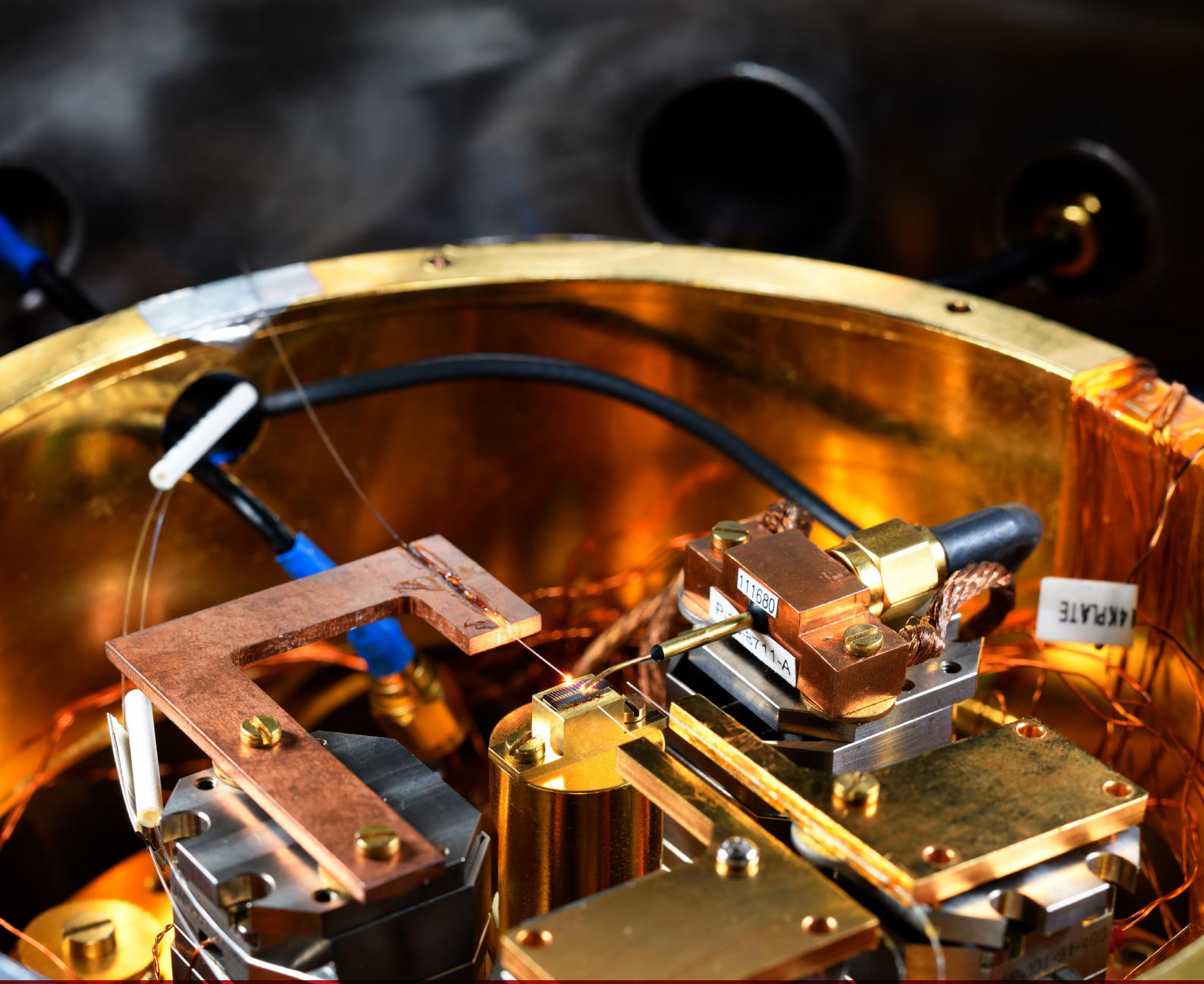
Evaluation of Arrowhead Framework for Timed Cyber-Physical Systems of Systems

Mahendra Kumar, Suman

Award date:
2020



MASTER THESIS

Evaluation of Arrowhead Framework for Timed Cyber-Physical Systems of Systems

Suman Mahendra Kumar
1334700
September 2020

DEPARTMENT OF ELECTRICAL ENGINEERING

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

ASML

This page is intentionally left blank

Department of Electrical Engineering
Electronic Systems Research Group

Evaluation of Arrowhead Framework for Timed Cyber-Physical Systems of Systems

Master Thesis Report

Student ID:1334700
Suman Mahendra Kumar
Embedded Systems Program, TU/e
s.mahendra.kumar@student.tue.nl

Academic Supervisors:

Prof. dr. ir. Jeroen Voeten
J.P.M.Voeten@tue.nl
Alireza Mohammadkhani, M.Sc
a.mohammadkhani@tue.nl

Company Supervisor:

dr. ir. Ramon Schiffelers
ramon.schiffelers@asml.com

Eindhoven, September 2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Abstract

Embedded Systems
Mathematics and Computer Science

Master of Science

Evaluation of Arrowhead Framework for Timed Cyber-Physical Systems of Systems

by Suman Mahendra Kumar

The ongoing research in Industry 4.0 has led the "Internet of Things" to become an increasingly growing topic both in academic and industrial settings. In the recent past, multiple frameworks for developing IoT-networks and Systems of Systems have emerged. One such framework is the Arrowhead framework developed as a part of the European research projects and Arrowhead Tools project. The focus of this framework is to enable interoperability between different industrial Cyber-Physical Systems, while also supporting real-time data handling, data security, system security, flexibility, and scalability of systems.

The goal of this MSc thesis is to evaluate the timing support of the Arrowhead Framework for Cyber-Physical Systems of Systems. A safety-critical, pothole detection and warning system is developed focusing on an Advanced Driver Assistance System for vehicles. Autonomous rover carts, in communication with one another are developed as Cyber-Physical Systems of Systems for demonstration and evaluation purposes.

Acknowledgement

I would sincerely like to thank my graduation thesis supervisor **Prof. dr. ir. Jeroen Voeten** at Eindhoven University of Technology (TU/e) for his continuous support throughout the graduation thesis phase. He is extremely prompt in providing feedback through the weekly discussions without which this work would have been impossible. His critical evaluation of my work kept me on my toes and helped me improve myself. I would like to specifically express my gratitude towards him for patiently reviewing my report and helping me to improve my presentation skills.

I also extend my gratitude to my daily supervisor **Alireza Mohammadkhani, M.Sc**. The hardware components that were related to my project were made available by him. By sharing his wisdom and experience he guided me throughout the project. I would like to thank him for being supportive. I would also like to thank **Dr. ir. Wilbert Alberts** who assisted me to get started with the Autonomous rover cart during my thesis.

I would sincerely like to thank my manager at ASML, **Dr. ir. Ramon Schiffelers**, for encouraging me throughout the thesis, along with providing me with an excellent environment to work. His support has been paramount at every stage of my thesis.

Finally, I would like to thank my family and friends. Without their effort and support, this thesis would not have been possible.

Suman Mahendra Kumar

Contents

List of Figures

List of Abbreviations

AA	Authorization and Authentication
AAA	Authorization, Authentication and Accounting
ADAS	Advanced Driver Assistance System
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitectire
CoAP	Constrained Application Protocol
CPS	Cyber-Physical System
CPSoS	Cyber-Physical Systems of Systems
DNS	Domain Name Service
DNS-SD	Domain Name Service - Service Discovery
ECU	Electronic Control Unit
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol Secure
IoT	Internet of Things
LWM2M	Light Weight Machine-to-Machine
MQTT	Message Queuing Telemetry Transport
NTP	Network Time Protocol
OCP	Open Connectivity Foundation
OPC-UA	Open Platform Communications-United Architecture
OBU	On-Board Unit
QoS	Quality of Service
RADIUS	Remote Authentication Dial-In User Service
REST	REpresentational State Transfer
RSU	Roadside Units
SOA	Service-Oriented Architecture
SoS	Systems of System
TU/e	Eindhoven University of Technology
URL	Uniform Resource Locator
VCPS	Vehicular Cyber Physical System
V2V	Vehicle-to-Vehicle
V2I	Vehicle-to-Infrastructure
WAVE	Wireless Access in Vehicular Environment
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

Today, the Internet of Things (IoT) has a significant impact on our daily lives by transforming various aspects of Information Technologies and providing inter-connectivity among them. IoT refers to an ecosystem of connected devices, which individually can be a Cyber-Physical System (CPS). A CPS consists of a physical system and a cyber system and is designed such that the physical system provides feedback to the cyber system. In CPS, cyber systems integrate sensing, control, computation, and networking into physical objects such as sensors and actuators, connecting them to the Internet and with each other [5]. Applications of CPS include smart power grids, chip manufacturing systems, autonomous automotive systems, medical monitoring, process control systems, distributed robotics, and automatic pilot avionics [6].

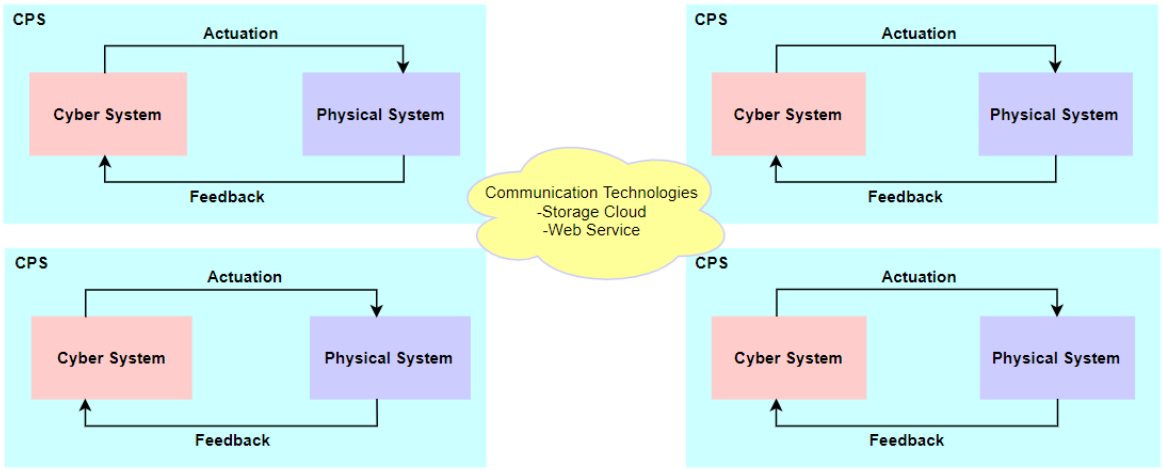


Figure 1.1: Cyber-Physical Systems of Systems

A Cyber-Physical Systems of Systems (CPSoS) is an integration of a finite number of constituent CPS which are independent and operable as shown in Figure 1.1. These CPS in CPSoS are networked together using communication technologies such as storage cloud and web services. Interaction between CPS requires the use of distributed CPS organized as dynamic peer-to-peer infrastructures, capable of autonomously restructuring in response to real-time changes. The realization of CPSoS is challenging due to the need for robust and

flexible interfaces to manage the heterogeneous data exchange between different CPS.

Emphasizing different criteria needed for the development of CPSoS such as real-time computation, architectural support, standardization, interoperability, and security, there are many industrial frameworks and commercial platforms developed for IoT applications. Some of the industrial frameworks are AUTomotive Open System ARchitectire (AUTOSAR), Arrowhead framework, FIWARE, IoTivity, Light Weight Machine-to-Machine (LWM2M), and commercial platforms like Azure IoT suite, Google Cloud IoT, and AWS IoT [1]. The industrial frameworks mentioned in the paper [1], mainly focus on centralized systems and do not support real-time data handling except the Arrowhead framework.

The Arrowhead framework began as a part of the European project Arrowhead (Artemis) and continued its development during the European project Productive 4.0. Presently, the development continues under the Arrowhead Tools project. The focus of the Arrowhead framework is on the collaboration between distributed systems through data and information exchange. The collaboration between systems is based on a Service Oriented Architecture (SOA) [7]. The Arrowhead framework claims to support real-time data handling i.e. giving timing guarantees in CPS [8]. Besides, it also assures data security, safety, flexibility, and scalability of systems.

1.1 Thesis Goal

ASML is the world's largest producer of various lithographic machines which are the key enablers to shrink the sizes of the transistors in accordance with Moore's law. The company is looking into a 'holistic approach' in order to shrink the size of the transistor through the concept of Holistic Lithography. The term Holistic Lithography, first used by ASML, refers to the intelligent integration of patterning, metrology, and modeling [9].

Holistic Lithographic systems use three holistic components - process window enhancement, process window control, and process window detection. These components use Advanced Lithography scanners such as YieldStar and TWINSCAN, Computational Lithography, and Optical and E-beam Metrology integrated with one another in order to improve the overall efficiency of patterning and metrology [9]. Each of these components consists of a cyber and a physical component, thus being a Cyber-Physical System (CPS) by themselves. Holistic Lithography paves the way for the integration of these components creating a Cyber-Physical Systems of Systems (CPSoS). In this direction, ASML wants to evaluate the suitability of the Arrowhead framework in order to use it for the development of Holistic Lithography.

One of the key challenges in the development of CPSoS is real-time data handling. Real-time data handling implies that data must be transmitted timely to meet the real-time requirement of task processing [10]. As mentioned in the previous section, the Arrowhead framework claims to guarantee real-time data handling. At present, the framework has not yet been validated based on timing support in CPSoS. Therefore, the goal of this MSc thesis is to evaluate the Arrowhead framework in the development of timed CPSoS based on timing support provided by the framework.

In order to evaluate timing support provided by the Arrowhead framework, we have developed a concrete use case in the safety-critical domain of Advanced Driver Assistance System (ADAS) where, meeting the timing requirement becomes crucial. The scope of this thesis is to evaluate the timing support of the Arrowhead framework in CPSoS, consisting

of two communicating autonomous rover carts. The purpose of the use case is to detect a pothole and send warning signals to the other carts using the Arrowhead framework. The implementation of the use case consists of a CPSoS which is Arrowhead framework compliant.

1.2 Thesis Outline

This thesis is organized into six chapters as follows:

- **Chapter 2 - Related Work**
  This chapter focuses on the literature review and compares different industrial frameworks based on their support for timing. Also, a brief description of the usage of the Arrowhead framework in similar applications is provided.
- **Chapter 3 - Arrowhead Framework**
  This chapter presents the theoretical background of this thesis. It starts with an introduction to Service Oriented Architecture (SOA) followed by the Arrowhead framework definitions, its architecture, and its management tool.
- **Chapter 4 - Use case, Concept, Design and Implementation**
  This chapter describes the use case, its design, and implementation carried out to achieve the goals presented in Chapter 1. In addition, we have proposed a connectivity solution for smart traffic to develop smart cities using the Arrowhead framework.
- **Chapter 5 - Results and Evaluation**
  This chapter provides the evaluation results achieved from the use case and evaluates the Arrowhead framework for the development of the timed CPSoS.
- **Chapter 6 - Conclusion and Future Work**
  In this chapter, the conclusion of the thesis is drawn, suggestions for future work are presented. In addition, we report on the challenges we faced during the implementation of our demonstrator.

Chapter 2

Related Work

This chapter presents a detailed literature survey about various frameworks developed as mentioned in Chapter 1 supporting the IoT movement. Section 2.2 elaborates on the frameworks that assure timing support. Furthermore, a brief overview of the applications developed using the Arrowhead framework is provided. The literature review conducted in this thesis is a combination of systematic literature review described in '*Guidelines for performing Systematic Literature Reviews in Software Engineering*' [11] and snowballing as described in '*Guidelines for snowballing in systematic literature studies and a replication in software engineering*' [12].

2.1 Frameworks for IoT applications

Derhamy et al. [13], summarize the commercially available IoT frameworks. The known approaches include the IoTivity framework by the Open Interconnect Consortium [14], the Light Weight Machine to Machine (LWM2M) framework by the Open Mobile Alliance [15], the FIWARE platform by FIWARE community [16], the Open Connectivity Foundation (OCF), and the AUTomotive Open System ARchitecture (AUTOSAR) [17]. Along with these, cloud-based IoT platforms such as AWS IoT, Azure IoT Suite, Google Cloud, and ThingWorx are used in the development of IoT applications.

Cristina et al. [1], surveyed most of the above mentioned IoT frameworks and presents an analysis of the mentioned frameworks in comparison with the Arrowhead framework based on automation and digitization key requirements such as real-time and runtime features, industrial support, interoperability, and security. Table 2.1 summarizes the characteristics of each framework.

The real-time feature mentioned in Table 2.1 concerns the capability of middleware to support and manage the application with real-time constraints. Having real-time support is an important aspect in many IoT applications, especially in health care, lithographic domain, and safety-critical applications of vehicles. In those cases, if operations are not performed logically and finished in a bounded time frame, the operation may be useless [18]. Most of the frameworks such as FIWARE, IoTivity, LWM2M, and OCF do not have support for real-time behavior, whereas AUTOSAR and Arrowhead framework claim to provide real-time features that assure fast response time [1].

Features	Arrowhead	AUTOSAR	FIWARE	IoTivity	LWM2M	OCF
Key Principle	SOA,local automation clouds	Runtime, ECU	Context awareness	Device-to-device communication	M2M, constrained networks	Resource Oriented REST, Certification
Real-time	YES	YES	NO	YES(IoTivity constrained)	NO	NO
Run-time	Dynamic Orchestration and authorization	Runtime Environment Layer	Monitoring, Dynamic service selection and verification	NO	NO	NO
Distribution	Distributed	Centralized	Centralized	Centralized	Centralized	Centralized
Open Source	YES	NO	YES	YES	YES	NO
Resource accessibility	High	Low	High	Medium	Medium	Low
Message Pattern	Request-reply, Publish-subscribe	Request-reply, Publish-subscribe	Request-reply	Request-reply, Publish-subscribe	Request-reply, Publish-subscribe	Request-reply
Transport Protocol	TCP,UDP, DTLS/TLS	TCP,UDP,TLS	TCP,UDP, DTLS/TLS	TCP,UDP, DTLS/TLS	TCP,UDP, DTLS/TLS	TCP,UDP, DTLS/TLS
Communication Protocol	HTTP,CoAP, MQTT,OPC-UA	HTTP	HTTP,RTPS	HTTP,CoAP	CoAP	HTTP,CoAP
Usage of 3rd Party legacy sw	YES	YES	YES	NO	NO	NO
Security Manager	AA system and AAA system	Crypto Service Manager	Identity Manger Enabler	Secure Resource Manager	OSCORE	Secure Resource Manager
Standardization	Use of existing standard	AUTOSAR standards	FIWARE NGSI	OFC Standards	Use of existing standards	OFC Standards

Figure 2.1: Characteristics of various IoT frameworkS[1]

2.2 Time-aware IoT Frameworks

2.2.1 AUTOSAR

AUTOSAR is a layered software framework for intelligent mobility, supported by AUTomotive Open System ARchitecture (AUTOSAR), a worldwide partnership of industrial automotive, electronics, and software companies [17]. The layered architecture style organizes a system into a set of layers each of which provides a set of services to the layer above. Adopting the layered architecture style in the AUTOSAR architecture enables the decoupling application development process from the underlying hardware. This property allows a software developer to develop an application for an ECU without being concerned about the hardware architecture, hence making the whole process function-centric rather than ECU-centric.

Figure 2.2 shows a block diagram of the AUTOSAR architecture which mainly includes an application layer, the Run-Time Environment (RTE) layer, and the Basic Software (BSW) layer, all building on top of the underlying micro-controller hardware. The application layer is hardware-independent, the RTE is the interface for applications and the BSW layer is divided into services, electronic control unit abstraction, and micro-controller abstraction. Among all the layers, the RTE is the most important layer which facilitates communication between the application software and the underlying BSW layer thereby decoupling the application layer from the hardware architecture. RTE also maps the runnable entities (a piece of code in the software component) to Operating System (OS) tasks. These runnable are triggered by events called the RTE events. On the occurrence of a certain RTE event, the runnable gets executed and the necessary function is performed.

As AUTOSAR is dedicated to Automotive ECUs and these ECUs are real-time systems that assure real-time data handling [17], the paper [19] attempts to validate the timing by calculating the end-to-end latency in an automotive software application. The end-to-end

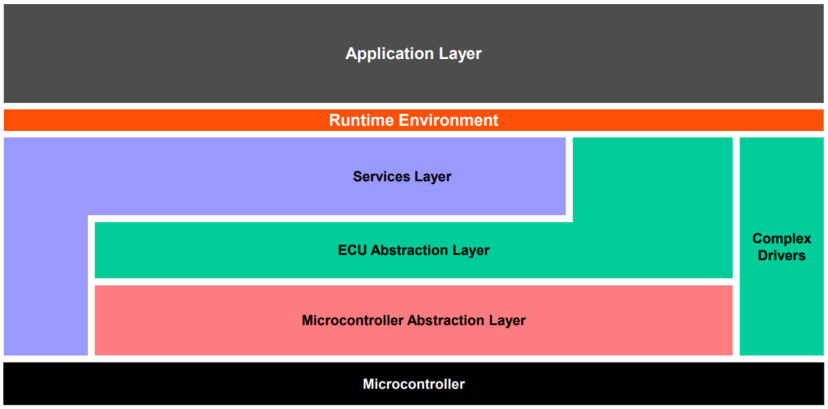


Figure 2.2: AUTOSAR functional block diagram [2]

latency mentioned in paper [19] covers the duration starting from the recognition of a sensor event and the corresponding action of an actuator. The software developed is based on AUTOSAR architecture and is specific to the ADAS system. The end-to-end latency achieved is 70.5 milliseconds. As per the survey conducted on Vehicle-to-Vehicle (V2V) communication for road safety applications, for the vehicle running at the speed of 120 kilometers per hour, the required latency is 100 milliseconds [19]. Therefore, the latency achieved based on the application developed using AUTOSAR is acceptable for safety-critical applications of vehicles.

2.2.2 Arrowhead Framework

The Arrowhead framework is a framework based on a Service-Oriented Architecture (SOA) [8]. It aims to provide automation capabilities such as scalability, security, real-time control, and engineering simplicity while enabling IoT and device interoperability at a service level. It supports SOA-based design principles such as standardized service contracts, late binding, and loose coupling.

The Arrowhead framework claims to guarantee real-time data handling. In addition, the framework also claims to guarantee reliability, scalability, security, and privacy. To meet all these guarantees, the Arrowhead framework introduced the concept of the '*Local cloud*'. A detailed description of the Local cloud and the Arrowhead framework architecture is given in Chapter 3.

2.3 Use of Arrowhead Framework in similar applications

The Arrowhead framework is used as an approach for establishing a condition-based monitoring system within the mining industry, to combat downtime of production and increase the efficiency of the maintenance performed on the equipment [20]. The solution delivers measured data from sensors on the equipment up to the cloud with the dynamic service composition from the Arrowhead framework as a key function within the system. Two different applications are presented in the thesis [21], which uses the Arrowhead framework as a demonstration of the SOA-based framework and the possibilities of collaborative interaction

between embedded devices.

To further demonstrate the usefulness of the Arrowhead framework, it is used to transform CAN bus in the vehicles as a service provider over the Internet whose data can be used by other ECUs acting as service consumers connected to it[22]. The main aim of this thesis was to reduce the load of data on the CAN bus and that was possible using the Arrowhead framework. The Arrowhead framework is used to support security in connected autonomous vehicle platoons to address the domain of secure communication among vehicles, especially the issues related to authentication and authorization of inter-vehicular signals and services carrying safety commands [23].

Chapter 3

Arrowhead Framework

The Arrowhead framework is based on the SOA approach, including the concept of exchange of services between the service provider and service consumer. In this chapter, Section 3.1 provides a brief description of SOA. The common keywords used in the Arrowhead framework are explained in detail in Section 3.2. Section 3.3 presents the fundamental theoretical concept of the Arrowhead framework.

3.1 Service-Oriented Architecture

CPSoS is a collection of CPS that collaboratively form a more complex system which can handle tasks that none of the systems can accomplish on their own. To create a collaborative environment among CPS, each system needs to be independent, distributed, and interoperable. To design such systems, SOA is used as opposed to other legacy client-server models of communicating systems to make collaboration possible [24] [7].

SOA is a design architecture for developing distributed systems that deliver application functionality as services to either end-user applications or other services [25]. The main purpose of a SOA is to facilitate data exchange between a service provider and service consumer through service as shown in Figure 3.1.

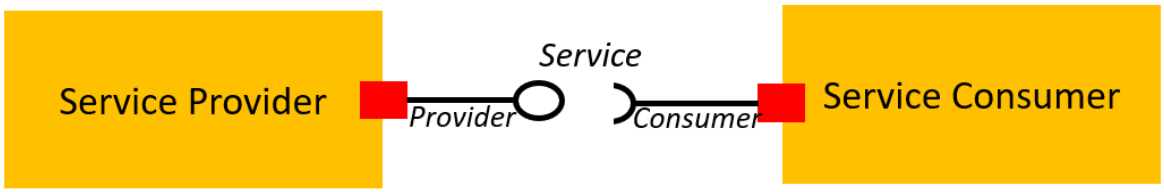


Figure 3.1: The provider and consumer of a Service

The building blocks of a SOA system is the services offered by the service provider. Furthermore, the services are consumed by the service consumer. The service consumer is a piece of software or an application that benefits from the service. The interaction between the service consumer and service is through an interface, which describes the rules for communication and the context of the messages used in communication [26].

Identification of service consumer and service provider is done based on an identifier called an *endpoint* which is required for the interaction on a network [26]. Communication between the service consumer and the service provider is based on sending the messages to these endpoints. A consumer needs to know the endpoint of the services to start communication. When the number of services increases, it becomes complex to manage them in the network. Hence, a Service Registry is used in SOA. Bean et al. [26] describe the Service Registry as a catalog of services hosted at the network. Therefore using the Service Registry, the service consumer can find the service and consumption of services are defined based on the Orchestration rules [27].

To ease the registration and orchestration process, each entity in a SOA plays one of the three roles of Service Provider, Service Consumer or Service Registry. These entities interact using find-bind-execute paradigm as shown in Figure 3.2. In this paradigm, the following operations are performed:

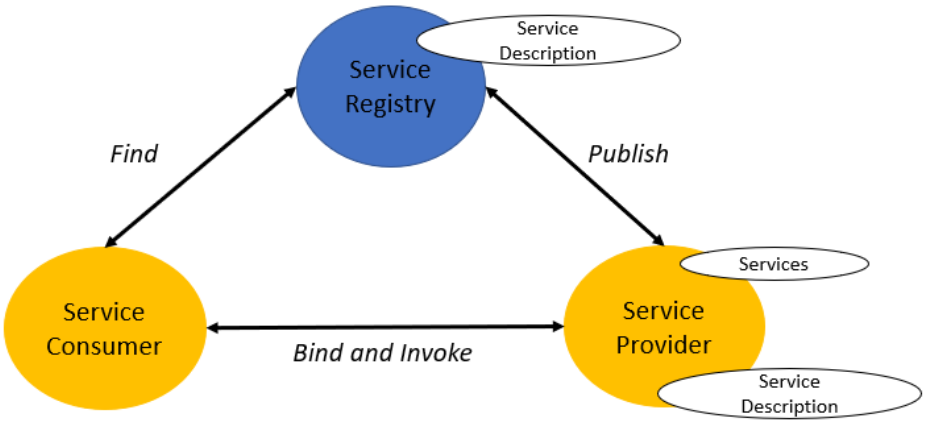


Figure 3.2: Service Oriented Architecture

- **Publish**: To be accessible, a service description must be published by the service provider to the Service Registry so that it can be discovered and invoked by the service consumer.
- **Find**: A service consumer locates a service by querying the Service Registry for the service that meets the required functionality.
- **Bind and invoke**: After retrieving the service description, the service consumer proceeds to invoke the service according to the service description.

The advantage of using a SOA architecture in the development of the software system is that the systems become loosely coupled, uses late binding while exchanging the data, and enable interoperability. Loose coupling means that the two SOA system need not know of each other beforehand, and identification of the systems and possible communication endpoints are discovered or acquired during run-time. By making use of the Service Registry and discovery mechanism supported by the Service Registry, the identification of systems is established. Late binding means that connections between two communicating components

are established first during run-time before data exchange is to occur. Interoperability enables the collaboration and the message exchange of services even though they are developed with different technologies [8]. However, there are a few challenges in using SOA. Simanta et al.[28], argues that the security of SOA is difficult to achieve because it is required that every element, interaction, and system to be secure. Therefore, a single unsecure block may compromise the end-to-end security in the complete SOA-based system [28].

3.2 Arrowhead Definitions

To discuss the Arrowhead framework in detail, a few important keywords are used. These keywords may have other definitions in different domains but in the Arrowhead framework they are defined as follows:

1. **Service**: An Arrowhead framework service is used to exchange information between a providing system and a consuming system using different SOA protocols such as REST (Appendix C.1.1), COAP, XMPP, MQTT or OPC-UA as shown in Figure 3.3. It must be possible for the Arrowhead framework compliant service to be able to register with the Arrowhead framework (Figure 3.6) and be consumed or provided by an Arrowhead framework compliant system [29].

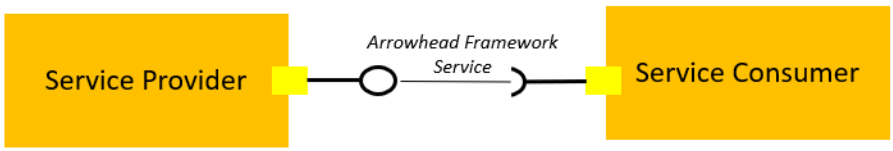


Figure 3.3: Arrowhead Framework Compliant Service

2. **System**: An Arrowhead framework compliant system (Figure 3.4) is one which is providing and/or consuming services. A system can have associated metadata and is implemented in software and executed on a device. An Arrowhead framework compliant system must be capable of consuming Arrowhead framework compliant services, producing Arrowhead framework compliant services, registering itself to the Service Registry system and releasing the authority of the service to the Authorization and Orchestration system [29]. Further details about the Service Registry, Authorization, and Orchestration system and their services are provided in Section 3.3.

3. **Device**: An Arrowhead compliant device is a piece of hardware or a machine capable of hosting one or several Arrowhead framework compliant systems and their services. In Figure 3.5, it is a laptop that is hosting the Arrowhead framework systems exchanging services.

4. **Local Cloud**: The Arrowhead Local cloud aims at providing interoperability by facilitating the service interactions within an operational boundary/ closed environment or at least separated automation environment [3]. The closed environment or operational boundary that exists within the Local cloud is either based on functionality, geography

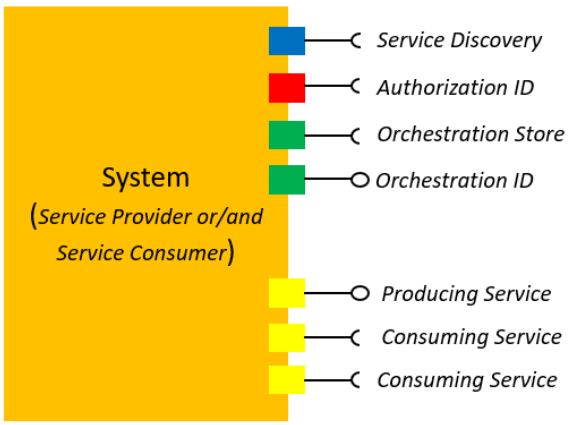


Figure 3.4: Arrowhead Framework Compliant System

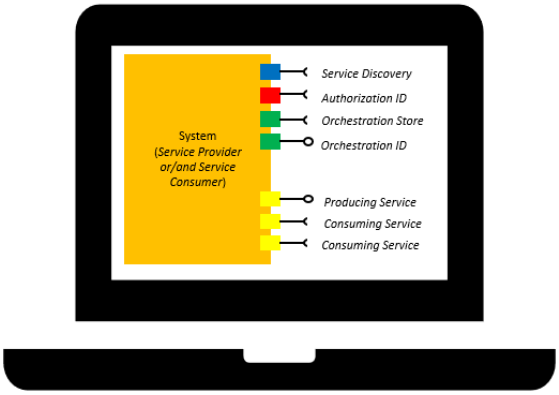


Figure 3.5: Arrowhead Framework Compliant Device

or network-based. Nevertheless, the Local cloud is governed by core systems as shown in the Figure 3.6. The core system resources (related to Service Registry, Authorization and Orchestration) are used by all kinds of entities - application systems in the network, and can also be implemented in a distributed way [30].

5. **Systems of Systems**: An SoS in the context of the Arrowhead framework (Figure 3.7) is defined as a set of systems, which are controlled based on the mandatory core systems and exchange information by means of services.

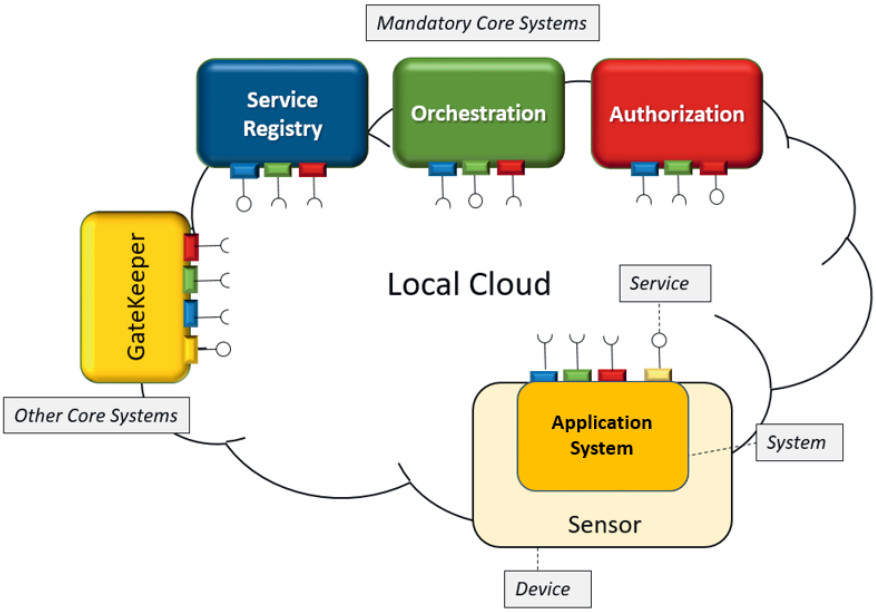


Figure 3.6: Arrowhead Framework Local Cloud [3]

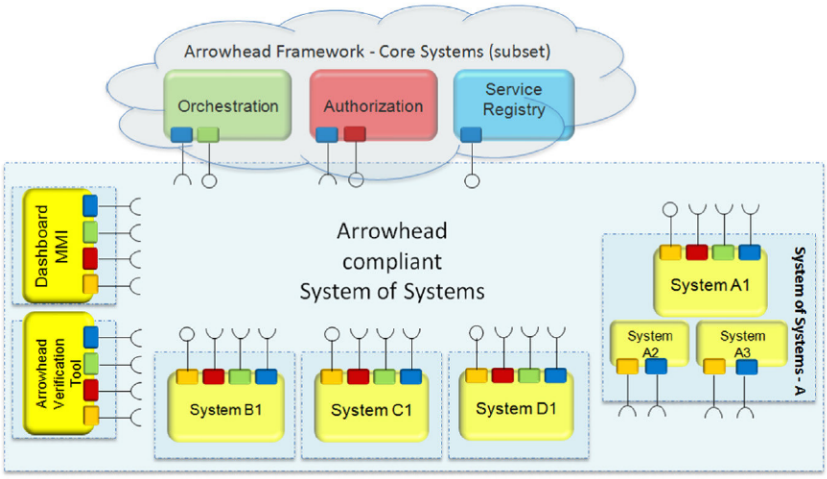


Figure 3.7: Arrowhead Framework Systems of Systems [4]

3.3 Arrowhead Framework Architecture

The Arrowhead framework uses the approach of an SOA to facilitate interoperability between any IoT device by providing service interactions within the Local cloud. A Local cloud is the collection of systems that work together via the Internet, which means the systems need not be at the same physical location. The Arrowhead framework Local cloud should host three mandatory core systems namely, Service Registry, Orchestration, and Authorization along with one or more application systems as shown in Figure 3.6. Along with mandatory core systems, the framework also supports nine automation core systems for extending the Local

cloud capabilities as listed in Figure 3.8.

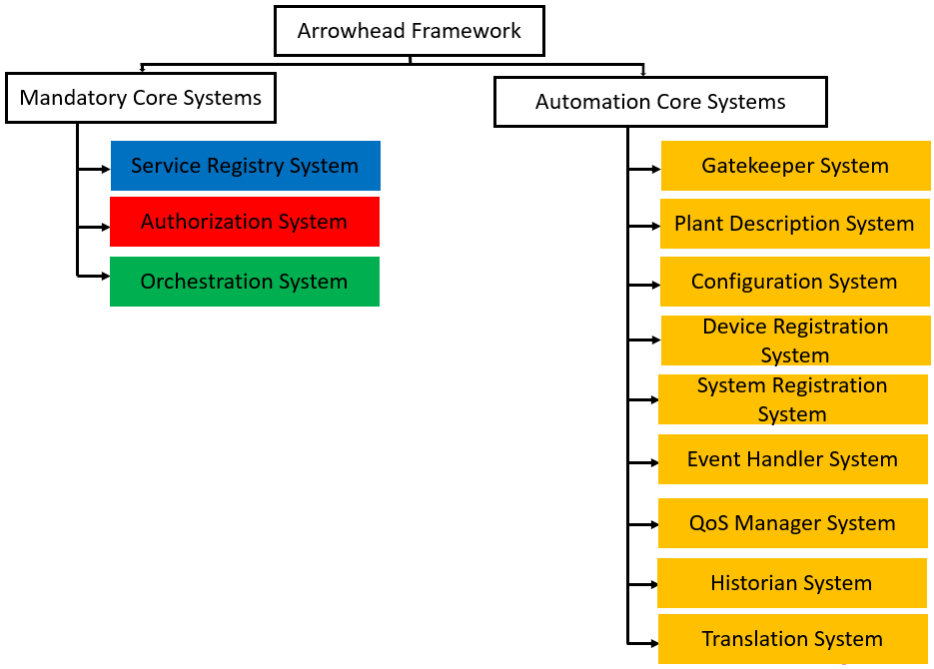


Figure 3.8: Arrowhead Framework Systems

The creation of a minimal working Local cloud based on the Arrowhead framework must incorporate three mandatory core systems. The services provided by the mandatory core systems are called mandatory core services. The mandatory core systems are:

1. **Service Registry System**: The objective of the Service Registry system is to provide storage of all systems that are available in the network and their service offerings within a Local cloud and enable the discovery of them [30]. Systems have to announce their presence and the services they can offer. The registry takes note of this information when the system comes on-line, and might have to revoke them when they go off-line.

   The Service Registry system provides one service and consumes no services as shown in Figure 3.9. The provided service is:

   - **Service Discovery**: It enables a service provider to publish all the services to the Service Registry system. It also allows service consumers to discover the service instances that it wants to consume.

   All the functionalities provided by the Service Registry system are based on the Domain Name Service (DNS) and Domain Name System-Service Discovery (DNS-SD) standards [29]. Using the DNS-SD engine, the services are published, un-published, or looked up in the Service Registry. The service contains a symbolic name and endpoint while publishing into the Service Registry system. The service consumer system can then easily find the service endpoint, IP address, and port using the lookup method provided by the Service Registry system. Furthermore, if the system disconnects from the Local cloud, its services are de-registered from the Service Registry System.

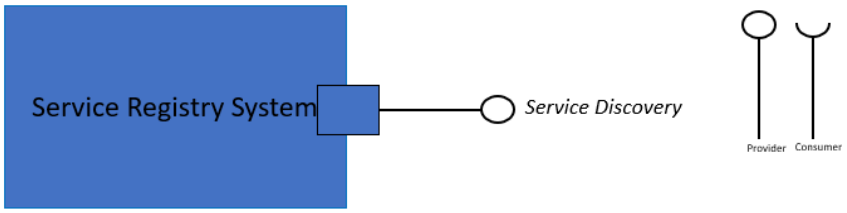


Figure 3.9: Service Registry System

2. **Authorization System**: The Authorization system ensures that a service can only be accessed by an authorized consumer. This system includes both the authentication of the consuming system and the authorization to consume a requested service. It provides two services and consumes one service as shown in Figure 3.10. Two different Authorization systems are defined within the Arrowhead framework: an Authorization and Authentication system (AA) and Authorization, Authentication, and Accounting system (AAA).

   - **AA system**: This system ensures that a service can only be accessed by an authorized consumer based on an X.509 certificate i.e. a public key certificate used in internet protocols [31]. It provides two services and consumes one service as shown in Figure 3.10 and it maintains a list of access rules to system services. The provided services are:

     (a) **Authorization Management**: This service provides the possibility to manage the access rules for each service.

     (b) **Authorization Control**: This service provides the possibility of controlling access to a service and particular resource within the Local cloud.

     The consumed service is:

     – **Service Registry**: Using this service, the AA system publishes all its provided services within the Service Registry system.

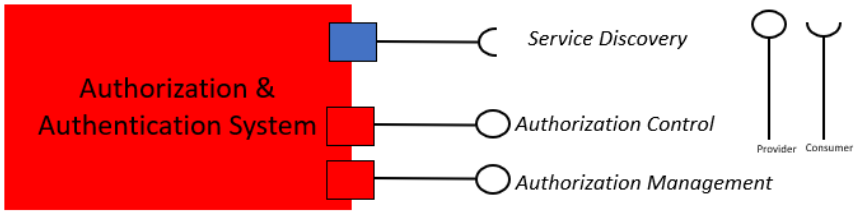


Figure 3.10: Authorization and Authentication System

   - **AAA system**: The AAA system implements an authorization system based on Remote Authentication Dial-In User Service (RADIUS) networking protocol [32] which provides centralized Authentication, Authorization and Accounting management for the users. This system provides three services and consumes one service as shown in Figure 3.11. The provided services are:

(a) **Authorization ID**: This service provides a challenge-response mechanism to get a valid ticket to authenticate each consumer on the Local cloud.

(b) **Authorization Management**: This service provides the possibility to manage the access rules for each service.

(c) **Authorization Control**: This service provides the possibility of controlling access to a service and particular resource within Local cloud.

The consumed service is:

– **Service Registry**: Using this service, the AAA system publishes all its provided services within the Service Registry system.

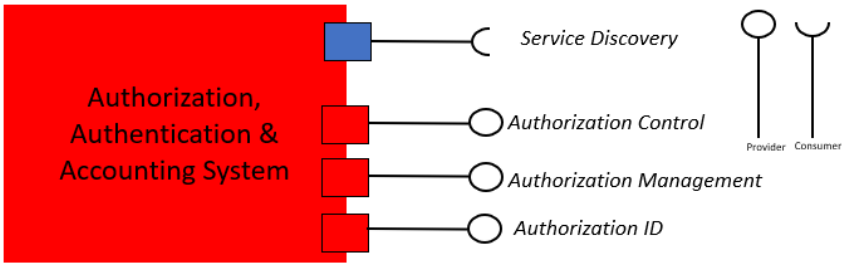


Figure 3.11: Authorization,Authentication and Accounting System

3. **Orchestration System**: The Orchestration system is a central component of the Arrowhead framework and also in any SOA-based architecture [29]. This system is responsible for finding and pairing service consumers and providers. In addition, this system also allows dynamic re-use of existing services and systems to create new systems and functionalities [33]. As shown in Figure 3.12, the Orchestration system provides three services and consumes three services. The provided services are:

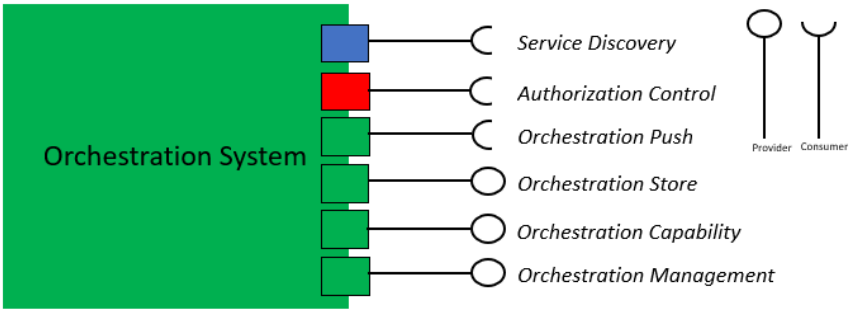


Figure 3.12: Orchestration System

(a) **Orchestration Store**: This service stores orchestration requirements and resulting orchestration rules.

(b) **Orchestration Capability**: This service provides the capability to extract the current state of the system and services based on provider and consumer.

(c) **Orchestration Management**: This service manages the orchestration configuration.

The consumed services are:

(a) **Service Registry**: Using this service, the Orchestration system publishes all its provided services within the Service Registry system.

(b) **Authorisation Control**: This service provides the possibility of controlling access to a service and a particular resource within the Local cloud.

(c) **Orchestration Push**: This service is used to push the orchestration configuration to an application system. The consumer of the service has orchestration rules and the producer of the service has application service. Therefore by pushing the configuration from the consumer using Orchestration push service to the provider, the receiving system will be able to connect its application service.

4. **Automation Core Service**: The objective of the automation core services is to provide support for inter-cloud service exchange and system and service interoperability. Currently, there are ten automation support systems and their associated services [29] are defined and they are:

(a) **Gatekeeper System**: The objective of this system is to enable discovery, orchestration, authentication, and authorization of the services present in the different Local clouds. This is achieved by supporting the inter-cloud service exchange. Inter-cloud service exchange also supports the scalability of a CPSoS.

(b) **Plant Description System**: The objective of the Plant Description system is to provide a basic common understanding of the layout of a plant or a site, providing possibilities for actors with different interests and viewpoints to access their view of the same data set provided by other sources.

(c) **Configuration System**: The goal of this system is to allow the systematic management of configuration information to configurable systems. The configuration system should be able to store and backup application configuration information.

(d) **Device Registry System**: The objective of this system is to store unique identities for the devices deployed within an Arrowhead Local Cloud. The importance of this system is that it provides metadata about all the devices which are stored in the Local cloud.

(e) **System Registry System**: The objective of the System Registry System is to provide the details regarding the systems that are registered within the Local cloud. This is achieved by holding unique system identities for the system deployed within the Arrowhead framework Local cloud.

(f) **Event Handler System**: The objective of this system is to provide support for event-based interaction between application systems along with eventual storage of events and the associated data.

(g) **Quality Of Service (QoS) Manager System**: The objective of this system is to verify, manage, and guarantee QoS for services within the Local cloud.

(h) **Historian System**: The functionality of this system is to provide the possibility to store payload data produced by service registered as well as to keep a historical record of the data produced within a Local cloud.

(i) **Translation System**: Interoperability is possible based on this system. The objective of this system is to provide the translation of protocol, encoding, semantics, and security between the service provider and consumer having non-interoperable SOA configurations.

But as of present framework implementation, three of the automation support core services (Event Handler, Gatekeeper System, and QoS Monitor) can be utilized in developing the application systems and services.

3.4 Arrowhead Framework Local Cloud

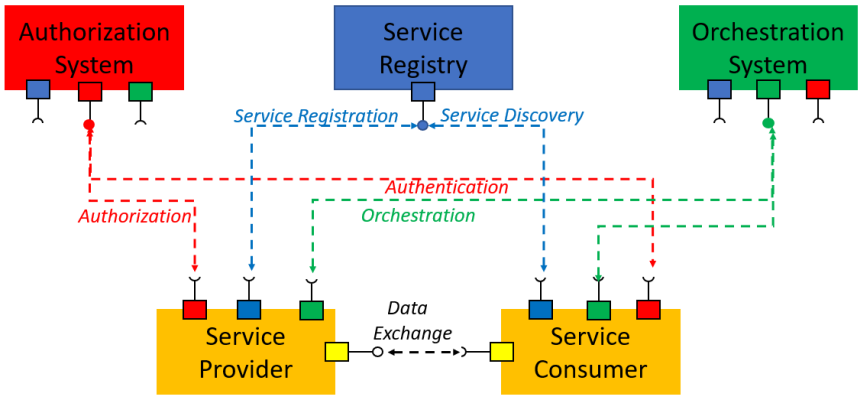


Figure 3.13: Interaction of the Mandatory Core Systems and Application Systems

The Arrowhead framework Local cloud must contain the mandatory core systems and their services and at least one application system. As already mentioned in Chapter 2, the concept of Local cloud was introduced to guarantee latency. The total latency of the Local cloud depends on the communication channel, bandwidth, protocol packet overhead, and the size of the data transmitted. That in turn depends on the communicating systems, MAC layer, type of compression techniques used while sending the data. Therefore, at least by specifying the communication systems (application systems), it is possible to provide certain timing guarantees within the Local cloud. The Service Registry takes care of registering the application systems in the Local cloud. The Service Discovery service provided by the Service Registry system does the registration and discovery of services, systems, and devices. The Local cloud also guarantees security which is provided by the Authentication and Authorisation system and the Orchestration system. The Authorization and the Orchestration system assembles all the individual services and helps service consumers to find the required service using the authorization and orchestration rules. Both service providers and service consumers are consumers of the core services.

Figure 3.13 illustrates how the mandatory core systems communicate with the application systems i.e. service provider and service consumer in the Local cloud. The choreography of interactions between all the systems within the Local cloud is described in Figure 3.14.

At startup, the service provider registers its service in the Service Registry. While registering its services, a provider must supply details about each service such as system name,

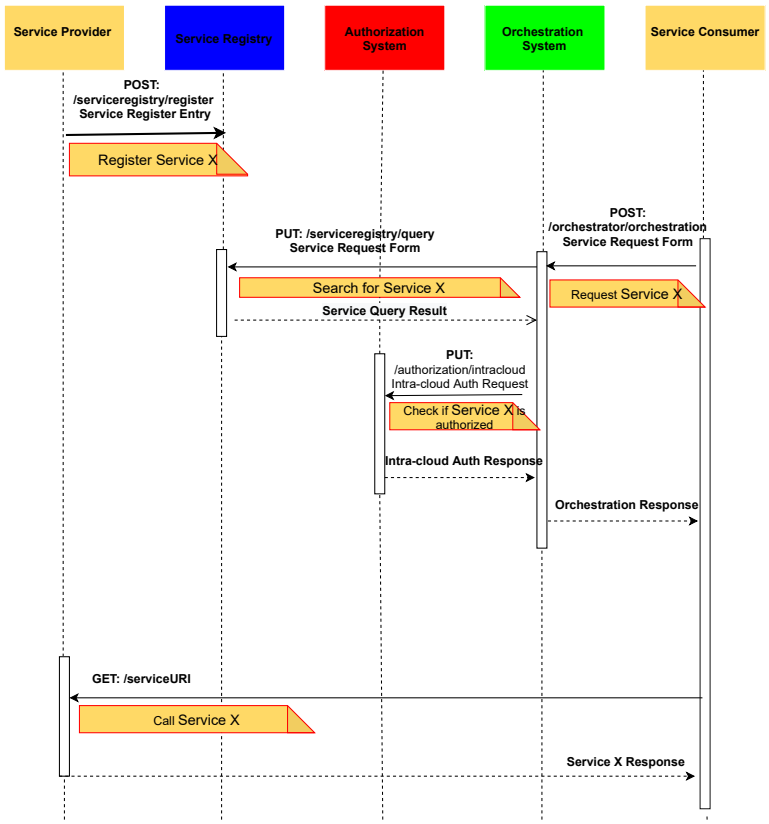


Figure 3.14: Sequence diagram for interaction between the service consumer and the Mandatory Core Systems to enable the consumption of the Service X produced by the service provider

address, port number, and communication protocol used, among others. This information is useful for the Orchestration to reference the best service provider at hand. To keep the Service Registry updated, the service provider refreshes the registration of its services. The sequence diagram presented in Figure 3.14 does not include repetitions or loops to make the figure readable. Similarly, the service request from the consumer to the provider is repeated at regular intervals or when needed. For cyber and IPR security, the services have to be authorized in interaction with the Authorization service. The system can provide its authenticity using a ticket or certificate-based method. Both of the methods are assigned a lifetime, which after expiration needs to be renewed.

When a service consumer requests service X to the Orchestration system using a service request form in the Local cloud that has a service provider for that request. The Orchestration system returns the provider's address to the consumer, as long as the consumer is properly authorized. The provider's address includes appropriate URL, payload, and HTTP methods (Appendix C.1.2) in the web services implementation. The service request form describes what is requested by the service consumer. The query information mentioned in Figure 3.15 is either composed in run-time or composed statically and stored partially in the Orchestration store (that is part of the Orchestration system). The service request form mainly consists of a payload which mainly describes which system is requesting orchestration and what service

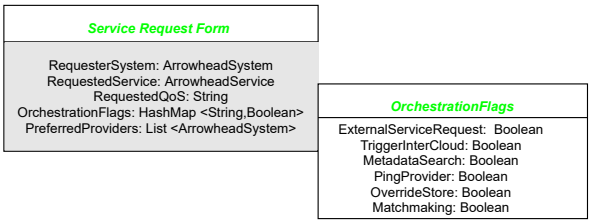


Figure 3.15: Service Request Form

is requested. The remaining fields control the orchestration process, for example, the service consumer can specify their servicing QoS expectations or can force certain preferred service provider they need to connect to.

3.5 Management Tool

The Arrowhead framework provides a management tool created by BnearIT [29]. The management tool connects to the core services and a few automation support systems and their services and presents a graphical user interface as shown in Figure 3.16. The management tool is useful in managing authorization and orchestration rules and also for monitoring the core systems.

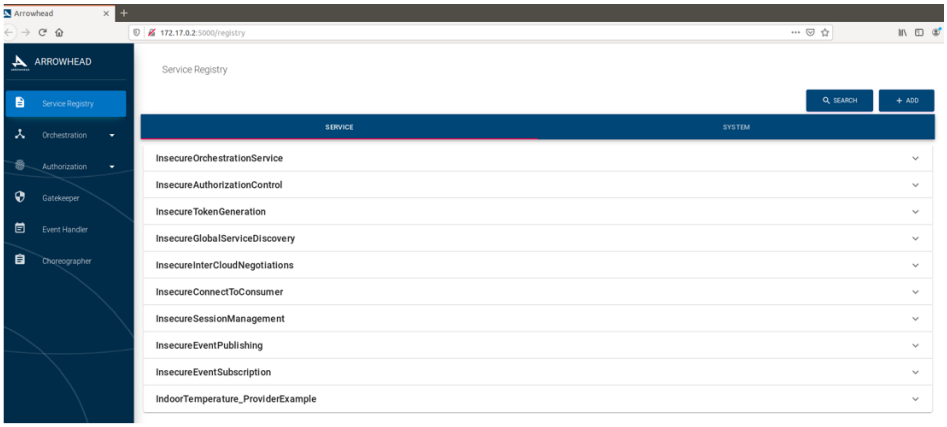


Figure 3.16: Arrowhead Framework Management Tool

Chapter 4

Use case, Concept, Design and Implementation

In this chapter, a detailed description of the use case developed to evaluate the timing support of the Arrowhead framework is presented. In Section 4.2, the use case is extended to the connectivity solution focusing on the development of Intelligent Transportation Services (ITS). Subsequently, the design and implementation of the use case is provided in Section 4.3 and Section 4.4 respectively.

4.1 Use case

This section explains the importance of timing requirements in CPSoS (Section 4.1.1) and also presents a use case in the domain of safety-critical applications of Advanced Driving Assistance Systems (ADAS) in vehicles. As vehicles are complex CPS and for the development of safety-critical applications, these vehicles together form an CPSoS for Vehicle-to-Vehicle (V2V) communication. A detailed description of the challenges faced in V2V communication is discussed in Section 4.1.2. Section 4.1.3 presents the advantages of the Arrowhead framework that can be used to overcome the challenges for V2V communication. Timing is a crucial factor in safety-critical applications and meeting timing deadlines is critical. Section 4.1.5 elaborates the timing requirement for V2V communication in safety-critical applications. Finally, the detailed description of the use case is presented in Section 4.1.4.

4.1.1 CPSoS Timing Requirements

CPSoS are systems that integrate various CPS to work cooperatively together and interact with the real-world to produce global behavior. These CPSoS work simultaneously and mostly in feed-forward loops where the computing process influences the physical process and vice versa. Time is the common entity that computing process and physical process in CPSoS share, and correct interfacing of that is essential to flawless functionality of a CPSoS [34].

To achieve the flawless functionality of a CPSoS, the timing requirement of the physical and computing processes need to be specified. Timing specifications of physical systems which include sensors and actuators are more clearly defined and easier to specify as time is continuous. Whereas in the computation systems the time is incremented in discrete units, making it complex to analyze and specify the timing specification of the computing process

[34]. Therefore, we need a framework or technology that can map the timed CPSoS application onto the platform in such a way that the timing requirement of this application is met.

4.1.2 Safety Critical Applications

In order to evaluate the timing support provided by the Arrowhead framework, a use case in the domain of safety-critical applications of Advanced Driving Assistance Systems (ADAS) in vehicles is developed. Each vehicle can be referred as vehicular CPS (VCPS). VCPS systems are based on coordination between a vehicle's physical mobility along with the cyber (computation and communication) features of the vehicle. ADAS, part of VCPS consists of assistance systems. Some assistance systems are critical to the safety of the driver while other systems may be an added convenience to help the driver to avoid minor accidents. As ADAS systems directly impact safety and accident avoidance, they are required to be real-time systems. These real-time ADAS system needs to be able to take inputs from various vehicle systems as well as the driver's input and produce output in real-time without missing the deadline [35]. Any failure to meet the deadline can result in an accident. Some of the applications of ADAS include Cruise control, Collision-warning, Lane-keeping systems, Blind spot detection, Platooning and Pothole detection.

Most of the applications mentioned above require vehicles to communicate and coordinate via Vehicle-to-Vehicle (V2V) communication and collaboratively process the shared data creating vehicular CPSoS (VCPSoS). These vehicles can communicate their location data using IoT devices such as GPS receivers, allowing the vehicles to send their location through V2V system. In addition, road sensors embedded in vehicles send data about road conditions through Vehicle to Infrastructure (V2I) communication [36]. In order to establish a distributed network for vehicles to achieve V2V communication, various wireless communication technologies focused on short-range communication are defined. Technologies such as IEEE 802.11p in the United States and ETSI ITS-G5 in Europe for low-cost, low-latency for connecting vehicles in distributed manner are used for V2V communication [37].

To achieve V2V communication and to have higher mobility, heterogeneous communication technologies (combination of wireless technologies and IoT) are needed to be utilized in a manner that is most efficient for the overlaying applications and the underlying wireless medium. The properties and characteristics of IoT in vehicles [37] are:

1. High Mobility: IoT must manage the high mobility of vehicles and their impact on wireless communication.

2. Safety-Critical Applications: high reliability and low latency requirements of the systems and applications.

3. Vehicle-to-Vehicle communication: short-range communication among vehicles.

4. Security: to avoid false data propagation from malicious agents in the network.

5. Privacy: driver data, driver behaviour and vehicular sensor data should be privately crowdsourced.

Joshua et al. [37] proposed an architecture to achieve the efficient use of heterogeneous technologies in vehicles (Figure 4.1). Vehicle clusters form concerning geographical locations or observed channel quality at vehicles. Vehicle clusters enable each vehicle within the group,

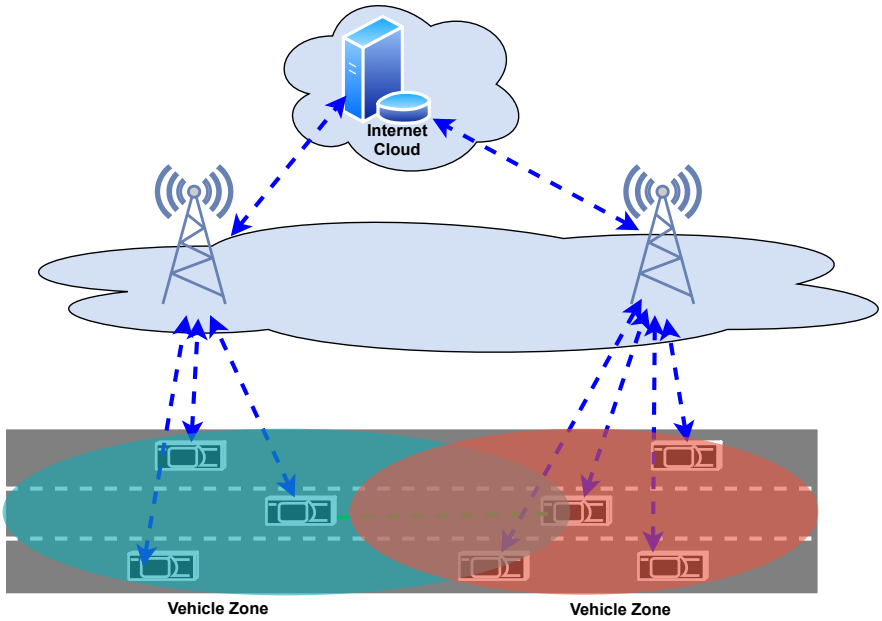


Figure 4.1: Vehicle-to-Vehicle communication using Internet Cloud

to collect data from Cluster Members (CMs) and aggregate this information to the Location server, which would decrease the demand for cellular resources. These clusters of vehicles form the vehicle zone. The architecture (Figure 4.1) raises concerns regarding latency, reliability, security and privacy since the communication channel is open to all vehicles within the network.

4.1.3 Arrowhead Framework

For applications with a strong requirement on timing, reliability, scalability, security and privacy, the Arrowhead framework Local cloud concept claims to fulfil these requirements [29]. Arrowhead Framework Local cloud acts as a protective fence towards any other internet communication. Each Local cloud has firewall protection to surrounding networks, resulting in the blocking of the external network traffic from reaching the interior of the Local cloud [29]. And within a Local cloud, each application system should be Arrowhead framework compliant capable of registration, authorization and orchestration.

4.1.4 Demonstrator

As a demonstrator, a pothole detection and warning system is implemented considering the criticality of time in ADAS, the usage of IoT in vehicles, and the timing support offered by the Arrowhead framework. A pothole is defined as a bowl-shaped depression or hollow regions in a pavement surface, formed by the erosion of rock, especially due to the action of water and traffic passing over the affected area [38]. These potholes can generate damages such as flat tires or wheel damage, impacting the vehicle and leading to major accidents. A vehicle that detects the pothole and informs other vehicles through warning messages could avoid

the occurrence of accidents, thus increasing the driver's safety and comfort.

Fundamentally, a pothole detection and warning system can be described as a group of vehicles communicating through vehicular networks with a closed feed-forward loop to exchange information. Pothole detection and warning system integrate the wireless communication interface on the On-Board Unit (OBU), which allows a running vehicle to collect information from its neighbors or the Roadside Units (RSU). Therefore, this is a CPSoS that integrates multiple CPS leading to the integration of the physical mobility and cyber features of the vehicles. The integration of computing, communication, and physical systems is essential to attain reliability, robustness, stability, efficiency, and performance of CPSoS. Most importantly, meeting the timing requirement is one of the main requirements for CPSoS [34].

The objective of the proposed system is to detect the pothole and inform other cars in its vicinity regarding the pothole by sending a warning message. In many safety-critical applications of vehicles, broadcasting is used to propagate safety messages to vehicles [37]. However, many problems and challenges occur while broadcasting the messages, for example, latency guarantee, high chance of packet loss, reliability, efficiency, and security.

To guarantee reliability and efficiency, different schemes have been adopted in vehicular communication networks which include location-based, position-based, cluster-based schemes and others. In location and position-based methods, broadcasting messages is based on the geographic area of the vehicle whereas in the cluster-based method, broadcasting messages are flooded to the platoon of vehicles that have a common path. Additionally, the growing threat of cyber-physical attacks makes it more challenging to secure the pothole detection and warning system.

Considering these challenges, the Arrowhead framework finds its advantage in the development of the demonstrator. As already mentioned in Section 4.1.3, the Arrowhead framework claims to support timing, reliability, safety, security, and scalability. Using the Arrowhead framework, the cars exchange their location information and whenever a pothole is detected, the location of the pothole and warning messages are sent to other cars through a Location server.

4.1.5 Demonstrator Timing Requirements

In safety-related applications, the broadcast messages sent for V2V communication are time-critical. According to Wireless Access in Vehicular Environment (WAVE) standard [39], each vehicle is recommended to broadcast safety messages in an interval of around 100 milliseconds [19]. These broadcast messages help communicating vehicles to exchange information about their position, movement, and road sensors. This in turn allows the vehicles to make necessary modifications such as speed and direction. For example, if a vehicle is traveling at a speed of 120 km per hour (km/h) and detects the pothole, the warning message should be sent to other vehicles below 100 milliseconds (ms). On the reception of the warning message, the vehicle can either change the lane or reduce the speed as a safety measure. The vehicle at the speed of 120 km/h would have traveled 3.367 meters in 100 ms and on receiving the warning message, the vehicle can further travel to change the lane to avoid a pothole on the road.

Therefore, the timing requirement between the communicating carts in the demonstrator should be below 100 milliseconds. This duration also includes the Service Registration, Authorization, and Orchestration process time of each cart.

4.2 Concept

A large number of cities around the world are investigating the application of a smart city model to enhance the utilization of city infrastructure and resources. Various advanced technologies and techniques supporting such models provide smart services to improve the performance and operations in healthcare, transportation, energy, education, and many other fields [40]. At the same time, these services reduce operational cost and resource consumption in smart cities. Examples of these technologies are Wireless Sensor Networks (WSNs), the Internet of Things (IoT), Cyber-Physical Systems (CPS), robotics, Unmanned Aerial Vehicles (UAVs), fog computing, cloud computing, and big data analytics. The utilization of these technologies provides many advantages and services in smart cities. One such example of smart services is intelligent transportation services that can be used to enhance route planning and congestion avoidance in city streets, provide intelligent light control and parking services, enhance vehicular safety and enable self-driving cars [41].

Along with smart city services and advanced technologies, smart cities require reliable and robust networking and communication infrastructure to enable the efficient exchange of messages among the different components of the system that provide the particular service [41]. In this thesis, we propose a connectivity solution that can be utilized for the development of smart cities. The connectivity solution focuses on vehicular safety contributing to the development of intelligent transportation. The basis for the development of the connectivity solution for smart traffic corresponds to the use of the Arrowhead framework and the characteristics of the IoT in vehicles.

4.2.1 Connectivity Solution for Smart Traffic

The connectivity solution for smart cities (Figure 4.2) consists of several vehicle clusters referred to as *zones* which are formed around the geographical locations. Each zone has a Location server referred to as *Tier 2 server* to collect data from each vehicle in that zone. A Location server maintains a database which receives updates as long as vehicles send their data depending on the presence of the vehicle in that particular zone. All the *Tier 2 servers* send their data to *Tier 1 server*. The *Tier 1 servers* are called as city servers because these servers contain location data of every vehicle present in the city. All these *Tier 1 servers* frequently send their data to *Tier 0 server*. The *Tier 0 server* referred to as the country server keeps an updated database of all the vehicles present in the country. Hence, *Tier 0, Tier 1, and Tier 2* servers collaboratively keep track of a larger set of vehicles and enable communication between the vehicles using the servers. The data, such as location data, pothole data, and speed limit data exchanged between vehicles would be useful in avoiding accidents (sending information through warning signals), and smooth driving (sending information regarding road conditions beforehand), and thus contribute to enhancing vehicular safety.

All the vehicles, as well as servers present in the connectivity solution, should be Arrowhead framework compliant systems capable of Service Registration, Authorization, and Orchestration. When a vehicle enters a zone, it registers itself and its service into the Service Registry. Furthermore, while moving from one zone to another, the intersection area of both the zones should be such that the vehicle can register itself in the new server as depicted in Figure 4.3. The prerequisite of this concept is that the vehicle has to be at least registered in one zone so that whenever the vehicle moves from one zone to another, either Tier 0, 1, or 2 would have the information regarding the vehicle.

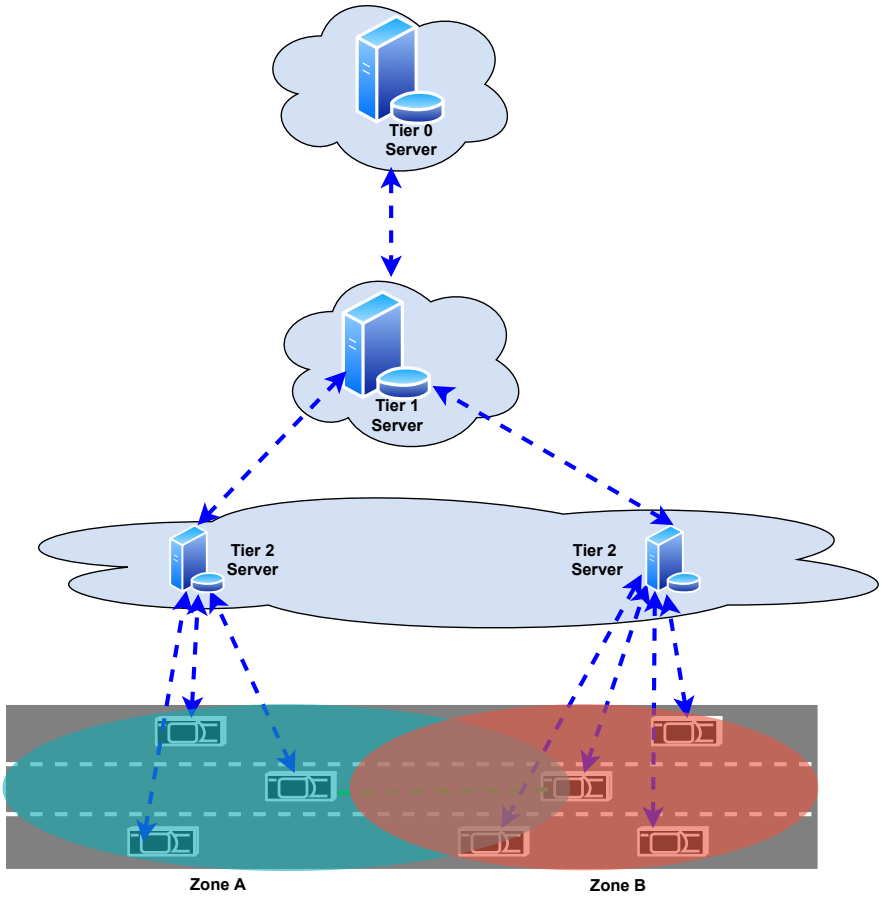


Figure 4.2: Connectivity Solution for Smart Cities

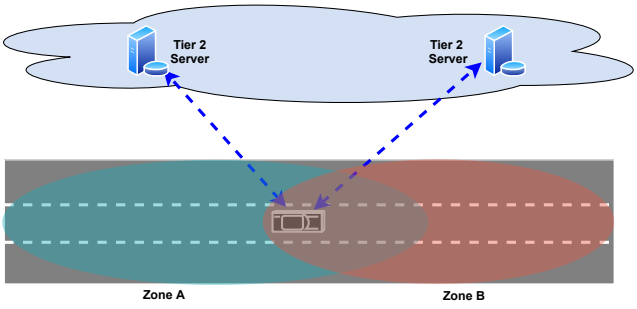


Figure 4.3: Intersection region of Zone A and Zone B

To enable the connectivity solution to work as an operational platform in a smart city, a proof of concept has been implemented and evaluated focusing on vehicles in a single zone and range of the intersection area between zones (Figure 4.3). In our proof of concept, we chose two communicating autonomous rover carts as CPSoS, and communication between them is achieved using the Arrowhead framework.

4.3 Design

4.3.1 System Overview

This section describes the major components of the system and the way they relate to each other. The system overview in Figure 4.4 gives a high-level description of the system. It includes an Autonomous rover cart A, an Autonomous rover cart B, a Location Server, and the Arrowhead framework.

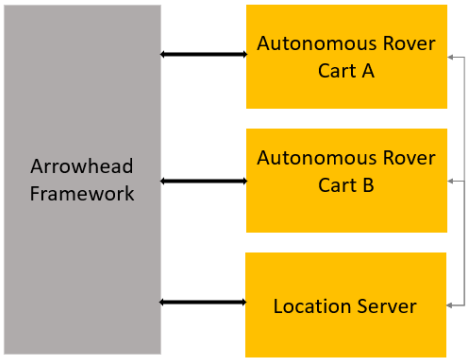


Figure 4.4: System Overview Diagram

- Arrowhead Framework: This framework is responsible for providing timing support, reliability, safety, security, and privacy to all the application systems. Arrowhead framework stores the connection data such as IP address, protocol, and authentication information regarding the systems and their service. It also displays the dataset using the management tool which helps in the easy addition of the systems and their services and deletion of the systems and their services if not used.
- Autonomous Rover Carts A and B: Each autonomous rover cart consists of Raspberry Pi and Arduino board and represents the CPS. The carts follows the lane, detects the pothole using line sensors, and recognizes its own location using the DWM1001C module (Appendix B).
- Location Server: The location server stores location data and warning messages received from the carts. It also provides an interface for accessing the location data of all the carts.

4.3.2 System Data Flow

Figure 4.5 shows a data flow diagram starting from the Autonomous rover cart line sensors to the Raspberry Pi (RPi) through the Arduino board. Figure 4.5 also shows how the Location server and software systems developed on RPi are interacting with the Arrowhead framework. The main data used in the system is primarily sent in JSON format (Appendix C.1.3) which consists of pothole data from Line sensors and location data from the DWM1001C hardware module. These data are sent using the HTTP protocol through the Arrowhead Framework to Location Server where the data is stored in a database. Figure 4.5 presents the core Arrowhead framework which includes Mandatory and Automation core systems.

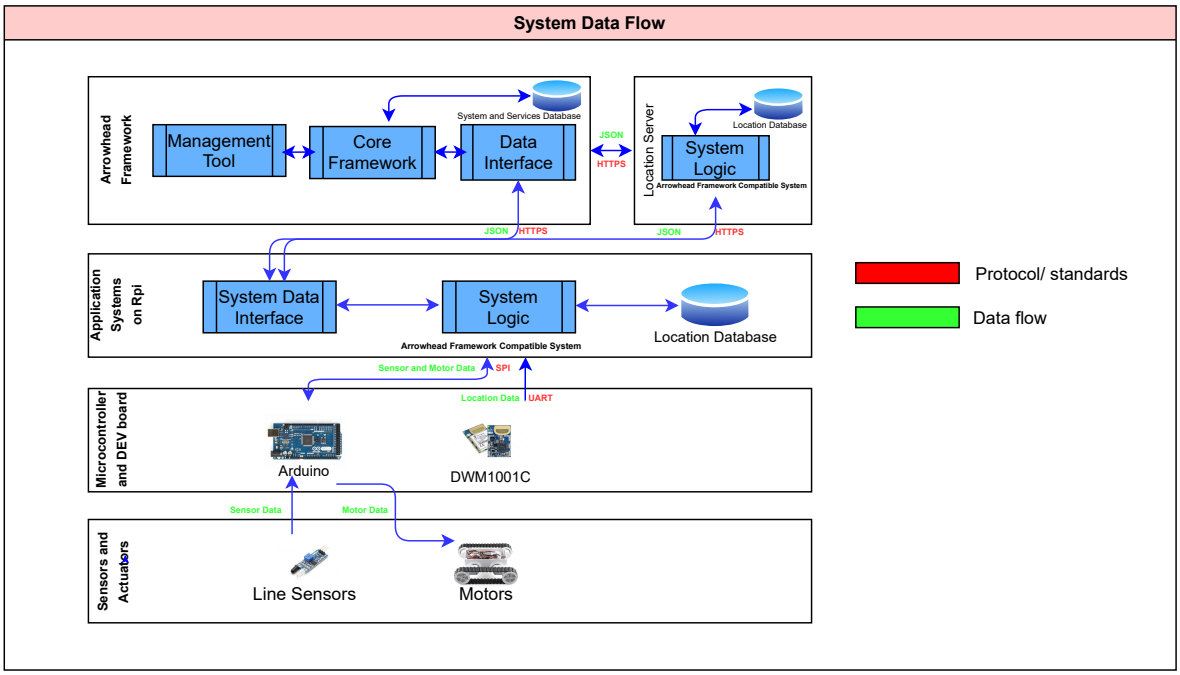


Figure 4.5: System Data Flow

4.3.3 System Design

In this section, we describe the components of the system as shown in Figure 4.6. As mentioned previously, our proof of concept targets a pothole detection and warning system. The components in the design are described in terms of our demonstrator, but the design can be easily applied to other types of services. A local IP based network is created to connect all the components mentioned in Figure 4.6. The components are :

- Arrowhead Framework
  The Arrowhead Framework acts as a tool to add/delete application systems, control their services, and orchestrate to arrange for the data to be delivered to an application system. The Arrowhead Framework is highly dependent on the Mandatory core systems and Automation systems for timing support. Using the Service Registry (Mandatory core system), the application system is added or deleted from the database. If the application system is registered as a service provider in the Service Registry, the service provided by the provider also needs to be registered. The application system registered as a service consumer should provide details regarding the service that it wants to consume using the service request form. A detailed description of the service request form is provided in Section 4.4. Once the application system is registered, it is necessary to authenticate for security and privacy concerns using the Authentication and Authorization system (Mandatory core system). The service consumer requests for the service using the Orchestration system (Mandatory core system). Then, if the service consumer is registered and authorized, it can consume the data from the requested service provider using RESTful Service and REST API. In addition, the Arrowhead framework provides a special interface to all the application systems through the management tool

which is useful to keep records of the application system's properties and its services.

- Autonomous Rover Carts A and B
  The Autonomous rover carts in this research represent the vehicles, which are equipped with line sensors and collision sensors to follow the lane and detect the pothole. They also contain DWM1001C hardware modules to detect the current location based on the Ultra-Wide Band (UWB) technology (Appendix B). The sensors send the data to the Arduino board, which forwards to the sensor system using the SPI interface. The sensor system calibrates the incoming data and sends it to the controller system. The controller system filters the data and processes the data using the Proportional-Integral-Derivative (PID) control loop mechanism. After the processing is done, the data is sent to the motor system to instruct the motor either to go left or right. The DWM1001C module sends the location data of the cart using the UART interface and is stored in the location database. The sensor system, controller system, and motor system act as a service provider and service consumer system. The service provider system of the cart provides a location and warning service and as a service consumer system, receives the location and warning information of the other carts present in the zone. Using the REST Interface, the service providers and service consumers interact with other components. Besides, the service provider and service consumer provide a REST API through which data is sent and received from the Location Server.

- Location Server
  The Location server is the main part of the system. It interacts with the Autonomous rover carts A and B to create the database that contains data regarding real-time location information and pothole warning detection information of each cart. Therefore, the Location server serves as a data storage system. The Location server uses RESTful API to ensure easy interfacing with other application systems and the Arrowhead framework. It also provides REST services through which data is sent and received by all the carts. The service provider system of Location Server receives location data and warning signal information of all the carts. The service consumer system sends location data and warning signal information of other carts present in the lane or zone to the individual carts.

4.3.4 Software Components

The demonstrator, pothole detection and warning system is a CPSoS which is composed of several software components as depicted in Figure 4.7. All the service providers and service consumers of the CPSoS are Arrowhead framework compatible capable of interacting with the Arrowhead framework Local cloud. The interactions of the systems with the framework are represented by the color code of Mandatory core systems. The systems acting as a service provider should be capable of Service Registration and Authorization, whereas the systems acting as service consumer should be capable of Service Registration, Authorization, and Orchestration. Each system, namely Autonomous rover cart A and Autonomous rover cart B, interact with the Location server to facilitate the exchange of location data and warning messages using Location and warning services.

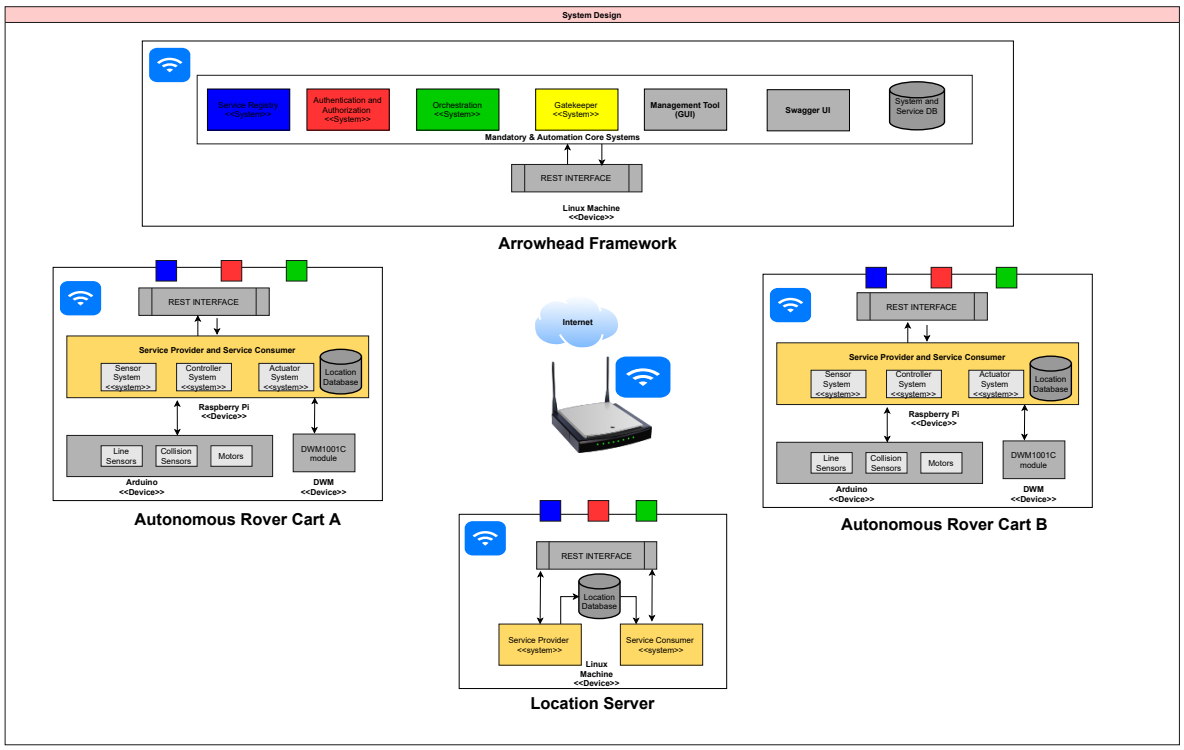


Figure 4.6: System Design

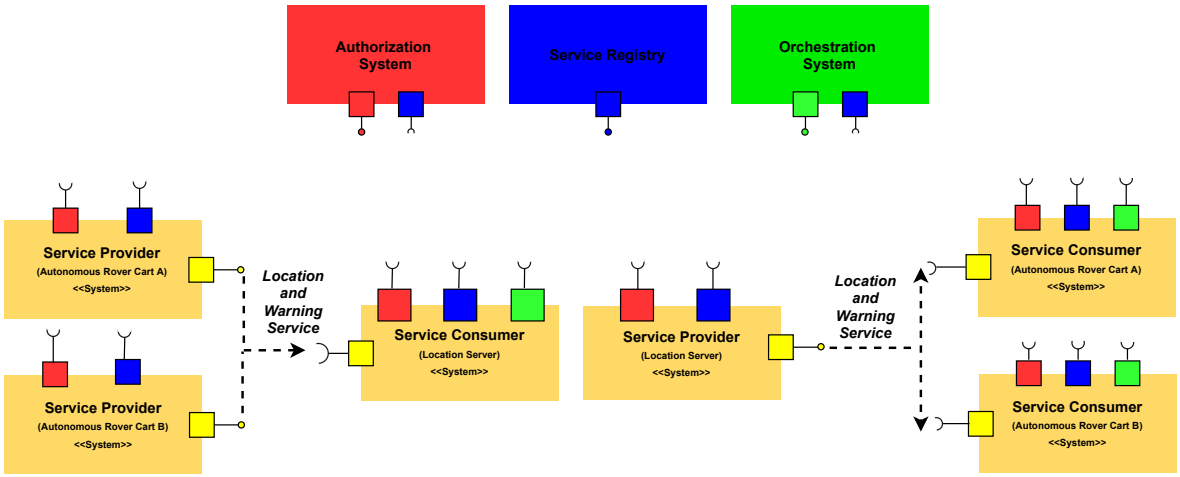


Figure 4.7: Software Components of the Demonstrator

4.4 Implementation

To evaluate the timing support of the Arrowhead framework, a pothole detection and warning system is implemented to calculate the worst-case end-to-end latency in sending location data and warning messages from one cart to another. Furthermore, the implemented application also demonstrates how the Arrowhead framework can be used to develop CPSoS. The pothole

detection and warning system include hardware devices, two autonomous rover carts including a Raspberry Pi, an Arduino, and DWM1001C module and software components (Figure 4.7). Section 4.4.1 provides a detailed explanation regarding the interaction between the software components of the developed application.

4.4.1 Interaction of application systems in Demonstrator

```
{
    "providedService":{
    "id":100,
    "serviceDefination": "Location_and_Warning_Service",
    "interface": [
          "REST-JSON-XML"
    ],
    "serviceMetadata": {
          "location_x": "value",
          "location_y": "value",
           "warning_message": "value"
        }
    },
    "provider": {
          "id": 5,
          "systemName": "RoverCartA",
          "address": "192.168.232.10",
          "port": "8452"
    },
    "port": 8452,
    "serviceURI": "this_is_location_data",
    "version": 1,
    "udp": false,
    "ttl": 0
}
```

Listing 1: Example of Autonomous Rover A registering to the Service Registry

The software components and the sequence diagram of the pothole detection and warning system are presented in Figure 4.7 and Figure 4.8. The demonstrator mainly consists of a local cloud with the Arrowhead framework containing three service providers and three service consumers. The service provider system of Autonomous rover carts A and B and the Location server post the location data and warning signal information by HTTP POSTing, using the World Wide Web's representational state transfer (REST) style. An example message is presented in Listing 1.

With this posting, the service provider system provides its name, its chosen communication semantics, and the path to the service it offers. By default or from an installation update, the service provider knows to register its services at the gateway host (host address and port). Referring to Figure 4.8, the service registry then builds a database of available services from

```
{
        "consumerID": "LocationServer",
        "requestForm": {
                "orchestrationFlags": {
                        "overrideStore": true,
                        "onlyPreferred": false,
                        "externalServiceRequest": false,
                        "enableInterCloud": true,
                        "enableQoS": false,
                        "matchmaking": false,
                        "metadataSearch": false,
                        "triggerInterCloud":false,
                        "pingProviders": false
                },
                "requestedService": {
                        "serviceDefinitionRequirement":
                                "Location_and_Warning_Service",
                        "interface": [ "REST-JSON-XML" ]
                },
                "requesterSystem":
                {
                        "address": "192.168.237.137",
                        "port": 8585,
                        "systemName": "LocationServer"
                }
        }
}
```

Listing 2: Example of Service Request Form sent from Location Server to the Orchestrator in JSON

all possible providers. As Arrowhead Service Registry is based on DNS-SD, an extension to DNS (Domain Name System) which is part of the Internet Protocol Suite's application layer, provides the address at which the location and warning services is hosted. The service discovery extension, DNS-SD, provides the ability to discover location and warning services, which the service consumer of cart A, cart B, and Location server need to use.

The service consumer of the Location server then enquires using a service request form (content can be seen in Listing 2) through the Orchestration system for the location and warning service. In response, it receives the address of the service providers of cart A and cart B with the path to the location and warning service. After receiving the address, the service consumer of the Location server directly contacts the service providers when it requires the desired information, which is also referred to as "pulling". The Location and warning service received from the service provider of cart A and cart B contains their location data and warning signal as shown in Listing 3.

In response, the service provider sends its location and warning data along with its name,

```
"{"
                "\"e\":[{"
                     "\"n\": \"this_is_location_data\","
                     "\"v\":" + sValue +","
                     "\"t\": \"" + sLinuxEpoch + "\""
                     "}],"
                "\"id1\":" + sValue +","
                "\"vx1\":" + sValuex +","
                "\"vy1\":" + sValuey +","
                "\"vb\":" + sWarning +","
                "\"bn\": \"this_is_location_data\""
           "}"
```

Listing 3: Example of Autonomous Rover A sending its Location data and warning message in SenML format [42]

the timestamp of measurement to the service consumer of the Location server. Alternatively, after being requested by a service consumer of the Location server, the service provider could "push" an update at a specific time interval or upon some agreed event. The Arrowhead framework supports both push and pull behavior. However, in our implementation, pull behavior is used.

Upon receiving individual location data and warning signal information of each cart, the service consumer of the Location server stores the data in its internal database created using MySQL (Appendix C.1.4). The internal database named Location database has a table with the following fields: *CART_ID, LOCATION_X, LOCATION_Y, WARNING_SIGNAL, and TIMESTAMP*. This internal database contains all the information of carts A and B and also including the location of the pothole. Thus, the Location server acts as a Tier 2 server that has location data of all the carts present in the zone. The multiple location servers present in different zones can communicate with each other through the Gatekeeper system, part of the Arrowhead framework using inter-cloud communication [8].

Using the location database, the service provider of the Location server sends the location data and warning signal of cart A to cart B and cart B to cart A. If cart B is following cart A in the same lane and cart A detects the pothole, then the warning signal is received by cart A to cart B through the Location server. On detection of the pothole, the cart A stops, and cart B changes the lane and moves forward. The Arrowhead framework provides a RESTful interface for communication between cart A, cart B, the Location server, and the Mandatory core systems. Furthermore, Figure 4.8 shows the sequence diagram of all the application systems involved in pothole detection and warning system interaction with all the three Mandatory core systems along with their services in a Local cloud.

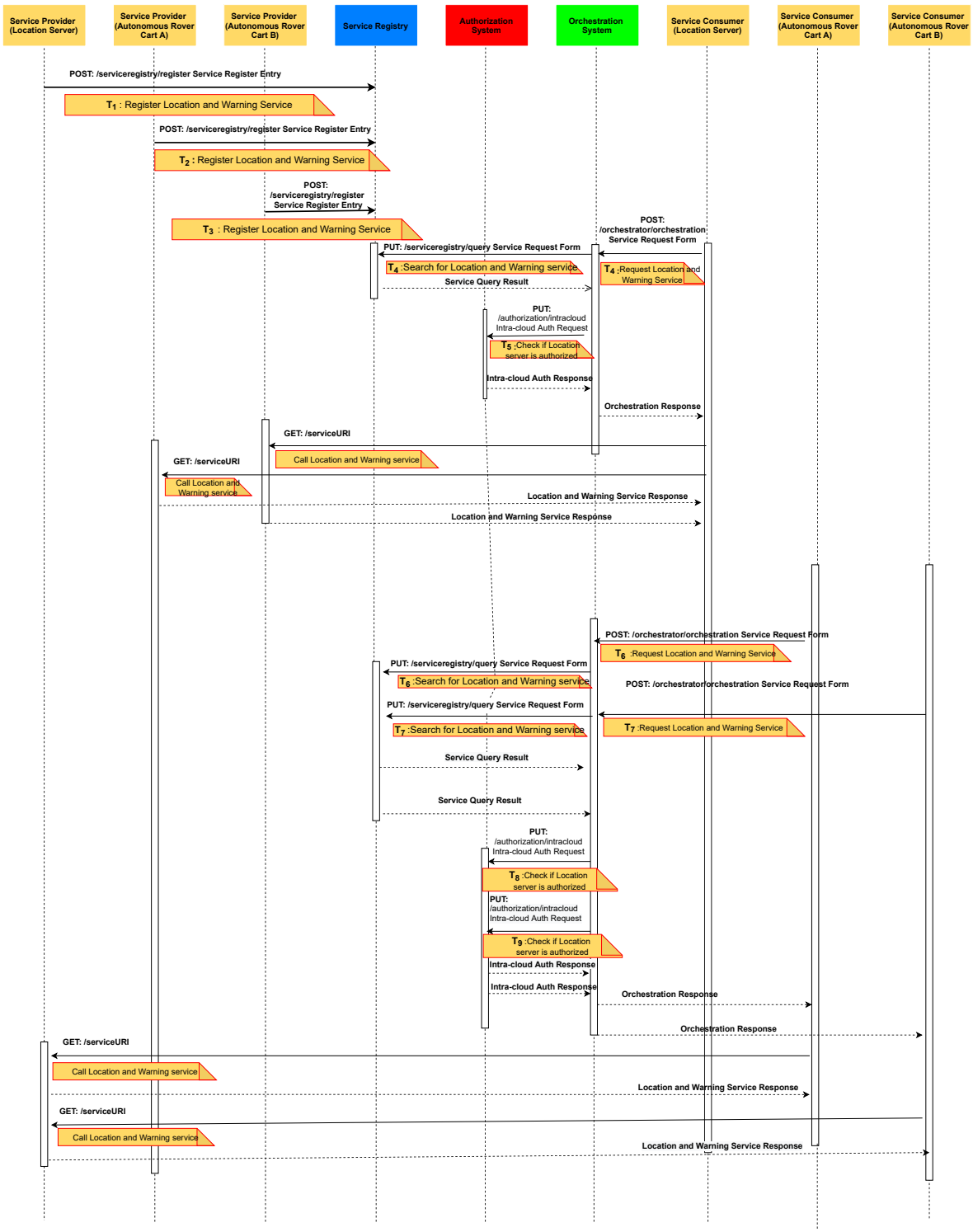


Figure 4.8: Sequence Diagram

Chapter 5

Results and Evaluation

As stated in the introduction chapter, the main goal of this thesis is to validate the timing support of the Arrowhead framework for CPSoS. To evaluate the suitability of the Arrowhead framework, the demonstrator is evaluated in the context of the timing requirements for V2V communication in safety-critical vehicles applications as defined in Chapter 4. Furthermore, the pros and cons of the Arrowhead framework used in Section 4.1 are discussed. Additionally, relevant suggestions concerning the Arrowhead mandatory core systems are presented. Finally, the evaluation is summarized.

5.1 Experimental Setup

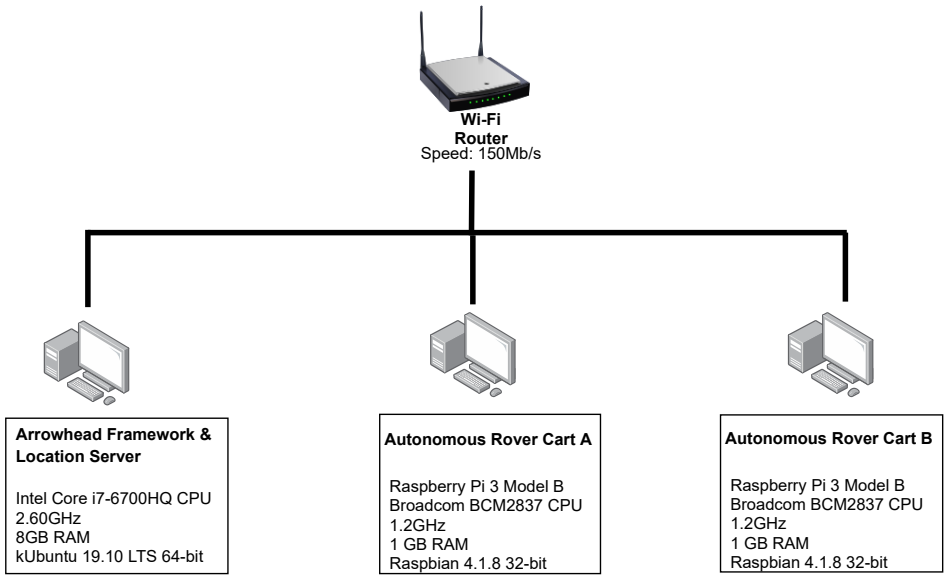


Figure 5.1: Testing environment for Pothole detection and Warning system

The experimental setup consists of the Autonomous rover cart A, Autonomous rover cart B, the Location server, and the Arrowhead framework Local cloud (which includes all the

Mandatory core systems). A general view of the setup is presented in Figure 5.1. Each autonomous rover cart consists of Raspberry Pi, a single board general-purpose computer with Raspbian (Debian based) operating system with support for Python 3.6. The service providers and service consumers of the carts A and B are deployed on respective Raspberry Pi controllers and interact with the Arduino board via the SPI and with the DWM1001C modules using the UART as shown in Figure 5.2. Location data is sent to the Raspberry Pi using DWM1001C via an UART interface. A pothole detection message is sent to the Raspberry Pi using the Arduino board over an SPI interface. The applications systems (service providers and service consumers) are developed on the Raspberry Pi provides an interface to interact with the Arrowhead framework core systems.

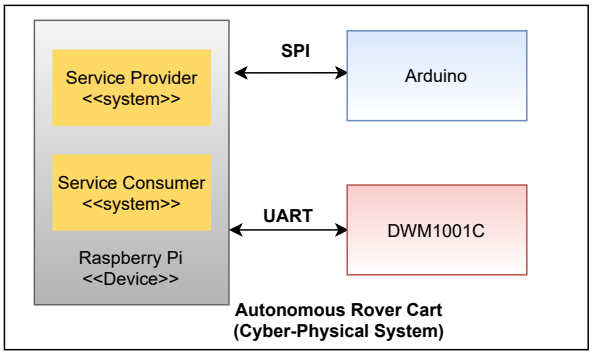


Figure 5.2: Block diagram of Autonomous Rover Cart

The Arrowhead framework which consists of the Service Registry, Authorization, and Orchestration systems along with other core systems is deployed on the Kubuntu 19.04 Linux virtual machine inside the Windows 10 environment. The requirement for the deployment of the Arrowhead framework is that the machine should at least have 4GB RAM [43]. An interactive GUI i.e. the management tool is provided by the Arrowhead framework which is installed separately using the docker image in the Linux virtual machine [43]. In addition to the framework running in the virtual machine, the service provider and service consumer of the Location server are also developed in the same environment.

To measure the latency between the systems, each time one of the application systems sends or receives an HTTP request, it outputs a message describing the action and current timestamp. In terms of the messaging workload, 100 requests are sent to the cloud's Orchestration system using the Service request form (see Listing 1), with a one-second delay between each message. Each request is sent on a 150 Mb/s Wi-Fi Router. Moreover, all system clocks were synchronized using a local NTP server, which provides accuracies in the range of 0.1 ms [44].

5.2 End-to-End Latency

To validate the timing support of the Arrowhead framework in CPSoS, the pothole detection and warning system was implemented successfully, consisting of six application systems as shown in Figure 4.7. The communication among the application systems is achieved using the Arrowhead framework Local cloud concept. A test was carried out using six application

systems of all the parties as depicted in Figure 5.1 and Figure 4.7. The experiment was performed with each service consumer and service provider sending 100 message requests to the Arrowhead framework to accurately measure the end-to-end latency in the pothole detection and warning system.

The pothole detection and warning system behave according to the visual order as depicted in the sequence diagram (Figure 4.8). After sending 100 requests, the minimum, average, and worst-case latency is measured for each message request and response in interaction with the Arrowhead framework and the same is denoted by timestamp in the Figure 4.8. The average and worst-case latency of each message are:

- T_1 corresponds to the Location server registration time. The Service Registry's database queries takes an average of 71.196 ms and a worst-case response time of 106.408 ms.
- T_2 and refers T_3 to the Autonomous rover carts A and B service registration time. The Service Registry's database queries takes an average of 463.707 ms with the maximum latency of 594.836 ms.
- T_4 is the elapsed time of the Location server until the service consumer of the Location server received the Orchestration response to its request. The worst-case end-to-end latency is 662.334 ms and the average end-to-end latency is 287.739 ms (Figure 5.5).
- T_5 refers to the Authorization's queries of the Location server. It takes an average of 14.186 ms and a maximum latency of 248.95 ms. The frequency distribution of the Authorization and also the Service Registry query latencies is shown in Figure 5.6.
- T_6 and T_7 corresponds to the elapsed time of the Autonomous rover carts A and B orchestration time. The worst-case end-to-end latency was 989.691 ms and the average end-to-end latency was 570.978 ms. The corresponding latency for each Orchestration response can be seen in Figure 5.3.
- T_8 and T_9 represents the time taken by the Autonomous rover carts A and B in interaction with the Authorization system. The service consumer of the Autonomous rover carts A and B takes an average of 171.293 ms with a maximum of 296.907 ms. Figure 5.4 shows the frequency distribution of the Authorization and also the Service Registry query latencies.

Table 5.1 shows latencies of the cart A, the Location server, and the cart B in interaction with the Arrowhead framework Local cloud. Along with the worst-case latency of each system, other latencies need to be considered to calculate the end-to-end latency to communicate location data and warning message from one Autonomous rover cart to the other cart. The DWM1001C module which is responsible for location data, updates at every 10 ms. Once the Orchestration, Service Registry, and Authorization is successful, every other data exchange takes place using an HTTP request and response. Every HTTP request and response from the Autonomous rover cart to the Location server takes at maximum of 55.40 ms and an average of 46 ms, whereas from the Location server to the cart the maximum latency is 9.667 ms and on average it took 6 ms.

As per the demonstrator, when a vehicle enters into the zone, it has to register in the Service Registry. The worst-case latency to register in the Service Registry is 594.836 ms. And in order to communicate with other vehicles in the zone, the vehicle has to orchestrate

Timing breakdown for End-to-end latency calculation			
	Min(ms)	Avg(ms)	Max(ms)
T_1 : Service Registration of Location server	52.364	71.19	106.408
T_2 : Service Registration of cart A	430.804	463.707	594.836
T_3 : Service Registration of cart B	430.804	463.707	594.836
T_4 : Orchestration of Location server	182.567	287.739	662.334
T_5 : Authorization of Location server	9.12835	14.186	248.95
T_6 : Orchestration of cart A	470.146	570.978	989.691
T_7 : Orchestration of cart B	470.146	570.978	989.691
T_8 : Authorization of cart A	150.5073	171.293	296.907
T_9 : Authorization of cart B	150.5073	171.293	296.907

Table 5.1: Timing breakdown of interaction of each system with the Arrowhead framework in the order of Sequence diagram (Figure 4.8)

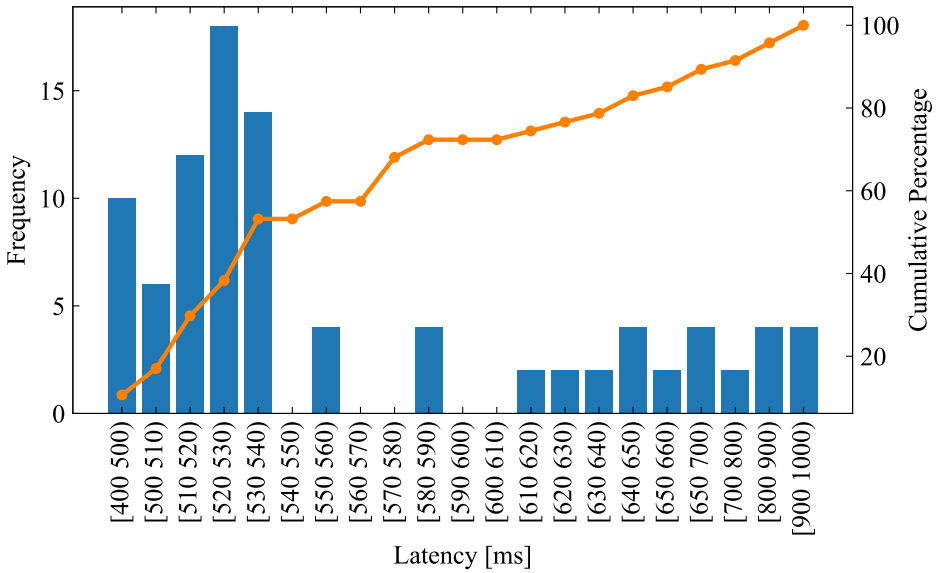
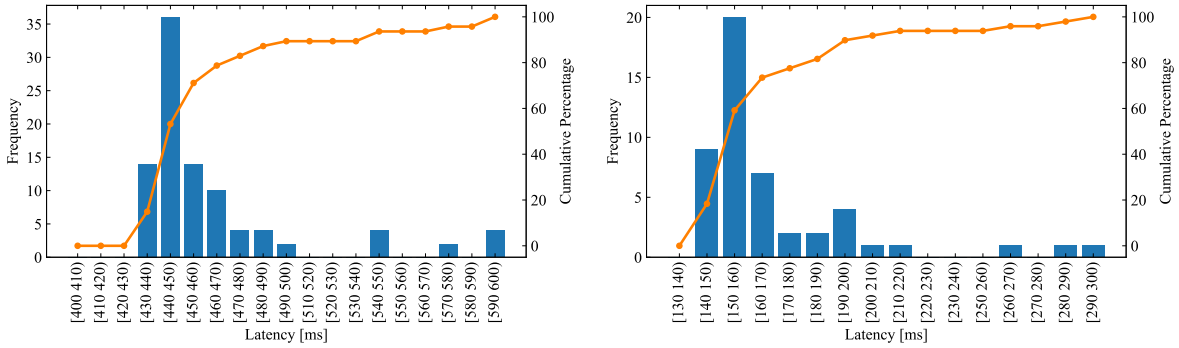


Figure 5.3: Orchestration Latency of Autonomous Rover Cart in milliseconds (ms)

and authorize it. The worst-case latency to authorize in the Authorization system is 296.907 ms. And the maximum latency to orchestrate using the Orchestration system is 989.691 ms. The maximum amount of time is spent in the orchestration process, where the vehicle finds the respective vehicles within the zone. Once the Service Registration, Authorization, and Orchestration is successful, then when the vehicle detects the pothole, it warns all the other vehicles present in the zone via the Location Server. The worst-case latency for the Service Registration, Authorization, and the Orchestration taken by the Location Server is 1017.692 ms.

Therefore, the worst-case end-to-end latency to send the first location data and warning message from the cart A to the cart B through the Location server is 4885.298 ms (4.8853 seconds) whereas the average end-to-end latency is 2.70433 seconds. This duration is the sum of all latencies of all six application systems of the demonstrator in communication with the



(a) Service Registration latency

(b) Authorization latency

Figure 5.4: Frequency Distribution of database query latency of Service Registry and Authorization of Autonomous Rover Cart

Arrowhead framework, the DWM1001C module, and the HTTP request-response. After the first message, every other location data and warning message exchanged between the vehicles takes at a maximum of 120.467 ms and an average of 98.06 ms. Considering the worst-case end-to-end latency, if the vehicle is moving at 120 km per hour (km/h), at least it would have travelled 162.81705 meters in 4.8853 seconds while exchanging the first location data and warning message with other vehicles within the zone. Regarding the intersection region between the zones based on the achieved worst-case end-to-end latency, it is recommended to be in the range of 150m to 250m for successful registration in the new zone.

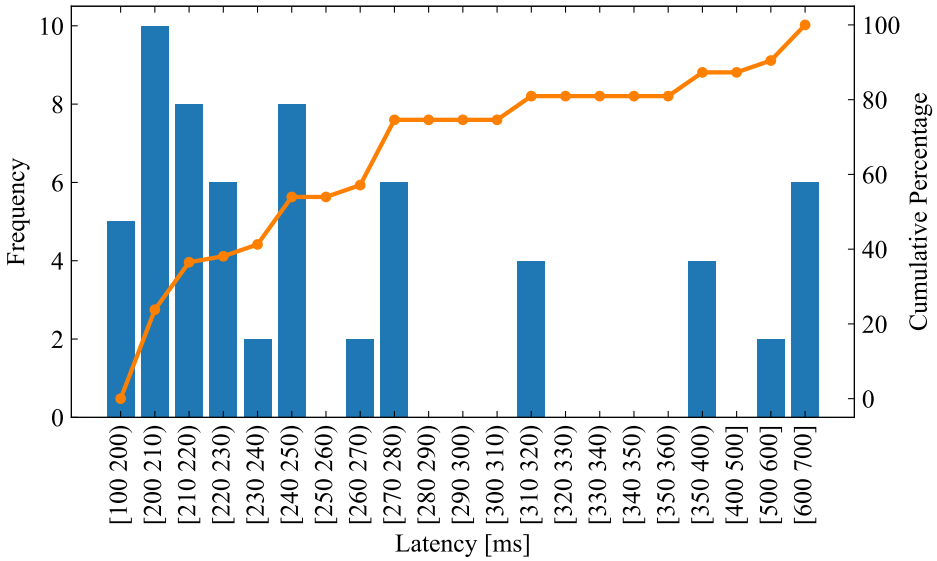
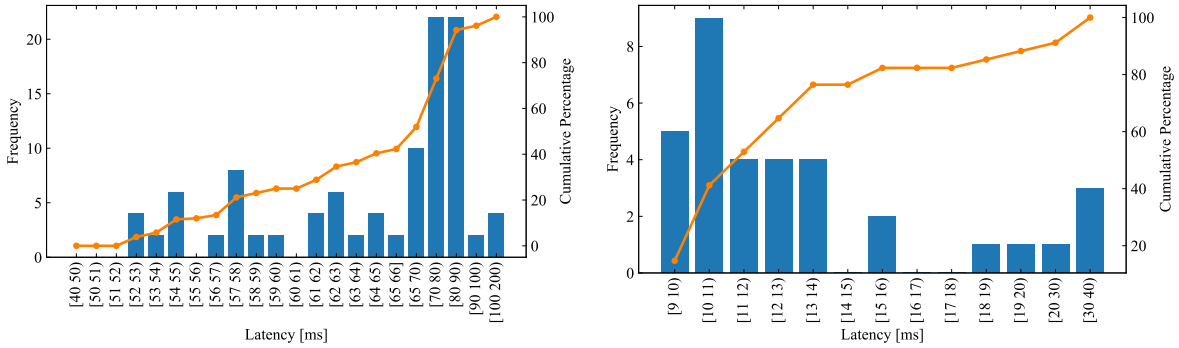


Figure 5.5: Orchestration Latency of Location Server in milliseconds (ms)

Based on the timing requirement for safety-critical V2V communication (Section 4.1.1), the end-to-end latency for road-safety applications should be within 100 ms. In our application, the worst-case end-to-end latency measured is 4885.298 ms which is lot more than 100 ms. One of the reasons for larger latency is the Orchestration process. Among all the latencies,



(a) Service Registration latency

(b) Authorization latency

Figure 5.6: Frequency Distribution of database query latency of Service Registry and Authorization of Location Server

the Orchestration from the Autonomous rover carts to the Location server took maximum time. The way the Arrowhead framework is implemented also introduces significant overhead in latency. A detailed explanation is provided in next Section 5.3.1 and suggestions to improve the Mandatory core systems are presented in Section 5.3.2.

5.3 Discussion

5.3.1 Timing support in the Arrowhead Framework

One of the reasons for the relatively high latency is the way the Arrowhead framework is implemented [43]. The Arrowhead Mandatory core systems and other automation systems use a REST-based architecture implemented on top of Grizzly [45] and Jersey [46]. Grizzly comprises of a core framework that facilitates the development of scalable event-driven applications using Java Non-blocking I/O API and the HTTP based services towards client-side as well as server-side. Jersey is a framework that facilitates the development of RESTful web services and its clients, by providing an implementation of the JAX-RS API (standard specification for developing REST services in Java) and some extensions. The default use of Jersey (which uses servlets as its underlying mechanism [47]) will lead to the creation of a new thread for each request that is destroyed after its work is completed. Thus, RESTful services using standard Jersey will slow down when there are thousands of requests sent at the same time or at a very fast pace. Rafael et al [48], proposed a solution to configure the thread pool which will reuse threads instead of destroying them on the Grizzly HTTP server module because Grizzly is responsible for creating new threads in Jersey's model. In our opinion, it is an insufficient solution for resource-constrained devices because creating a thread-pool limit the number of threads that can be created, this in turn limits the HTTP requests or to increase the execution of the multiple numbers of threads, the threads need to execute in parallel which is possible using GPU or other high-end devices.

Another reason is that the Arrowhead framework is developed based on the SOA style. In SOA, as all the services and systems communicate with each other over the network, leads to an increase in the latency [49]. One of the examples for this is the Orchestration response of the Autonomous rover cart. The average orchestration latency is 570.978 ms which appears to be slightly larger, given that this process consists of a simple service query and authorization

validation on a private network. Furthermore, all the core Arrowhead framework systems are located inside the same machine, the requests between the Service Registry, Orchestration, and Authorization systems are sent to the local host, so it is expected to consume less time.

5.3.2 Mandatory Core Systems in the Arrowhead Framework

The Arrowhead framework version number 4.1.3 (major.minor.patch convention used in open-source projects) of the evaluated framework [43] is used in the thesis work. However, this version was not mature yet. Currently, the framework is only suitable for small scale test setups. As already mentioned, the Orchestration System, Authorization System, and Service Registry form the Mandatory core systems of Arrowhead Framework. We discovered a few flaws in these systems:

- Service Registry: In the implemented demonstrator, both the location data and warning message are combined in a single service to avoid multiple registrations of a single system with multiple services. The issue with the Service registry is that if the system provides multiple services, then it is not possible to register all the services with one registration call. This means that every-time the system has to register per-service basis. A similar problem exists during the service de-registration process.

- Authorization System: Before a consumer can discover anything using the Orchestration system, the authorization rules are mandatory to be added in the Authorization system mentioning which system can consume which service. This information gets stored in local authorization tables through which the Orchestration system interacts. Thus, the consumer and provider are tightly coupled to each other through the Authorization system. This strictness of adding authorization rules and the tight coupling can affect the flexibility to the user to find the respective service.

- Orchestration System: Though the service discovery is possible through the Orchestration system, it still lacks features that would be necessary for it to be usable in the industry. One example of this is the infancy of the metadata-based service discovery. While registering services in the service registry, the application system can specify a set of arbitrary key-value based metadata fields. Another example is the configuration feature ping-provider service provided in the Service request form which checks whether the provider is online or not. If the provider is inaccessible it removes the provider from the list. However, the orchestration system is not capable of fully leveraging either metadata-based service discovery or ping-provider service.

5.4 Summary of Evaluation

The pothole detection and warning system detects the pothole and informs all the other carts in the zone based on warning messages as presented in Chapter 4. The advantage of using the Arrowhead framework is that the development of Systems-of-Systems is well supported. Also, the concept of the Local cloud in the Arrowhead framework supports the discovery of services via the Service Registry and making sure that they are authorized to be used for the interaction and information exchange. Theoretically, a vast network of CPSoS is possible using the Arrowhead framework Local cloud concept.

However, the pothole detection and warning system could not utilize the Arrowhead framework to meet the timing requirements. To some extent, the Arrowhead framework does provide provisions to reduce the latency on communication channels by encoding the payload in JSON or XML [29]. Section 5.3.1 and Section 5.3.2 enlisted issues that were found during the development of the application, the point of view was especially on findings that would have an impact if the Arrowhead framework is used in timed CPSoS. However, many minor but still essential things like the quality of code where warnings are suppressed and heavy memory usage are not explicitly mentioned.

Currently, the Arrowhead framework development is in process, and since it already has proven to have CPSoS integration, and lot of the issues mentioned in Section 5.3.2 are fixable. While this thesis was written, version 4.2.0 of the framework was released with additional automation systems such as Event handler, Gatekeeper, and QoS monitor along with few fixes concerning the Orchestration system. However, due to time constraints in this masters thesis, it was not possible to update from version 4.1.3 to 4.2.0 and verify the concerns mentioned in Section 5.3.2.

Chapter 6

Conclusion

This chapter recaps the most relevant points of this dissertation and describes the results obtained from this work, explains the future improvements, and the challenges faced during the implementation of this thesis work.

6.1 Results from the Dissertation

In this master's thesis, an IoT framework known as the Arrowhead framework was evaluated for its claim on timing support, by developing a demonstrator. The demonstrator, a pothole detection and warning system consists of two communicating CPS (Autonomous rover carts) and the interaction between them is implemented using the Arrowhead framework to create data exchange in CPSoS. The goal was to evaluate the end-to-end latency for data exchange from one CPS to another in a local network. The demo setup consists of the Autonomous rover carts equipped with sensors and DWM1001C modules for pothole detection and location data. In addition, a Location server was used to store the location data of the carts and pothole location. Both the Autonomous rover carts and the Location server are Arrowhead framework compliant, capable of interacting with the Arrowhead framework (version 4.1.3) deployed on a Linux virtual machine.

The demonstrator achieved the functionality of detecting a pothole and warning the carts regarding the pothole. The end-to-end latency of the implemented demonstrator was measured to validate the timing support of the Arrowhead framework. We could not utilize the Arrowhead framework to meet the timing requirements (Section 4.1.5) of the pothole detection and warning system. However, the Arrowhead framework was useful in the development of CPSoS where each CPS was capable of sending their location data and warning messages from one cart to another cart via the Location server.

6.2 Future Work

This thesis has only scratched the surface of what is possible to do and achieve using the Arrowhead framework in the development of CPSoS. Even though the resulting implementation is successful as a pothole detection and warning system and evaluating the timing support of the Arrowhead framework, there is still much more that can be done to show the full potential of the Arrowhead framework. For instance, the Gatekeeper system is not utilized in this implementation. Developing an application that requires multiple Local clouds interactions in

the smart cities and utilizing all the potential of the Gatekeeper system along with other core systems of the Arrowhead framework is outstanding, but considering the time and resources for the scope of this thesis, this was not a realistic goal and can be carried out as future work to explore the framework.

In the current implementation, the HTTP protocol is used for interaction between the application systems which might be contributing to the increased latency. Future work could be extended to use CoAP protocol in resource-constrained embedded systems such as Raspberry Pi as the Arrowhead framework claims to support different SOA protocols such as HTTP, CoAP, MQTT, and XMPP [3].

6.3 Challenges

Firstly, the Autonomous rover cart used for the experiment had limited documentation, due to which, getting started with the cart was quite challenging. The Arrowhead framework, as mentioned in the previous sections is currently under development, resulting in limited support and documentation. The installation of the Arrowhead framework on the Raspberry Pi board of the Rover carts failed due to incompatible RAM requirement between the RPi and the Arrowhead framework version 4.1.2. This information was not found on the open-source support platforms which resulted in an unforeseen delay in the initial implementation phase. Also since the framework is not yet in a matured development phase, the Arrowhead framework Github support community [43] has only a few active developers. The documentation of the framework is either old or non-existing. Due to this, more time was spent to try and understand the process flow of the framework, and also some issues faced during the installation were resolved with a significant delay.

As previously mentioned, the Arrowhead framework is still under development and the new versions roll out quickly requiring re-installation of the entire Arrowhead framework during the thesis. These versions (version 4.1.2 and 4.1.3) were significantly different from one another and this caused an unforeseen delay in the implementation.

The COVID-19 pandemic has caused some delays in procuring the hardware for the implementation of the thesis as well. For a few weeks, parcel service was not active and procuring various components, which would otherwise have been easier, became a hurdle during the implementation and resulted in the implementation to stall for a few days before continuing.

The Autonomous rover cart had not been thoroughly tested before by the developers and in the implementation phase of this thesis, there were issues with the collision sensors which were not working and the line sensors with low sensitivity which became a problem when developing an application which required the rover to follow a line. When the second rover was received, the solder on the sensors was worn out. To solder them back and test them, there was a need for a lab that was closed due to the pandemic. This resulted in a lot of to and from movement to get the components soldered which resulted in significant delay as well. The DWM1001C modules to provide location data which were ordered needed to be soldered as well and few of them got short-circuited due to the wrong placement of batteries and the heavy load current in the circuit. This resulted in a limited number of modules to test.

The above-mentioned points were the significant challenges faced during the implementation of this thesis work.

Bibliography

[1] C. Paniagua and J. Delsing. Industrial frameworks for internet of things: A survey. *IEEE Systems Journal*, pages 1–11, 2020.

[2] Durisic D Staron M. *AUTOSAR Standard. In: Automotive Software Architectures.* Springer Publishing Company, Incorporated, 1st edition, 2017.

[3] Ned Smith Sunil Cheruvu, Anil Kumar and David M.Wheeler. "Demystifying Internet of Things Security". pages "7–10". "CRC Press", 2017.

[4] Pal Varga, Fredrik Blomstedt, Luis Lino Ferreira, Jens Eliasson, Mats Johansson, Jerker Delsing, and Iker Martnez de Soria. "Making system of systems interoperable the core components of the arrowhead framework". *J. Netw. Comput. Appl.*, 81(C):85–95, March 2017.

[5] Etal. Matthew N O Sadiku. "Cyber-physical systems: A literature review". In *European Scientific Journal*, 2017.

[6] Matthew Sadiku, Yonghui Wang, Suxia Cui, and Sarhan Musa. Cyber-physical systems: A literature review. *European Scientific Journal*, 13, 12 2017.

[7] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga. "The arrowhead approach for soa application development and documentation". In *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pages 2631–2637, Oct 2014.

[8] Jerker Delsing. *IoT Automation: Arrowhead Framework*, chapter 1, pages 7–10. CRC Press, 2017.

[9] ASML. 2019 Integrated Report based on IFRS. Available at `https://www.asml.com/en/investors/annual-report/2019`, Accessed on 9-10-2020.

[10] Y. Liu, Y. Peng, B. Wang, S. Yao, and Z. Liu. Review on cyber-physical systems. *IEEE/CAA Journal of Automatica Sinica*, 4(1):27–40, 2017.

[11] Kitchenham BA and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. 2, 01 2007.

[12] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. *ACM International Conference Proceeding Series*, 05 2014.

[13] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. "A survey of commercial frameworks for the internet of things". In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sep. 2015.

[14] IoTivity. Linux Foundation,2019. `accessed`, 2020.

[15] OMA SpecWorks. Lightweight M2M,2019. `accessed`, 2020.

[16] FIWARE. The Open Source Platform for our smart digital future,2019. `accessed`, 2020.

[17] AUTOSAR. Autosar Offical Website. `https://www.autosar.org/`, Accessed on 8-19-2020.

[18] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[19] M. Vulić, V. T. Ponović, A. Davidović, and I. Kaštelan. Overview of end-to-end event chain in advanced driver-assistance software following autosar. In *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, pages 308–312, 2019.

[20] Henrikki Hoikka. Evaluation of arrowhead framework in condition monitoring application. 12 2019.

[21] T.Latvala. Deployment of service oriented automation platform for integrating smart city applications. 02 2016.

[22] Andreas Månsson. Can-bus system for vehicle actuation and data logging with arrowhead framework, 2019.

[23] R. Passerone, D. Cancila, M. Albano, S. Mouelhi, S. Plosz, E. Jantunen, A. Ryabokon, E. Laarouchi, C. Hegedűs, and P. Varga. A methodology for the design of safety-compliant and secure communication of autonomous vehicles. *IEEE Access*, 7:125022–125037, 2019.

[24] A. W. Colombo, F. Jammes, H. Smit, R. Harrison, J. L. M. Lastra, and I. M. Delamer. "Service-oriented architectures for collaborative automation". In *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, pages 6 pp.–, Nov 2005.

[25] Y. Baghdadi. "A framework to select an approach for web services and soa development". In *2012 International Conference on Innovations in Information Technology (IIT)*, pages 277–282, March 2012.

[26] J.Bean. *SOA and Web Services Interface Design.* The MK/OMG Press, Boston, 2010.

[27] IBM. "Patterns: Service Oriented Architecture and Web Services,2004". `accessed`, 2019.

[28] L. White, N. Wilde, T. Reichherzer, E. El-Sheikh, G. Goehring, A. Baskin, B. Hartmann, and M. Manea. Understanding interoperable systems: Challenges for the maintenance of soa applications. In *2012 45th Hawaii International Conference on System Sciences*, pages 2199–2206, 2012.

[29] Jerker Delsing. *IoT Automation: Arrowhead Framework*, chapter 3, pages 43–86. CRC Press, 2017.

[30] C. Hegedűs, D. Kozma, G. Soós, and P. Varga. Enhancements of the arrowhead framework to refine inter-cloud service interactions. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 5259–5264, 2016.

[31] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C.Adams. "X. 509 internet public key infrastructure online certificate status protocol-ocsp", RFC 2560, Tech. 1999.

[32] A. DeKok and A. Lior. "Remote authentication dial in user service (radius) protocol extensions", RFC 6929, Tech. 2013.

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

[34] A. Shrivastava, P. Derler, Y. L. Baboudr, K. Stanton, M. Khayatian, H. A. Andrade, M. Weiss, J. Eidson, and S. Chandhoke. Time in cyber-physical systems. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2016.

[35] Adnan Shaout, Dominic Colella, and S.s Awad. Advanced driver assistance systems - past, present and future. *Proc. 7th Int. Computer Engineering Conf*, 12 2011.

[36] V. Vibin, P. Sivraj, and V. Vanitha. Implementation of in-vehicle and v2v communication with basic safety message format. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 637–642, 2018.

[37] Josh Joy, Vince Rabsatt, and Mario Gerla. Internet of vehicles: Enabling safe, secure, and private vehicular crowdsourcing. *Internet Technology Letters*, 1:e16, 11 2017.

[38] Seung-ki Ryu, Taehyeong Kim, and Young-Ro Kim. Image-based pothole detection system for its service and road management system. *Mathematical Problems in Engineering*, 2015:1–10, 09 2015.

[39] Ieee guide for wireless access in vehicular environments (wave) - architecture. *IEEE Std 1609.0-2013*, pages 1–78, 2014.

[40] Maninder Kaur and Piyush Maheshwari. Building smart cities applications using iot and cloud-based architectures. pages 1–5, 03 2016.

[41] Imad Jawhar, Nader Mohamed, and Jameela Al-Jaroodi. Networking architectures and protocols for smart city systems. *Journal of Internet Services and Applications*, 9, 12 2018.

[42] J.Arkko C.Jennings, Z.Shelby and A.Keranen. Media Types for Sensor Measurement Lists, 2018. `accessed`, 8-11-2020.

[43] Arrowhead Consortia. Arrowhead Framework 4.2.0. GitHub, Inc. . Available at `https://github.com/arrowhead-f/core-java-spring`, Accessed on 8-16-2020.

[44] David Mills. Network time synchronization research project. `https://www.eecis.udel.edu/~mills/ntp.html,Accessed`, 8-14-2020.

[45] Oracle Corporation. Project Grizzly . Available at https://javaee.github.io/grizzly/, Accessed on 8-16-2020.

[46] Eclipse Foundation Oracle Corporation. Project Jersey . Available at https://eclipse-ee4j.github.io/jersey/, Accessed on 8-16-2020.

[47] Oracle Corportation. Interface Servlet. Available at https://docs.oracle.com/middleware/12213/wls/WLMBR/core/index.html, Accessed on 8-16-2020.

[48] R. Rocha, C. Maia, L. L. Ferreira, and P. Varga. Improving and modeling the performance of a publish-subscribe message broker. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 5493–5498, 2019.

[49] O'Brien, Liam, Liam, Bass, Len, Merson, Paulo, and Paulo. Quality attributes and service-oriented architectures. 01 2005.

[50] M. Malajner, P. Planinšič, and D. Gleich. Uwb ranging accuracy. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 61–64, 2015.

[51] Zafer Sahinoglu, Sinan Gezici, and Ismail Gvenc. *Ultra-Wideband Positioning Systems: Theoretical Limits, Ranging Algorithms, and Protocols*. Cambridge University Press, USA, 2011.

[52] Decawave. Decawave Tech forum. https://decaforum.decawave.com/, Accessed on 8-19-2020.

[53] Wikipedia. Representation State Transfer. https://en.wikipedia.org/wiki/Representational_state_transfer, Accessed on 8-24-2020.

[54] Wikipedia. Hypertext Transfer Protocol. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, Accessed on 8-24-2020.

Appendix A

A.1 Setting up the Arrowhead Framework v4.1.3

A.1.1 Install Linux

sudo apt update
sudo apt dist-upgrade

A.1.2 Install MySQL

sudo apt install mariadb-server

A.1.3 Install Java

sudo apt install openjdk-11-jdk-headless

A.1.4 Install Apache Maven

sudo apt install maven

A.1.5 Download/Install Arrowhead framework

Build Arrowhead Debian package

Clone the repository: git clone `https://github.com/arrowhead-f/core-java-spring.git`
Build them with : mvn package

Install Arrowhead Core Debial package

Maven creates a package by name target folder in the arrowhead folder. Go to the path <arrowhead>/target/ and run:
sudo dpkg -i arrowhead-*.deb

Swagger UI

Each core systems of the Arrowhead framework offers Swagger UI to discover its REST interfaces. This UI is available at the /api/root path. For example, the REST interface of the Service Registry is available at `https://localhost:8443/api/` by default.

A.2 Hints after installing

- Log files (log4j) are available in : /var/log/arrowhead/*
- Output from system are available at : journalctl -u arrowhead-*.services
- Restart services: sudo systemctl restart arrowhead-.service
- Configuration and certificates are found under: /etc/arrowhead
- Mysql database : sudo mysql -u root, to see the Arrowhead tables, use commands: use arrowhead;
  show tables;
  The output of the show tables is as shown in the Figure A.1

```
asml@asml-virtual-machine:~$ sudo mysql
[sudo] password for asml:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 118
Server version: 8.0.18-0ubuntu0.19.10.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use arrowhead;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------------------------------------------------+
| Tables_in_arrowhead                                       |
+-----------------------------------------------------------+
| authorization_inter_cloud                                 |
| authorization_inter_cloud_interface_connection            |
| authorization_intra_cloud                                 |
| authorization_intra_cloud_interface_connection            |
| choreographer_action                                      |
| choreographer_action_action_step_connection               |
| choreographer_action_plan                                 |
| choreographer_action_plan_action_connection               |
| choreographer_action_step                                 |
| choreographer_action_step_service_definition_connection   |
| choreographer_next_action_step                            |
| cloud                                                     |
| cloud_gatekeeper_relay                                    |
| cloud_gateway_relay                                       |
| event_type                                                |
| foreign_system                                            |
| location_DB                                               |
| logs                                                      |
| orchestrator_store                                        |
| relay                                                     |
| service_definition                                        |
| service_interface                                         |
| service_registry                                          |
| service_registry_interface_connection                     |
| subscription                                              |
| subscription_publisher_connection                         |
| system_                                                   |
+-----------------------------------------------------------+
27 rows in set (0.00 sec)
```

Figure A.1: Contents of Arrowhead framework MySQL table

- To remove everything related to arrowhead : sudo apt purge arrowhead-

Appendix B

B.1 DWM1001C

Usually, the Global Position System (GPS) is used to track vehicles and other devices with a relative high accuracy, but not upto centimeter or millimeter and therefore is not suitable for indoor position. Hence, Ultra-wide band (UWB) finds its advantage in indoor environment. Also, UWB has a higher accuracy and can track devices with a higher accuracy than GPS [50]. The Decawave module, DWM1001C uses UWB to find the location of the device.

DWM1001 module can be configured as "tag" or "anchor" to find the location of the device. The term tag is being referred to a device that is being located or positioned, in another words "the target" whereas anchors are the devices that has fixed position and are helpful in locating the tag. In order to calculate the distance between tag and anchor, the formula $d = v * t$ is used where v is the velocity, d is the distance and time is unknown but based on Time of Arrival (ToA), distance is calculated [51]. There are three ways to calculate the ToA, and they are

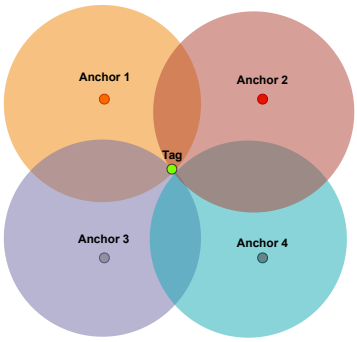


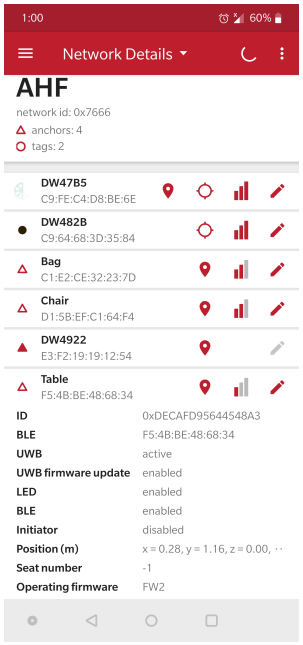
Figure B.1: Illustration of Trilateration using the distance between anchors and tags

- One way ranging which depends on the synchronization between two nodes.
- Two way ranging measures how long it takes for a signal to get from the sender and back, also called round trip time (RTT).
- Trilateration, this method is used in DWM1001C module to calculate the distance. This method involves using the estimated range between object positioned at known location (anchors) and a target object at an unknown position (tags) to estimate the location

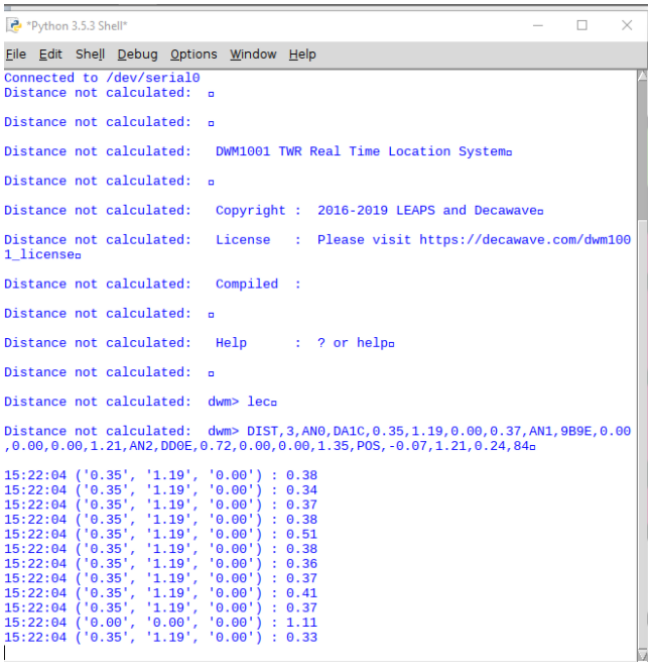
of the target object i.e. tags relative to that of the known location i.e. anchors in the context of the DWM1001C module terminology [51].

Therefore, when visualized in 2-dimensional coordinate frame, the process of finding the exact location of the device consists of 4 or more anchors at known position with circles around them, intersecting with the location of the target object as shown in Figure B.1.This same methodology can be used in a 3-dimensional coordinate frame utilizing spheres instead of circles.

In this thesis, the anchors are placed in the four corners of the room and the tags are attached with the Raspberry Pi and the communication between tag and Raspberry Pi is achieved using UART. The distance calculation of the Autonomous Rover Cart or tag is updated at every 10 seconds and is stored in the MYSQL database. The Decawave also provides an android application (DRLTS) through which we can configure the DWM1001 modules as tags or anchors and the position of the tag is visible based on anchors. The implementation details (Listings 4), configuration details (Figure B.2a) and the location data information (Figure B.2b) is provided below.



(a) Configuration of DWM1001 modules as tag or anchor



(b) Location data output of the tag

```
#DWM Serial Parser
import serial
import time
import datetime
import mysql.connector

mydb=mysql.connector.connect(host="localhost", user="pi", passwd="raspberry",
        database="distanceDB")
sql = "update location set Xvalue= %s, Yvalue= %s WHERE ID=1"

DWM=serial.Serial(port="/dev/serial0", baudrate=115200)
print("Connected to " +DWM.name)
DWM.write("\r\r".encode())
time.sleep(1)
DWM.write("lec\r".encode())
time.sleep(1)
while True:
    try:
        line=DWM.readline()
        if(line):
            if len(line)>=140:
                parse=line.decode().split(",")
                x_pos=parse[parse.index("POS")+1]
                y_pos=parse[parse.index("POS")+2]
                val = (x_pos,y_pos)
                mycursor=mydb.cursor()
                mycursor.execute(sql, val)
                mydb.commit()
                print(datetime.datetime.now().strftime("%H:%M:%S"),"(",x_pos,",
                ",y_pos,")")
            else:
                print("Position not calculated: ",line.decode())
    except Exception as ex:
        print(ex)
        break
DWM.write("\r".encode())
DWM.close()
```

Listing 4: Python code to extract the location data from DWM1001 module interfaced using UART to Raspberry Pi [52]

Appendix C

C.1 Design Technologies and Motivation

For the design, development and implementation of this project, different technologies were used.

C.1.1 REST

REpresentational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web [53]. REST depends on stateless client- server cacheable communication protocol. The advantages of REST are:

- REST is platform independent i.e. it does not matter if the server is Windows, the client is a Mac or anything else.
- REST is language independent and standards-based i.e. it runs on top of HTTP.
- As it runs on the standard HTTP, it can be used in the presence of firewalls.
- REST architecture supports flexibility and scalability.

Therefore, REST architecture style is suitable for the achieving distributed systems such as the CPSoS, which is one of the targets of this project.

C.1.2 HTTP

Hypertext Transport Protocol (HTTP) is a request-response protocol used in client-server computing model. HTTP establishes the connection between client (a web browser) and server (a application hosting a website). The advantages of HTTP are:

- HTTP is an application layer protocol which uses reliable transport layer protocol i.e. Transport Control Protocol (TCP).
- Identification and location of HTTP resources on the network is based on unique Uniform Resource Identifier (URI) or Uniform Resource Locator (URL).
- It offers reduced network congestion.
- Handshaking is done at the initial connection establishment stage. Hence, it offers reduced latency in subsequent requests as there is no handshaking.

Different methods such as GET, POST, PUT, DELETE options are used for requests and responses to indicate the desired action to be performed on the resource [54]. HTTP also uses various status code for indicating successful connection, server error or client error. Hence, the Arrowhead framework and also in this project HTTP protocol is used for communication between systems in CPSoS.

C.1.3 JSON

JavaScript Object Notation (JSON) is a readable data interchange format which is text-based open standard developed for easy readability. The advantages of JSON are:

- JSON is easy to generate and ease to parse.
- JSON is directly and readily supported by JavaScript, used in the development of the Arrowhead framework.
- JSON is language independent, with available parsers for most languages.

Though the Arrowhead framework can support both XML and JSON, JSON format is used throughout the project it is suitable for JavaScript application and also estimated to parse faster than XML [8].

C.1.4 MySQL

MySQL is a powerful open source database assuring high performance and reliability. The advantages of MySQL are:

- MySQL is distributed with Linux based OS and is free of cost.
- MySQL is widely used and lot of support materials and tools are available.
- It is easily possible to integrate with the developed software.

The choice of using MySQL is motivated by the reason that the Arrowhead framework uses MySQL for storing the information regarding systems and services [43]. Therefore for storing the location data and warning messages, MySQL is used.

This page is intentionally left blank

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

DEPARTMENT OF ELECTRICAL ENGINEERING