

Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis

Testing in Mobile Cyber-Physical Systems

Sagar Chande

Course of Study: INFOTECH

Examiner: Frank Leymann (Prof. Dr. Dr. h. c.)

Supervisor: Kálmán Képes (M.Sc.)

Commenced: April 24, 2018

Completed: October 24, 2018

Acknowledgement

I would like to thank Prof. Dr. Dr. h. c. Frank Leymann for offering me such a wonderful opportunity to pursue my master thesis in the Institut für Architektur von Anwendungssystemen.

I express my sincere thanks to my supervisor, M.Sc. Kálmán Képes for all the guidance and consistent support. I am grateful to him for his continuous encouragement and guidance throughout the thesis. His friendly nature and thoughtful insights created suitable work environment as well as truly encouraged and motivated me during thesis work.

Finally, I express my deep sense of gratitude to my parents Mrs. Uma Chande and Mr. Sanjay Chande for their everlasting support and encouragement without which this accomplishment would not have been possible.

Sincerely yours

Sagar Sanjay Chande

Abstract

This master thesis presents a method of testing the mobile Cyber Physical System (CPS) during normal operation. CPSs are considered as complex heterogeneous systems combining information technology and physical elements such as sensors, actuators etc. Being deployed in physical environments, safety and security are major challenges for CPSs. Improper usage of Mobile CPSs can cause either self-damage or to people and property around. This implies a pre-requisite for the mobile CPS to be more reliable, predictable and secure than their non-mobile counterparts. Frequent testing or continuous monitoring of mobile CPS can reduce the risk of improper functioning or failure of the complete system. This can be ensured by introducing a run-time testing mechanism to the system. Run-time testing needs to be scheduled such that no interference with normal system operation occurs. The test system proposed here, provides the facility to locally deploy the given test case on mobile CPS with redundant hardware, alternatively on a remote test server taking all of the network related problems into consideration. Care has been taken such that the proposed system should not interfere with normal system operation. We proposed this run-time testing system on the basis of constraint-based approach and Built In Self Test (BIST). We define exemplary ‘Test Points’ in mobile CPS i.e. an ideal test case deployment strategy and a non-influential execution of the same.

Contents

1	Introduction	15
1.1	Motivation	15
1.2	Research Objective	16
2	Basics	17
2.1	Introduction to CPS	17
2.2	Testing of CPS	18
2.3	Run Time Testing	20
2.4	Mobile Computing	21
2.5	Fault Diagnosis	22
2.6	Middleware	23
2.7	Application Integration in Distributed Systems	26
2.8	Message Oriented Middleware (MOM) – Details	28
2.9	Messaging Queues	30
3	Related Work	33
4	Concept	43
4.1	Research Questions and Assumptions	43
4.2	Architecture Concept Overview	45
4.3	Architecture Modules	49
4.4	Local vs. Remote	50
4.5	Local Test System Configuration	51
4.6	Remote Test System Configuration	51
4.7	Messaging Channels	52
4.8	Message Interaction	53
4.9	System Behaviour - BPMN	53
5	Validation	57
5.1	Implementation Overview	57
5.2	Hardware and Software	58
5.3	Local Testing Approach	61
5.4	Remote Testing Approach	62
5.5	Prototype	63
5.6	System Flow Chart	66
5.7	Use Case	70
5.8	System Analysis	71
6	Conclusion and Outlook	73

List of Figures

1.1	Thesis Strategy	16
2.1	Design of Cyber-Physical Systems [MBED12]	18
2.2	Life Cycle Model for CPS [AIH15]	19
2.3	Example of Proxy [SDEF98]	22
2.4	Fault Diagnosis Spaces [TBH+01]	23
2.5	Offline BIST	24
2.6	Middleware Classification [BK03]	25
2.7	File Transfer Mechanism [HW04]	26
2.8	Shared Database Mechanism [HW04]	27
2.9	Remote Procedure Invocation Mechanism [HW04]	27
2.10	Messaging Mechanism [HW04]	28
2.11	Basic MOM Concept [CUR04]	29
2.12	Message Structure [HW04]	30
2.13	Application Communication via Message Queue [HW04]	31
2.14	ZMQ Connections [ZMQ]	32
2.15	ZMQ String Data Type Representation [ZMQ]	32
3.1	Three Layer Test Architecture [BUZL11]	33
3.2	T-UPPAAL Engine [MLN]	34
3.3	Timed Specification State Machine [MLN]	34
3.4	UPBOT Test Bed Architecture [CB10]	35
3.5	Monitor-Actuator Pair [KW16]	36
3.6	Online Testing Classification [AMH98]	37
3.7	Testlet for Headlight Testing [BK06]	38
3.8	TPT Test Process [BK06]	38
3.9	Design Spaces [TBH+01]	39
3.10	Lock Step Duel Architecture [BFM+03]	40
3.11	General View of Remote and Local Test Approach [MAU05]	41
4.1	Application with Different Processing Power Hardwares	45
4.2	System Proposal with Concept at Glance	46
4.3	Constraints Modelling	48
4.4	System Architecture Modules	49
4.5	Local Test System Configuration	51
4.6	Remote Test System Configuration	52
4.7	Messaging Channel	52
4.8	Socket Structure	53
4.9	Local Test behaviour	54
4.10	Remote Test behaviour	55

5.1	Implementation Overview	57
5.2	mBot from Makeblock [MAK]	58
5.3	mBot Sensors [MAK]	59
5.4	Raspberry Pi 3 B+ Module [RPI]	59
5.5	Arduino Uno Microcontroller [ARD]	60
5.6	Distributed System Implementation Outline	61
5.7	Local Test System Approach	61
5.8	Remote Test System Approach	62
5.9	System Prototype	63
5.10	System UI	64
5.11	Complete System Behaviour	65
5.12	Monitoring Algorithm	67
5.13	Local Approach Algorithm	68
5.14	Remote Approach Algorithm	69
5.15	Use Case with mBot	70
5.16	Testing and Monitoring	71
6.1	Server Redundancy	74

List of Tables

4.1	Initial Test Strategy	45
5.1	Monitoring Constraints	70
5.2	Interrupting Constraints	71

List of Abbreviations

API	Application Programming Interface.	30
ASIL	Automotive Safety Integrity Level.	36
BIST	Built In Self Test.	5
BPMN	Business Process Modelling Notation.	43
CAN	Controller Area Network.	44
CPS	Cyber Physical System.	5
CRUD	Create,Read,Update,Delete.	27
CTM	Client Test Manager.	44
DB	Database.	25
FIFO	First In First Out.	30
HRM	Hardware Resource Manager.	33
HSM	Hardware Scheduling Manager.	33
HTM	Hardware Test Manager.	33
IDE	Integrated Development Environment.	16
IOT	Internet of Things.	18
ISR	Interrupt Service Routine.	53
LPU	Local Processing Unit.	44
MOM	Message Oriented Middleware.	25
OBD	On Board Diagnosis.	44
ORB	Object Resource Broker.	25
OTA	Over The Air.	15
QoS	Quality of Services.	23
RPC	Remote Procedure Call.	27
RPI	Remote Procedure Invocation.	27
RPU	Remote Processing Unit.	46
TM	Test Manager.	44

List of Abbreviations

TPT Time Partition Testing. 37

ZMQ Zero Message Queue. 30

1 Introduction

Cyber-physical systems are a new revolution in modernized computing and are highly distributed in nature. As the name suggests, it is combination of physical and computational systems which work in harmony and regulate complete operation of the system. CPSs are used in wide range of applications such as embedded computing. In addition, it is widely used in automotive domain for e.g. vehicles, drones etc. Mobile CPSs are highly distributed and safety-critical systems. Malfunctioning in mobile CPS can cause the fatality. Therefore, the testing of CPS is extremely important.

1.1 Motivation

In current era of automation where devices are having the control of the operation or decision authority, safety place a critical role. Imagine a car receiving software updates over the air and customer updates existing software using these updates. How a user can test after the update whether the system is functioning properly or not? For a non technical user it would be difficult to test the system due to complexity involved. Hence many researchers are contributing in order to improve the safety of cyber physical system. The solution is an end to end testing system, which facilitates user to test the effect of software update or test the performance of the car during run time and assure all sensors and actuators acting within boundary conditions. Most of the automobile manufacturers have testing bays and maintenance stations for complete inspection and fault diagnosis. But, this won't be the case in run time testing. Every time after Over The Air (OTA) software update, it is in-feasible for a user to reach these maintenance stations/bays and opt for the maintenance services. Giving test authority to the user either locally (i.e., within the car) or remotely (over the network) is the feasible solution. If a fault gets diagnosed during this user testing, a user can safely roll back current software update and resume his normal operation informing the manufacturer. Testing can be performed in safe environment and it can be monitored continuously with the help of sensors. Hence a framework can be designed with the help of sensors which is safe and can be used for run time testing. run time testing solution to the user is not an easy task. there are certain challenges in the designing of run time test system such as: WHEN? i.e., what is an appropriate time when a car can be shifted to test mode? Another job of application engineer is to think about where to deploy the existing test case for run time testing? Will it be locally on the car itself or the remote testing unit via network? The decision should base on ideal deployment strategy and of course non-influential (to normal operation) execution of test cases.

1.2 Research Objective

The primary objective of this thesis is conceptualizing the run time testing system for mobile cyber-physical systems (e.g. Car). The system which is proposed in this thesis is responsible for the non-influential execution of test cases. As we are dealing with the run time testing scenarios, non-influential implementation is the mandatory criteria. It is essential to study and understand safety requirements in mobile cyber-physical systems. In addition to it, many questions to be answered concerning vehicle safety criteria. Since a car is the most complex embedded system, the requirements for testing the same are equally complicated. According to our studies performed, we are dividing testing in local testing and remote testing approach.

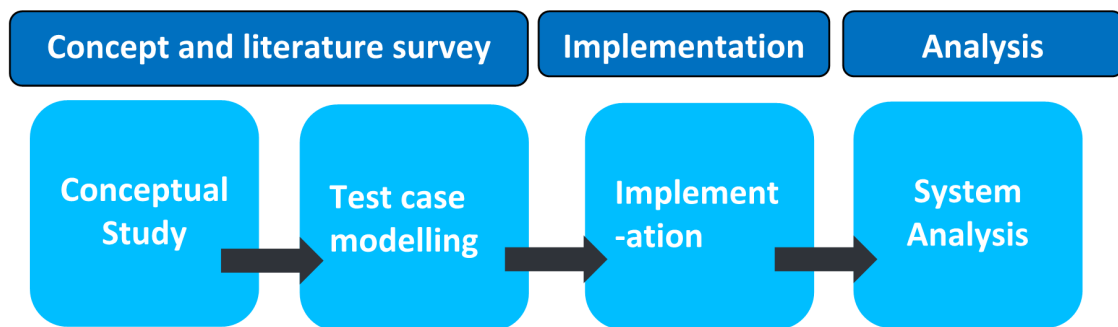


Figure 1.1: Thesis Strategy

For the prototype development of our system, we are going to use mBot educational robot kit from Makeblock. mBot will serve the purpose as a prototype for the vehicle under test. As stated above, we are implementing local and remote testing approach. Hence, for local testing strategy, we are deploying system application on general purpose processor for eg. Raspberry Pi (RPi). The whole software will be written in Python since it's a high-level scripting and object-oriented language. In order to test the system prototype, we need test cases that will cover the maximum aspect of run time testing. Since the mBot is Atmega 328p based robot, we are going to write our test cases in Arduino Integrated Development Environment (IDE) and Arduino language. We will deploy Test cases on testing nodes as simple byte files (.hex files). This master thesis mainly focuses on the realization of the testing system for mobile CPSs. Considering the time restrictions and the complexity of the work, we are mainly covering our conceptual idea with small prototype and keep our system scalable so that it can be integrated within all mobile CPSs irrespective of platform, Programming language, and hardware. We will perform an analytical study on our prototype so that our system can be used efficiently in real-world testing scenarios. Researchers and developers can use this study in the field of Embedded systems and CPSs. Cyber-physical systems exist everywhere, and safety is one of the essential features. Hence, a large group of engineers and researchers working in security and testing domain will use this study.

2 Basics

This chapter explains about prerequisites and background work for this thesis. Also, it focuses on all used technologies, hardware and software.

2.1 Introduction to CPS

Computation processes and physical processes combined to build a cyber-physical system (Figure 2.1). Feedback loops with efficient monitoring play an essential role in the overall functioning of a cyber-physical system. While dealing with cyber-physical systems, predictability and reliability are significant concerns. We cannot underestimate these terms while designing a robust system in fields such as health care, safety for automotive, etc. [LEE08] Designing of cyber physical systems involves high level of complexity. They are heterogeneous, combining information technology and physical elements such as sensors, actuators, etc. [MBED12].

CPSs are hard real-time systems and interact with the world in real time. Their design is equally efficient at the physical and network levels. In this work, we are focusing on mobile CPSs, i.e., Cars, Airplanes, and Drones, etc. These systems consist of numerous computers, sensors, and actuators. The primary intention is to orchestrate these computers and sensors altogether [WOL09].

The orchestration is very important for proper functioning of CPS. Concerning these crucial timing requirement areas viz. automotive safety, health care, etc., CPSs need precise timing compliance during normal operation or recovery operation as well. Being distributed over a wide area of network and dealing with the number of physical processes, we cannot underestimate the fact that CPSs are more prone to errors and defects. The abnormal functioning of the physical component or sudden network failure can cause whole system failure. System failure is highly crucial, and one needs to avoid it while dealing with safety-critical applications. Time line maintenance along with validation of system is equally essential for CPSs while working in health care, automotive safety, and other safety-critical areas [PSRL09] [MQM11].

Lee [LEE08] and Trainor [TRA08], in their work, explain the main difference between embedded systems and cyber-physical systems. It is evident that over the decades, embedded systems set the benchmark for two critical characteristics in the computing world, i.e., Predictability of a system and its reliability. CPSs further extended these benchmarks in safety-critical applications. Performance of CPS led to the inclusion of CPS in high security and safety-critical applications. Moreover, CPS can deal with the high unpredictability of operation into the physical world.

Jazdi [JAZ14] also discussed about embedded systems and cyber-physical systems in the context of a new industrial revolution, i.e., industry 4.0. Embedded systems serve as standalone systems whereas, CPSs are having networking capabilities. Hence, the CPS is an idea for distributed infrastructure. Jazdi explains various aspects of extension of an embedded system to make it compliance with CPS. He confirms the study with an interesting example of the innovative prototype developed for linking coffee machine with the network. With the micro-controller extension approach, one can

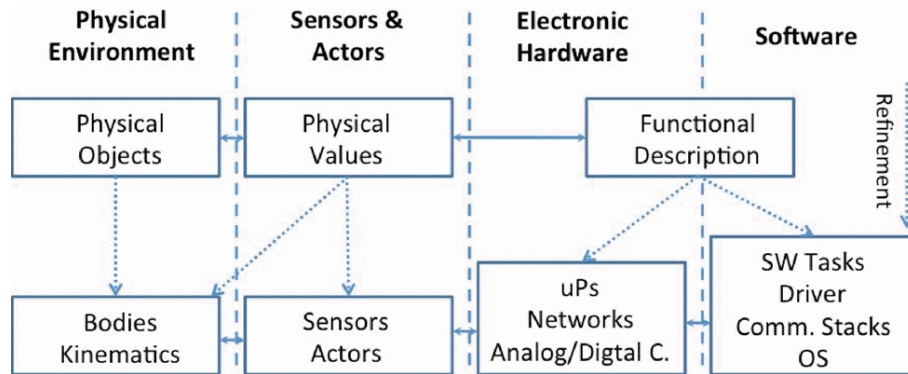


Figure 2.1: Design of Cyber-Physical Systems [MBED12]

control, maintain and customize the coffee machine functionality. Moreover, a software update can be provided over the network. Hence, a simple networking approach converts a standalone embedded system into a distributed CPS.

Pal et al. [PSRL09] focused on a significant threat to CPS, i.e., security issues. Being distributed in nature and spread over the wide area, CPS has to face security issues related to data and information interchange. In such a heterogeneous system, eliminating all threats or timely detection of them is a challenge. CPS handles a substantial amount of data. Furthermore, the data is distributed over a broad network. Hence, results in contamination with noise due to system failure or malfunction in the network. Such contaminated data may result in wrong input values to sensors and actuators and hence result in system malfunction at a physical level. Data is not the only problem in distributed CPS. Along with data, these systems share essential and highly sensitive information within inter-organization networks. Over such a large networking platform, it is easy to violate the privacy of the data. A minor fault within a network or malfunction by physical parameters might add the adverse effect on the system. Challenge is to avoid the impact of these effects on the complete system. Timeliness of the operation is highly crucial in the design of the cyber-physical system. But it is a significant challenge concerning the system and architecture design. Cyber-physical systems are highly reliable, and hence, they are mostly involved in safety-critical applications. External attackers violate timing constraints and remarkably hold the whole system functioning at stake. Of course security of cyber-physical systems is a significant issue. But various researches are going on in making CPS more reliable and safe for distributed and safety-critical applications.

2.2 Testing of CPS

Cyber-physical systems are highly responsive systems. Considering the size of these systems, performance management is crucial. The only possible way is continuous monitoring and early detection of problems. Adequate testing is inevitable for CPS to sustain in the competition. Testing of such hybrid systems is difficult. It is not only because of their size and complexity but also because of the interrelation of various input and output parameters. Also, the continuous behaviour of input and output signals make the testing scenarios difficult. Individual test case needs consideration of multiple parameters. Resulting in increased complexity of the overall test suite. Mostly testing of CPS follows constraints-based approach. Satisfactory condition based on test input can be calculated and relate the same with boundary constraints [ZHY13] [SGLW09].

Asadollah et al. [AIH15] compare the testing of CPS with V-model. Software testing verification essentially uses the traditional V-model. Testing levels of CPS compared with V-Model levels. CPS being a considerable complex physical and cyber system. Therefore to develop a sophisticated testing system, we need to consider various aspects. To validate the reliability and efficiency, testing of CPS is essential at different system levels, i.e., software, network, and hardware testing. Architectural design and functional testing are important aspects as well. CPS consists of hardware and software

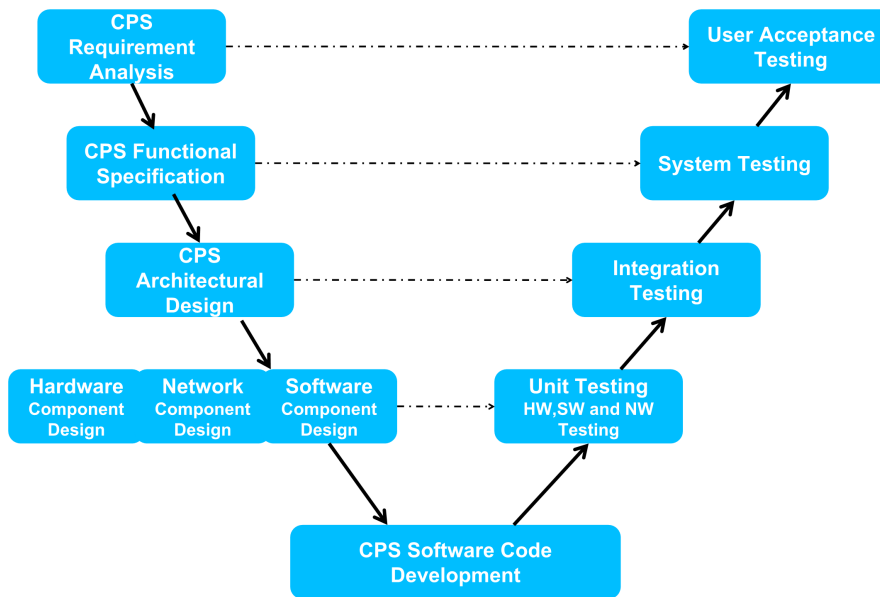


Figure 2.2: Life Cycle Model for CPS [AIH15]

co-design and distributed over the broad area of the network. Therefore, as seen in Figure 2.2, unit testing in CPS performed at hardware, network, and software levels separately. V model further adapted to CPS by integration testing which tests the various modules in the system and finally with system testing where end to end system performance can be validated.

According to Asadollah et al. [AIH15], there are total six levels defined in CPS, and they are depicted in V-model in Figure 2.2 above. At the hardware testing level, the functionality of each component can be tested. Performance of hardware is verified by assuring the quality and performance under testing conditions. The structural design of the program and its computational functionality can also be tested at the unit testing level according to V-model for CPS. Network testing at the unit testing level is performed for the validation of communication protocol. Combining and consequently synchronizing various software modules depicts integration testing. Early detection of problems in a specific module is possible with integration testing. If a test fails, corrective measures can be taken at the integration testing level itself instead of the entire system testing level. Since different modules are communicating with each other, network testing at the integration testing level is inevitable. At the system testing level, the complete application functionality and the dependency of all the system modules are verified. System testing is naturally performed at software and hardware levels. System testing evaluates end to end performance of the application. After system testing, finally user acceptance testing is inevitable for verifying the performance of the end application. Conceptualization and development of a semi- or fully automated system to validate the testing is an ongoing research area related to CPS.

Marilyn Wolf [WS18] explains the safety and security in the context of CPS and Internet of Things (IOT). In his work, the notion of a safety-critical system is provided. For safety-critical systems, testing is necessary at both run time and the development time. Both testing methods have entirely different approaches for system verification. For the verification of system properties, testing the boundary line performance at design time is essential. System failures, security threats can only be identified during run time. It is always advisable to develop a safe and secure system. But that is not the ideal case. Modelling of a system gives designers, the opportunity to check and ensure system safety at development time. On the other hand, run time testing deals with characteristics analysis. The system gets continuously monitored for its behaviour and real-time fault detection. Using diagnosis methods, a fault can be detected during run time and closely examined. One can thus work towards minimizing the errors in the system.

Cyber-physical systems are prominently used in distributed and loosely coupled network applications. These applications are highly dispersed in the network and demand real-time resources, making ubiquitous computing more complex. But there is an increasing threat to the safety and security of these applications. It is application engineer's job to ensure safety and security in CPS using various techniques and doing constant research in the field [KSKH10].

2.3 Run Time Testing

Majority of the embedded CPS's are involved in the automotive and aviation range. In these domains, safety is of top priority. Various ISO standards define stringent safety standards in these fields (e.g., ISO 26262 for automobile safety). Hence, these applications are called 'Safety Critical' applications [BUZL11] [CAR10] [KEN15]. Mobile CPS's can be damaged or harm the people around them during uncontrolled operation. This uncontrolled behaviour cannot be detected at design time. In spite of careful system design, some unwanted faults can occur during usage of the system. Those can be intermittent or transient. These are the two types of Operational faults [AMH98] [AVI76]. Bradatsch et al. [BUZL11] explicitly mentioned the requirement of run time tests, i.e., requirement to detect the above mentioned intermittent and transient faults. The critical issue in run time testing is test execution must be non-influential to the normal system operation. Timing considerations are of primary importance regarding resource sharing for run time tests. Test application and regular system application have dedicated time slots. These time slots should allow simultaneous access to the dedicated hardware resource. Shared access to hardware resources is the primary objective to be achieved using run time testing. The most complex embedded systems are vehicles. These systems comprise hundreds of processors working together with multiple ECUs. Testing of vehicular CPS is not an easy job for engineers. Now a days cars are smart. They are connected via the mobile network and represent huge distributed cyber-physical systems. Vehicular CPSs are getting smarter, leading to maintenance and testing challenge.

The most important and viable parts in a vehicle are driving, power control and brake assembly. These systems are termed as 'Safety critical' systems. Testing of these components needs to be performed in separate time slot dedicated for test purpose. It is not advisable to perform diagnosis for such safety-critical systems during normal operation. So during failure unconstrained operation of these systems can be avoided [MAU05].

Bradatsch et al. [BUZL11] mentioned testing approach for such safety-critical modules. He divided the system testing into two essential modes, i.e., concurrent and non-concurrent modes. The concurrent method works well for less safety critical areas of the vehicle system. Whereas, for

safety-critical operations such as the braking system, driving, etc. non-concurrent testing mode is suitable. The significant difference between both methods is testing during normal system operation and testing while discontinuing normal system operation temporarily. Run time testing is essential in mobile/vehicular CPS to detect run time hazards. Precautionary measures can be taken if these hazards get noticed before catastrophic events. A non-influential and easily integrated test system approach is of prime requirement. It is a job of an application engineer to facilitate system architecture to satisfy these requirements.

2.4 Mobile Computing

Now a days, machines, computing devices are getting smaller in size. Compact processing devices such as mobile phones are majorly used worldwide. E.g., everyone preferred laptops over large size desktop computers placed at one static location. In short, mobility is the prime requirement in the current scenario. Along with portability and mobility, communication is equally important. Mobile phones, as worldwide distributed systems always communicate with each other. Various computing stations share data among themselves. In short, these devices are nothing but huge distributed systems. It is evident that for such widely spread distributed systems; communication among them is a challenge. Hence, mobile computing is an extensive research area by distributed system engineers all around the world [SG14].

Fundamental constraint on mobile devices is resource-poor nature. It is evident that heavy workload handling is not possible in a pocket-size computer. Damage due to improper handling and theft is profoundly affecting portable devices. Network availability is one of the major concerns. It is important to take corrective action if the connectivity lost during data transmission. Precious data loss is unacceptable. Mobile devices do not have heavy duty power supplies. They have to rely on small inbuilt batteries, and battery backup is never long-lasting [SAT96].

Every distributed system exhibits client-server architecture. In mobile systems, various access points are used to access the network. Unlike in standard communication system, mobile computing does not have fixed client-server nodes. Therefore portable devices use these access points to get network connectivity [BBIM93].

As explained by Seitz et al. ([SDEF98]), multiple users of the mobile system desire to use the network and various resources simultaneously. All of them need shared access to numerous applications and to the shared data. It sounds easy regarding fixed network architecture. But wireless links are not trustworthy. They are not reliable regarding continuous availability. Various proxy server based approaches are available for these issues. Proxy is an agent which receives requested information and makes it available to the requesting client (Figure 2.3). But here, application dependency exists prominently. Hence, these approaches are inflexible and not able to scale due to application dependent nature of proxies. Proxies cannot be a universal solution related to all applications. Hence, the middleware approach is necessary.

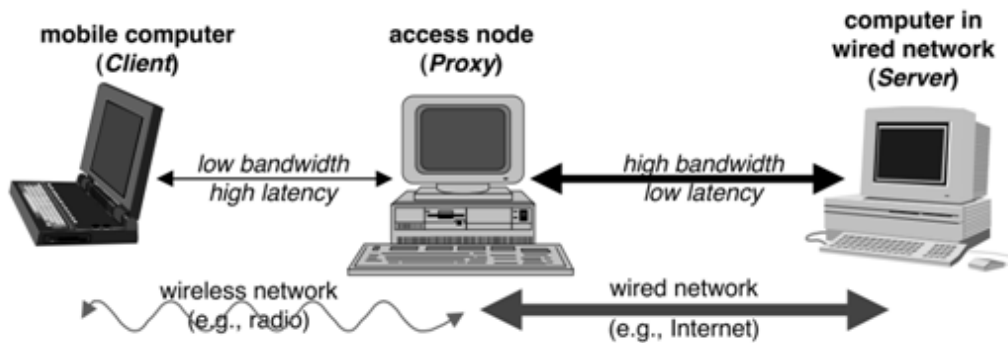


Figure 2.3: Example of Proxy [SDEF98]

2.5 Fault Diagnosis

2.5.1 Fault

A fault is an unwanted obstacle in a system or an application. Fault produces results in the form of an error. Errors must be minimized for streamlined use of a system. Hence, fault build up has to be avoided. Faults can be related to hardware or software or both. Systematic analysis of time-lines describes appropriate behaviour and nature of faults in a system. In this work, we are focusing on run time testing. We need to concern about the operation of the system and looking forward to detecting operational faults. Operational faults further categorized on the basis of their place of occurrence, frequency and impact time. As the name suggests, permanent faults always occur at the same location and timings can be predictable as well. Physical damage is one of the leading causes of permanent faults to be dominant in the system. Faults those who are transient, one cannot predict their occurrence or behaviour. Intermittent faults show mostly recurring nature. Faults do not put system into fail mode, hence to identify and avoid them in initial stage is required. On the other hand ignoring would lead to permanent defects, hence system failure [BUZL11].

2.5.2 Diagnosis Methods

Hamilton et al. produce research on fault diagnosis related to the automotive domain. He divided the fault sphere for vehicle systems in three different categories as shown in the Figure 2.4 below. Every fault must be observed by observation/monitoring technologies. Robust monitoring techniques help in early detection and hence facilitate diagnosis. One has to consider the fact that detection of entire fault space is impractical. Also, there are constraints in observation space and diagnosis space. One to one approach for a sensor to diagnosis architecture limits the operation capability of observation space and diagnosis space. As a consequence, fault detection gets affected and hence diagnosis. It would be too ambitious to build a perfect fault diagnosis system within a vehicle to track down all faults at the correct time. Moreover, plenty of time and money will be wasted in this ideal scenario planning. An alternate solution is proving more realistic. Multidimensional monitoring approach is at high priority. This approach achieved with multiple sensors connected to the vehicle. Each sensor monitors its dimension and helps in the diagnosis procedure.[TBH+01].

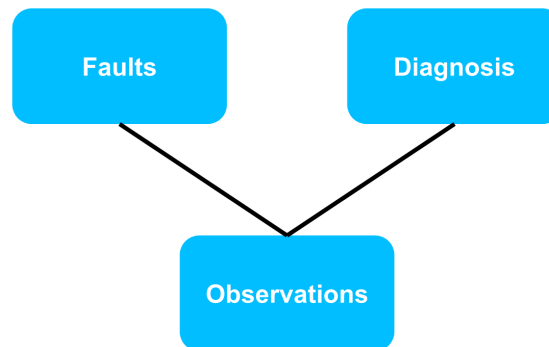


Figure 2.4: Fault Diagnosis Spaces [TBH+01]

According to [BUZL11], run time testing can be accomplished using two modes, i.e., concurrent and non-concurrent. Concurrent method primarily focuses on the run time testing, i.e., system during its operational state. Boundary conditions of the software have been observed (observations phase). Fault detection does not only limit to software but depends on timing dependency, information, and hardware duplication for comparison as well. Fault detection techniques applied during normal operation for fault diagnosis. Non-concurrent mode behaves exactly opposite. The basic idea behind non-concurrent testing is the sharing of hardware resources. Concurrent access of hardware resources for testing and application running purpose is not possible. If done so can interfere into the normal application mode and can be hazardous. Hence the option is to decide time slots for them to access resources one by one. Temporary switch the control of hardware to test/diagnosis system is what is done during the non-concurrent testing approach. The most important goal is to achieve obliging safety assuring smooth system functioning using online diagnostic methods.

Kewal et al. [SS88] focused mainly on concurrent testing approach using off-line BIST method (Figure 2.5). His primary research focus is to use system standard inputs as testing inputs. This approach proves beneficial to perform tests during normal system operation. Hence, no separate testing time slot is needed. Furthermore easy and speedy detection of faults can be achieved. Since the system is in normal operation, It is easy to detect fault location in the system. No independent analysis is required for the same. Hence Kewal et al., finally describe the advantages of using concurrent testing approach. Intermittent and transient fault diagnosis becomes effortless. System analysis with reduced maintenance is achieved. Moreover, availability of overall system increases.

2.6 Middleware

Distributed systems are widely spread and developed by the number of engineers in various organizations. Development of these systems is not a job of a single organization with one developer. Smart automotive systems, i.e., a car development involve multiple engineers in various domains. Internet and intelligent sensors connected to numerous functionalities of the car. This connection increases demand for mobile computing. Diverse applications continuously collect data of car performance and information related to driver operations over the internet. Hence need of interfacing these sensors with internet are of primary concern [APW+11]. All these applications might developed on different platforms; using different languages. How can we integrate these systems developed

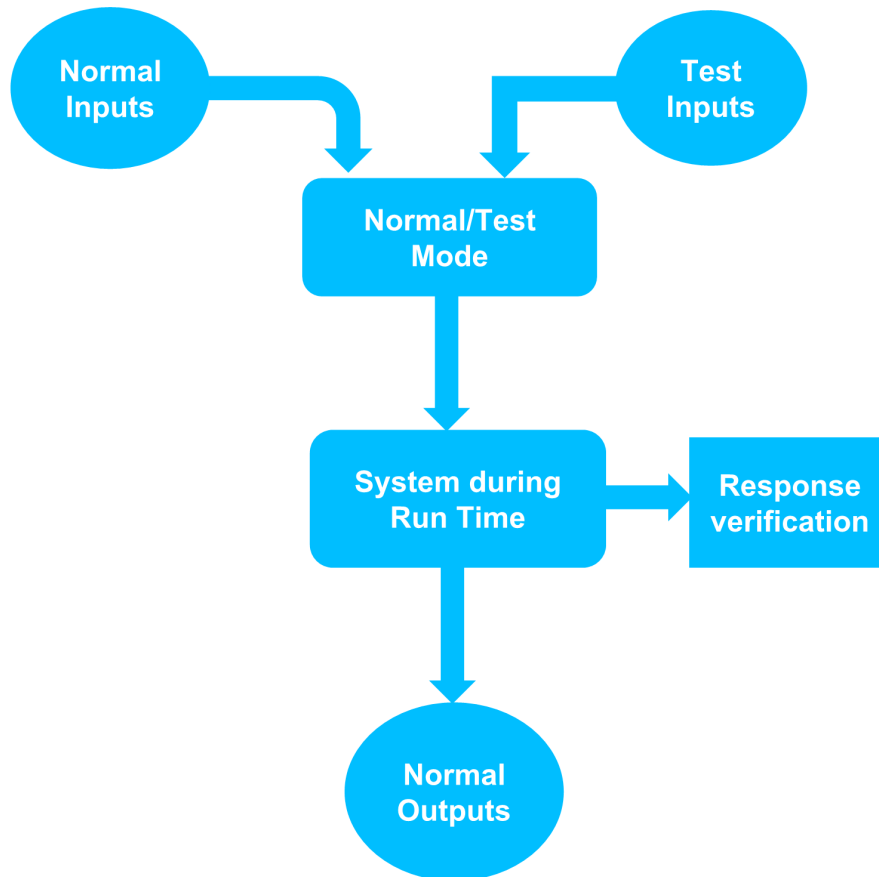


Figure 2.5: Offline BIST

by multiple people and by diverse organizations? The answer is by using a middleman who is responsible for maintaining communication between them. The solution is the middleware [BER96]. Pietzuch [PB02] explains the characteristics of middleware. Scalability is the prime importance of middleware. Any complex distributed system can have multiple clients and servers. Middleware also needs to be distributed in nature to handle distributed client-server traffic. Undistributed middleware induces heavy traffic resulting in the bottleneck. Another essential feature of middleware is inter-operability. Any middleware must be platform independent. It is also language independent and hence maintains harmony between the numbers of components in multiple systems. Various fault tolerance mechanisms make middleware reliable. Reliability is essential characteristics when it comes to Quality of Services (QoS). Expressiveness related to events and data facilitates abstraction for all subscribers in a system. This abstraction must be incorporated smoothly in the programming language used by the application. Hence, middleware usability can be justified.

Bishop et. Al. [BK03] stated that middleware is additional program/software which links multiple software applications. Software is a generic term. There are many sub-cases, and Bishop et al. explained each of them with the systematic classification of middleware. One cannot generalize middleware as a single service providing software. Based on this, grouping gets further maturity. According to Figure 2.6, middleware has two major categories to be divided in. Integration type middleware have a specific protocol for implementation. They are implemented in systems with a

predefined semantics. It is evident that they use various communication protocols, which facilitates their further classification. On the other hand, application middleware types are function oriented middleware. They are categorized by application dependency. Application interaction and application dependency are at the highest priority for application type middleware. They are further classified depends on access type, well-timed behaviour and the field of operation.

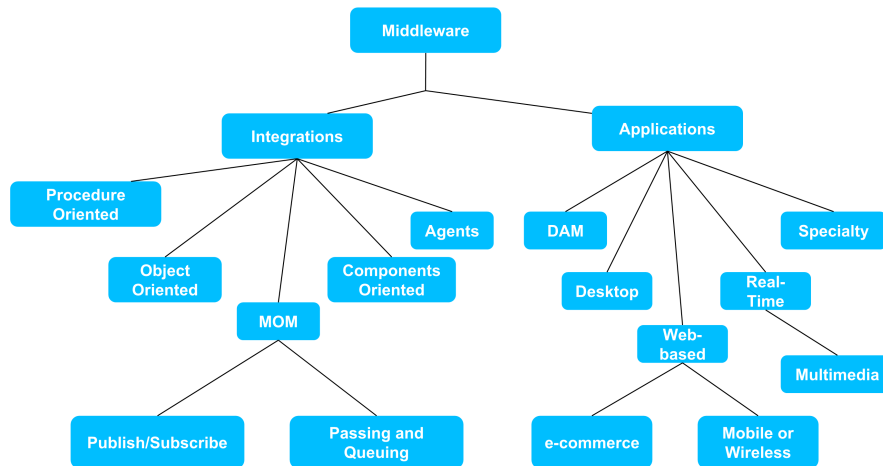


Figure 2.6: Middleware Classification [BK03]

These two significant categories of middleware are further divided into various types. Their operating manner, as well as application assistance, decides this further classification. Procedure-oriented middleware is synchronous, whereas object-oriented middleware based on Object Resource Broker (ORB) approach. Message Oriented Middleware (MOM) provides client-server and pub-sub approaches. Component-oriented middleware is mainly having an advantage in that its reconfiguration is possible according to requirements. Agent middleware is the combination of several elements. They are flexible. On the other side, application middleware category consists of Data Access Middleware (DAM), i.e., Database (DB) management middleware. Desktop middleware control requested user data according to the presentation needed. Web-based middleware as in the name itself is used in internet browsing. Real-time middleware has higher time lines compare to others. Specialty middleware is a particular case of middleware for multi-campus connections, medical field, etc. We need integration type of middleware due to establishing vehicle to remote unit communication. Our plan is to make vehicle receive messages via remote unit which is dedicated for testing our test cases. Hence, the choice is obviously integration middleware type Message Oriented Middleware (MOM). It offers client-server approach. In the client-server approach, messages are getting queued in a message queue, and via the channel, they will reach to the server. MOM will perform this message queuing task as a message broker. MOM works on predetermined protocols. We can use MOM queues as FIFO queues. Also, priority based message separation can be performed. If needed, load balancing algorithms can be used. It is one to one, i.e., peer to peer application. MOM can be used in the pub-sub mode as well so that one publisher can send a message to multiple subscribers. Pub-Sub approach is an event-driven approach for MOM [BK03].

2.7 Application Integration in Distributed Systems

Within distributed systems, multiple applications function together to achieve predetermined system goal. To synchronize these applications, their integration is of prime importance. One of the prime requirements is, the integration should not affect the complete system. This implies that applications should be loosely coupled and highly tolerant to changes. But there are multiple challenges in loose coupling of enterprise applications. Multiple development platforms, programming languages, development by third party organization etc. are few of them. Several integration criteria need to be followed by integration engineers. Need of application integration should be precise and clear. As stated earlier, loose coupling is of highest priority. It means that application intended functionality must be achieved but flexibility must be provided to easily incorporate changes if any. When applications integrated, they share data among themselves. Ideally, data should be in same format. But this is not the case always. Applications have to admit on a data format to start sharing seamlessly. Timing latency needs to be dealt with while integrating enterprise applications. Integration involves sharing of data and functionality between two applications. Remote communication needs to be asynchronous in nature. Due to uncertain network availability and asynchronous nature of communication, reliability of the communication is at stake. The fixed integration solution is not possible after considering all the challenges. Integration solutions developed over the years, and now four significant patterns exist: [HW04]

2.7.1 File Transfer Pattern

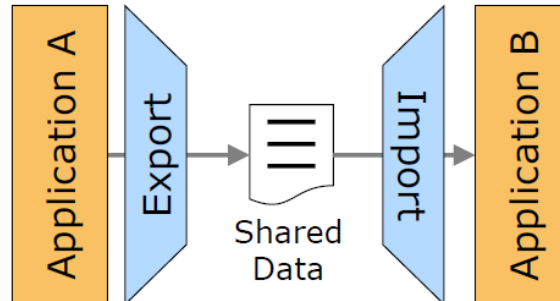


Figure 2.7: File Transfer Mechanism [HW04]

As shown in Figure 2.7, a file is the standard mode of communication when two applications needed simple data transfer, economical solution. Without having application insights, file transfer pattern can be integrated. Loose coupling can be achieved. But whole communication can go out of sync due to late updating of files. Frequent file updates result in increased processing cost.

2.7.2 Shared Database

Each integration mechanism tries to overcome problems within its predecessor. File sharing mechanism was highly inefficient in maintaining timeliness. Also, data connotations might be different within different files. Hence, to avoid these problems along with frequent synchronization,

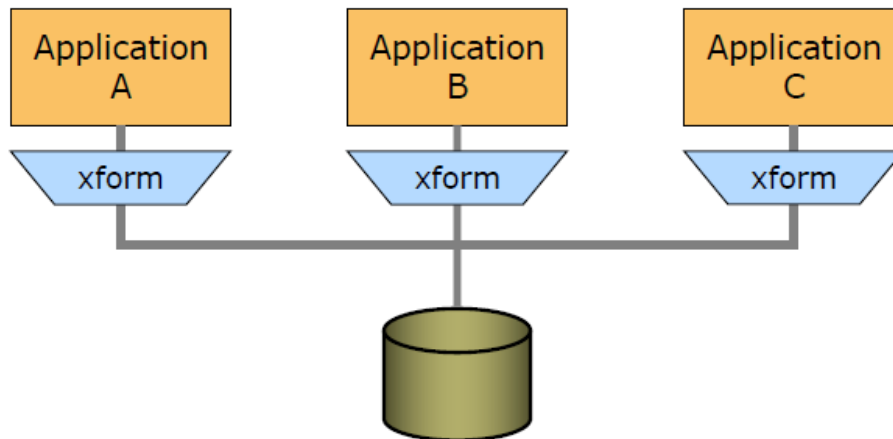


Figure 2.8: Shared Database Mechanism [HW04]

shared database solution has been proposed in Figure 2.8. As expected, semantics problems have been solved using a shared database, but the new question arises. Designing such a database itself proved a challenging task. The speed of DB over WAN is slow. Hence, a new solution is required.

2.7.3 Remote Procedure Invocation

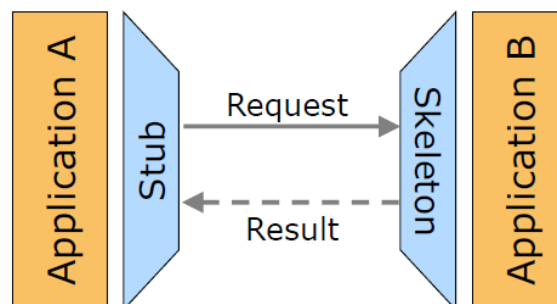


Figure 2.9: Remote Procedure Invocation Mechanism [HW04]

Remote Procedure Invocation (RPI) Figure 2.9 required when only data sharing is not sufficient for application integration. Few operations need to be performed along with data sharing. It is a mechanism where one application is responsible for function invocation in another application. Here interface is necessary for both applications so that external code can interact with application code. Direct communication between applications can be established along with data sharing. Create, Read, Update, Delete (CRUD) operations can be performed on data using RPI. The significant issue associated with RPI is no loose coupling between applications. Applications are tightly coupled, and hence no flexibility has been offered. Loose coupling is of high priority regarding distributed systems. Therefore, practical solution over RPI is messaging mechanism.

2.7.4 Messaging

The fundamental shortcoming of Remote Procedure Call (RPC) is tight coupling. Timely integration and loose coupling are primary requirements in the distributed systems. A remote procedure call is comparatively a slower functionality. Widely distributed system applications need a communication system which is capable of high-speed communication, which makes the transfer process easy, having an alert mechanism and asynchronous communication mode. The solution is ‘Messaging’ Figure 2.10. Messaging facilitates loose coupling within numerous applications, although it needs

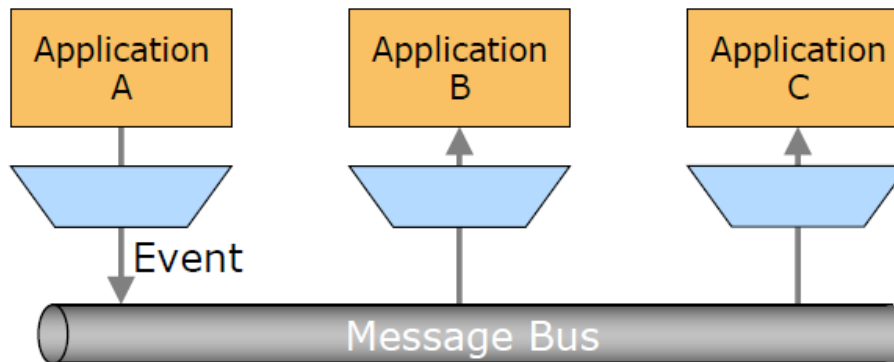


Figure 2.10: Messaging Mechanism [HW04]

more complex software interface. Since messaging is asynchronous and faster mode of communication between distributed applications, we are going to implement the same within this thesis. Test application and regular vehicle application are two distinct applications which will communicate throughout the testing process via messaging communication mechanism.

2.8 Message Oriented Middleware (MOM) – Details

Large distributed systems, dispersed among different organizations, need to exchange the information amongst various applications. For this reason, these systems need to communicate with each other. But within two different enterprises, communication is not a straightforward approach. These enterprises need to be integrated to establish information exchange facility. Loose coupling is another aspect to be considered for remote information exchange within two distinct applications. While integrating two applications within enterprises, application code should not be affected due to an integration mechanism, as well as communication should be achieved asynchronously. In the asynchronous mode of communication, a client does not need to wait for the response from the server. Hence, waiting time reduces, and it improves the throughput of the entire system. When the requirements mentioned above are inevitable, traditional communication protocols become absolute. One cannot integrate enterprise applications using the synchronous communication protocol like RPC. Also, peer-to-peer messaging methods like REST protocol are inapplicable due to variations in application programming languages and deployment platform. Hence requirement arises for message transfer communication architecture like MOM [TGG02].

In MOM, a client does not connect to the server, i.e., no point to point communication. A middleman is available to validate both server side and client side communication. This broker connects two different systems and establishes a connection between them via sending messages just like a

message carrying person. Moreover, the client does not need to wait before receiving a reply from the server. That's the paramount advantage of MOM which increases the throughput of the complete communication system – asynchronous nature of communication. Figure 2.11 gives a basic idea for MOM which acts as a middleman and enables messaging communication between various enterprise applications. If two applications deployed on the same OS, they will be sharing

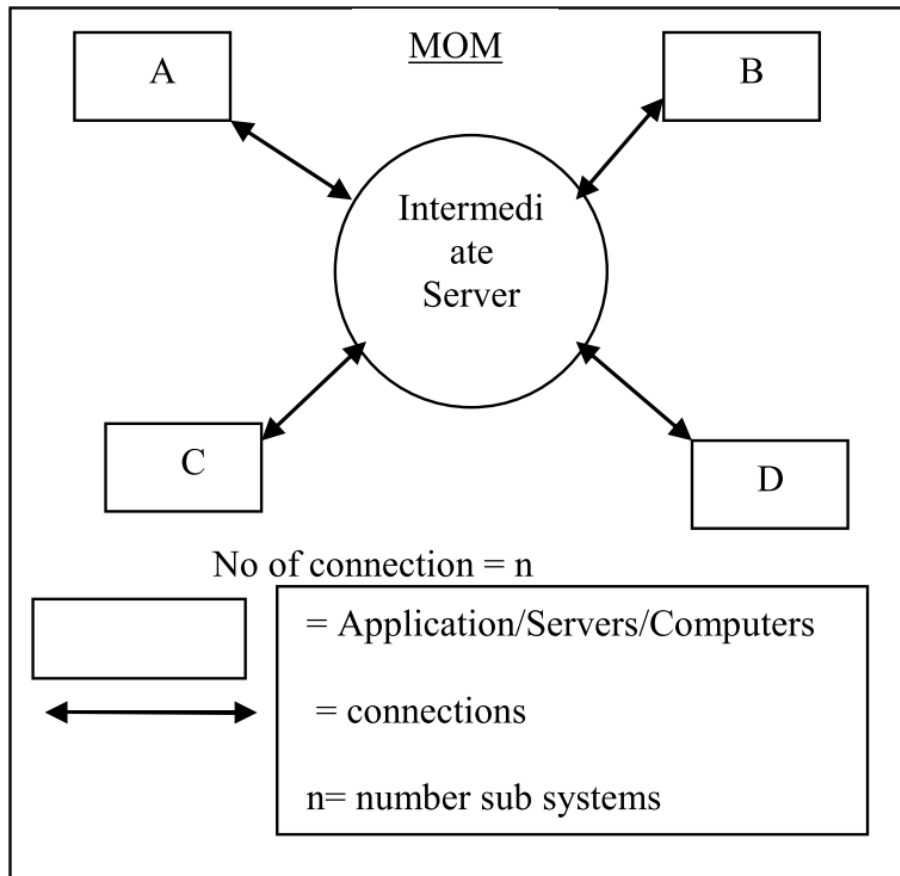


Figure 2.11: Basic MOM Concept [CUR04]

a shared memory. Data sharing won't be an issue. But it is not the case with distributed remote application systems. Messaging is an only viable option to communicate along with data. Platform independence and generic approach towards programming language are essential for this messaging service. One cannot guarantee that two remote applications will have the same OS or they are written in the same language. Messaging system works as an interpreter between these applications by exchanging information as messages. A critical feature of messaging is the asynchronous mode of communication. "send and forget" approach allows sender application become free after sending a message. i.e., no waiting till reply. This approach increases the throughput of the complete system. So here, since the sender is not waiting at all for the receiver to answer back, it can work in its own time lines and receiver can work with its separate timeliness. Hence, maximum utilization at both the ends can be achieved. This independence in timing avoids overload on the system and therefore, no degradation of the performance. Messaging communication exhibits reliability. Since it can store every message before its gets deliver, there is no threat of loss of data. As we know, network connections are the one which cannot be relied upon. Communication via messaging queues

facilitates applications to synchronize after re-availability of the network. Consistent operating behaviour, acting as a mediator and effective listener management for call back procedures prove communication via messaging is inescapable for distributed system applications. There are some challenges in communication via messaging approach. Messaging is event-based protocol. Hence, programming scenarios can be complicated. Since its asynchronous mode, message delivery is guaranteed but timing. Therefore, rearranging message sequence could be an overhead. Not all applications can communicate using asynchronous communication. Synchronicity to some extent is needed. But, we have a solution – “Request-Reply pattern” [CUR04].

Data is not always ready in messaging form. It has to be converted into messages Figure 2.12. This conversion adds extra responsibility to the system. There is a lack of platform support for communication via messaging. Communication between two messaging systems itself can be difficult due to no common implementation protocols [HW04].

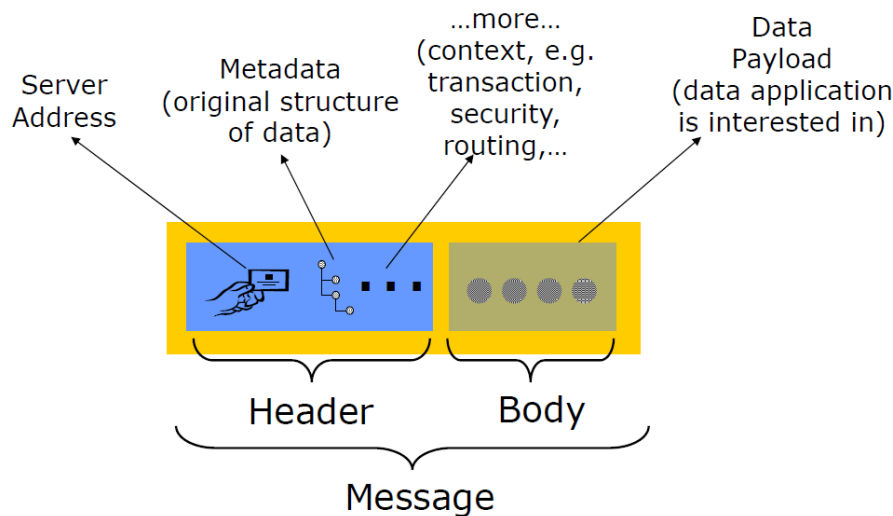


Figure 2.12: Message Structure [HW04]

2.9 Messaging Queues

Queuing is lining up something. The same happens with multiple messages. While communicating with a remote application, data transfer generates several messages. These messages are lined up in a message queue on First In First Out (FIFO) basis. Hence, the queue is a medium for information exchange. The storage of the message and its forward circulation depends on the queue structure [VT06]. The primary advantage of the queue approach is application independence. Knowledge of the pairing mechanism is not required. As shown in Figure 2.13, an interface helps in the abstraction of queuing protocol from application code. Messaging queue connects to this interface. Applications usually communicate with the interface provided by the queue. Queue in general, a database with multiple messages. Queue follows receiving order while delivering messages from one application to another. Priority-based execution can change this paradigm [HW04].

FIFO architecture of queue explicitly replicates a pipe structure. Input at one end and output with an assigned location at another end. The queue can deliver messages to applications as well as to

remote/other local queues itself. It depends on the routing algorithm of the respective application. Before routing, messages need to be stored at a location. Persistence of the queue storage depends on queue architecture itself [VT06].

Hohpe [HW04] has also mentioned about various methods in Message Queue Interface. This Application Programming Interface (API) is used to open and close connections, establish routing, etc. It is very efficient to use queue architecture for communication between distributed applications. There are various queuing protocols exists such as Advanced Message Queuing Protocol (AMQP), Zero Message Queue (ZMQ). We are using ZMQ for remote application testing approach.

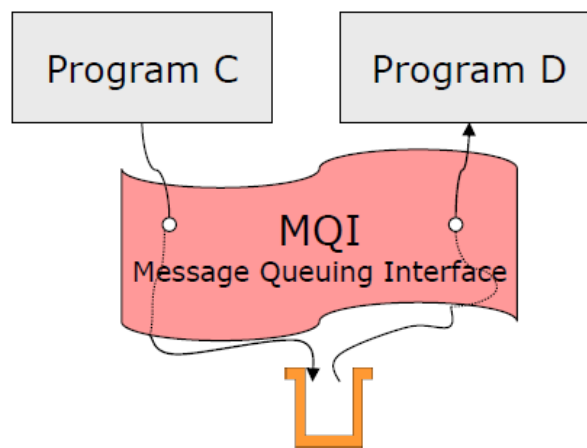


Figure 2.13: Application Communication via Message Queue [HW04]

2.9.1 ZeroMQ

IMatix has developed ZeroMQ. As the name indicates, it is a messaging queue. But unlike regular MOM, it is a messaging library, primarily developed in C language. ZeroMQ can be used with multiple languages, and we are planning to use its python versioning, i.e., pyzmq. ZeroMQ works in request-reply pattern as well as the publish-subscribe pattern using TCP type sockets connections. Using zeroMQ multiple clients can be connected to one server. ZMQ communication does not depend upon the data type, it is based on the number of bytes transferred over the socket. For e.g. Length of data is required in order to transfer the string, as depicted in figure 2.15. Also it cannot differentiate between a null character or other normal character. Length needs to be always specified from strings. Now a days, many applications are distributed in nature and connected across networks (LAN, WAN), etc. Communication between these applications using messaging is the most stable and suitable option. Typical TCP/UDP connection protocols do the needful, but reliability is at high priority. There is a huge difference in achieving messaging with high reliability using MOM and standard communication protocols. ZeroMQ is a message-oriented middleware which is a library. It does not behave like a broker within the client-server application. To reduce the complexity of complete architecture, it is a good idea to introduce a message broker. In contrast, it also means adding a point, on which complete system is dependent and its failure can cause the complete system to collapse. Its maintenance is moreover complicated. ZeroMQ avoids this broker-centric approach. In addition, it is entirely OS and programming language independent. Little maintenance efforts and least operating cost with high portability make ZeroMQ unique to use in all messaging based

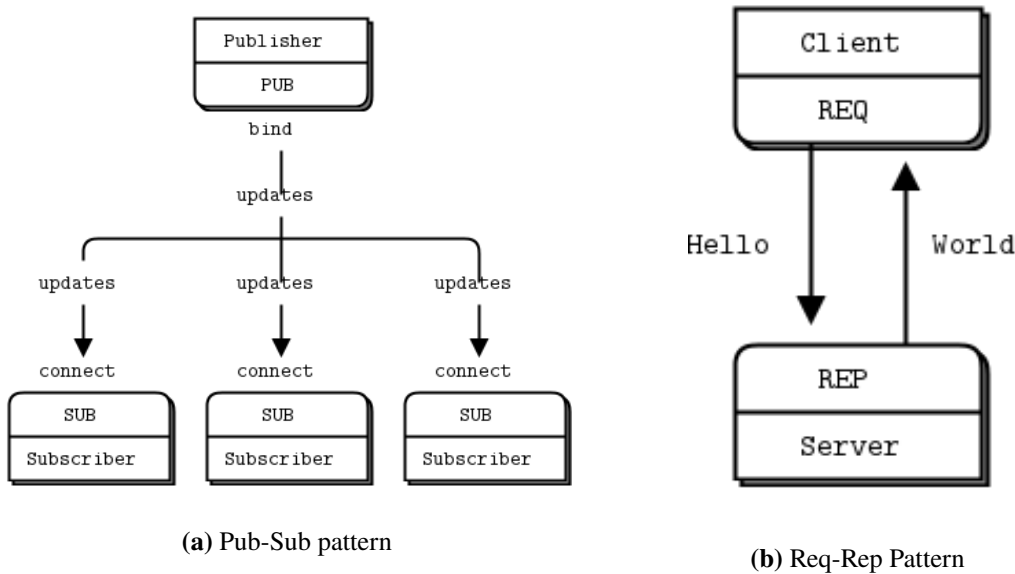


Figure 2.14: ZMQ Connections [ZMQ]

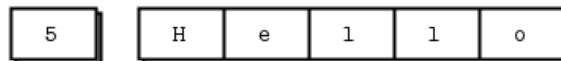


Figure 2.15: ZMQ String Data Type Representation [ZMQ]

applications. Along with ease of operability, ZeroMQ provides various advantages and features regarding message-based communication. The most important one is asynchronous communication. In a client-server application, a client does not need to wait for the server’s response. Moreover client can continue with its own task. Hence, the overall throughput of the system increases by increasing communication speed. It gracefully handles overload situations by either discarding message, or blocking communication to particular sender. Also it supports the usage of wide range of protocols. For, e.g., TCP, inter-process communication, multiple broadcasting, etc. All protocols will be used with no change in the base coding of ZeroMQ. One can establish peer-to-peer communication using ZeroMQ (Figure 2.14b) or even publish-subscribe type (Figure 2.14a) approach can be used. ZeroMQ allows easy creation of proxies. It delivers a complete message with high accuracy. There is no limitation as such defined in the zeroMQ protocol for message length. Network is very critical part of any distributed system. ZeroMQ has various options to handle network errors gracefully avoiding loss of data. ZMQ is a socket based messaging library. Multiple bindings using single socket is possible in ZMQ server. Multiple clients can connect to these bindings. There is a significant difference between standard TCP sockets and ZMQ sockets. All operations of ZMQ sockets always take place behind the normal system operation. No application thread gets affected due to ZMQ messaging overhead. ZMQ makes the message available at a queue so that the asynchronous communication protocol will handle communication IO thread [AKG13] [ZMQ].

3 Related Work

Testing of mobile CPSs combined with security of people using them is considered of prime importance in this thesis. It is more critical to test the system during its run time. It also adds a risk of interfering with on going operation. Therefore a massive research is going on in this field to create a safe and reliable run time test environment. Run time testing in automotive and embedded field has been proposed by Bradatsch et al. [BUZL11], for online testing of system operational faults detection. The three layered architecture proposed by Bradatsch is based on concurrent access to the system hardware. As run time testing is proposed, it is relevant that system will perform testing while continuing with normal operation. Bradatsch et al. also made use of the Built In Self Test (BIST) principle, which is further used in the thesis. Having faced hardware concurrent access problem as well, the thesis deals with this by implementing the hardware redundancy solution within the architecture. Figure 3.1 highlights a three layer architecture. The Hardware Resource Manager

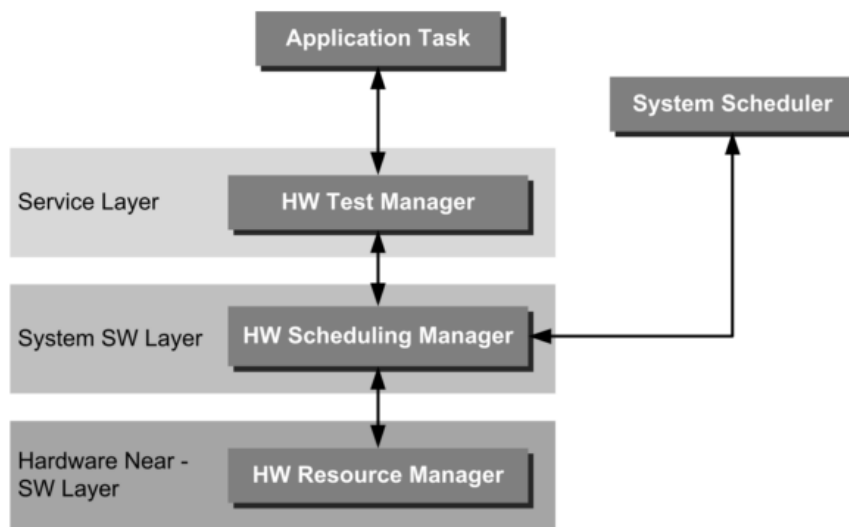


Figure 3.1: Three Layer Test Architecture [BUZL11]

(HRM) layer allocates the resources to the test system. The Hardware Scheduling Manager (HSM) is responsible for the Hard real-time system timing constraints. HSM further notifies the system scheduler and the HRM about all the timing information. HSM subsequently receives input from the Hardware Test Manager (HTM) regarding the test requests. Employing this architecture, resources are allocated based on the HRM information. Resources either get allotted to the test system or to the normal system operation, according to the scheduler. The system is advantageous with respect to the maintenance of hard real time deadlines and stringent safety standards in accordance with the ISO standards. The system is further subjected to limited overhead during run time testing, owing to the proper scheduling of normal operations and the test systems. However, such a scheduling demands

strong handshake between the resource consumers and the scheduling mechanisms, resulting into reduced flexibility of the complete architecture. Furthermore, all the embedded systems do not possess such handshaking capability and hence the architecture proposed by Bradatsch et al. helps reduce its resilience.

Mikucionis et al. [MLN] proposed the idea of model based testing approach for real-time systems. Testing of the embedded systems utilize critical resources like memory, execution time, and hence it should be systematic, well founded and automatic in real-time. Therefore, testing is divided in three basic activities – generation, testing and result. Hence, Mukucionis et al. developed a tool named T-UPPAAL for model-based testing approach for the embedded systems. T-UPPAAL uses an online testing approach where generation and execution is a simultaneous process. In this scenario, the system under testing is connected to T-UPPAAL via an adapter to translate the data actions.

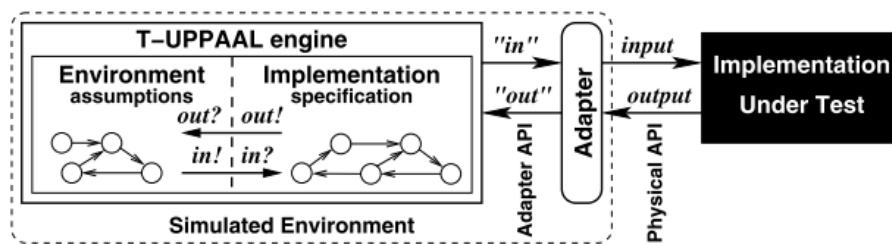


Figure 3.2: T-UPPAAL Engine [MLN]

Modelling of the real-time systems is popularly performed by timed automation, which is a detailed finite state machine (Figure 3.3). T-UPPAAL works on the basis of online testing by receiving input i.e. test case creation and execution parameters from these models. Online testing proves advantageous in multitudinous possible ways such as long run testing and overcoming memory issues which are prominently visible during offline testing. Additionally, online testing methods are

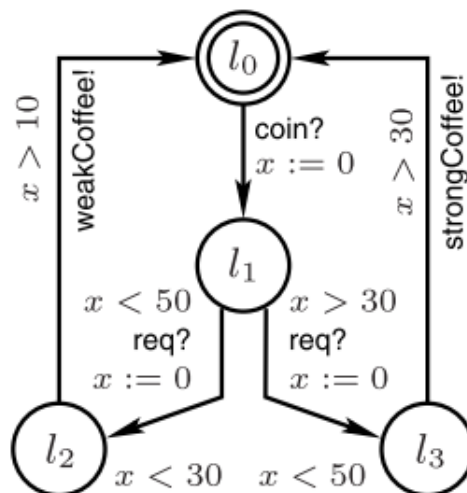


Figure 3.3: Timed Specification State Machine [MLN]

adaptive in nature. T-UPPAAL works on the basis of online testing algorithms, which are further implemented for testing of mobile CPSs in this thesis. This is a run-time testing approach with minimal additional test time required.

Crenshaw et al.[CB10] designed a test bed for cyber physical systems named Upbot, with the primary objective of confirming the intended working of the software, protecting the CPS from malicious attacks and therefore increasing the human safety. It provides a test bed for the valid cyber physical system. It consists of mainly three parts – firstly, the body i.e. the hardware, secondly the nerves i.e. the communication medium and lastly the brain i.e. the processor. As shown in

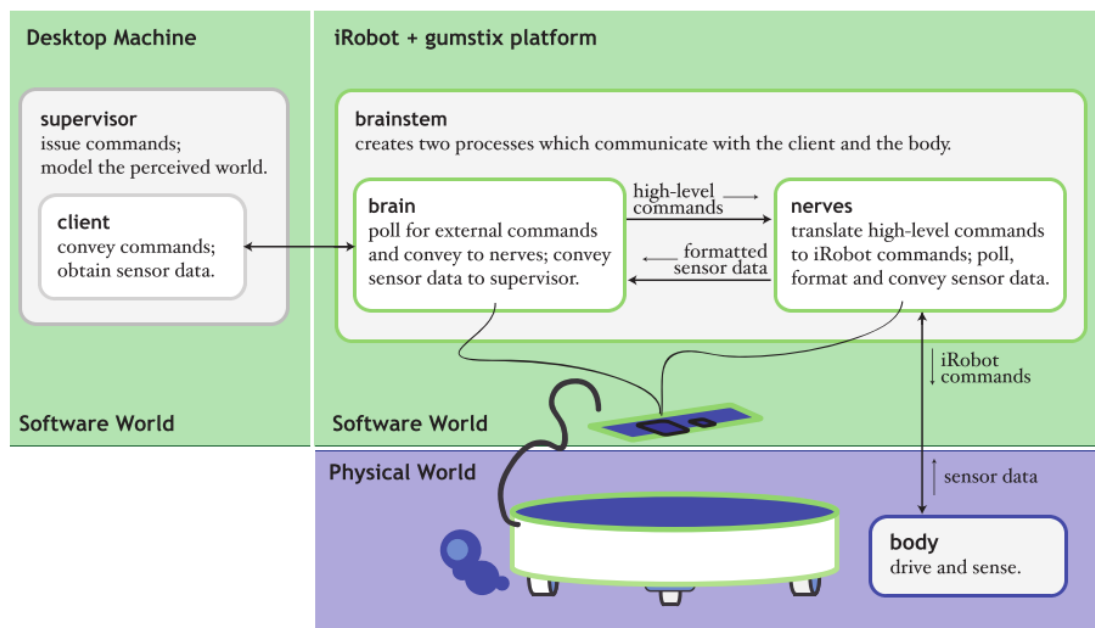


Figure 3.4: UPBOT Test Bed Architecture [CB10]

the Figure 3.4, architecture of Upbot works on local decision-making and remote decision-making principle, which is also implemented in the thesis architecture. The nerves of the system continuously poll the hardware to obtain real-time current sensor data which is then fed into the brain for further processing. Here upbot utilizes the concept of continuous monitoring as it requires real-time sensor data. The architecture demonstrated in this thesis implements this concept of continuously monitoring of the sensor data before starting the test case execution. The brain of architecture is the actual processing on the data collected which is further conveyed by the nerves. The brain either processes the information locally and replies to the sensors or simply passes on the data to the supervisor over the network using the socket connection. The thesis demonstrates the socket connection using message-oriented middleware in the architecture. Upbot has been designed keeping in mind the test system for CPS, artificial intelligence being one among them which makes use of the hierarchy in the architecture. In such an architecture the brain's ability to make local decisions is limited to small decisions and requires a further involvement of the supervisor for the higher computing decisions. This hierarchical approach is appreciable however in the thesis architecture, this decision will be solely dependent on the type of the mobile CPS application and not on computation requirements, as the thesis provides a redundant hardware with processing power. Unlike Upbot, the thesis does not utilize multiple sockets and pipes for inter-process communication, instead the local testing

approach which increases the hardware capacity of the complete system. Further, for the remote testing approach, the thesis considers the network and socket related issues using MOM ZeroMQ, which is further validated with a small system prototype.

The challenges considered by Koopman et al. [KW16] in vehicle testing and validation were proposed in compliance with ISO 26262 vehicle safety standards V model. The thesis mainly aims in achieving the Automotive Safety Integrity Level (ASIL). For achieving high ASIL, Koopman et al. suggested monitor-actuator architecture with hardware redundancy, wherein the actuator performs the main function and redundant hardware monitor does the job of data validation. Figure 3.5

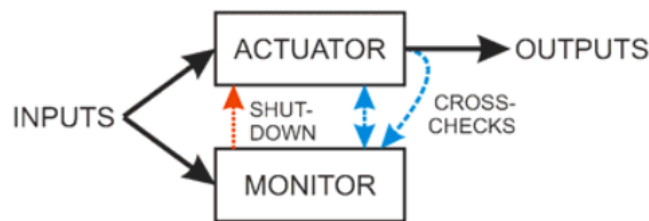


Figure 3.5: Monitor-Actuator Pair [KW16]

demonstrates the architectural approach suggested by Koopman et. al. to achieve a high ASIL, which is the continuous monitoring of actuator/sensor data and further using this data for testing and validation purposes. The monitor actually validates if the data is within the predefined range or not. If the actuator behavior is not compliant as per the desired constraints, then monitor shuts the actuator system down resultantly creating a fail silent system. Nevertheless, this can be proved as a disadvantage for the mobile CPSs. Shutting down of a actuator function may not be feasible in each case and thence, Koopman et. al.'s architecture demonstrates this disadvantage.

In the test monitoring system, the sensor and actuator data is monitored and if the data is found to be out of bound during the testing, the actuator function is not shut down, instead the user is informed about the test case failure and guided through the further steps to be taken to achieve a fail-safe mode. Further, the actuator is also made to be within the constraints and achieve fail safe conditions making use of the system safety guards (fail safe condition scenarios in case of safety critical system fault). The entire actuator-monitor system defined by Koopman et. al. is based on the fail operational behavior i.e. having multiple system redundancy. However, the fail safe operation is preferred as it will be difficult to implement the multiple actuator-monitor pairs in a huge mobile CPS which are needed for the fail operational approach. Moreover, it will be an additional expense in an already expensive mobile CPSs. Our solution therefore is economical and also serves the purpose of run time testing while providing continuous monitoring of the sensors and actuators data in mobile CPSs.

Al-assad et al. [AMH98] proposed a test system with concept of Built In Safe Test (BIST) for embedded systems. Mainly focused on hardware testing, it dividesthe various embedded system testing in multiple parameters. Critical applications like automotive safety are the primary conditions of online testing. Operational faults in these systems can be avoided using BIST.

They mainly divided online testing in concurrent and non-concurrent approaches (Figure 3.6). Non-concurrent testing is event or time triggered and this testing approach is also focused upon in this thesis. BIST is the solution which covers all type of faults. BIST is developed on the basis of four important parameters – i.e. the fault coverage area, the size of the tests set, the hardware

performance overhead and the performance cost. This master thesis implements the BIST principle.

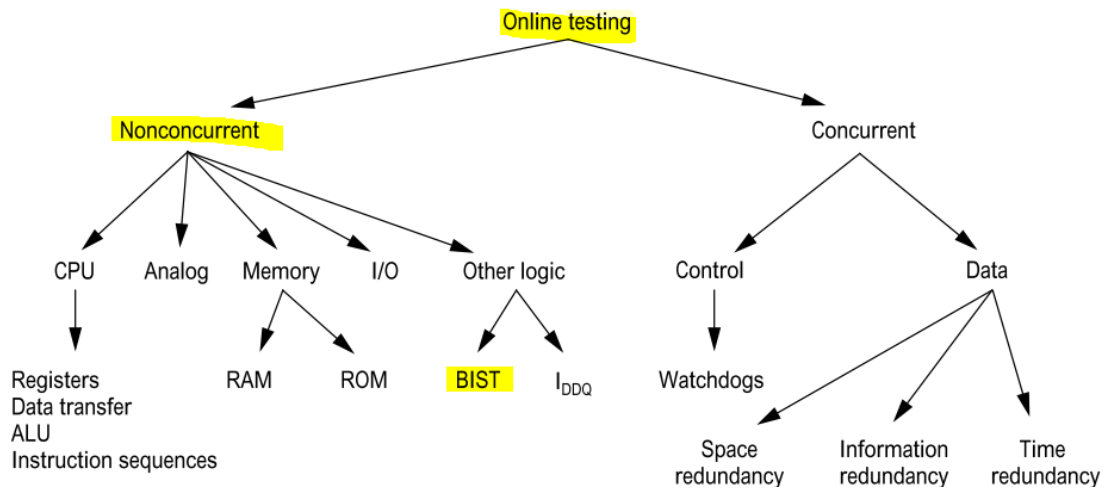


Figure 3.6: Online Testing Classification [AMH98]

BIST fundamentally utilizes the same normal system inputs as the test system inputs. When in the normal operation mode, the test system is not activated. However, when the user wants to perform the testing, the normal system inputs will be used as the test system inputs by digital logic circuits. Once the test starts, normal system inputs get monitored by monitoring and test system and testing is performed according to the test logic specified by the user. Once the testing is completed, a report is generated and information gets presented to the user. After testing, the normal system operation resumes. The BIST approach in the embedded systems, requires no separate time out for testing. Also, the overhead on the hardware is comparatively less due to no involvement of additional system inputs. BIST is a basic feature in critical safety and high availability applications. Therefore, the thesis has integrated hardware and software in the architecture with the use of BIST to enhance the safety and the reliability in mobile CPSs.

Bringmann et al. [BK06] proposed a testing approach for automotive embedded systems. This research is based on the graphical notations of the test cases. Automotive systems work with continuously changing data. The rate of change of data being very high requires continuous monitoring thus implying the need of continuous behavior monitoring. Time Partition Testing (TPT) is the approach suggested for this task. TPT selects the test case systematically and performs automated real-time testing. To select the test case, a strong descriptive test language is essential. A test language must be capable enough to describe a single test case. Also, it should optimize the test case avoiding the redundancies and the initial modelling of test cases carried out by graphical notations. Textual test case requirements noted down graphically. This has advantage of simplifying complex scenarios and formal description of the complete procedure. Graphical notation also makes test case sets modelling easier. When two test cases are almost identical, there needs to be an unique functional requirement. If not, test cases will be considered as identical i.e. redundant. TPT identifies this difference and hence avoids redundancy in test cases. It essentially models the test cases together in an abstract model called “testlet” Figure 3.7 depicts testlet for headlight testing. Here three variation pints exist which can alter the output of the testing. As the system gets complicated, the number of test combinations grow exponentially to 39 dealing with this problem, TPT makes use of classification tree method with graphical notation which offers high flexibility.

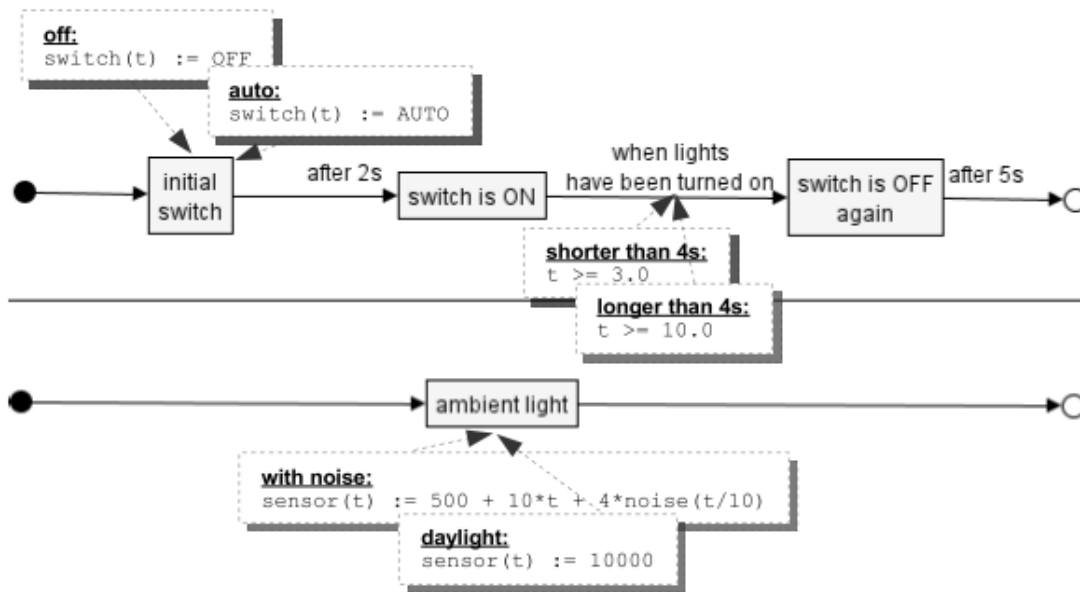


Figure 3.7: Testlet for Headlight Testing [BK06]

TPT test process is divided in multiple blocks wherein each block works independently from the stage of requirement gathering until generation of report. Figure 3.8 depicts these blocks and provides overview of TPT. The TPT provides support and automation at all levels of testing as much

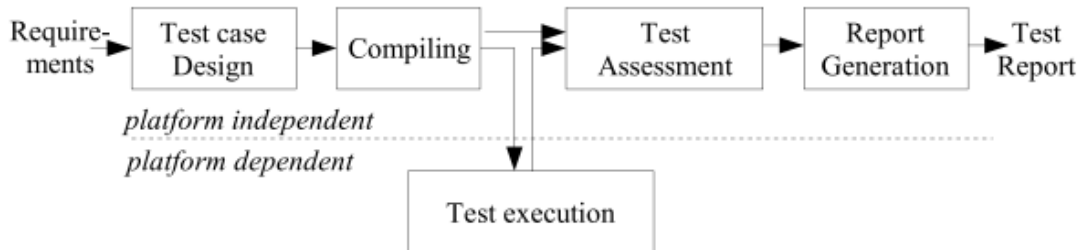


Figure 3.8: TPT Test Process [BK06]

as possible. It is platform independent approach. Current research is focusing on integration of TPT with advanced test methodologies to achieve automation when it is difficult to specify input signal. This thesis demonstrates pre-designed test cases with predefined input signal. In its approach, the thesis test start decision depends on some logical constraints. Once the testing is started, the algorithm is responsible of automated testing and assessment.

Hamilton et al. [TBH+01] mentioned about vehicle diagnosis model in his research. As described in Section 2.5, knowledge of vehicle performance can be obtained by three different spaces i.e. fault space, Diagnosis space and Observations space (Figure 3.9). These spaces are interrelated and exhibit interdependency. Fault space denotes all possible faults in a vehicle and its size could be infinite. The control over fault space is not in the hands of the designer. However, he can extend the control of the observation space by increasing the number of sensors. This thesis deploys all the vehicle sensors as an observation space and hence try to cover as much faults as possible, thus

exhibiting direct relation between the observation space and the diagnosis space. According to Hamilton et al., this ratio is 1:1 i.e. one sensor responsible for explicit diagnosis. Further, it is completely possible to boost diagnosis capability using poly-dimensional approach. This implies the use of the same sensor for monitoring multiple aspects of the vehicle. This will keep the observation space of the same size enhancing only the diagnosis space. Appreciating this idea of the sensor space the same has been tried to be implemented in this master thesis thus using each vehicle sensor with all possible dimensions and trying to achieve maximum diagnosis space. Hamilton et al. also

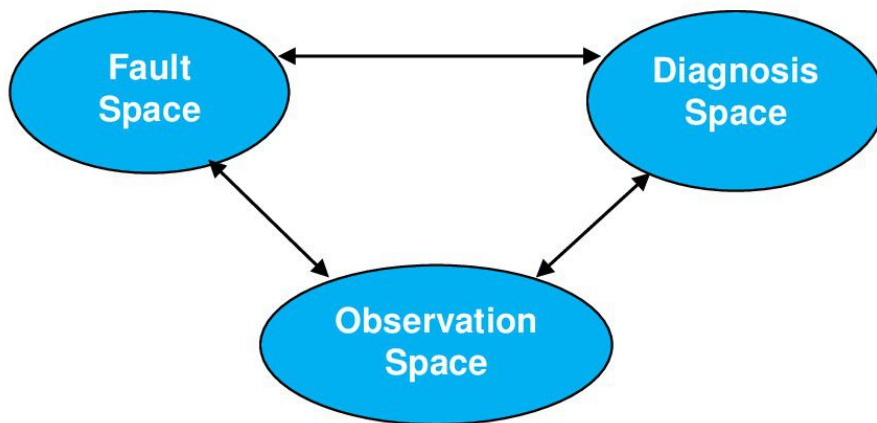


Figure 3.9: Design Spaces [TBH+01]

suggested the use of miscellaneous knowledge about vehicle data and behavior for fault diagnosis. Mainly the manufacturer makes provision to generate this knowledge as a part of the vehicle design process. But the knowledge related to vehicle performance history is a part of diagnosis process and generated over time. To relate this knowledge to fault diagnosis, the fault diagnosis methods must be equipped with modularity support, machine learning technology and advanced systems. Thus this fact of relating fault diagnosis systems with machine learning algorithms to take care of vehicle historical data is well appreciated. This definitely strengthens the diagnosis space for vehicles and would be the future scope for our master thesis.

Baleani et al. [BFM+03] suggested fault tolerant approaches for automotive safety-critical applications. In the current times, electronics has become an inevitable part of the automotive industry. With increase in use of electronics, risk of system failure also increases. Hence, fault tolerance mechanisms must exist. Every fault tolerance mechanism is based on the fact of redundancy. Each differs from another by means of handling redundancy. Fault tolerance systems make automotive systems reliable. To achieve this, fault tolerance system must satisfy hard real time requirements and must be economical. Using separate power supply and clock sources, multiple processor architectures can be used. Common mode failure is the primary issue with single chip systems. It is always advisable to design complete system with full hardware redundancy. Need of isolation of power source and clock is paramount. This ensures that fault should not cross the line between two systems and single point of failure won't result in complete system failure. Baleani et al. suggested

several architectures to achieve fault tolerance using multiple processor architecture. These methods are useful when one needs to achieve cure from the faulty condition. This master thesis focuses on basic prevention of the fault by allowing run time testing by the user. Thus considering the lock step duel processor architecture suggested in this research the same kind of approach is implemented in the test system. As shown in Figure 3.10, two redundant processors are used. Master processor

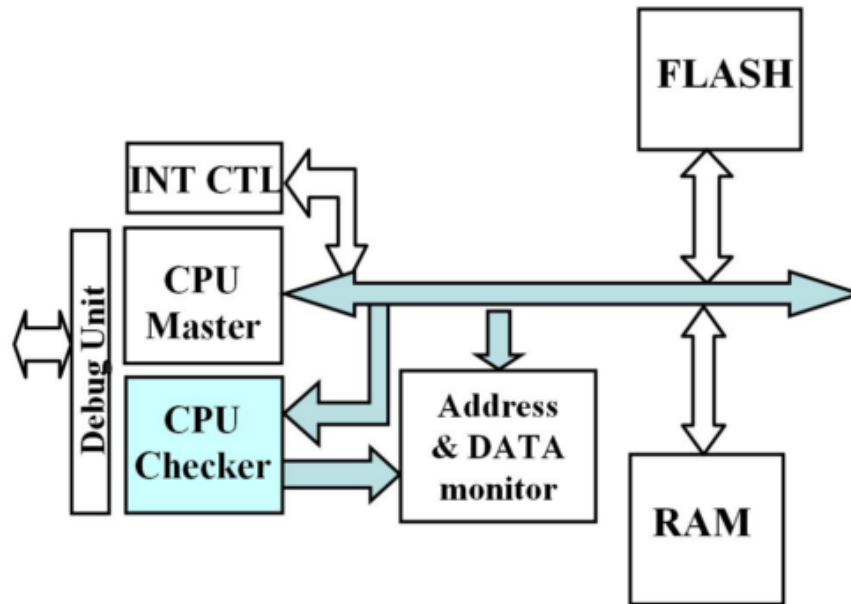


Figure 3.10: Lock Step Duel Architecture [BFM+03]

is the main processor having access to system peripherals and memory, whereas checker checks the state of the bus. The output of the Checker feeds to the comparator and checks consistency for the data. Any discrepancy in data is considered as a fault and analyzed further. During such analysis the system must be held in the fail-silent mode. This logic is optimum for the test system. We are using a redundant processor hardware to separate test code from master (application) code. Master processor has access to complete system and its peripherals. Since our idea is to prevent faults by testing, we run our test code on test redundant processor. As a comparator logic, we have decided some constraints to be satisfied in each test case. If these constraints do not get satisfied, we consider this as a fault condition. Our system is fail-silent too. This architecture is suitable for large applications as vehicles and also for small applications as mini drones. Architecture deals with high computational requirement and safety critical applications.

Cancellara et al. [MAU05] published patent for remote and local vehicle diagnosis. There are three different modes explained in this architecture (Figure 3.11). First is Local testing mode. Other two are remote testing mode with passive and active configuration. Here ECI is a vehicle communication unit. In the local test approach, ECI is having a small display. This display is connected to local test systems via wireless link, thus avoiding any connection of ECI with the service bay. In the remote passive testing approach, ECI connects to a local PC which further acts as a gateway and connects to the remote system in the external network via the client server type connection. In remote active testing approach, ECI itself establishes connection with the remote server network. The complete system architecture is having a modular approach. Processor, in vehicle communication via CAN

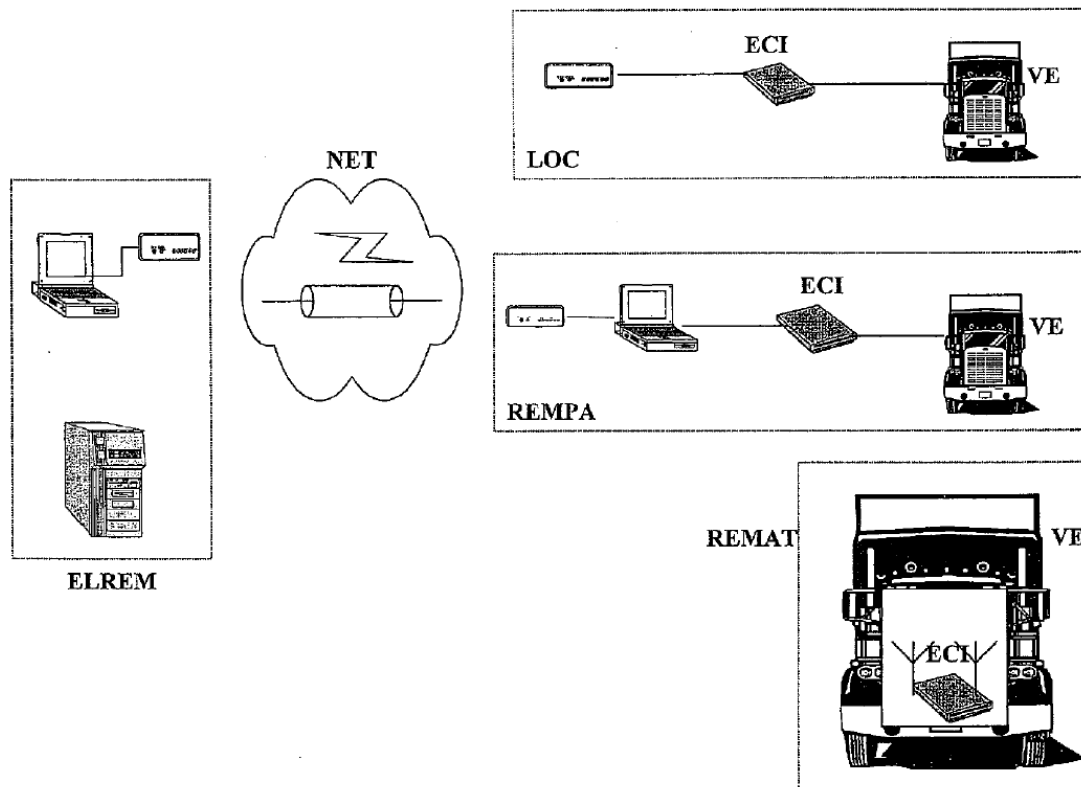


Figure 3.11: General View of Remote and Local Test Approach [MAU05]

bus, local external device communication via USB, RS232, communication module for remote external networks i.e. GSM, GPS etc. are basic building blocks of the architecture. This type of modular approach makes the software flexible. Additionally, it makes all the modules loosely coupled with one another i.e. change in one module hardly affect the other one. The complete summary of the architecture starts with the protocol management module. In our system, it is serial communication with in-vehicle electronics. A module which communicates with the outside network is an endpoint for message socket as per the message oriented middleware. Middleware part is responsible for external communication and processing. In our thesis, this middleware is message oriented and is responsible to carry out the commands, data and test results from local to remote network. The application layer part is the actual test application. The consideration here is this whole patent is designed for testing along with test bays either locally or remotely. In our thesis, we are not depending on test bays. We need to provide the required authority to the user to perform the testing by himself in a running system. Hence we have modified the approach and redesigned it with the appropriate hardware redundancy and software coding.

4 Concept

This chapter describes our system concept overview. Our idea is to develop an architectural approach for run time testing of mobile CPS. We have discussed researched questions, focused on assumptions and own architectural ideas in the upcoming sections of this chapter. Our research questions and assumptions are defined in Section 4.1. Section 4.2 provides brief overview of our architecture concept. Each architecture module explained briefly on Section 4.3. We separate local approach and remote approach in Section 4.4 and explained them in detail. Configuration we used for each approach explained briefly in further sections. Details about introduction to messaging channel and message interaction are provided within Sections 4.7 and 4.8 respectively. We conclude this chapter with Business Process Modelling Notation (BPMN) behaviour diagrams in Section 4.9.

4.1 Research Questions and Assumptions

Our focused on testing, especially in mobile CPSs. There are many research questions to be answered in this field. Our primary focus is to cover most of them during our work.

Research Question 1: When is the correct time to execute test case in the mobile CPSs during run time?

It is equally important not to interrupt normal working of the vehicle. No driver would prefer testing while driving if it affects normal operation of the vehicle. Our idea is to perform testing using same inputs as normal vehicle operation and perform built-in self test (BIST). This facilitates testing during normal vehicle operation. Hence, no specific time slot is required for the test scenario.

Research Question 2: “What kind of constraints must be met when executing tests in mobile cps at run time?”

Even though we are using BIST approach, each and every time it is not possible for a vehicle to be in testing mode. For example, ABS testing is required and car is at a speed of 100mph, one cannot put breaks and perform ABS testing. For this particular test, car has to be below the predefined speed limit. This limit is addressed as constraints on a particular test case. In this master thesis, we are proposing constraints based testing approach for mobile CPSs.

Research Question 3: “Where do you execute tests in mobile cps during run time?”

The original system hardware is not equipped with test hardware. Adding extra hardware will increase additional cost. Moreover, some OEMs do not allow modifications or access to their application code. we handled this issue by adding extra hardware which is dedicated for testing. This additional hardware is with minimal capabilities and need not be as powerful as the original system hardware. This reduces extra cost due to hardware redundancy.

Research Question 4: "How do you perform tests in hardware constrained mobile cps?"

This question is important in context with small mobile CPSs for e.g. Drones who have original hardware with low processing power where we cannot add additional test hardware and test node with the whole system. Remote testing is inevitable and hence, we need to perform testing on remote server instead of local redundant test node hardware.

Assumptions

Mobile CPS is a wide sphere of applications. It could include heavy vehicles as well as small size drones. Each of them has different hardware and hence different processing power. Huge vehicle hardware with well implemented CAN data bus allows insertion of hardware redundancy. Whereas for small size drones, it is even difficult to do small modification in hardware because of small payloads and processing power. Hence, we cannot use a generalized approach for our test system architecture and it is supposed to be application dependent. So we have constraints to design architecture which should be flexible according to the application. Major obstacle in flexibility of the architecture is integration with application code. We must consider a fact that most of the times application code can not be accessed via interfaces provided by manufacturers like Controller Area Network (CAN) or On Board Diagnosis (OBD). Vehicle manufacturers do not grant access to core functionality for all third party companies. One cannot modify core functionality within the original application. Manufacturers only provide access to various sensor data via OBD. We are obliged to design our test system architecture using this sensor data. Very obvious solution is to add additional hardware which will take care of the whole test system code without interfering with the original application code. We named this hardware as Local Processing Unit (LPU). As mentioned earlier, we planned to keep system independent of operating system and platform. So we focus on designing test cases as .hex files, and these byte files need to be flashed on a hardware platform to perform the testing. Flashing of test files on original application hardware is not feasible because it will interrupt running application during run time. Moreover manufacturers may not provide programming and debugging functionality with the system hardware. Hence we need to add a 'test node' i.e. a dedicated hardware to only flash test cases and perform testing accordingly. this hardware can be on sleep mode, except during the testing phase to avoid any interference. As stated earlier, we have flexible architecture according to the application type and we consider a complete overview of the same with our assumptions which are related to system requirements and hardware dependency.

Assumption 1: We have to assume preconditions about system, hardware and software in order to proceed with our work. we are assuming that mobile CPS must be equipped with real time operating system (RTOS) For eg. FreeRTOS [FRE]. As we want to perform the run time testing, it is highly important to keep the system running in its normal operation while the test application is running. Priority of the tasks have to be assigned and they must be scheduled appropriately.

Assumption 2: Separation of application hardware and the test system hardware is only possible if the application consists of high processing power hardware (e.g. Vehicle). Else, there has to be an overlap between the application system hardware and the test system hardware i.e. both application and test codes can be deployed on same hardware chips. We need to consider this assumption due to hardware restrictions and variations according to application. As shown in the Figure 4.1a, the architecture is specific for high processing hardware units like vehicles. Hence, redundant LPU can be added as an extra hardware chip. On the other side, in Figure 4.1b the overlap between two applications is inevitable due to low hardware processing power. Single hardware chip is used. Above mentioned constraints are highly important from the architectural design point of view.

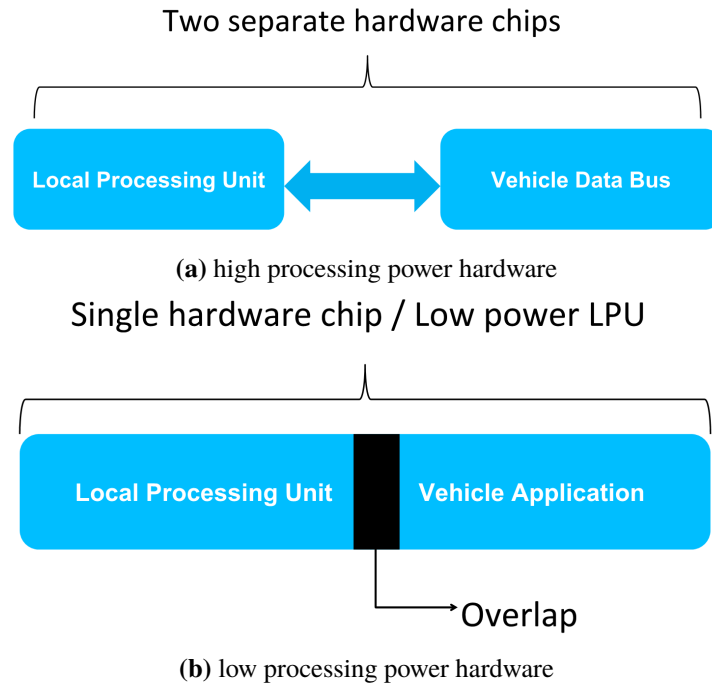


Figure 4.1: Application with Different Processing Power Hardware

4.2 Architecture Concept Overview

The aim is to provide testing functionality for the mobile CPS to assure safety during the run time. Initial approach for this system was based on client-server pattern. The main idea behind the system was to divide the whole functionality into normal operation and testing operation. We have started with two possible testing scenarios in mind. We want our system as much as free from all

Data Monitoring	Testing
Local	Local
Local	Remote

Table 4.1: Initial Test Strategy

constraints related to language/platform etc. This means, it should be easily scalable and flexible as well. Hence, we decided to write test cases and convert them to .hex file (byte files) and perform testing using these '.hex' byte files. These byte files are smaller and easy to handle in an application. This approach removes system dependency from any particular programming language/platform up to some extent. Now the system is generic in terms of test cases and can be used with any SW language/ HW platform combinations. Testing operation is based on two major system blocks i.e. Test Manager (TM) and Client Test Manager (CTM). We were focusing on including client test manager in the vehicle itself as shown in the figure below. The main idea behind test manager was a complete testing dedicated program deployed in the laptop. Architecture conceptualized mainly of two major components i.e. TM and CTM. Initially we tried with Local-Local (Table 4.1)

approach. Test cases (.hex) files were in TM and we established a serial communication with CTM. The major disadvantage of this approach was that the whole application code needs to be modified in order to make CTM communicate with TM. Communication and testing needs to be handled in the application code itself. This is exactly what we were expecting in our study in terms of system scalability and flexibility. Hence, we decided to opt for more redundant and powerful hardware.

4.2.1 Abstract Overview

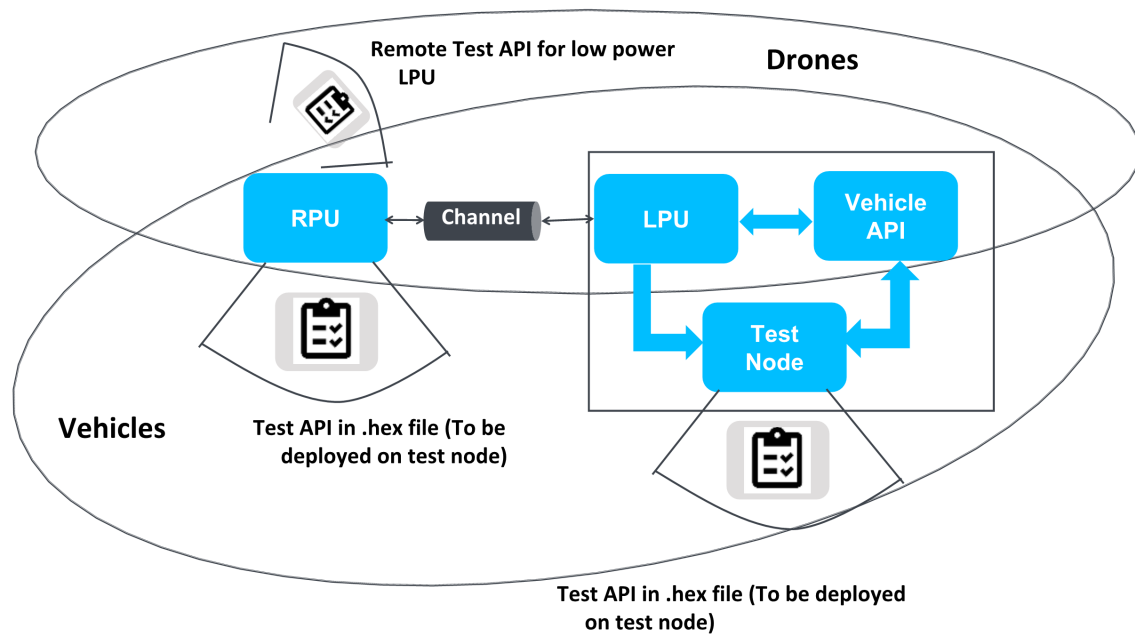


Figure 4.2: System Proposal with Concept at Glance

As depicted in Figure 4.2, there are two circles who define whole architecture concept with assumptions. These circles represent space of vehicle and drone i.e. hardware with high processing power and hardware with low processing power. This actually distinguishes between two approaches from the architectural perspective. The focus here is on both i.e. local testing and remote testing. From architecture point of view, test node block exists only in local test approach and hence not in the space of both the spheres. For local testing approach, consider vehicle space with LPU, Application interface and Test Node. This space also contains channel and Remote Processing Unit (RPU) only if new test case is needed from the manufacturer or test engineer. Testing will be performed on test node. .hex i.e. test file on RPU will only be sent via network to repository in LPU on demand. For remote testing approach, we need to consider drone space. No test node i.e. redundant hardware is involved. LPU and the vehicle application can be on the same hardware chip due to low power and capacity of the whole system. Overlap might exist between these two different applications. Channel plays an important role in sending message from LPU to RPU. Message might contain either command, data or result depends on the socket topic. In this case, complete testing has to be performed in RPU. Hence RPU must be equipped with API. No test case files needed separately. API in RPU will take care of the complete testing procedure. According to the assumptions, as the

hardware processing power reduces, we have to consider overlapping of LPU and the application code. This overlap is in terms of hardware, i.e. a single hardware chip would be in use. Original application must have an interface to communicate with LPU.

4.2.2 Constraints - Overview

To develop a run time testing system, we need to decide fix test points. Test points can answer one important question which we have already mentioned in our research questions (Section 4.1) – i.e. When? When to deploy test case?(local approach) or When to perform testing?(remote approach). We define test points on basis of ‘Constraints’. There are two types of constraints we are using – Monitoring constraints and Interrupting constraints. Monitoring constraints activate test mode along with normal operation and interrupting constraints are used in test case execution.

Monitoring Constraints

Monitoring constraints decide whether system should enter test mode or not. These constraints differ for each test case. Once the user selects particular test case, monitoring constrains for that test case is fed to the system. Via continuous monitoring, system checks if these constraints get satisfied and if so, system proceeds to the test mode. If not satisfied, system remains in normal operation mode notifying user to satisfy constraints first. since the monitoring constraints decide whether the test mode should be started or not, they are a part of the LPU in the system architecture.

Interrupting Constraints

Interrupting constraints can be monitoring constraints and additional test constraints or depends on the test cases. Once test mode gets started, during testing user has to think about test interrupting factors. Test can get interrupted due to various factors other than system failure. Test getting interrupted due to factors other than system failure should not be considered as failed test. User should get notification of the test interruption due to ‘Safety Guards’ and test decision should get modified accordingly. One can consider this test interruption due to safety guards as idle time and can try to eliminate this time from the final test result. This may impact total test decision or at least can be used as a user notification. Interrupting constraints are part of test cases and hence exists at location where testing is performed. Interrupting constraints consists of safety guards as well as they are safety guards for respective test case. User should not get confused between test fail due to system malfunction and test fail due to violation of safety guards (interrupting constraints). If there is a test fail due to system malfunction, our test system must notify the user about the same. If there is a test fail due to safety guard violation, our system must inform our user that there is no problem with the system functioning, and hence should be adviced for re-testing.

Safety Guards

There are some safety guards for monitoring constraints. These are nothing but constraints which make whole system fail safe. If safety guards get violated, system has to stop its operation and enter in fail safe mode. Example of safety guard can be an obstacle avoidance in autonomous cars. If the car detects a person in front it should stop immediately. We have such safety guards in our system and any violation of them must be given high priority in test or normal operation mode.

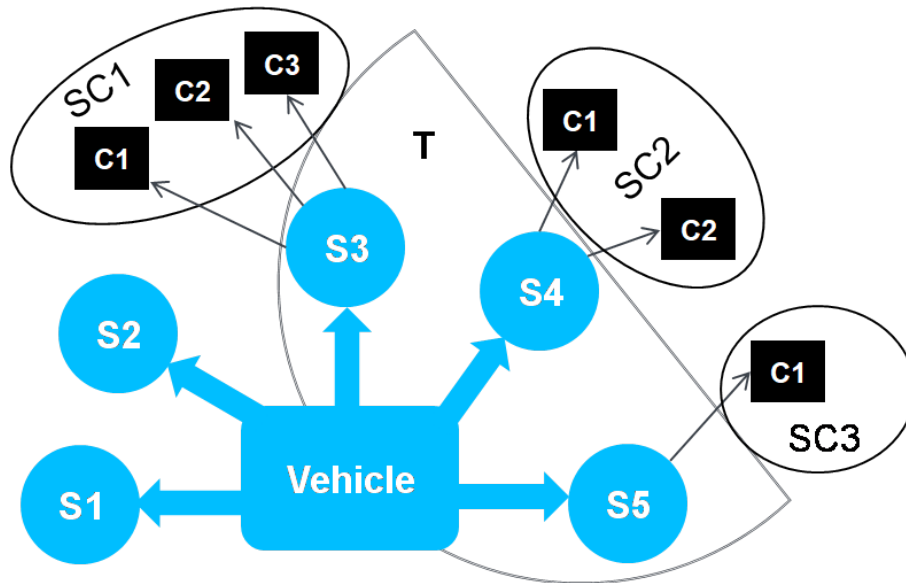


Figure 4.3: Constraints Modelling

4.2.3 Constraints - Modelling

Figure 4.3 implies that, to perform test case T for vehicle with five sensors (S1,S2,S3,S4 and S5), we need to take care of constraints for each sensor which are test case dependent. Explaining further, to perform test case T, we need three test constraints C to satisfy (C1,C2,C3) for sensor S3, two (C1,C2) for sensor S4 and one (C1) for sensor S5. Constraint for a sensor is the value of that sensor within predefined boundary condition. We need to group these constraint according to sensor i.e SC1 for S3, SC2 for S4 and SC3 for S5. To perform test case T, SC1 SC2 and SC3 must be satisfied. The complete modelling is applicable to monitoring constraints as well as interrupting constraints.

4.3 Architecture Modules

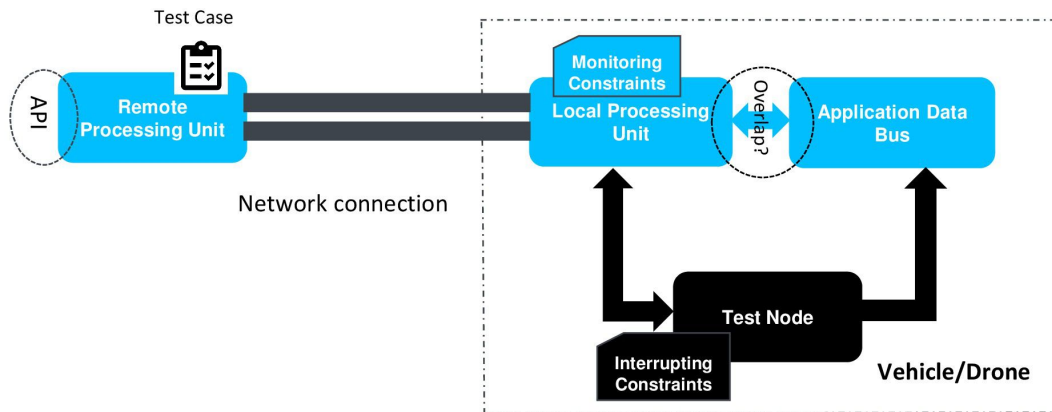


Figure 4.4: System Architecture Modules

Figure 4.4 gives the overview of the architectural concept of the run time test system. This architecture is flexible and could be modified according to the applications need. The basic building blocks of our architecture are: Local Processing Unit (LPU), Remote Processing Unit (RPU) and Test Node.

Application Data Bus

Application data bus is responsible for interfacing between test system and original application. In real scenarios it can be CAN bus with on board diagnosis (OBD) connector to get data from all vehicle sensors. It is mainly an interface to the original system.

Local Processing Unit

Local processing unit is mainly responsible for the complete test application. This is an additional hardware to be added in the system. LPU and application interface with data bus are two separate hardware chips in case of local testing approach. For remote testing approach with low processing power hardware applications, LPU and application interface are on the same hardware chip and hence the overlap with original application code is inevitable. No external communication between LPU and application interface exists. They will be a part of the same code separated by an interface.

Remote Processing Unit

While the local testing approach, Remote Processing Unit (RPU) is only responsible to add new test case file on demand in local repository of LPU. Further testing will be performed by LPU locally. In remote testing approach, RPU is a server and Mobile CPS behaves as a client. The whole system will be considered as a distributed system communicating via messaging. During this approach, RPU is also responsible to perform testing i.e. test cases will be written on RPU. No test case .hex file is needed for remote testing approach. Network related issues needs to be addressed while performing remote testing. To perform a test on the server i.e. RPU, it needs to have a complete API of the application. Only then test cases can be written and testing can be performed remotely. If the application changes, API needs to be changed at the RPU front without interrupting the logic of RPU. For Local testing approach, no API needs to be written in RPU.

Test Node

Test node exists only in local testing approach. Higher the processing power of the application hardware more will be the redundancy and less involvement of the network. Test node is low power, low capacity redundant hardware which is only responsible for testing purpose i.e. test case .hex file can be flashed over it and it can perform testing along with LPU. Once the testing is done, test node must go to the idle state to avoid any interference during normal system operation.

Communication Medium

For local testing approach, we must follow wired communication approach. Network involvement is not required as the system is directly approachable for testing purpose. For remote testing approach, client and server are communicating over the network. All network related issues are to be taken into consideration. Since, our system works on communication using messaging approach, appropriate message oriented middleware (MOM) needs to be used.

4.4 Local vs. Remote

By considering both the assumptions (Section 4.1) and basic implementation strategies, we have decided the concept of our system architecture. Main focus points are Local and Remote Testing.

Local Testing Approach

Network is full of latencies. It is unreliable and brittle. If possible, it's better to avoid network involvement in the application communication. Hence, we have come up with an approach of Local Testing. Basic idea behind local testing is monitoring data from mobile CPS locally and performing testing on local hardware of the system itself. Important factor to be considered here is 'run time' testing. There is no possibility of the system shut down. That's because the system is in operating mode and one cannot shut down the system abruptly just for testing purpose. Moreover, application with only the test mode is not feasible due to run time testing. Normal system inputs to be used as test inputs. But it is impossible to interrupt normal operation of the system and hence standalone hardware is needed for handling testing application. This architecture block is called as Local Processing Unit (LPU). LPU must interact with the system main hardware. Interaction with system hardware needs to be done via an interface to application code. No overlapping or modification in application code is acceptable. System data is needed for monitoring purpose is taken from CAN bus interface which is most common in vehicle system and to be used as test system inputs. Once the test mode is confirmed, test case is deployed on test node i.e. a redundant hardware dedicated to test case deployment in local testing approach. Within Local test approach, a provision can be made in architecture to connect test system to manufacturer server and request for new test case file over the network if needed. But final testing has to be performed locally on test node.

Remote Testing Approach

According to assumption we mentioned in Section 4.1, as the processing power of mobile CPS hardware goes on decreasing, it becomes impossible to add hardware redundancy. Consider case of small drones. Adding additional hardware for LPU in such small system is impossible. If LPU cannot be added on additional hardware, then it must exist on existing system hardware itself. In this scenario, it will be difficult to perform testing locally since heavy modifications in original application code are needed. Hence, our architecture facilitates remote testing approach. Remote testing approach makes network involvement inevitable. Monitoring is performed locally and testing

performed on remote server. Redundant test node hardware is not required since deployment of test cases does not exist. Testing is done at the server end, whereas the data monitoring process is done at the client end i.e. application end (vehicle, drone etc).

4.5 Local Test System Configuration

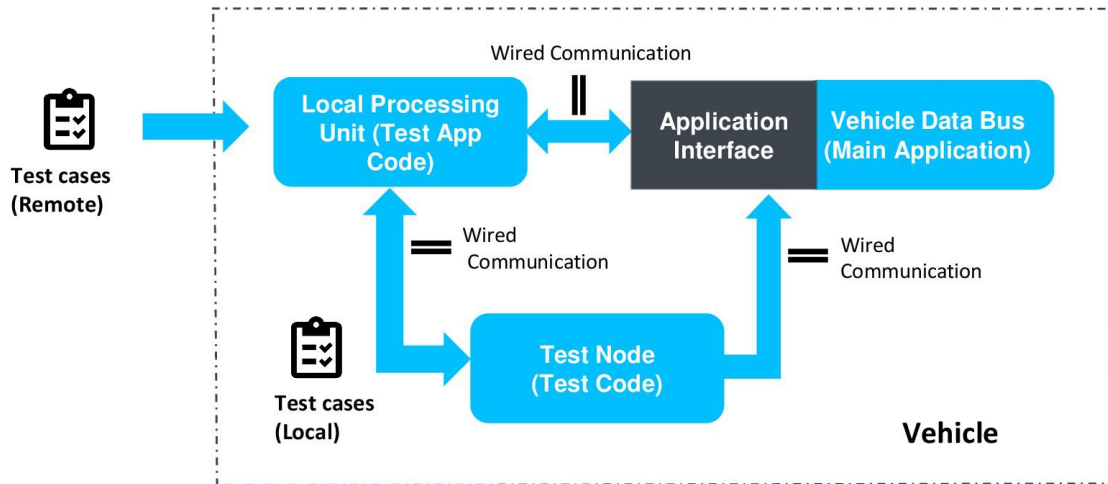


Figure 4.5: Local Test System Configuration

Figure 4.5 shows local test system configuration. Here LPU and test node lies inside the vehicle. Remote connection to server is established only to get new test case from the server. All test cases stored inside LPU repository. Vehicle data bus i.e. main application is having application interface. This application interface allows LPU to connect to vehicle data bus. Application interface is mandatory since we do not need to interfere in original application. Test node is needed to flash test case code i.e. .hex files. This test mode is needed to avoid flashing of test cases on the system controller on which system operation is running. Communication exists between LPU and vehicle data bus, LPU and test node and test node and vehicle data bus. All communication lines are wired communication lines.

4.6 Remote Test System Configuration

Figure 4.6 Shows remote system configuration. For remote system, we need to focus on client server architecture. As explained before, we are considering client as small size drones. A server is a Remote processing unit. Original application must have an interface to interact with LPU. LPU and main application have wired communication in between. RPU server and LPU client are communicating via wireless communication. This test application is distributed in nature and hence have messaging as a communication medium. Both local and remote applications exchange data and commands via messages. To communicate via messaging, both applications need to be connected via a channel.

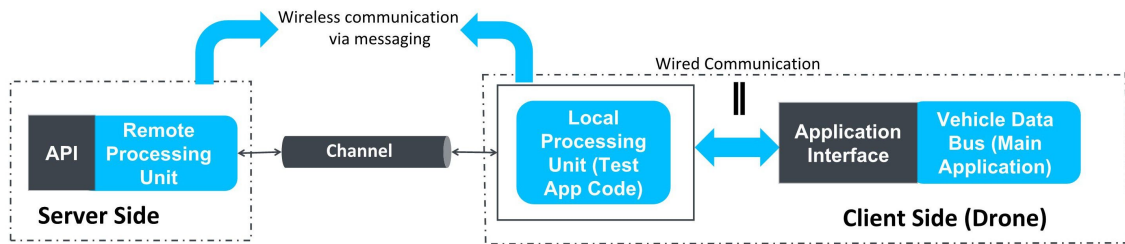


Figure 4.6: Remote Test System Configuration

As shown in Figure 4.6 an API is used at the server side unit i.e. RPU. This is needed since testing is performed within RPU. Without API, it is not possible for RPU to decode messages received from original application. No test cases are needed and hence no deployment on any redundant hardware. Hence, this approach is ideal in case of low power hardware devices like drones, where system is not capable of adding extra hardware. Also, there is a possibility of overlap between LPU logic and original application logic if single hardware chip is used. One important thing to notice is, LPU logic will always be the same in our architecture, any change in original application will result in change at API in RPU. But LPU and RPU i.e. major blocks of our architecture are stable.

4.7 Messaging Channels

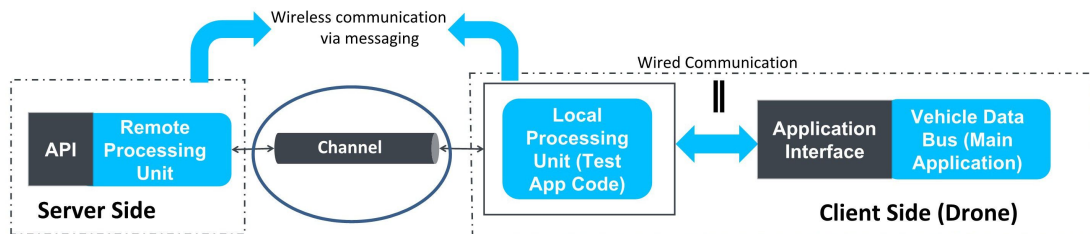


Figure 4.7: Messaging Channel

Important use of channel is to introduce two unknown applications. We are going to introduce LPU and RPU via channel (Figure 4.7). LPU and RPU are loosely coupled. No identification is needed to be sent and receive data from LPU or RPU over the channel. Channel just acts as an address for the data to be placed. We are using publish-subscribe channel type. In pub-sub pattern publisher publishes data to all subscribed clients. So pub sub channel has one inlet and multiple outlets. Each outlet will be having same data to be paced by channel. Pub-Sub channels handle event messages [HW04]. We have three types of events defined i.e. command event, data event and result event. The detailed explanation will be provided for all these mentioned events.

4.8 Message Interaction

For remote testing approach, we need to establish communication between applications via messaging. Application defines endpoints i.e. sockets for messages. We are using PUB-SUB socket type. Figure 4.8 shows socket structure in our test system. Each of them is pub-sub type. We have

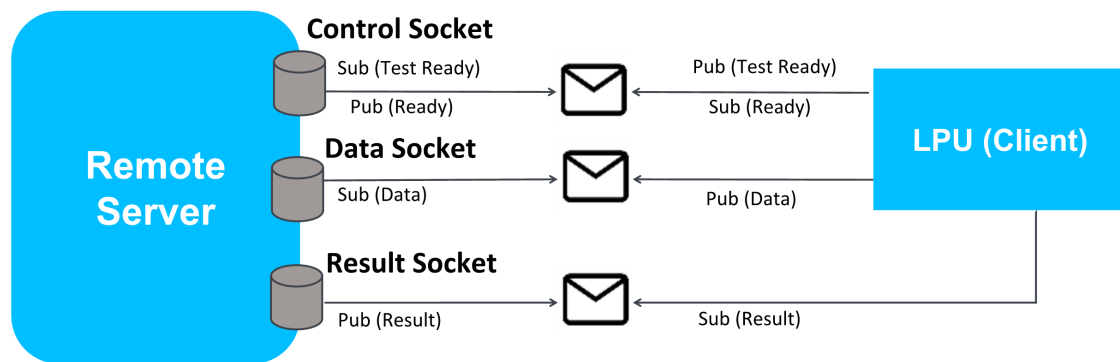


Figure 4.8: Socket Structure

three endpoints in our system. Remote server communicates with LPU. LPU i.e. vehicle connects to sockets bonded on remote server. Basic structure consists of Control, Data and Result Endpoint. Via control endpoint, LPU initiates communication to remote server. It publishes the command to start testing. Server subscribes to the control topic and initiates server ready acknowledgement. Server publishes this acknowledgement via control topic which gets subscribed by LPU. Once test communication sets up, i.e. when LPU gets acknowledgement, it starts publishing data via data socket. Server subscribes to this continuous data stream via Data topic. This data stream is used to monitor vehicle behaviour and also to perform testing via RPU i.e. server. Once testing is finished, server publishes result via result topic which gets subscribed by LPU. The result may be test fail or test successful. Once result get published by result topic, server stops testing procedure and LPU starts data monitoring again till user requests for new testing routine.

4.9 System Behaviour - BPMN

Figure 4.9 depicts local test system behaviour. Local testing works with three distinct hardware processes i.e. LPU, Vehicle and Test Node. User once started monitoring, vehicle will keep providing data to LPU. LPU needs test case to be selected by user. Depending on test case, LPU keep checking for monitoring constraints satisfaction. Once constraints are satisfied, it will invoke test node and start testing process. If testing gets interrupted, test node informs LPU and wait in Interrupt Service Routine (ISR) till interrupting constraints get satisfied by LPU. After testing, Test node will inform results to the user via LPU. Figure 4.10 depicts remote test system behaviour. Remote test system has LPU, vehicle and remote server as distinct processes. The major difference in remote and local approach is connection with remote server. LPU connects to remote server

within specified time. If timeout exceeds, user will get notified. Vehicle provides data to remote server since testing is performed at remote server. Interruption and constraint based approach is same as in case of local test system. Remote server informs user about test result via LPU.

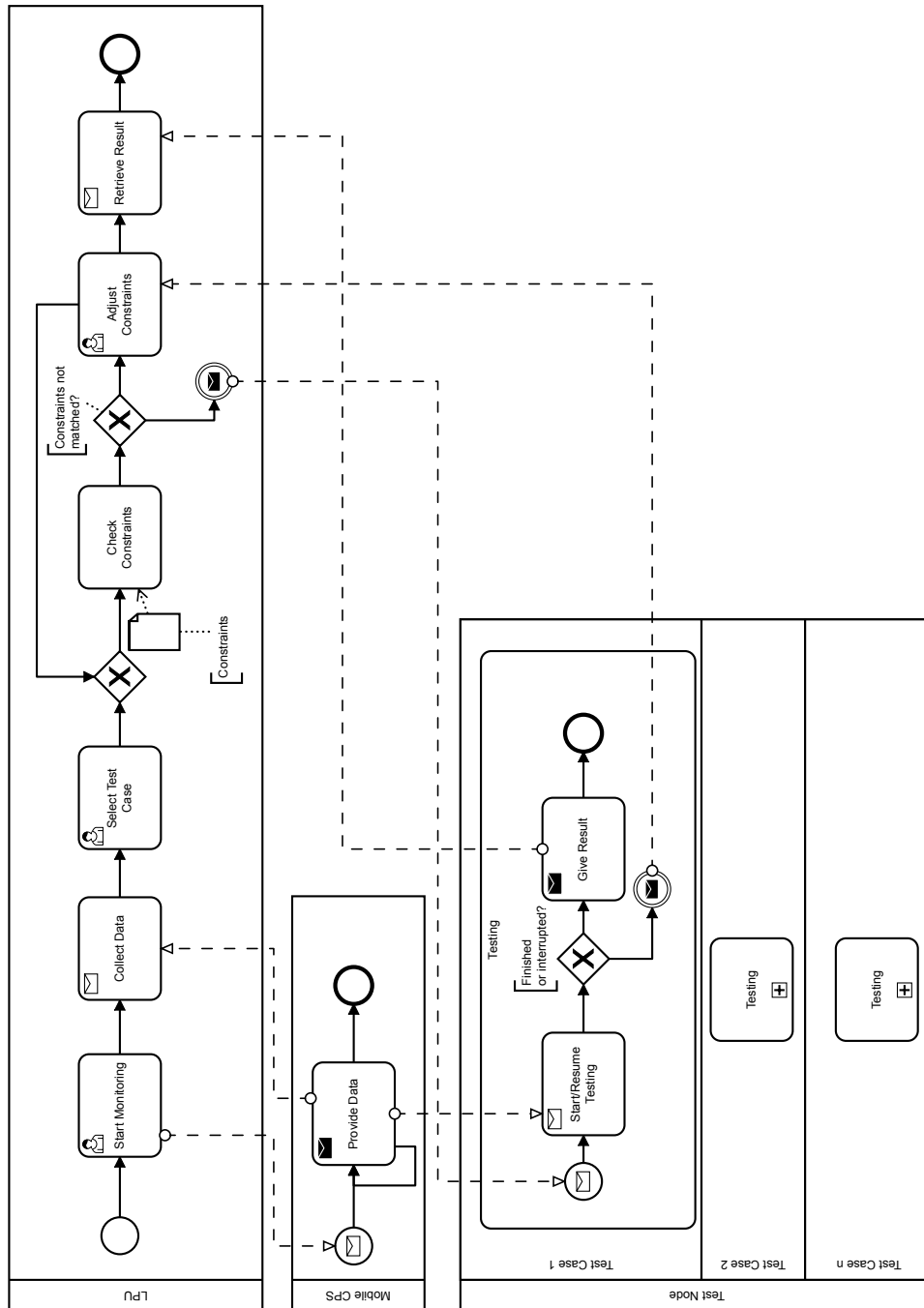


Figure 4.9: Local Test behaviour

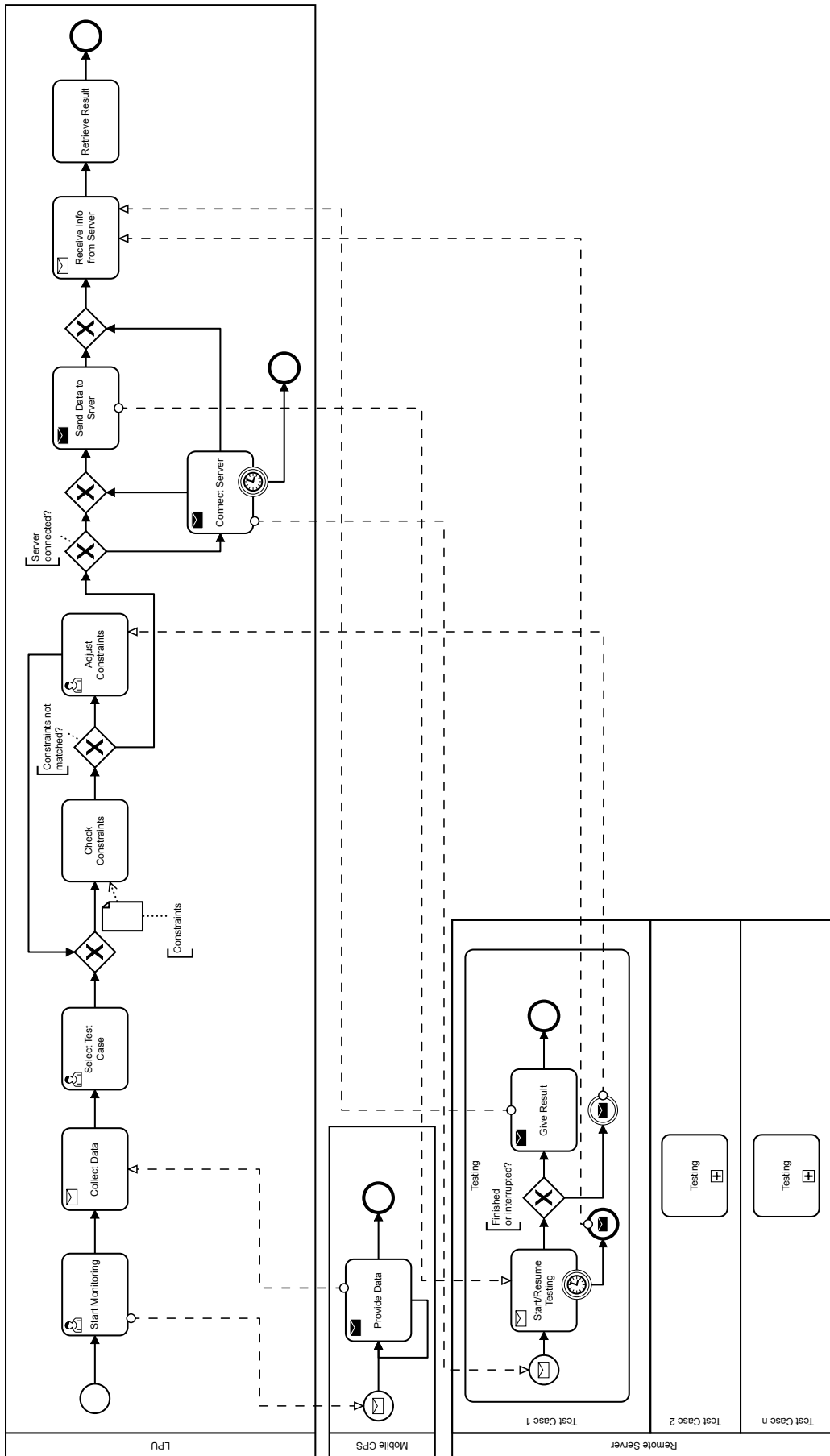


Figure 4.10: Remote Test behaviour

between two concepts. The big circle denotes a vehicle in the local testing approach. Here, we have high processing power hardware and hence, we can use redundant test hardware i.e. arduino, also a separate local processing unit i.e. Raspberry Pi. We have also provided remote connectivity to server i.e. a laptop via Zero Message Queue MOM. The smaller circle denotes low processing power hardware for eg. a drone. As stated low processing power, it is impossible to add an extra redundant hardware test node i.e. arduino. Hence remote testing is inevitable. In our prototype, this approach architecture consists of Raspberry Pi as LPU, mBot as low power hardware mobile CPS and a laptop as remote Server to perform testing. Laptop contains all test case code and application API needed to perform testing.

5.2 Hardware and Software

We are supporting our proof of concept by implementing a system prototype. We have decided to implement this prototype with a robotic vehicle as a mobile CPS and off the shelf components. As a mobile CPS, we are going to use robotic vehicle mBot from makeblock [MAK] (Figure 5.2). MBot is an education robot kit based of Atmega328p micro-controller. It has mcore as main control

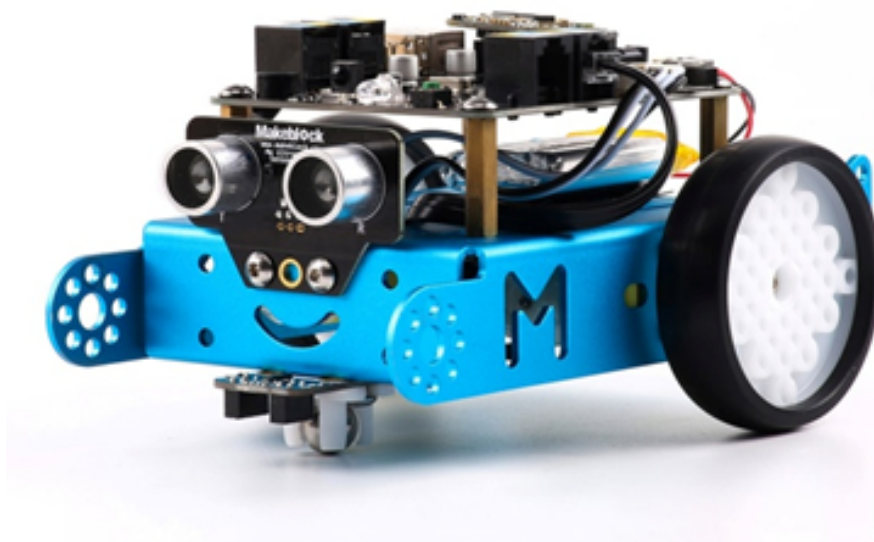
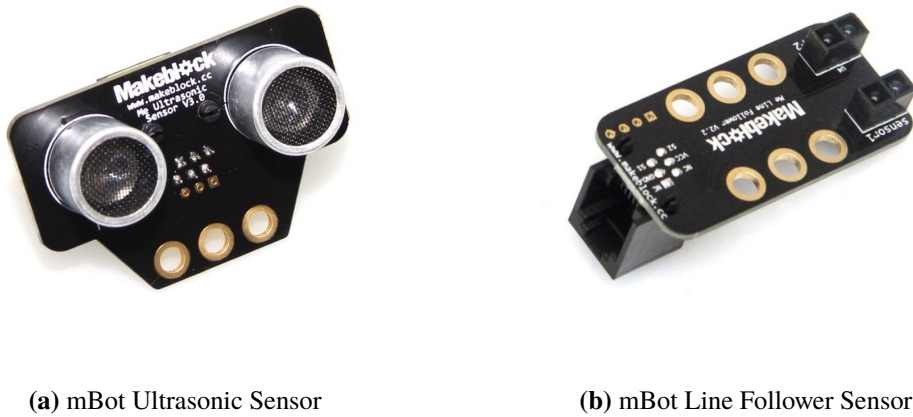


Figure 5.2: mBot from Makeblock [MAK]

board. Mcore board has four expansion slots for various sensors. Two of them are already occupied by ultrasonic sensor and line follower sensor provided by mBot manufacturers. Other slots can be used as further extension slots. This tiny ultrasonic sensor is used to measure distance and avoid obstacles. It has RJ425 connector for easy interface with mcore main board (Figure 5.3a). We are using this sensor as one of the real time vehicle sensor in our implementation. Another sensor which we are going to include in our use case is mBot line follower sensor (Figure 5.3b). Mblock line follower module is compatible with arduino IDE and mCore main controller with RJ25 connector. MBot is an educational kit and to use it as CPS prototype, we need to strengthen its capabilities to transform it to CPS. According to Crenshaw et al., A CPS test bed must have network control. CPS must show enforceable physical properties and they must be composed of off the shelf components.



(a) mBot Ultrasonic Sensor

(b) mBot Line Follower Sensor

Figure 5.3: mBot Sensors [MAK]

Hence, we have decided to expand mBot capabilities with a redundant hardware Raspberry Pi 3 B+ as shown in the Figure 5.4. Raspberry pi 3 B+ is high power processor with WiFi/Bluetooth and wired connectivity. It is an off the shelf component in embedded domain and have easy interface with atmega micro-controllers which is main board of mBot [RPI]. Raspberry pi 3 B+ is the perfect choice to use for strengthening capabilities of mBot and use it as a mobile CPS prototype. Raspberry

**Figure 5.4:** Raspberry Pi 3 B+ Module [RPI]

pi (RPI) preferred as Local Processing Unit (LPU) in our architecture. There are several reasons behind the choice we made. One of the most important reason is processing power. RPi has its own processor i.e. Broadcom BCM2837B0 quad-core A53 (ARMv8). It has several possibilities to communicate with vehicle hardware interface i.e. CAN or K-Serial lines. RPi supports bidirectional communication. Hence, data can be sent in and out from the vehicle. It supports multiple common interfaces i.e. I2C, USB, SPI etc. We are prototyping a distributed system and hence, we need to implement remote server communication. RPi 3 B+ module supports wireless connectivity with onboard WiFi module. As explained in Section 4.2, our conceptual architecture as two important aspects i.e. Local testing and Remote testing. For Local testing approach, we have decided to use .hex files as test case files. This approach has been followed to avoid any specific platform or

language dependency. Our architecture is completely flexible and hence we decided to maintain this flexibility with test cases too. Moreover, .hex i.e. byte files are small in size and easy to handle in an application.

In local test approach, to avoid interfere of our test system with original application code, we have decided to add another redundant hardware i.e. a test node. This hardware must be capable of deploying test case byte files and to perform testing by communication with main controller and LPU during run time. Our choice for the same is Arduino uno microcontroller board (Figure 5.5) [ARD]. We made this choice since arduino Uno has same processor Atmega 328p as mBot Mcore



Figure 5.5: Arduino Uno Microcontroller [ARD]

hardware. Hence the interface between mBot and Arduino will no longer be a challenge. Our idea is, this redundant hardware must not be as powerful as main system hardware since its job is to perform testing when demanded and then remain to be in power down mode to avoid any interference.

For second concept of our architecture i.e. remote testing concept, we are using messaging as a communication medium. We have already explained about messaging as a communication medium for distributed systems in Section 2.7 of this master thesis. To communicate with remote server via messaging, we need a message oriented middleware (MOM) as explained in Section 2.8. In our prototype, we are using Zero Message Queue (ZeroMQ) as MOM. Choice of ZeroMQ has been made since it is a simple python library. It provides many features along with normal socket connection and it is a dedicated MOM. For more details about zeroMQ, please refer to Section 2.10 of this master thesis report.

In remote testing approach, we need to set RPU. LPU and RPU will communicate with each other and perform testing as well (Figure 5.6). As RPU, we are using a laptop which is a stable remote server. We are writing logic for RPU in python language. In remote test approach, we do not need to use .hex file and redundant hardware node i.e. arduino uno. Complete testing will be performed at RPU and test cases will be written in python i.e. same digital logic programming language as RPU. Considering flexibility of the architecture concept, the components, devices and programming

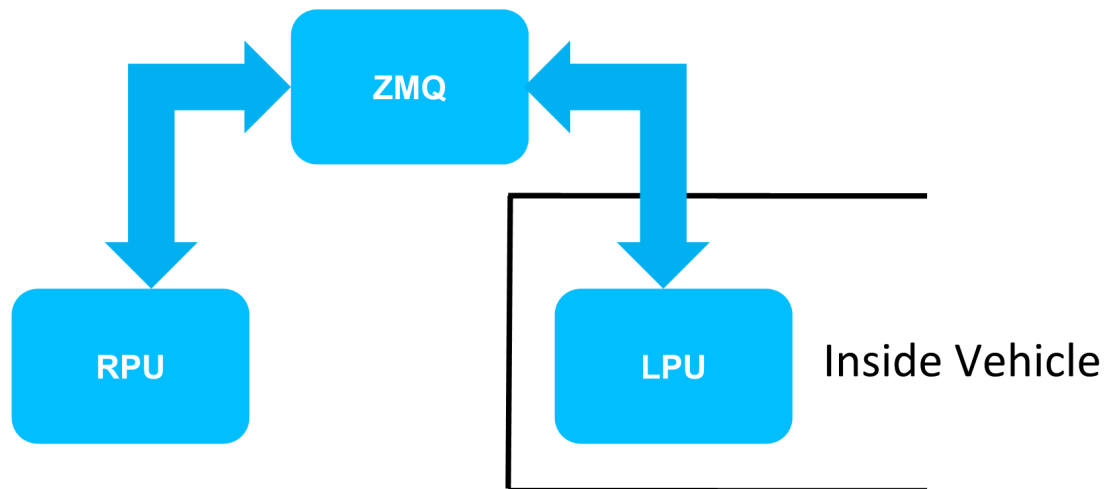


Figure 5.6: Distributed System Implementation Outline

languages we used here in our validation, are need not be the same to replicate our system. We have chosen these components and methods for our consideration and to provide proof of concept in this master thesis. Our architecture can be realized using any mobile CPS with LPU and RPU devices. Also, distributed system can be realized using any middleware. RPU logic need not be written in python only and can be in any other language with test case code.

5.3 Local Testing Approach

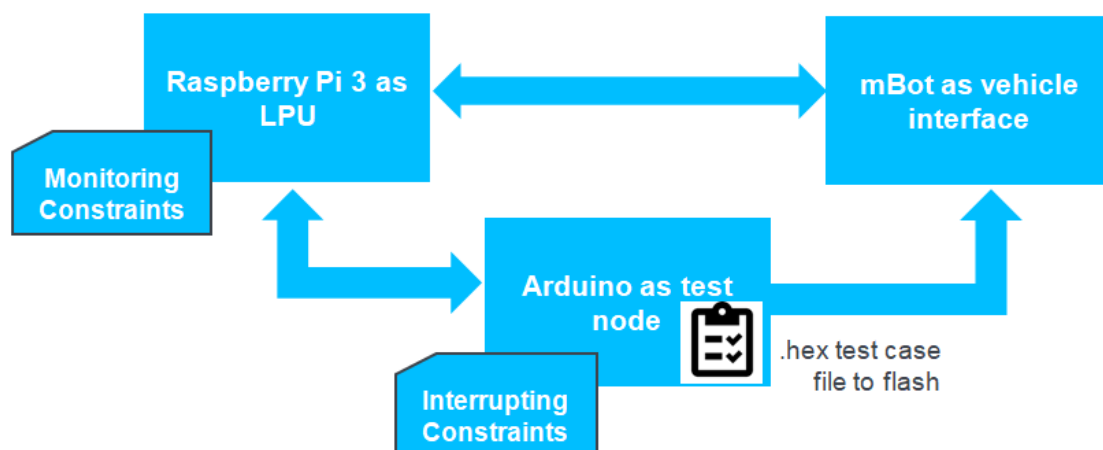


Figure 5.7: Local Test System Approach

Figure 5.7 explains local testing approach. As main architecture block LPU, we are using Raspberry pi 3 B+ module. mBot is vehicle prototype. We wrote an interface for communication of Raspberry pi with mBot application. RPi and mBot are communicating via serial USB communication. It is a wired communication interface.

In local approach, we are adding functionality for remote connectivity to receive new test cases from manufacturers or test engineers. Hence, we are using Zero Message Queue (ZMQ) as MOM. Our server for new test case generation and provision is on Laptop. Laptop and raspberry pi are connected via ZMQ MOM. We send .hex test case file from laptop to Raspberry Pi via ZMQ. Raspberry pi as LPU, responsible to deploy this test file on redundant test hardware arduino in test mode. Arduino further perform run time testing with mBot. The test mode enabling decision must be taken by logic implemented in LPU. Hence, LPU always monitors test constraints to be satisfied during vehicle normal operation. When these constraints get satisfied, LPU informs user about test mode enable and start deploying selected test case on redundant hardware. Hence, monitoring constraints are implemented in a .xml file which exists in LPU.

Once LPU enables test mode, redundant hardware i.e. arduino gets interrupt as wake up call. After wake up from power down mode, arduino receives .hex file from LPU repository and establish wired connection with mBot vehicle via interface logic we wrote. Arduino performs testing and gives feedback to LPU in real time. During testing, it is critical to determine the test interrupting constraints. What if safety guards gets violated? What if test halts/fails due to external environmental conditions and not because of vehicle internal fault? To handle these conditions, we have designed interrupting constraints with each test case along with general vehicle safety guards. These interrupting constraints and safety guards exists in Test node arduino and monitor behaviour of vehicle during testing operation. If these constraints gets violated, there altogether effect has been analyzed and final test result will be based on outcome of testing including this analysis with interrupting constraints. In our prototype, if interrupting constraints gets violated in local test approach, we halt the operation of redundant hardware till interrupt removal.

5.4 Remote Testing Approach

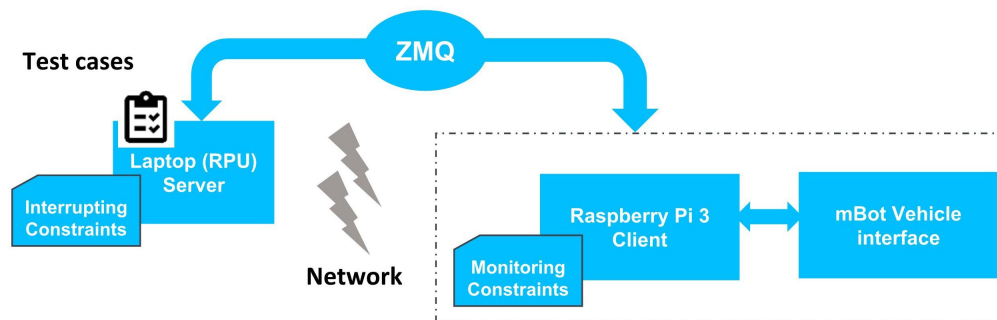


Figure 5.8: Remote Test System Approach

Figure 5.8 explains our second i.e. remote testing approach. As explained in Section 5.1, this approach is specifically essential for low processing power hardware mobile CPSs. Additional redundant hardware test node i.e. arduino does not exist. Hence, no deployment of test case .hex files is possible. Moreover, testing has to be performed remotely on a server. Hence it is inevitable to have API in remote server. In our prototype, we are using a laptop as a remote server and hence we are adding an API for mBot in the laptop. We have also written a test case code in the laptop server. We have used a specifically Python version 3.5.

Laptop i.e. RPU is communicating with Raspberry Pi i.e. LPU. Raspberry pi is communicating with mBot i.e. vehicle locally i.e. via wired communication interface. For remote communication between RPU and LPU, i.e. laptop and RPi, we are using communication via messaging approach. Communication established using WiFi connectivity. Zero MQ is used as message oriented middle ware in this prototype. RPi is responsible for handling monitoring constraints to enable test mode. This approach is aligned with local test approach. Once the test mode is enabled, communication establishes between laptop and Raspberry Pi using messages. Test case code exists in laptop and test commands conveyed to raspberry pi via publish and subscribe under the topic called "Command". Based on these commands, run time data of the vehicle provided back to laptop i.e. server using data topic. Server is responsible to perform testing on this data using internal logic and feed back result to RPi and notifies user about test result using result topic.

Safety guards and interrupting constraints must be monitored in remote testing approach too. These constraints exists in an .xml file as previous approach. This .xml file with interrupting constraints and safety guards exists on the remote server i.e. on laptop in case of our architecture prototype.

5.5 Prototype

Figure 5.9 displays our system prototype. Educational robotic kit mBot connected with RPi 3 B+ module and arduino. Complete system depicts prototype of mobile CPS. We have designed a User

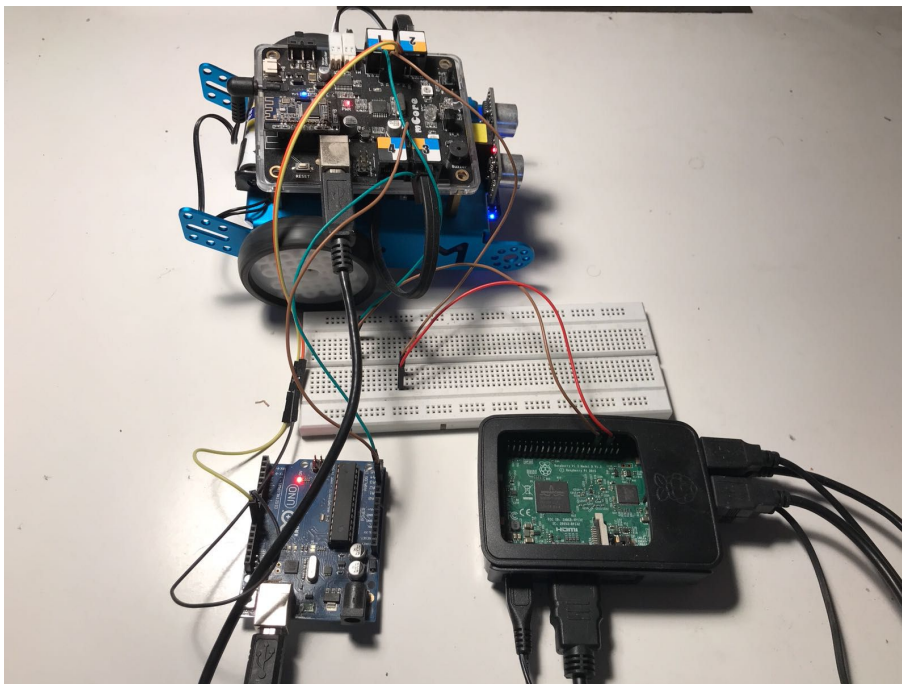


Figure 5.9: System Prototype

Interface (UI) for the system and developed developed an exemplary test case named T1. Consider the mBot i.e. vehicle in normal operation mode. User has a system UI shown in a display fitted inside the vehicle. The UI is as shown in the Figure 5.10. As mentioned before, UI is designed for one test case T1 with push button named T1. Further it can be improved with drop down or simple

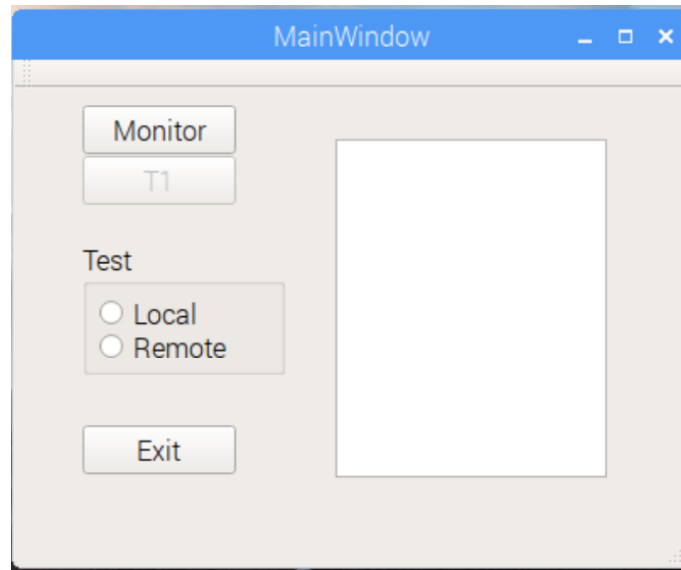


Figure 5.10: System UI

test box for the user to enter test name in case of multiple test cases. Before testing, user can simply monitor the real time vehicle data by selecting 'Monitor' option (Figure 5.11a). Once the real time monitoring starts, user can select test case to be tested. He can do this by selecting test case name option. Once the monitoring constraints are satisfied, system will prompt user about start of test mode. If monitoring constraints are not satisfied, system will tell user which constraints are not yet satisfied and boundary conditions for the constraints (Figure 5.11b). User has to bring his vehicle within this boundary range and satisfy constraints manually. Then system will enable testing mode. For prototype development, we are using only one CPS prototype i.e. mBot and hence we are offering local and remote testing approaches to the same. In test mode, therefore user has authority to select test mode to be local or remote (Figure 5.11c). This could be hard-coded if system does not support both approaches i.e. local and remote testing. Hereafter system will be in testing mode and keeps prompting user about test case results. If interrupting constraints get violated, system will be in ISR and will take ISR effect into the consideration for final test case evaluation. Effect of ISR on final test result is test case dependent and vary for each test case. User will be notified once the test is completed and outcome of the test can be observed on the display in the vehicle (Figure 5.11d).

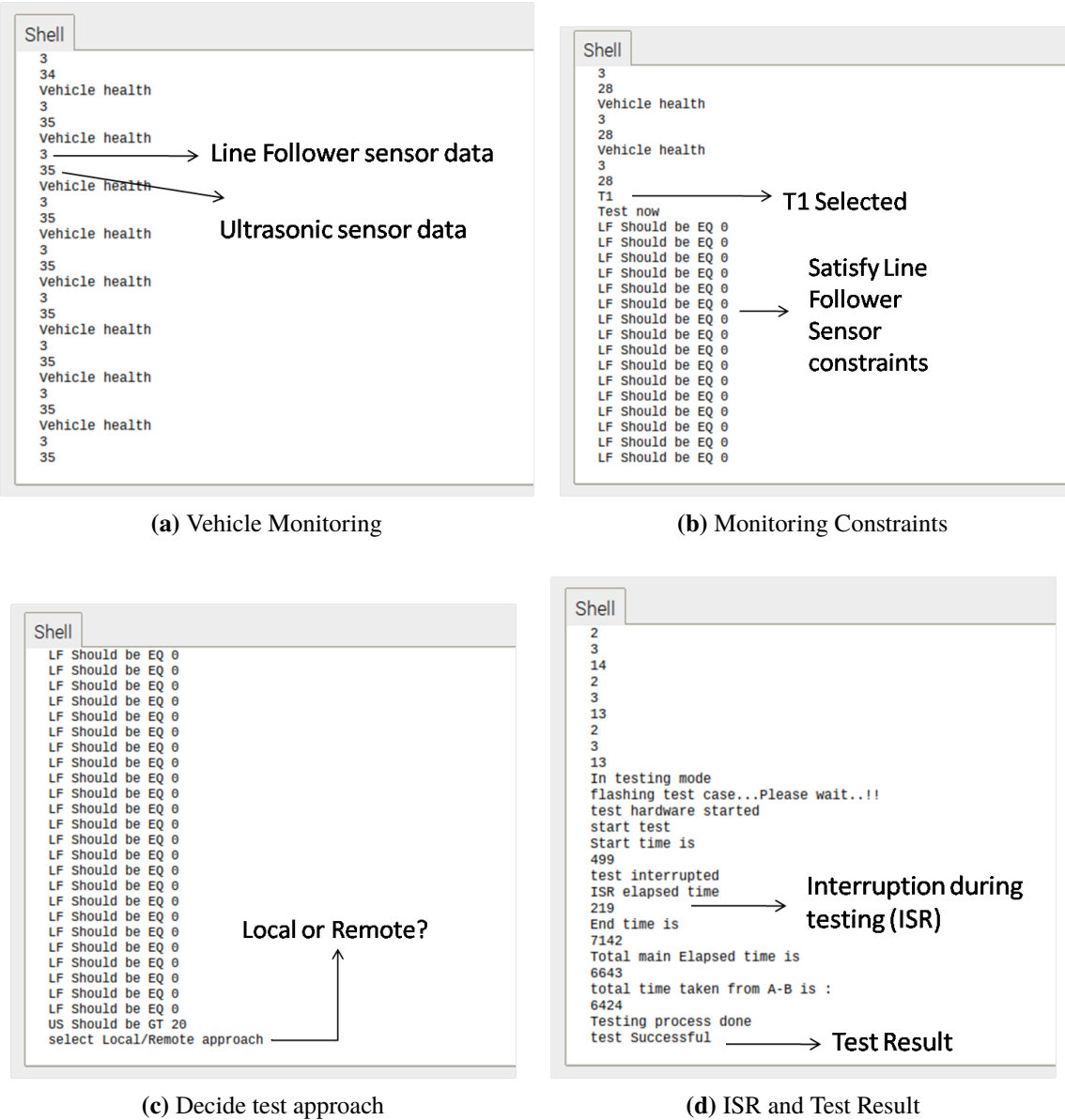


Figure 5.11: Complete system behaviour

5.6 System Flow Chart

Figure 5.12 describes the algorithm we used for our run time testing system. Whether the system is using local approach or remote approach, this architecture part is similar in both the cases. Once started, system will perform monitoring for all sensors data. In the mean time, user must select the test case he wants to use. System will check for constraints requirement fulfillment and keep updating user the current status. Once constraints are satisfied, it is user's choice to perform local or remote testing in our prototype. In future implementations, we can neglect this decision step and directly implement local or remote testing logic according to the type of CPS.

Figure 5.13 describes local test algorithm. Once user selects local test approach, system will deploy .hex file for the test case selected. Testing will be started on redundant node. If testing gets interrupted, system will be in ISR till interrupting constraints get satisfied. Once out of ISR, system will check for testing finished. After finishing the testing, system will update user with result.

Figure 5.14 depicts remote testing algorithm. If user selects remote testing approach, then system will try to establish connection with remote server. If this connection timed out, system will inform user about network issue. Once the connection established testing will be initiated. If testing gets interrupted, system will be in idle mode. When connection timed out occur, user will be informed and system will be shut down. If no time out occur and system resumes normal operation, testing will be continued and user will be updated with the test information.

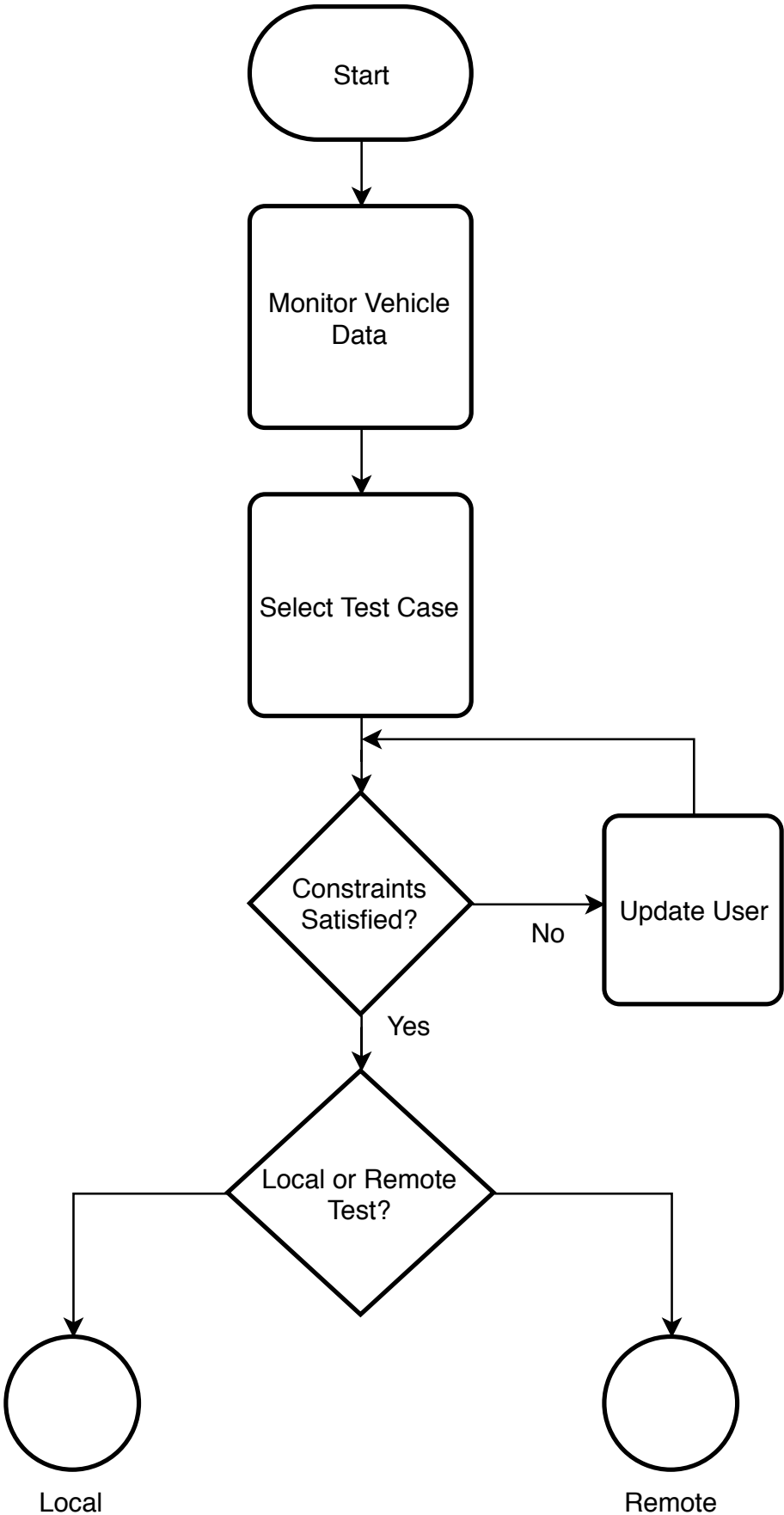


Figure 5.12: Monitoring Algorithm

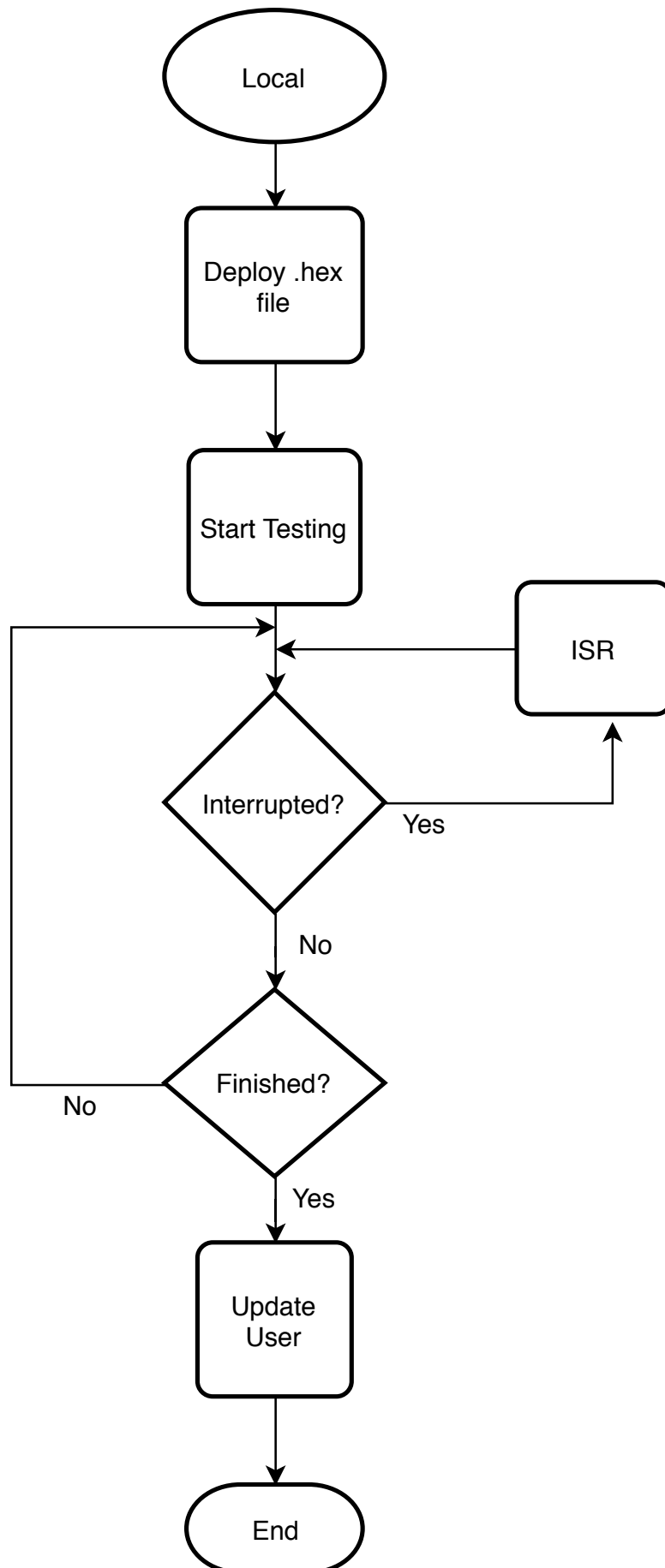


Figure 5.13: Local Approach Algorithm

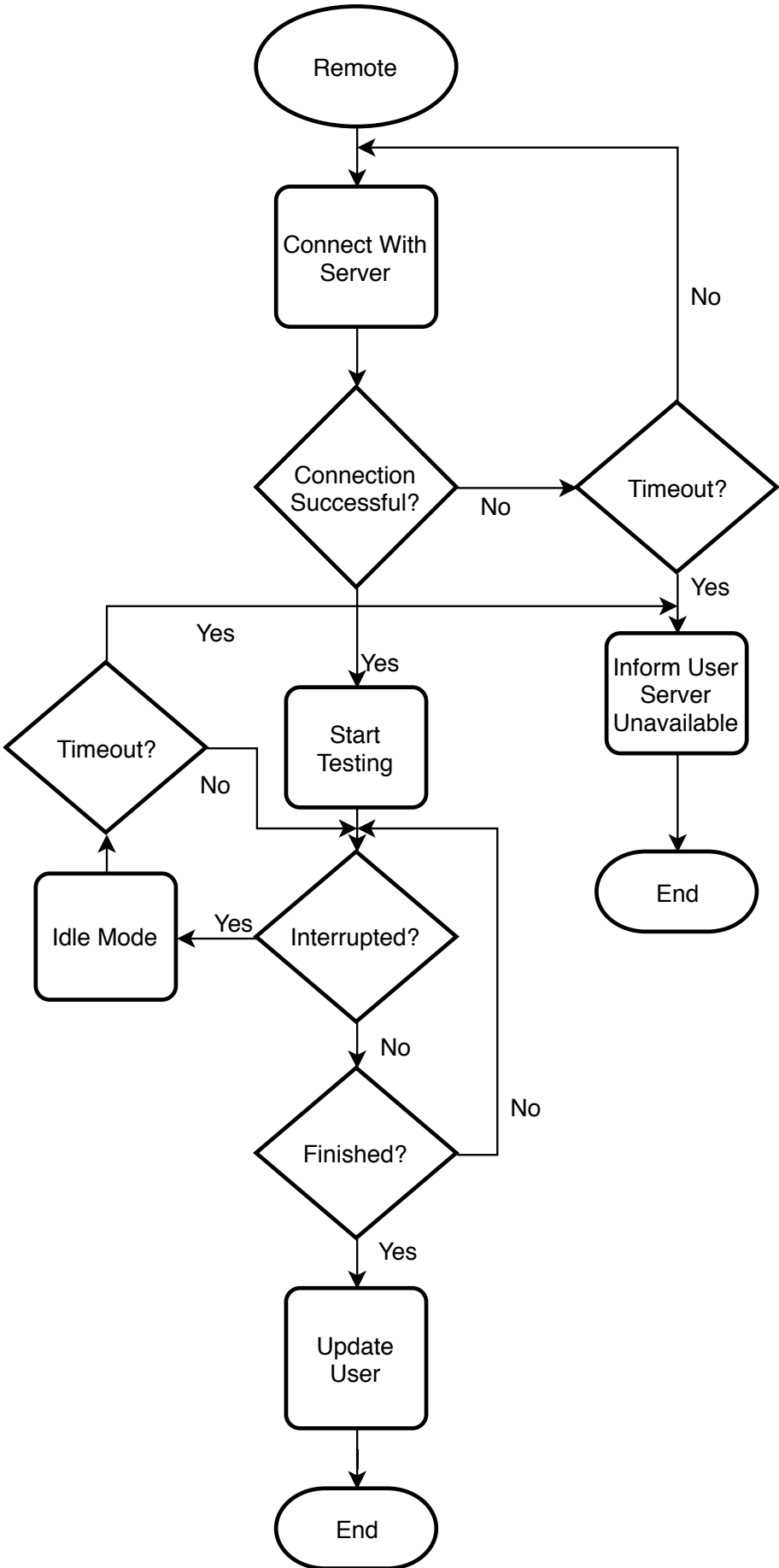


Figure 5.14: Remote Approach Algorithm

5.7 Use Case

To validate our implementation, we implemented an exemplary use case with remote testing approach.

Use Case: Vehicle should travel from point A to point B within 5 to 10 seconds (Figure 5.15).

mBot will start from a point A when line follower sensor detects black color block. It will travel along a white line till point B. At point B it will detect another black box and it must stop and return time elapsed value.



Figure 5.15: Use Case with mBot

For this use case we have to set prerequisite constraints i.e. monitoring constraints (See Table 5.1). Tc1 implies that no obstacle should exist in front of the vehicle. Tc2 defines detection of white line

Constraint Name	Related Sensor	Value
Tc1	Ultrasonic Sensor	>20
Tc2	Line Follower Sensor	=0

Table 5.1: Monitoring Constraints

by line follower sensor i.e. vehicle is not reached at point A. Once the testing is started, we should monitor it for interruptions which are external. Test should not fail i.e. vehicle should not return time value more than actually elapsed due to external conditions. Interrupting constraints are as shown in Table 5.2. Here Tc1 is the constraint for interruption by sudden obstacle appear in front of vehicle. Tc2 and Tc3 states interruption due to vehicle deviated from its path and not moving across white line between points A and B.

Constraint Name	Related Sensor	Value
Tc1	Ultrasonic Sensor	<=20
Tc2	Line Follower Sensor	=1
Tc3	Line Follower Sensor	=2

Table 5.2: Interrupting Constraints

Considering all above constraints we should get value of Elapsed time viz.
 Elapsed time = End Time – (Start time + ISR/Idle time)

5.8 System Analysis

We performed performance and feasibility analysis of our architecture and prototype using use case. Main observations of our system are as follows: As explained in Figure 5.16, we use continuous



Figure 5.16: Testing and Monitoring

monitoring approach while testing. This implies that monitoring and testing works hand in hand. Our system monitors the data, make assessments and take the decisions based on the real time analysis. System decisions are based on change in data in real time.

In this work, we completely focused on run time testing and hence used Built In Self Test (BIST) approach. This approach eliminates need of separate test time and facilitates testing during normal system operation. We developed this approach on basis of constraints i.e. preconditions for all sensors and actuators of a vehicle varying according to each test case. A simple .xml file is capable of handling this constraints approach. Our system architecture is completely flexible and application oriented. LPU and RPU logic will always remains the same for any variant of mobile CPS. Changes required in application interface and not in core logic of the system.

Due to network unreliability and brittleness, our system faced connectivity problems and communication problems. This problem observed in remote testing approach. We could managed to handle the network connection issue with the help of MOM ZMQ. We used its timeout property to verify the connection availability for specific time duration. After timeout, test system turns off gracefully informing user about network issue. As we are following BIST approach, even though testing gets terminated, normal system operation remains unaffected.

6 Conclusion and Outlook

In this master thesis we have described as well as demonstrated a testing system of the mobile cyber physical systems. The principal idea guiding this thesis is facilitating the testing of the mobile CPSs during run-time, by utilizing additional resources as redundant hardware and off the shelf components to conceptualize the idea. As detailed in Section 4.4, the system has been developed with two main approaches i.e. Local Testing and Remote Testing. Selection of the approach chiefly depends on the type of the CPS. As mentioned in our assumptions, hardware processing power of the CPS decides the approach to be selected. Our system consists of a testing architecture. The main building blocks of the architecture are primarily the Local Processing Unit (LPU) and the Remote Processing Unit (RPU). With regard to the local testing approach, we integrated additional test node hardware in the complete system. The idea of testing, proposed in this research is based on the constraint based testing during run-time. The focus of the thesis lays on two main constraints types i.e. monitoring constraints and interrupting constraints. Monitoring constraints decides whether the test mode is to be started or not, whereas the interrupting constraints are responsible to handle the interrupts in the testing mode. In order to implement the run time testing, we had no choice rather than to keep the system in continuous operation as testing cannot be performed in separate time slots. Hence, we decided to follow the built-in self-test (BIST) approach. In the BIST approach, we deployed the normal system inputs as test system inputs and performed the testing during normal system operation. This facilitates the testing phase to be completely independent than the normal system operation. However, this entire discussion is pointless if we do not consider the assumptions we made in Section 4.1. We have assumed that real-time OS i.e. RTOS is essential for this test system. The reason therefore being scheduling and the higher priority of the testing task which needs to be performed in parallel to the normal task. Another assumption would be to utilize the separate LPU for high processing power HW systems and overlapping the LPU logic with application code for low processing power HW systems. We propose to follow the local test approach for separate LPU whereas follow the remote test approach for merged LPU.

Finally, the system is validated with a prototype in Section 5.5. We deploy the off the shelf components for our prototype i.e. mBot robot as a vehicle, Raspberry Pi B 3+ as the LPU, arduino as the test node and laptop as the RPU with Python 3.5 as the programming language. The prototype is further tested with a simple use case explained in Section 5.7. As focused in this thesis, the use case tests the vehicle during normal operation. Further the system is validated as we performed complete testing using the remote test approach, wherein the network issues were focused on as well. ZeroMQ is used as MOM in our system. ZeroMQ socket timeout handles network connection related problems and helps in the graceful degradation for the system's operation. Conclusively, run time testing in mobile CPSs is possible with constraints based approach. With it, we not only defined the exact test points about when the run-time testing of the mobile CPSs can be initiated but also simultaneously performed with the normal system operation.

Outlook

Having presented and implemented the prototype of run-time test system for mobile CPSs, the system currently works on the principle of non-influential execution of the test cases, i.e. no influence on the normal system operation while performing the testing, which is tried to be achieved by adopting the constraints based approach. In order to perform run-time testing, we made the effective use of built in self-test (BIST). Our architecture is further conceptualized with two major blocks viz. the LPU and the RPU. Additionally, we carefully handled the network related issues during the remote testing approach and implemented the system with off the shelf components. From a futuristic perspective, our system can be implemented with real vehicles and deploying real ECUs. Our prototype use the USB as a communication bus which can be implemented with CAN bus for vehicle sensor data monitoring and diagnosis. Furthermore, the complete remote test system implementation is possible to be implemented over the cloud, making the system accessible from a wide range of network. In our validation, we are using off the shelf components to support our system concept. Hence, we are not following the hard real-time requirements needed for mobile CPSs. Our system is capable of handling these hard real time requirements in actual mobile CPSs itself. For our prototype we are using one robotic vehicle which communicates with the server. However in the actual scenario, several vehicles will communicate with the main server and hence the server must be capable enough to handle this redundancy. It should maintain the state by providing multiple sessions to multiple vehicle requests as illustrated in Figure 6.1. Proposing this future scope and summary, we conclude our architecture for the run-time testing of the mobile CPSs. Therefore, our research proves to be useful for researchers working in the field of testing of the cyber-physical systems and run time testing.

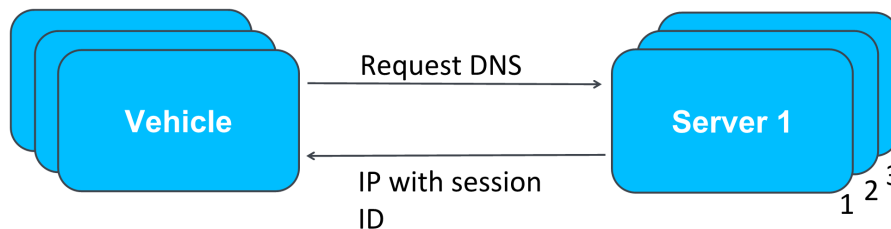


Figure 6.1: Server Redundancy

Bibliography

- [AIH15] S. A. Asadollah, R. Inam, H. Hansson. “A survey on testing for cyber physical system”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9447.1 (2015), pp. 194–207. ISSN: 16113349. DOI: [10.1007/978-3-319-25945-1_12](https://doi.org/10.1007/978-3-319-25945-1_12) (cit. on p. 19).
- [AKG13] F. AKGUL. *ZeroMQ*. Packt Publishing, 2013. ISBN: 178216104X, 9781782161042 (cit. on p. 32).
- [AMH98] H. Al-Asaad, B. T. Murray, J. P. Hayes. “Online BIST for embedded systems”. In: *IEEE Design and Test of Computers* 15.4 (1998), pp. 17–24. ISSN: 07407475. DOI: [10.1109/54.735923](https://doi.org/10.1109/54.735923) (cit. on pp. 20, 36, 37).
- [APW+11] H. Abid, L. T. T. Phuong, J. Wang, S. Lee, S. Qaisar. “V-Cloud”. In: *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies - ISABEL '11* (2011), pp. 1–5. ISSN: 15308669. DOI: [10.1145/2093698.2093863](https://doi.org/10.1145/2093698.2093863). arXiv: [arXiv:1405.1155v1](https://arxiv.org/abs/1405.1155v1). URL: <http://dl.acm.org/citation.cfm?id=2093698.2093863> (cit. on p. 23).
- [ARD] ARDUINO. *makeblock*. URL: <https://www.arduino.cc/> (cit. on p. 60).
- [AVI76] A. AVIZIENIS. “Fault-Tolerant Systems”. In: *IEEE Transactions on Computers* C-25.12 (1976), pp. 1304–1312. ISSN: 00189340. DOI: [10.1109/TC.1976.1674598](https://doi.org/10.1109/TC.1976.1674598) (cit. on p. 20).
- [BBIM93] B. R. Badrinath, A. Baker, T. Imielinski, R. Marantz. “Handling Mobile Clients: A Case for Indirect Interaction”. In: *4th Workshop on Workstation Operating Systems* (1993), pp. 91–97 (cit. on p. 21).
- [BER96] BERNSTEIN. “Philip A. Bernstein 86”. In: *Communications of the ACM* 39.2 (1996). ISSN: 00010782 (cit. on p. 24).
- [BFM+03] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, S. Pezzini. “Fault-tolerant platforms for automotive safety-critical applications”. In: *Proceedings of the international conference on Compilers, architectures and synthesis for embedded systems - CASES '03* (2003), p. 170. DOI: [10.1145/951710.951734](https://doi.org/10.1145/951710.951734). URL: <http://portal.acm.org/citation.cfm?doid=951710.951734> (cit. on pp. 39, 40).
- [BK03] T. Bishop, R. Karne. “A survey of middleware”. In: *Proceedings of the ISCA 18th International Conference Computers and Their Applications, Honolulu, Hawaii, USA, March 26-28, 2003* April (2003), pp. 254–258. DOI: [10.1.1.468.4907](https://doi.org/10.1.1.468.4907). URL: <http://triton.towson.edu/~%7B~%7Dkarne/research/middlew/surveyem.pdf> (cit. on pp. 24, 25).

- [BK06] E. Bringmann, A. Krämer. “Systematic testing of the continuous behavior of automotive systems”. In: *Proceedings of the 2006 international workshop on Software engineering for automotive systems - SEAS '06* (2006), p. 13. ISSN: 02705257. DOI: 10.1145/1138474.1138479. URL: <http://portal.acm.org/citation.cfm?doid=1138474.1138479> (cit. on pp. 37, 38).
- [BUZL11] C. Bradatsch, T. Ungerer, R. Zalman, A. Lajtkep. “Towards runtime testing in automotive embedded systems”. In: *SIES 2011 - 6th IEEE International Symposium on Industrial Embedded Systems, Conference Proceedings* (2011), pp. 55–58. ISSN: 2150-3109. DOI: 10.1109/SIES.2011.5953679 (cit. on pp. 20, 22, 23, 33).
- [CAR10] K. CARLSON. “The Number One Complaint.” In: *Collector (0010082X)* 75.12 (2010), pp. 26–30. ISSN: 0010082X. DOI: 10.1109/ICSE.2002.1007998. URL: <http://search.ebscohost.com/login.aspx?direct=true%7B%5C%7Ddb=bth%7B%5C%7DAN=51707471%7B%5C%7Dsite=ehost-live%7B%5C%7Dscope=site> (cit. on p. 20).
- [CB10] T.L. Crenshaw, S. Beyer. “UPBOT : A Testbed for Cyber-Physical Systems”. In: *Computer* (2010), pp. 1–8. URL: <http://dl.acm.org/citation.cfm?id=1924551.1924552> (cit. on p. 35).
- [CUR04] E. CURRY. “Message Oriented Middleware”. In: *Middleware for communications* (2004). URL: <http://onlinelibrary.wiley.com/doi/10.1002/0470862084.ch1/summary%7B%5C%7D5Cnhttp://www.edwardcurry.org/publications/curry%7B%5C%7DMfC%7B%5C%7DMOM%7B%5C%7D04.pdf> (cit. on pp. 29, 30).
- [FRE] FREERTOS. *FreeRTOS*. URL: <https://www.freertos.org/> (cit. on p. 44).
- [HW04] G. Hohpe, B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004 (cit. on pp. 26–28, 30, 31, 52).
- [JAZ14] N. JAZDI. “Cyber physical systems in the context of Industry 4.0”. In: *2014 IEEE Automation, Quality and Testing, Robotics* (2014), pp. 2–4. ISSN: 1844-7872. DOI: 10.1109/AQTR.2014.6857843. URL: <http://ieeexplore.ieee.org/xpls/abs%7B%5C%7Dall.jsp?arnumber=6857843> (cit. on p. 17).
- [KEN15] M. KENNEDY. “An intrinsic characterization of C*-simplicity”. In: 43.4 (2015), pp. 91–97. arXiv: 1509.01870. URL: <http://arxiv.org/abs/1509.01870> (cit. on p. 20).
- [KSKH10] M. Kim, M. O. Stehr, J. Kim, S. Ha. “An application framework for loosely coupled Networked Cyber-Physical Systems”. In: *Proceedings - IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2010* (2010), pp. 144–153. DOI: 10.1109/EUC.2010.30 (cit. on p. 20).
- [KW16] P. Koopman, M. Wagner. “Challenges in Autonomous Vehicle Testing and Validation”. In: *SAE International Journal of Transportation Safety* 4.1 (2016), pp. 2016–01–0128. ISSN: 2327-5634. DOI: 10.4271/2016-01-0128. URL: <http://papers.sae.org/2016-01-0128/> (cit. on p. 36).
- [LEE08] E. A. LEE. “Cyber Physical Systems : Design Challenges University of California , Berkeley”. In: (2008), pp. 363–369. DOI: 10.1109/ISORC.2008.25 (cit. on p. 17).
- [MAK] MAKEBLOCK. *makeblock*. URL: <https://www.makeblock.com/> (cit. on pp. 58, 59).
- [MAU05] S. MAURO. “(12) Patent Application Publication (10) Pub . No . : US 2005 / 0251304 A1”. In: 1.19 (2005) (cit. on pp. 20, 40, 41).

- [MBED12] W. Mueller, M. Becker, A. Elfeky, A. DiPasquale. “Virtual prototyping of Cyber-Physical Systems”. In: *Design Automation Conf. (ASP-DAC), 2012 17th Asia and South Pacific* (2012), pp. 219–226. ISSN: 2153-6961. DOI: [10.1109/ASPDAC.2012.6164948](https://doi.org/10.1109/ASPDAC.2012.6164948) (cit. on pp. 17, 18).
- [MLN] M. Mikucionis, K. G. Larsen, B. Nielsen. “T-U PP A AL : Online Model-based Testing of Real-time Systems”. In: *Science ()* (cit. on p. 34).
- [MQM11] C. A. Macana, N. Quijano, E. Mojica-Nava. “A survey on cyber physical energy systems and their applications on smart grids”. In: *2011 IEEE PES Conference on Innovative Smart Grid Technologies Latin America SGT LA 2011 - Conference Proceedings* (2011). ISSN: 9781457718014. DOI: [10.1109/ISGT-LA.2011.6083194](https://doi.org/10.1109/ISGT-LA.2011.6083194) (cit. on p. 17).
- [PB02] P. R. Pietzuch, J. M. Bacon. “Hermes: A distributed event-based middleware architecture”. In: *Proceedings - International Conference on Distributed Computing Systems* 2002-Janua (2002), pp. 611–618. DOI: [10.1109/ICDCSW.2002.1030837](https://doi.org/10.1109/ICDCSW.2002.1030837) (cit. on p. 24).
- [PSRL09] P. Pal, R. Schantz, K. Rohloff, J. Loyall. “Cyber-physical Systems Security- Challenges and Research Ideas”. In: *Workshop on Future Directions in Cyber-physical Systems Security* June 2014 (2009), pp. 1–5. URL: http://165.230.77.67/positionPapers/CPSS%7B%5C_%7DBBN.pdf%7B%5C%7D5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.5435%7B%5C%7Drep=rep1%7B%5C%7Dtype=pdf (cit. on pp. 17, 18).
- [RPI] RPI3. *Raspberry Pi*. URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> (cit. on p. 59).
- [SAT96] M. SATYANARAYANAN. “Fundamental Challenges in Mobile Computing”. In: *Annual ACM Symposium on Principles of Distributed Computing* (1996), pp. 1–7. DOI: [10.1145/248052.248053](https://doi.org/10.1145/248052.248053) (cit. on p. 21).
- [SDEF98] J. Seitz, N. Davies, M. Ebner, A. Friday. “A CORBA-based Proxy Architecture for Mobile Multimedia Applications”. In: *2nd IFIP/IEEE International Conference on Management of Multimedia Networks and Services MMNS 98* (1998). URL: <http://www.prism.uvsq.fr/conferences/1998/mmns98/> (cit. on pp. 21, 22).
- [SG14] C. Shravanthi, H. S. Guruprasad. “Mobile Cloud Computing As Future for Mobile Applications”. In: (2014), pp. 2319–2322. ISSN: 1556-5068. DOI: [10.2139/ssrn.2485229](https://doi.org/10.2139/ssrn.2485229) (cit. on p. 21).
- [SGLW09] L. Sha, S. Gopalakrishnan, X. Liu, Q. Wang. “Cyber-physical systems: A new frontier”. In: *Machine Learning in Cyber Trust: Security, Privacy, and Reliability* (2009), pp. 3–13. DOI: [10.1007/978-0-387-88735-7_1](https://doi.org/10.1007/978-0-387-88735-7_1) (cit. on p. 18).
- [SS88] K. K. Saluja, R. Sharma. “A Concurrent Testing Technique for Digital Circuits”. In: 7.12 (1988), pp. 1250–1260 (cit. on p. 23).
- [TBH+01] N. Taylor, K. E. Brown, K. Hamilton, D. Lane, N. Taylor, K. Brown. “Fault Diagnosis on Autonomous Robotic Vehicles with RECOVERY : An Integrated Heterogeneous-Knowledge Approach Fault Diagnosis on Autonomous Robotic Vehicles with RECOVERY :” in: November 2015 (2001), pp. 3232–3237. DOI: [10.1109/ROBOT.2001.933116](https://doi.org/10.1109/ROBOT.2001.933116) (cit. on pp. 22, 23, 38, 39).

- [TGG02] P. Tran, P. Greenfield, I. Gorton. “Behavior and performance of message-oriented middleware systems”. In: *Proceedings - International Conference on Distributed Computing Systems* 2002-January (2002), pp. 645–650. ISSN: 0013094X. DOI: [10.1109/ICDCSW.2002.1030842](https://doi.org/10.1109/ICDCSW.2002.1030842) (cit. on p. 28).
- [TRA08] S. TRAINOR. “Tobacco control in New Zealand: a history”. In: (2008), pp. 731–736. ISSN: 0738-100X. DOI: [10.1145/1837274.1837461](https://doi.org/10.1145/1837274.1837461) (cit. on p. 17).
- [VT06] S. Vinoski, I. Technologies. “Queuing Protocol”. In: December (2006), pp. 87–89 (cit. on pp. 30, 31).
- [WOL09] W. WOLF. “Cyber-physical systems”. In: *Computers* 42.3 (2009), pp. 88–89. ISSN: 0018-9162. DOI: [10.1109/MC.2009.81](https://doi.org/10.1109/MC.2009.81) (cit. on p. 17).
- [WS18] M. Wolf, D. Serpanos. “Safety and Security in Cyber-Physical Systems and Internet-of-Things Systems”. In: *Proceedings of the IEEE* 106.1 (2018), pp. 9–20. ISSN: 0018-9219. DOI: [10.1109/JPROC.2017.2781198](https://doi.org/10.1109/JPROC.2017.2781198). URL: <http://ieeexplore.ieee.org/document/8232537/> (cit. on p. 20).
- [ZHY13] L. Zhang, J. He, W. Yu. “Test Case Generation from Formal Models of Cyber Physical System”. In: 6.3 (2013), pp. 15–24 (cit. on p. 18).
- [ZMQ] ZMQ. *ZMQ_Guide*. URL: <http://zguide.zeromq.org/page:all> (cit. on p. 32).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature