($velocityCount == 1)
   #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 2)
#literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 3)
 #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "),
#end
#if($velocityCount == 4)
 #literal()
new CarRequestDTO
#end
 ("$vp.name ", "$vp.defaultValue.value "));
#end
#end
#end
#end
#end


        #[[ for (final CarRequestDTO carRequestDTO : carsToCreate) {
           logger.info("Create a car request:");
           printOut(carRequestDTO);
           final String token = orchestrationResult.getAuthorizationTokens() == null ?
               null : orchestrationResult.getAuthorizationTokens().get(getInterface());
           final CarResponseDTO carCreated =
               arrowheadService.consumeServiceHTTP(CarResponseDTO.class,]]#
#foreach ($c in $Service)
#set($x = $$c.tags.get(2))
#foreach ($t in $x)
   #if($velocityCount == 9)
         $t
         #end
#end
#end
#[[.valueOf(orchestrationResult.getMetadata().
           get(CarConsumerConstants.HTTP_METHOD)),
                 orchestrationResult.getProvider().getAddress(),
                     orchestrationResult.getProvider().getPort(),
                     orchestrationResult.getServiceUri(),
                 getInterface(), token, carRequestDTO, new String[0]);
           logger.info("Provider response");
           printOut(carCreated);
        }
```

```
    }
  }


  public void getCarServiceOrchestrationAndConsumption() {
    logger.info("Orchestration request for " +
        CarConsumerConstants.GET_CAR_SERVICE_DEFINITION + " service:");
    final ServiceQueryFormDTO serviceQueryForm = new
        ServiceQueryFormDTO.Builder(CarConsumerConstants.GET_CAR_SERVICE_DEFINITION)
                                                    .interfaces(getInterface())
                                                    .build();

    final Builder orchestrationFormBuilder =
        arrowheadService.getOrchestrationFormBuilder();
    final OrchestrationFormRequestDTO orchestrationFormRequest =
        orchestrationFormBuilder.requestedService(serviceQueryForm)
                                                            .flag(Flag.MATCHMAKING, true)
                                                            .flag(Flag.OVERRIDE_STORE,
                                                                true)
                                                            .build();

    printOut(orchestrationFormRequest);

    final OrchestrationResponseDTO orchestrationResponse =
        arrowheadService.proceedOrchestration(orchestrationFormRequest);

    logger.info("Orchestration response:");
    printOut(orchestrationResponse);

    if (orchestrationResponse == null) {
       logger.info("No orchestration response received");
    } else if (orchestrationResponse.getResponse().isEmpty()) {
       logger.info("No provider found during the orchestration");
    } else {
       final OrchestrationResultDTO orchestrationResult =
           orchestrationResponse.getResponse().get(0);
       validateOrchestrationResult(orchestrationResult,
           CarConsumerConstants.GET_CAR_SERVICE_DEFINITION);

       logger.info("Get all cars:");
       final String token = orchestrationResult.getAuthorizationTokens() == null ? null
           : orchestrationResult.getAuthorizationTokens().get(getInterface());
       @SuppressWarnings("unchecked")
       final List<CarResponseDTO> allCar =
           arrowheadService.consumeServiceHTTP(List.class,]]#
#foreach ($c in $Service)
#set($x = $$c.tags.get(2))
#foreach ($t in $x)
   #if($velocityCount == 9)
        $t
        #end
#end
#end
 #[[.valueOf(orchestrationResult.getMetadata().
        get(CarConsumerConstants.HTTP_METHOD)),
```

```
                    orchestrationResult.getProvider().getAddress(),
                        orchestrationResult.getProvider().getPort(),
                        orchestrationResult.getServiceUri(),
                                                    getInterface(), token, null, new
                                                        String[0]);
        printOut(allCar);

        logger.info("Get only blue cars:");
        final String[] queryParamColor = {orchestrationResult.getMetadata().get
        (CarConsumerConstants.REQUEST_PARAM_KEY_COLOR), "blue"};
        @SuppressWarnings("unchecked")
        final List<CarResponseDTO> blueCars =
            arrowheadService.consumeServiceHTTP(List.class, ]]#
#foreach ($c in $Service)
#set($x = $$c.tags.get(2))
#foreach ($t in $x)
   #if($velocityCount == 9)
        $t
        #end
#end
#end
 #[[ .valueOf(orchestrationResult.getMetadata().
        get(CarConsumerConstants.HTTP_METHOD)),
                    orchestrationResult.getProvider().getAddress(),
                        orchestrationResult.getProvider().getPort(),
                        orchestrationResult.getServiceUri(),
                                                    getInterface(), token, null,
                                                        queryParamColor);
        printOut(blueCars);
     }
   }
   private String getInterface() {
     return sslProperties.isSslEnabled() ? CarConsumerConstants.INTERFACE_SECURE :
         CarConsumerConstants.INTERFACE_INSECURE;
   }

   private void validateOrchestrationResult(final OrchestrationResultDTO
       orchestrationResult, final String serviceDefinitin) {
     if (!orchestrationResult.getService().getServiceDefinition().equalsIgnoreCase
     (serviceDefinitin)) {
        throw new InvalidParameterException("Requested and orchestrated service
            definition do not match");
     }

     boolean hasValidInterface = false;
     for (final ServiceInterfaceResponseDTO serviceInterface :
         orchestrationResult.getInterfaces()) {
        if (serviceInterface.getInterfaceName().equalsIgnoreCase(getInterface())) {
          hasValidInterface = true;
          break;
        }
     }
     if (!hasValidInterface) {
        throw new InvalidParameterException("Requested and orchestrated interface do not
            match");
     }
   }
```

```
    private void printOut(final Object object) {
      System.out.println(Utilities.toPrettyJson(Utilities.toJson(object)));
    }
}]]#
```

LISTING C.1: CarConsumerMain.java

Bibliography

[1] Fuentes-Fernández, L. and Vallecillo-Moreno, A., 2004. An introduction to UML profiles. UML and Model Engineering, 2(6-13), p.72.

[2] Arrowhead Project, *Arrowhead Framework-This is it*, Arrowhead Project, viewed February 2020, <https://arrowhead.eu/arrowheadframework/this-is-it>

[3] Varga, P., Blomstedt, F., Ferreira, L.L., Eliasson, J., Johansson, M., Delsing, J. and de Soria, I.M., 2017. Making system of systems interoperable–The core components of the arrowhead framework. Journal of Network and Computer Applications, 81, pp.85-95.

[4] Delsing, J. ed., 2017. Iot automation: Arrowhead framework. CRC Press.

[5] IBM Knowledge Center, Rational Rhapsody 8.4.0, IBM, viewed March 2020, URL <https://www.ibm.com/support/knowledgecenter/SSB2MU_8.4.0/com.ibm.rhp.homepage.doc/helpindex_rhapsody.html>

[6] Blomstedt, F., Ferreira, L.L., Klisics, M., Chrysoulas, C., de Soria, I.M., Morin, B., Zabasta, A., Eliasson, J., Johansson, M. and Varga, P., 2014, October. The arrowhead approach for SOA application development and documentation. In IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society (pp. 2631-2637). IEEE.

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

[8] SysML Org., *What is SysML? Who created SysML?, SysML*, SysML Org., viewed March 2020, <https://sysml.org/>

[9] ISO, 2017, *ISO/IEC 19514:2017 [ISO/IEC 19514:2017] Information technology — Object management group systems modeling language (OMG SysML)*, ISO, viewed March 2020, <https://www.iso.org/standard/65231.html>

[10] Nagorny, K., Harrison, R., Colombo, A.W. and Kreutz, G., 2013, July. A formal engineering approach for control and monitoring systems in a service-oriented environment. In 2013 11th IEEE International Conference on Industrial Informatics (INDIN) (pp. 480-487). IEEE.

[11] Arrowhead Project, 2019, *Support Core Systems and Services*, Fusion Forge, viewed February 2020, <https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Support_Core_Systems_and_Servicess>

[12] Lollini, P., Mori, M., Babu, A. and Bouchenak, S., 2016. Amadeos sysml profile for sos conceptual modeling. In Cyber-Physical Systems of Systems (pp. 97-127). Springer, Cham.

[13] IBM Knowledge Center, Rational Software Architect 9.7.0, IBM, viewed March 2020, URL <https://www.ibm.com/support/knowledgecenter/SS8PJ7_9.7.0/com.ibm.rsa_base.nav.doc/topics/c_node_overview.html>

[14] Arrowhead Project, *Arrowhead Consortia*, Github, viewed March 2020, <https://github.com/arrowhead-f/>

[15] No Magic Inc., MagicDraw 18.5 Documentation, MagicDraw, viewed March 2020, URL <https://docs.nomagic.com/display/MD185/MagicDraw+Documentatio>

[16] Arrowhead Project, *Arrowhead-f/core-java-spring*, GitHub Repository, viewed March 2020, <https://github.com/arrowhead-f/core-java-spring/blob/master/documentation /development/developer_tmp.txt>

[17] Arrowhead Project IncQuery Labs, *IncQueryLabs/arrowhead-tools*, GitHub Repository, viewed March 2020, <https://github.com/IncQueryLabs/arrowhead-tools>

[18] Wikipedia, *Profile(UML)*, Wikipedia, viewed March 2020, <https://en.wikipedia.org/wiki/Profile(UML)>

[19] UML Diagrams Org., *UML Profile*, UML Diagrams Org., viewed January 2020, <https://www.uml-diagrams.org/profile.html>

[20] Alhir, S.S., 2006. Guide to Applying the UML. Springer Science Business Media.

[21] Object Management Group. OMG Systems Modeling Language TM(OMG SYSML), 2019. URL <https://www.omg.org/spec/SysML/1.6/PDF>

[22] Object Management Group. OMG Unified Modeling Language TM (OMG UML), 2015. URL <https://www.omg.org/spec/UML/2.5/PDF>

[23] Wikipedia, Systems Modeling Language, Wikipedia, viewed September 2020, <https://en.wikipedia.org/wiki/Systems_Modeling_Language>

[24] Palm, E., Bodin, U. and Schelén, O., 2020. Kalix: A Java 11 Library for DevelopingEclipse Arrowhead System-of-Systems. In 2020 IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, September 8-11.

[25] Bean, J., 2009. SOA and web services interface design: principles, techniques, and standards. Morgan Kaufmann.

[26] Josuttis, N.M., 2007. SOA in practice: the art of distributed system design. " O'Reilly Media, Inc.".

[27] Maier, M.W., 1998. Architecting principles for systems-of-systems. Systems Engineering: The Journal of the International Council on Systems Engineering, 1(4), pp.267-284.

[28] Delsing, J., Eliasson, J., van Deventer, J., Derhamy, H. and Varga, P., 2016, December. Enabling IoT automation using local clouds. In 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT) (pp. 502-507). IEEE.

[29] Heart of the Art, The Life of Living Systems, What is a System of Systems and Why Should I Care?, URL <https://www.heartoftheart.org/?p=4514>

[30] The SOA Source Book, Open Group, Service-Oriented Architecture Ontology Version 2.0 – Service, ServiceContract, and ServiceInterface, URL <https://www.opengroup.org/soa/source-book/ontologyv2/p4.htm>

[31] Varga, P. and Hegedűs, C., 2017. Inter-cloud communication through gatekeepers to support IoT service interaction in the arrowhead framework. Wireless Personal Communications, 96(3), pp.3515-3532.

[32] Github, Arrowhead Consortia, core-java-spring, URL <https://github.com/arrowhead-f/core-java-spring/tree/master>

[33] Blomstedt, F., Ferreira, L.L., Klisics, M., Chrysoulas, C., de Soria, I.M., Morin, B., Zabasta, A., Eliasson, J., Johansson, M. and Varga, P., 2014, October. The arrowhead approach for SOA application development and documentation. In IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society (pp. 2631-2637). IEEE.

[34] Arrowhead Framework Kalix Library, ar:kalix, Industrial IoT with Java 11, URL <https://arkalix.se/>

[35] Selic, B., 2007, May. A systematic approach to domain-specific language design using UML. In 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07) (pp. 2-9). IEEE.

[36] Weilkiens, T., Lamm, J.G., Roth, S. and Walker, M., 2015. Model-based system architecture. John Wiley Sons.

[37] Weilkiens, T., 2014. Systems engineering with SysML / UML: requirements, analysis, architecture. With a foreword by Richard Mark Soley. dpoint. publishing company.

[38] Grady, R.B., 1992. Practical software metrics for project management and process improvement. Prentice-Hall, Inc..

[39] Robertson, S. and Robertson, J., 2012. Mastering the requirements process: Getting requirements right. Addison-wesley.

[40] Güdemann, M., Kegel, S., Ortmeier, F., Poenicke, O. and Richter, K., 2010, June. SysML in digital engineering. In Proceedings of the First International Workshop on Digital Engineering (pp. 1-8).

[41] Van Deursen, A., Klint, P. and Visser, J., 2000. Domain-specific languages: An annotated bibliography. ACM Sigplan Notices, 35(6), pp.26-36.

[42] Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C., Visser, E. and Wachsmuth, G., 2013. DSL engineering: Designing, implementing and using domain-specific languages (p. 558). dslbook. org.

[43] Sharafi, Z., Mirshams, P., Hamou-Lhadj, A. and Constantinides, C., 2010, May. Extending the UML metamodel to provide support for crosscutting concerns. In 2010 Eighth ACIS International Conference on Software Engineering Research, Management and Applications (pp. 149-157). IEEE.

[44] Robak, S., Franczyk, B. and Politowicz, K., 2002. Extending the UML for modelling variability for system families. International Journal of Applied Mathematics and Computer Science, 12, pp.285-298.

[45] No Magic Inc., Cameo Systems Modeler, Dassault Systemes, viewed March 2020, URL <https://www.nomagic.com/products/cameo-systems-modelerfeatures>

[46] Andova, S., van den Brand, M.G., Engelen, L.J. and Verhoeff, T., 2012, June. MDE Basics with a DSL Focus. In International School on Formal Methods for the Design of Computer, Communication and Software Systems (pp. 21-57). Springer, Berlin, Heidelberg.

[47] MagicDraw Documentation, MagicDraw 18.5 Documentation, Report Wizard Documentation, Template Creation Introduction, URL <https://docs.nomagic.com/display/MD185/Introduction>

[48] Harrop, R., 2004. Pro Jakarta Velocity: From Professional to Expert. Apress.

[49] The Apache Velocity Project, User Guide, Velocity, viewed August 2020, URL <http://velocity.apache.org/engine/1.7/user-guide.html>

[50] Wikipedia, *Object Constraint Language*, Wikipedia, viewed July 2020, URL <https://en.wikipedia.org/wiki/Object_Constraint_Language>

[51] Object Management Group. OMG Object Constraint Language (OMG OCL), February 2014. URL <https://www.omg.org/spec/OCL/2.4/PDF>

[52] Clark, T. and Warmer, J. eds., 2003. Object modeling with the OCL: the rationale behind the Object Constraint Language (Vol. 2263). Springer.