

LB/DON/01/2019

CS 02/77

**PERCEPTION.JS – A FRAMEWORK FOR CONTEXT  
ACQUISITION, PROCESSING AND PRESENTATION**

**Supun Dissanayake**

**(148210U)**

**LIBRARY  
UNIVERSITY OF MORATUWA, SRI LANKA  
MORATUWA**

**M.Sc. in Computer Science**

**Department of Computer Science and Engineering**

**University of Moratuwa**

**Sri-Lanka**



TH3708

**May 2018**

TH 3708

+

CD-ROM

004"18"

004(043)

**TH 3708**

**PERCEPTION.JS – A FRAMEWORK FOR CONTEXT  
ACQUISITION, PROCESSING AND PRESENTATION**

**Supun Dissanayake**

**(148210U)**

**Thesis/Dissertation submitted in partial fulfillment of the  
requirements for the degree M.Sc. in Computer Science Specializing  
in Software Architecture**

**Department of Computer Science and Engineering**


**University of Moratuwa  
Sri-Lanka**

**May 2018**

## DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: .....

Date: 05/06/2018

Name: Supun Dissanayake

The supervisor/s should certify the thesis/dissertation with the following declaration.

I certify that the declaration above by the candidate is true to the best of my knowledge and that this report is acceptable for evaluation for the CS6997 MSc Research Project.

Signature of the supervisor: ... *UOM Verified Signature*

Date: 2018/06/06

Name: Dr. Malaka Walpola



## ABSTRACT

Context Awareness which is an area of Pervasive Computing that enables a person to accomplish his day-to-day tasks by seamlessly interacting with the “smart” devices that are embedded in the environment (smart space). In contrast to how a user interacts with a desktop computer or a mobile device using various input/output devices, Pervasive Computing paradigm acquires user’s context using sensors embedded in the surrounding environment, and identifies the actions the user would need to perform in a specific context. The Pervasive Computing application would then perform the required action on behalf of the user or it would give recommendations on the action the user would need to perform.

Because the number of smart devices are being produced and increased rapidly the demand for context awareness applications also increase, software developers can exploit the new computing paradigm to provide more innovative user-centered software solutions. However, the biggest obstacle for Context Awareness application development is its high complexity due to its broad technical areas (i.e. handling sensor imperfections, modelling smart environments, inferencing context, integrating with heterogenous systems or sensors, etc.) Hence software developers fail to provide quality context awareness applications that meet end-user requirements or fail to accurately identify context. Additionally, such software development increases project schedules, and could increase its bug rate.

This research project addressed the above problems by developing a software framework that enables the software developers to develop their applications using the fundamental features of Context Awareness such as Context Acquisition, Context Processing and Context Presentation. Apart from its functionality this research project focused on enhancing the quality of the framework by introducing quality attributes such as extensibility (which enables the developers to address the problem of heterogeneity), portability (which enables toe developers to use the framework in various devices and platforms), and usability (which enables the framework more usable by the developers).

From the technical perspective, the framework is based on the architecture of Sentient Object which this project aims to implement the architecture using a JavaScript technology stack. JavaScript enables to mitigate the problem of heterogeneity because the technology stack that will be used to develop the framework consist of Apache Cordova which enables to implement sensing mechanisms in a broad range of smart devices, and Node.js which enables to execute the context server in multiple platforms.

One of the most prominent aspect of the framework is that when the framework is embedded in a JavaScript application, the framework can transform the application into a Sentient Object. A Sentient Object can acquire contextual information using sensors, model and processes the context using an inferencing engine, and to respond to context changes using actuators. Scalability can also be achieved through Sentient Objects which can separate contextual information capture from context processing using a context server approach.



## ACKNOWLEDGEMENTS

I would like to express profound gratitude to my advisor, Dr. Malaka Walpola, for his invaluable support by providing relevant knowledge, materials, advice, supervision and useful suggestions throughout this research work. His expertise and continuous guidance enabled me to complete my work successfully.

Further I would like to thank all my colleagues for their help on finding relevant research material, sharing knowledge and experience and for their encouragement. I am as ever, especially indebted to my parents for their love and support throughout my life. Finally, I wish to express my gratitude to all my colleagues at 99X Technology, for the support given me to manage my MSc research work.

# Table of Contents

DECLARATION .....	i
ABSTRACT .....	ii
ACKNOWLEDGEMENTS.....	iii
LIST OF FIGURES .....	vi
LIST OF TABLES.....	viii
LIST OF ABBREVIATIONS.....	ix
INTRODUCTION .....	1
1.1. Overview of Pervasive Computing.....	2
1.2. Overview of Context Awareness .....	3
1.3. Problem Background .....	4
1.4. Problem Statement.....	6
1.5. Research Objectives.....	6
1.6. Solution Outline.....	6
1.7. Outline of the Thesis.....	8
LITERATURE REVIEW .....	9
2.1. Pervasive Computing.....	10
2.2. Context Awareness .....	12
2.2.1. Introduction.....	12
2.2.2. Context acquisition .....	13
2.2.3. Context processing.....	18
2.2.4. Context presentation .....	25
2.3. Related Work .....	27
2.3.1. Context-aware frameworks.....	27
2.3.2. Context-aware systems .....	28
2.4. Software Reusability and Software Frameworks.....	30
2.4.1. Usability of Frameworks.....	31
2.4.2. Architectural Pattern for the Framework .....	35
2.5. JavaScript and its Related Technologies.....	38
2.5.1. JavaScript programming paradigm .....	38
2.5.2. JavaScript design patterns.....	38
2.5.3. JavaScript Development Stack .....	40
DESIGN AND IMPLEMENTATION .....	43
3.1. Introduction.....	44
3.2. Requirement Analysis.....	44
3.2.1. Requirements of the framework.....	44



3.2.2. Use case scenarios.....	47
3.3. Architectural Design.....	48
3.2.1. Architectural considerations .....	48
3.2.2. Implementation of the architectural baseline.....	52
3.4. Detailed Design and Implementation.....	53
3.4.1. Portability.....	53
3.4.2. Bootstrapping process .....	54
3.4.3. Extensibility .....	55
3.4.4. Context representation .....	58
3.4.5. Context processing.....	59
3.4.7. Context acquisition and presentation.....	65
3.5. Integrating with Existing Technologies .....	66
3.5.1. Integrating with technologies to acquire contextual information .....	66
3.5.2. Integrating with technologies to store contextual information .....	70
3.5.3. Integrating with technologies to process contextual information .....	71
3.6. How to Use the Framework .....	72
3.6.1. Getting Started .....	72
3.6.2. Using Framework Features .....	77
3.6.3 Best Practices .....	80
TESTING AND ANALYSIS .....	83
4.1. Introduction.....	84
4.2. Unit Testing .....	84
4.2. System Testing.....	93
4.2.1. Testing of functionality, and extensibility .....	93
4.2.2. Testing of context server approach, and portability.....	97
4.3. Usability Testing.....	100
4.3.1. Test goals .....	100
4.3.2. Test scenario .....	100
4.3.3. Test process.....	101
4.3.4. Test findings .....	106
4.3.5. Analysis of usability .....	108
CONCLUSION.....	111
5.1. Summary of the Developed Framework.....	112
5.2. Advantages of the Framework.....	112
5.3. Limitations.....	113
5.4. Future Enhancements.....	113
5.5. Conclusion .....	114
REFERENCES .....	116



## LIST OF FIGURES

		Page
Figure 1.1	Definition of context	3
Figure 2.1	Ubiquitous computing framework	10
Figure 2.2	Layered conceptual framework for context-aware systems	14
Figure 2.3	Use of aggregation to obtain more accurate contextual information	16
Figure 2.4	Implementation effort to handle uncertainty in various methods	17
Figure 2.5	CONON ontology for context awareness applications	22
Figure 2.6	Domain-Specific ontologies and upper ontologies in CONON	23
Figure 2.7	Learning barriers for APIs and how they relate each other	34
Figure 2.8	Sense compute control architectural pattern	36
Figure 2.9	Sentient object model architecture	36
Figure 2.10	Non-blocking I/O feature of Node.js	41
Figure 2.11	PhoneGap architecture	42
Figure 2.12	Platform support for Apache Cordova	42
Figure 3.1	Use case diagram of Perception.js	47
Figure 3.2	Architecture of Perception.js	48
Figure 3.3	Internal representations of Perception.js	50
Figure 3.4	Class diagram of Perception.js	51
Figure 3.5	Integrating extensions	55
Figure 3.6	Default ontology in Perception.js	58
Figure 3.7	Internal representation of contexts using expression trees	60
Figure 3.8	How actuators are triggered	61
Figure 3.9	How expression trees are created for sub classes	62



Figure 3.10	Polling consumer integration pattern	67
Figure 3.11	How polling consumer integration pattern is used	67
Figure 3.12	Event driven consumer integration pattern	68
Figure 3.13	How event driven consumer integration pattern is used	68
Figure 3.14	How storage extensions can be used to integrate storage mechanisms	70
Figure 3.15	Integrating external context processing systems	71
Figure 4.1	Architecture of the first sample application	94
Figure 4.2	Ontology of the first sample application	94
Figure 4.3	Architecture of the second sample application	98
Figure 4.4	Architecture of the third sample application	102
Figure 4.5	Class diagram of the client application of the third sample application	103
Figure 4.6	Class diagram of the context server of the third sample	104
Figure 4.7	Work breakdown structure of the third sample application	105

## LIST OF TABLES

		Page
Table 2.1	Appropriateness indications of context modelling approaches	21
Table 4.1	Unit testing results of the component: Helpers	84
Table 4.2	Unit testing results of the component: AsyncIterator	85
Table 4.3	Unit testing results of the component: EventManager	86
Table 4.4	Unit testing results of the component: ConfigurationManager	86
Table 4.5	Unit testing results of the component: RuleManager	87
Table 4.6	Unit testing results of the component: TransportManager	88
Table 4.7	Unit testing results of the component: OntologyManager	88
Table 4.8	Unit testing results of the component: OntologyClass	89
Table 4.9	Unit testing results of the component: OntologyProperty	90
Table 4.10	Unit testing results of the component: Scheduler	90
Table 4.11	Unit testing results of the component: ExpressionTree	91
Table 4.12	Unit testing results of the component: BinaryExpression	91
Table 4.13	Unit testing results of the component: UnaryExpression	92
Table 4.14	Test results of the first test scenario	96
Table 4.15	Test results of the second test scenario	99
Table 4.16	Developer feedback questionnaire	106
Table 4.17	Time taken complete each task	107



## LIST OF ABBREVIATIONS

API	Application Programming Interface
CONON	Context Ontology
GPS	Global Positioning System
GPIO	General Purpose Input/Output
HCI	Human Computer Interaction
HMM	Hidden Markov Model
HTML	Hyper Text Markup Language
IO	Input Output
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
MQTT	Message Queueing Telemetry Transport
NPM	Node Package Manager
NoSQL	Not Only Structured Query Language
OO	Object Oriented
OWL	Ontology Web Language
QOS	Quality of Service
REST	Representational State Transfer
RFID	Radio-Frequency Identification
SCC	Sense Compute Control
SDK	Software Development Kit
SOA	Service Oriented Architecture
TCP	Transmission Control Protocol
SDK	Software Development Kit
SOCAM	Service Oriented Context Awareness Middleware
SWRL	Symantec Web Rule Language
UML	Unified Modelling Language
URL	Uniform Resource Locator
W3C	Worldwide Web Consortium
WBS	Work Breakdown Structure
XML	Extensible Markup Language

# CHAPTER 1

## INTRODUCTION



## 1.1. Overview of Pervasive Computing

As the number of “smart” devices such as smart watches and smart phones are being produced and increase rapidly the world is moving towards a physical environment which will be more saturated with computing and communication entities [1]. According to Mark Weiser [2] this computing paradigm allows the computers themselves to immerse in the environment and be part of the day-to-day life. From the user's perspective, this emerging trend introduces new means of interacting with the computer via smart devices. The dictionary of engineering [3] defines Pervasive Computing as “the presence of interconnected computing devices, such as motes, smart appliances, and wearable computers, which permeate a given environment” which is also known as ubiquitous computing. Pervasive Computing is a broad area which will be described in the later chapters. The area of Pervasive Computing which is concerned in this project is known as context awareness which enables the users of smart devices to accomplish their day today tasks with a little or no interaction with any device. Following is an example scenario of this technology;

Bhathiya is at the canteen of University of Moratuwa working on an assignment which requires to be submitted in a few minutes of time. He also needs to attend to the next lecture in a few minutes of time. He had worked on many documents and files which are to be submitted to Moodle the wireless connection at the canteen. Unfortunately, the bandwidth is miserable because many users are connected to the wireless network at the canteen. The Pervasive Computing system installed in Bhathiya's laptop discovers that with the existing wireless connection he won't be able to submit all the files to Moodle before the deadline and it discovers that the wireless bandwidth is excellent at the university library. Then the Pervasive Computing System displays a dialog box in Bhathiya's laptop screen suggesting him to move to the canteen in order to submit the assignment files on time. Bhathiya accepts the advice and walks to the library and while the files are being uploaded to Moodle he reads one of the latest magazines until the assignment upload is complete. Once the submission is completed on time he can also attend the next lecture.

From the technical perspective, to implement Pervasive Computing Mattern [3] assumes that a collection of IT techniques such as mobile and heterogeneous technologies are required [4] and the integrating such components and services are of the main importance. In addition to that the interaction between software, hardware, concepts and processes must also be taken place. Due to its broad area to practically implement in real world Pervasive Computing applications many specialists for HCI, process engineers, and software developers should be involved [5].



## 1.2. Overview of Context Awareness

The word 'context' can be described as related circumstances or background, whereas the word 'awareness' as knowledge, self-consciousness, recognition, conscious, consideration or the like [6]. Dey et al. [7] defines context as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Mehra [8] defines context in a broader perspective which explains that "To work with user context, systems and services must go beyond exploiting location data from GPS coordinates or interpreting words in queries and documents. They must understand places. They must understand what users know (such as their areas of interest), who they know, and who or what they trust (their social network and the social context within which they communicate and collaborate)". Mehra [8] presents the following as an illustration of the context.

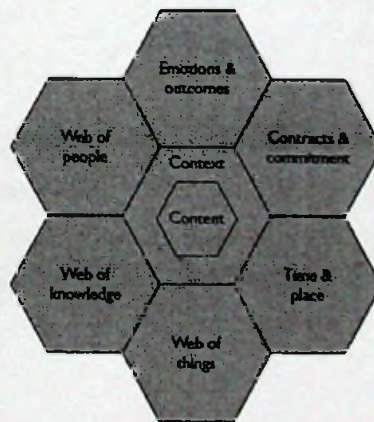


Figure 1.1: Definition of context [8]

Context-aware computing was first discussed by Schilit and Theimer [9] in 1994 to be software that "adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time." Providing a similar definition Dey et al. [7] defines context awareness as a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. Therefore, it can be concluded that the context awareness is the process which uses contextual information that will perform various processing to identify the situation of the user and to adopt accordingly.



In contrast to mobile computing which reacts to discrete events related to location and mobility management, pervasive computing is more proactive due to characteristic of invisibility. Invisibility enables a user to seamlessly accomplish his day-to-day tasks [1]. Therefore, intelligent environments are a prerequisite to pervasive computing. In pervasive computing two concepts exist in order to provide proactivity which are the Adaptation and Context Awareness. The difference between adaptation and context awareness is that, an adaptation system identifies a computational resource (i.e. wireless network bandwidth, energy, computing cycles, or memory) scarcity and adjusts itself accordingly while a context awareness system aims identifies the need of the user and adjusts itself accordingly. Both concepts relate to the invisibility in pervasive computing [10].

For proactivity to be effective Satyanarayanan [10] describes that a pervasive system must be capable of identifying user intent and the system surroundings in order to modify its behavior which is known as context awareness. Additionally, such systems may also try to make assumptions about the end-user's current situation [11].

### **1.3. Problem Background**

Complexity in developing context awareness applications is a common problem despite of its opportunities. Complexity can be further described by taking the functionality that the context awareness application developers are required to develop. Being a subset of Pervasive Computing, a context awareness system may require the following functionality to be implemented [10];

- **Effective use of Smart Spaces:** A space could be a room or any area which becomes a smart space when smart devices are embedded in the space. Pervasive computing focuses on the effective use of the smart devices in smart spaces.
- **Invisibility:** Allows the mobile user to perform his tasks seamlessly in the environment which is the core area of research of this project.
- **Localized Scalability:** As the number of smart devices increase in smart spaces the interactions between the smart devices and the user's personal computing space increase. Localized scalability aims to achieve scalability in a way that would enable to reduce interactions between distant entities.
- **Masking Uneven Conditioning:** The rate of penetration of pervasive computing will introduce many heterogeneous infrastructures. Masking uneven conditioning means that



the user's personal computing space must interact with the smart spaces transparently without concerning about the underlying technology. If such interaction is impossible, one way to solve this issue is to compensate for "dumb" environments.

Taking the above areas into consideration, Context Awareness emphasizes in the Invisibility characteristic of Pervasive Computing. To develop such systems with the above functionality Saha and Mukherjee [1] describes the following issues and challenges in Context Awareness application development which describe the non-functional requirements or quality attributes that such an application should consist of;

- Scalability: As environmental smart-ness grows, so will the number of devices connect to the environment and the intensity of human-machine interactions.
- Integration: Though pervasive computing components are already deployed in many environments, integrating them into a single platform is still a research problem. The problem is similar to what researchers in distributed computing face, but the scale is bigger.
- Heterogeneity: Conversion from one domain to another is integral to computing and communication. Assuming that uniform and compatible implementations of smart environments are not achievable, pervasive computing must find ways to mask this heterogeneity - or uneven conditioning
- Invisibility: A system that requires minimal human intervention offers a reasonable approximation of invisibility

Taking the above context awareness application development challenges into consideration that complexity is the major obstacle that application developers must cope with. Not being able to address this issue may result in the following negative outcomes;

- From the developer's perspective, the developers focus more on solving technical problems (i.e. performance, scalability or programming problems) rather than focusing more on solving the actual context awareness problem. As mentioned before to develop context awareness applications an End-User centered design approach must be followed in contrast to the actual approach which the developers are following which in System Centered Design which violates the development paradigm of context awareness applications.



- From the end-user's perspective, this could result in poorly constructed context awareness which lacks performance, doesn't meet his requirements, higher bug rate or provide inaccurate results.
- From the development project manager's perspective, this would result in increased development schedules.

#### **1.4. Problem Statement**

Developing a context awareness application is a difficult task because its technical complexity. Specialists from various disciplines are required such as HCI specialists, software developers, and process engineers [5] which adds more complexity into the software development. When such applications are being developed in a team to achieve a business objective, due to its complex nature the team would not be able to deliver the required functionality. In addition, the application development time would increase, it would increase in bugs, and it would not meet its quality criteria.

#### **1.5. Research Objectives**

The main objective of this research project is to research and develop a reusable and extensible software framework that enables the software developers to develop context awareness applications. Research was mainly conducted to identify on how to facilitate effective context awareness application development which include context acquisition, context modelling, context inferencing, and context presentation. The research was also conducted on the existing approaches, and previous work that was used to solve such problems related to context awareness application development. Such research had revealed information on identify the requirements and to construct the architecture the framework.

#### **1.6. Solution Outline**

Research was conducted on the area of framework usability which enables the developers to effectively and efficiently use the framework to build their domain-specific context-aware applications. In addition to that, research was conducted on how to use various



JavaScript design patterns which provide non-functional requirements of the framework such as extensibility, and portability.

Taking the problem background into consideration a context awareness framework was developed to address the problems. The framework enables the developers to architect and develop context awareness applications that provides context acquisition using sensors, context modelling using ontologies, context reasoning using rules, and context presentation using actuators. The framework is also capable of building context servers that can be used for context processing.

Context awareness application development also requires integrating and interacting with many heterogeneous devices. Therefore, extensibility quality factor was taken into consideration to integrate with many heterogeneous technologies such as dissimilar communication protocols (i.e.: web sockets, or TCP), and various sensing mechanisms (soft sensing mechanisms such as social network integration, and various hard sensing mechanisms). In addition to addressing the problem of heterogeneity through extensibility, this quality attribute also enables advanced developers to extend the framework according to their domain-specific requirements.

In addition to making the software framework reusable and to make the framework usable by software developers API usability was taken into consideration which facilitates the developers by enabling them to deliver less error prone and effective solutions in less time. To achieve this objective usability features were added to the framework to eliminate counter-productive areas in software projects and frameworks.

In addition to the above proposed solutions for heterogeneity and usability, and taking the programming perspective into consideration, JavaScript was selected as the programming language to implement the framework. The framework had implemented various design patterns in the framework that can be used to easily use context awareness features with JavaScript, to reduce the learning curve, and to provide extensibility in the framework. Since JavaScript can be used to write cross-platform applications, the framework was written as a portable library that can be included in applications in the web browser, mobile devices, and node.js servers. JavaScript also uses non-blocking IO to retrieve data from IO devices which



makes the JavaScript applications more efficient at IO bound tasks. This feature facilitates context-aware applications to retrieve data from IO devices such as sensors efficiently.

### **1.7. Outline of the Thesis**

The second chapter of this thesis presents the literature review which aims to critically evaluate the research and the subject area of context awareness. The chapter includes areas related to context acquisition, context processing, context presentation, software frameworks, usability of frameworks and JavaScript related design patterns.

The third chapter consists of the requirements, the architecture, the detailed design and the implementation of the solution framework. The detailed design sections present the internal implementations of various functionalities related to context awareness and internal implementations of quality attributes such as extensibility, portability and usability.

The fourth chapter of this thesis consists of testing of the framework in order to verify whether the solution framework solves the problems presented in the first chapter. For testing few scenarios were presented to verify the functional and nonfunctional features of the framework. For the final scenario, a real-world usability testing was performed with two software developers to identify whether the solution framework meets the usability criteria. Additionally, the chapter analyses the findings of the test results and critically evaluates the findings against the solution outline presented in the first chapter.

# CHAPTER 2

## LITERATURE REVIEW



## 2.1. Pervasive Computing

According to Mark Weiser [2] Pervasive Computing is not a consequence of technology but of human psychology because it focuses more on end-user's day-to-day tasks rather than the technology being the main focus. This approach is known as User Centered Design which focuses on creating environments and products that are usable by the end-users in contrast to System Centered Design which focuses on the best use of technology. This enables the end users to focus more on their end goals they would need to achieve from an application rather than focusing on technical details on how to operate a device to achieve his end goals.

In order to get a holistic view on Pervasive Computing Holzinger et.al [11] had presented the following Ubiquitous Computing Framework that describes the integrations and interactions between entities involved in a Ubiquitous Computing environment.

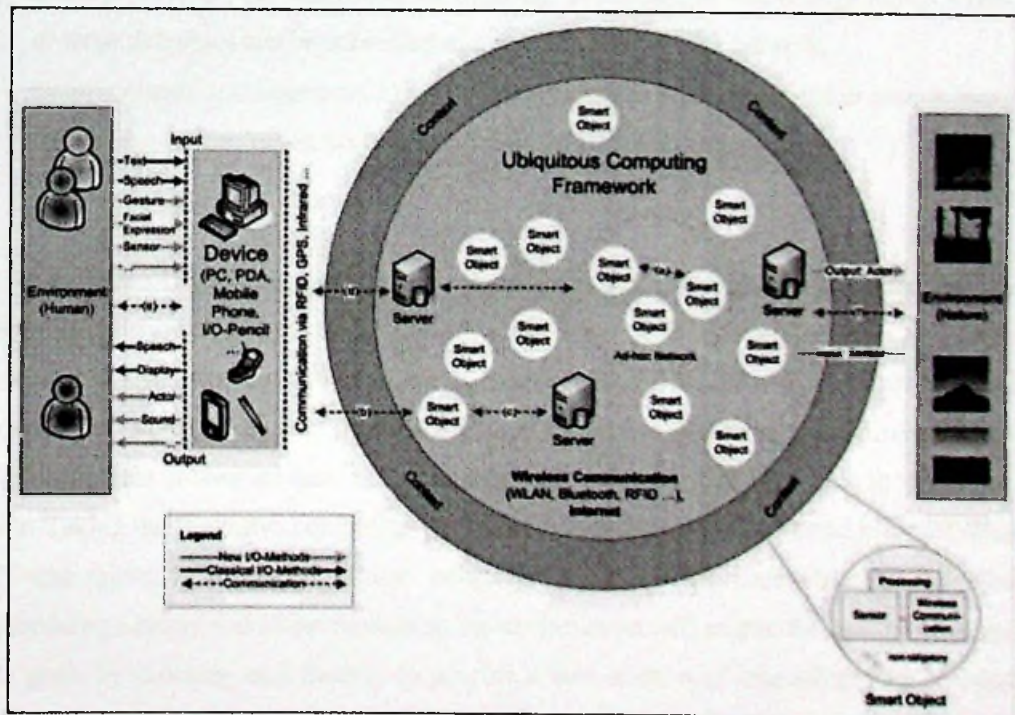


Figure 2.1: Ubiquitous computing framework [11]

- Input and output: The environment produces different types of input and output, which originates from users or from the natural environment. To interact with the environment, sensors are used. To interact with the users' various types of I/O devices are used.



- **Devices:** Under some circumstances it is important to interact with the network via I/O devices. These devices mainly take over I/O tasks between the end-user and the network. The device can be a PC, smart watch or mobile phone, or an electronic pencil, etc.
- **Context:** A context describes the object's background and provides end-user services depending on the context.
- **Smart Objects:** The core network consists of so-called smart objects, which can be smart things as well. Smart objects are aware of their environment, can quasi perceive their surroundings through sensors, can collaborate with peers using short-range wireless communication technologies, and provide context-aware services to end-users in smart environments [13].
- **Ad-hoc Network:** For some applications, smart objects can form ad-hoc networks. Examples are Mobile ad-hoc networks, which are self-configuring and mainly work autonomously.
- **Internet:** Web services, which are used by the components of the network, must be offered (e.g. smart objects, servers).
- **Remote Servers:** Some functions that need high processing as well as high storage capacity or large databases can be submitted to a remote server on the network.
- **Communication:** Communication is an integral part of our concept and is mainly handled by wireless technologies, such as Bluetooth, RFID, GPS, and infrared

The above Ubiquitous Computing Framework relates to the Pervasive Computing goals described by Saha and Mukherjee [1] which are Invisibility, Masking of Uneven Conditioning, Localized Scalability and Effective use of Smart Spaces. This framework illustrates clearly that pervasive computing uses sensors to acquire contextual data from the environment, smart objects process contextual data, and actions perform a specific task that is required by the end user. Taking the Pervasive computing paradigm that was initially introduced by Mark Weiser [2] and taking this framework into consideration, it can be concluded that seamlessly embedding sensors and smart devices in the environment will enable the user to accomplish his goals by utilizing such devices to provide a new method of interaction with computers which accomplishes their goals with little or no intervention with any device.



## 2.2. Context Awareness

### 2.2.1. Introduction

To identify the characteristics in Context Awareness this section of literature review is separated into three subsections which are Context Acquisition, Context Processing and Context Presentation. This approach will enable to critically evaluate the academia and adopt the best methods in order implement the functional and non-functional requirements of the framework.

From the technical perspectives of extensibility and reusability in context awareness Ranganathan et al. [14] presented a layered architecture which augments layers for detecting and using context by adding interpreting and reasoning functionality which be used to provide the invisibility aspect of Pervasive Computing. The objective of presenting this layered architecture is that it will serve as a solid foundation to conduct the research to address the extensibility requirements of the framework.

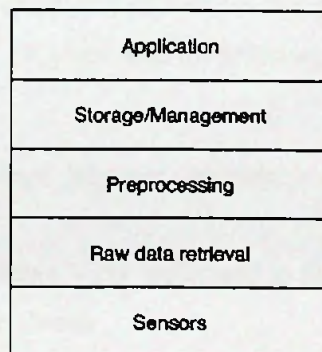


Figure 2.2: Layered conceptual framework for context-aware systems [14]

- **Sensors:** Consists of a collection of different sensors which includes hardware or software entities which provide contextual information.
- **Raw data retrieval:** Makes use of appropriate drivers for physical sensors and APIs for virtual and logical sensors. Often the query functionality is implemented in reusable software components.
- **Preprocessing:** This layer is responsible for tasks such as information aggregation, reasoning and interpreting contextual information. Context aware systems consists of multiple context data sources therefore a single value may not provide important

information a process known as “aggregation” or composition is conducted to acquire important information.

- Storage and Management: Organizes the gathered data and offers them via a public interface to the client which enables to retrieve contextual information for machine learning or analytical purposes.
- Client: The reaction on different events and context-instances are implemented in this layer which is used to present the context changes.

### 2.2.2. Context acquisition

Initially, in the process of Context Awareness the relevant contextual information must be acquired from the environment and from the user’s smart device. This section aims to evaluate the context acquisition mechanisms, sensing types, imperfections of sensing information and how to deal with imperfection of sensing information to adopt the best context acquisition mechanisms for the framework.

Context awareness requires contextual information to make assumptions about the current state of the user. Holzinger et al. [11] describes the following types of contextual information.

- Situation:
  - Demographic Data (age, language, education): authentication [15] can adapt the viewable features.
  - Optimization of Display View: optimized to the corresponding device (i.e. PC, smart watch, mobile phone)
- Location: the end-user’s current location and surroundings, sent via RFID GPS or infrared
- Activity: the end-user’s activity, e.g. recognition via wearable computing devices, mobile phone

In addition to the above contextual information Mehra [8] describes the following additional contextual information;

- What the users know (such as their areas of interest).
- Who the users know.
- Who or what they trust (social network and the social context within which they communicate and collaborate)



To capture above described contextual information sensing mechanisms can be used. Chen et al. [16] Presents three different approaches on how to capture contextual information.

1. Direct sensor access: This approach is used with devices which have inbuilt sensors which has a drawback that it doesn't consist of an additional layer for gaining and processing sensor data. Therefore, this approach is not suited for distributed systems
2. Middleware infrastructure: This approach uses layered architecture encapsulates low level sensing details. Hence it promotes reusability and extensibility because its client which is accessing the sensor doesn't required to change its code due to its strict encapsulation.
3. Context Server: This approach enables multiple clients to access to remote data sources. This extends middleware based architecture by introducing an access managing remote component which is known as the Context Server which is responsible for gathering of sensor data. This approach relieves clients from resource intensive operations which makes this suitable for devices with limited resources such as computational power, or disk space.

To simplify the context acquisition process by eliminating the previously mentioned issues and challenges such as scalability, integration, heterogeneity and invisibility the best approach is the context server. Even though a context server approach addresses these issues the focus of this research project is a framework approach which follow two contrasting approaches to solve the same problem. In order to resolve the conflict, the best approach to follow is to combine the capabilities of a context server and a framework based approach. This approach will enable the developers to transform their application into a context server by embedding the solution framework of this project.

Since the context server is deployed in a server it requires contextual information available in smart devices. Therefore, in our approach the process of context acquisition involves integration and interaction with the smart devices and the context server. Hence the framework should extend its scope further to support this feature.



To capture contextual information in smart devices sensors can be used which can be classified into three groups as follows [17].

1. Physical sensors: Also known as hardware sensors which are the most frequently used sensors which could capture almost any physical data.
2. Virtual sensors: Also known as software sensors which capture context data from software applications or services. For example, it is possible to determine an employee's location not only by using tracking systems physical sensors but also by a virtual sensor, (i.e. by browsing an electronic calendar, a travel-booking system, emails etc.) for location information.
3. Logical sensors: A combination of physical and virtual sensors with additional information from databases of various other sources. For example, a logical sensor can be constructed to detect an employee's current position by analyzing logins at desktop PCs and a database mapping of devices to location information.

Taking the extensibility factor into consideration the solution framework should allow the developers to extend by developing extensions for sensing mechanisms.



### 2.2.2.1 Handling sensor imperfections

Although sensors can be used to obtain low level sensor data, to make such data meaningful for the context aware system for reasoning, low level contextual data should be abstracted to a higher-level context [18]. However, when sensors provide data, the sensors could provide uncertain data which causes inaccuracy ambiguity, and incompleteness of sensed context. [18]. According to [19] such uncertainties in sensing can be caused by the following reasons;

1. Lack of knowledge: Caused by incomplete information both at the model level or if the information is not provided by the sensors.
2. Lack of semantic precision: Caused by semantic mismatches in the notion of the information.
3. Lack of machine precision: Caused by machine sensors that provide data that are imprecise and ambiguous.

To address such situations Kramer et.al [20] suggests that aggregation can be used which can combine multiple sources of sensors and aggregate such sensors to provide more accurate contextual information. When uncertainties described above is taken into consideration, aggregation enables to overcome those issues. The following table shows some examples on how aggregation is used to get more accurate contextual information;

Class	Description	Sensor
Context of Muffin		
State	Moving	Accelerometer,
	Top side	Ultrasonic range finder
	Direction	Compass, Accelerometer
	Held by a user	Skin resistance sensor, Grip sensor
Context of a environment		
Air	Air temperature	Air temperature sensor
	Air humidity	Relative humidity sensor
	Air pressure	Barometer
Sound	Ambient noise	Microphone
	Talking voice	

Class	Description	Sensor
Context of a user		
Activity	Standing or sitting	Accelerometer,
	Walking or running	Ultrasonic range finder,
	Going up/down stairs	Compass
Geographical information	Location	GPS
	Orientation	Compass
Physical condition	Level of stress	Skin resistance sensor, Skin temperature sensor, Pulse sensor, Grip sensor
		Alcoholic breath
	Emotion	Exciting
Surprising		Skin temperature sensor,
Fearing		Pulse sensor, Accelerometer

Figure 2.3: Use of aggregation to obtain more accurate contextual information [20]



The following methods can also be used for handle uncertainty of sensor data;

- Bayesian reasoning: can be used to dealing with uncertainty which is commonly used for location tracking [18]. Bayesian probability model can also be used to extend an ontology based model to reason about uncertainty [18]. This method can be used to effectively handle machine precision and lack of knowledge. This is well suited for handling mechanisms caused by vague information in terms of probability [19].
- Dempster-Shafer theory: A weighted Dempster-Shafer evidence combining rule is introduced based on the historically estimated correctness rate of sensors [18]. This method can be used to effectively handle imprecise sensor information and lack of semantic precision.
- Fuzzy Logic: Is used to model uncertainty in understandable form. But they mainly cope with uncertainty caused by the lack of human precision [18].
- Machine Learning: Uses data-driven rather than model driven approach for reasoning. They allow for handling both uncertainties due to lack of knowledge and lack of precision [19].
- Certainty Factors: are used to describe uncertainties related to lack of knowledge and related to lack of machine precision. One of the main advantages of certainty factors over other uncertainty handling mechanisms is that they can be easily incorporated into existing rule-based system without the necessity of redesigning or remodeling knowledge base. They also require a very low implementation effort [18].

	Uncertainty source			Implementation effort
	Lack of knowledge	Semantic imprecision	Machine imprecision	
Probabilistic	●	○	●	High
Fuzzy Logic	○	●	●	Medium
Certainty Factors	●	○	●	Low
Machine learning	●	○	●	High

Figure 2.4: Implementation effort to handle uncertainty in various methods [19]

In perception.js to handle sensor imperfections, a callback method can optionally be provided to add preprocessing to sensed data. Preprocessing feature can be used to handle uncertainties in sensed data. Extensions were not used for perception.js because such different mechanisms for handling uncertainties are more domain-specific. Therefore, applying extensibility to handle uncertainties would not provide any value in the framework.



### **2.2.3. Context processing**

#### *2.2.3.1. Context modelling*

In addition to using a context server for context acquisition a context server can also be used for context processing to reduce complexity in the context-aware application because awareness is an integration between the physical environment and the technology [21]. Additionally, it consists of functionality related to context aggregation, and analysis operations and provide mechanisms for context notification to applications [21].

When taking the term “model” in to consideration which can be defined as an abstraction of the system which provides its problems in a concise fashion [22]. Models aims to focus on the most important elements of interest.

According to Henricksen et al. [23] contextual information has the following characteristics;

1. Context information exhibits a range of temporal characteristics: Context information can be characterized as static or dynamic. Static context information describes those aspects of a pervasive system that are invariant, such as a person’s date of birth. As pervasive systems are typically characterized by frequent change, the majority of information is dynamic.
2. Context information is imperfect: Information may be incorrect if it fails to reflect the true state of the world it models, inconsistent if it contains contradictory information, or incomplete if some aspects of the context are not known. First, pervasive computing environments are highly dynamic, which means that information describing them can quickly become out of date. These factors can lead to large delays between the production and use of context information. Second, context producers, such as sensors, derivation algorithms and users, may provide faulty information.
3. Context has many alternative representations: Much of the context information involved in pervasive systems is derived from sensors. There is usually a significant gap between sensor output and the level of information that is useful to applications, and this gap must be bridged by various kinds of processing of context information.
4. Context information is highly interrelated: Several relationships are evident between people, their devices and their communication channels (for example, ownership of devices and channels and proximity between users and their devices).

To overcome the problems of imperfections of sensor readings data preprocessing can be used. Taking the above mentioned layered architecture into consideration, the preprocessing layer is responsible for reasoning and interpreting contextual information [24]. In order to store



context data in a machine process able form a context model is needed [24]. According to Da et al. [21] context model is the format used by the context middleware to provide context information for the upper layer which is structured, consistent, decomposable, assemblable and extensible. Context modelling solves the problem of representing context internally in this scenario inside the context server. Ubiquitous computing systems make high demands on any context modeling approach in terms of [25]:

1. Distributed composition: Ubiquitous computing system is a derivative of a distributed system which lacks centralized maintenance of data and services. Instead, composition and administration of a context model and its data varies with notably high dynamics in terms of time, network topology and source.
2. Partial validation: Since information is available in many nodes, it is necessary to validate information on one node. This is particularly important because of the complexity of contextual interrelationships, which make any modeling intention error-prone.
3. Richness and quality of information: The quality of information captured by sensors varies overtime and information provided by different kinds of sensors characterize information delivered by a sensor varies over time, as well as the richness of information provided by different kinds of sensors characterizing an entity in a ubiquitous computing environment may differ. Thus, a context model appropriate for usage in ubiquitous computing should inherently support quality and richness indication.
4. Incompleteness and ambiguity: The context model enable to deal with incompleteness and ambiguity in the contextual information. This should be covered by the model, for instance by interpolation of incomplete data on the instance level.
5. Level of formality: It is highly desirable that each participating party in ubiquitous computing interaction shares the same interpretation of data exchanged and the meaning behind it. For example, to perform the task "print documents on printer near me" means to me.
6. Applicability to existing environments: From the implementation perspective, it is important that a context model must be applicable within existing the infrastructure of ubiquitous computing environments, e.g. a service framework such as Web Services.



In [26], in the point of view of data structure, context modelling can be divided into the following categories.

1. Key-value models: This approach is the simplest approach of context modelling which is easier to manage and less error risk prone. However, it has a drawback if quality meta-information or ambiguity is considered. Frequently used in distributed service frameworks which the services itself are usually described with a list of simple attributes in a key-value manner [27].
2. Markup scheme models: Taking the partial validation requirement into consideration markup scheme models are the best which suit the requirement. There usually exists a scheme definition and a set of validation tools which can be used for type checking, even for complex types. Its drawback is that incompleteness and ambiguity should be handled proprietary on the application level.
3. Graphical models: This kind of approach is particularly applicable to derive an ER-model [28]. UML is one example of a graphical model which can be used due to its generic structure from it, which is very useful as structuring instrument for a relational database in an information system based context management architecture [25].
4. Object oriented models: Common to object oriented context modeling approaches is the intention to employ the main benefits of any object-oriented approach - namely encapsulation and reusability to cover parts of the problems arising from the dynamics of the context in ubiquitous environments. Applicability to existing object oriented ubiquitous computing runtime environments is given, but has usually strong additional requirements on the resources of the computing devices – requirements which often cannot be fulfilled in ubiquitous computing systems [25].
5. Logic based models: A logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inference) from a set of other expressions or facts. Logic based context models may be composed distributed, but partial validation is difficult to maintain. Their level of formality is extremely high, but without partial validation the specification of contextual knowledge within a logic based context model is very error-prone [25].
6. Ontology based models: Ontologies enable the developer to specify concepts and interrelations [29]. They are particularly suitable to project parts of the information describing and being used in our daily life onto a data structure utilizable by computers. Using ontologies provides a uniform way for specifying the model's core concepts as well as an arbitrary number of sub-concepts and facts, altogether enabling contextual knowledge sharing and reuse in a ubiquitous computing system [28]. This contextual knowledge is evaluated using ontology reasons.



Table 2.1: Appropriateness indications of context modelling approaches [25]

Approach	Distributed Composition	Partial Validation	Quality of Information	Incompleteness	Level of Formality	Applicability
Key-Value Models	-	-	--	--	--	+
Markup Scheme Models	+	++	-	-	+	++
Graphical Models	--	-	+	-	+	++
Object Oriented Models	++	+	+	+	+	+
Logic Based Models	++	-	-	-	++	--
Ontology Based Models	++	++	+	+	++	+

Even though each modelling approach has its strengths and weaknesses, taking the above evaluation into consideration it can be concluded that ontology based models provide many features required for context awareness.

Taking the extensibility requirement of the solution framework into consideration, although the framework can be extended for the developer to develop extensions for various modelling approaches it would not enable the developers to overcome the above-mentioned issues and challenges in pervasive computing which are invisibility, heterogeneity, scalability and integration. For example, key-value models are poor in handling incompleteness and ambiguity while graphical models are poor meeting the distributed composition requirement. Hence as a context modelling approach for this framework ontology based models would meet the requirement.

Data storage is another area in context process which extensibility can be adopted. To store data according to Ruijn [30] there are three possible ways which are the file system, database and cloud storage. Depending on the use case the architecture must focus on how to guarantee fast access to the data anytime and anywhere. Databases are the best solution for different purposes because they include access and privacy mechanisms. On the contrary Cloud storage needs Internet access and data privacy is always a problem. However, Cloud storage can suit to self-adaptive platforms.





Wang et.al [31] has presented an ontology for context known as CONON for awareness applications.

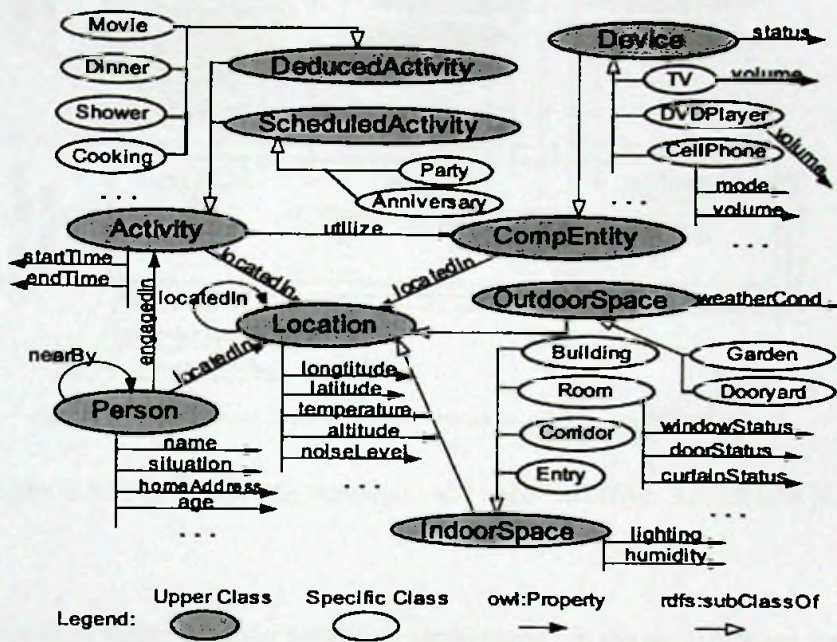


Figure 2.5: CONON ontology for context-aware applications [31]

According to Wand et.al [31] the entities Location, Person, Activity, and Computational Entity are the most fundamental context for capturing the information about the executing situation. In addition to that, this approach enables the developer to easily model sub domains for different intelligent environments [31]. The upper ontology on CONON consists of a more generic set of entities that can be used in any domain whereas the lower ontology consists of a more domain specific entities which are inherited from the upper generic ontology [31].



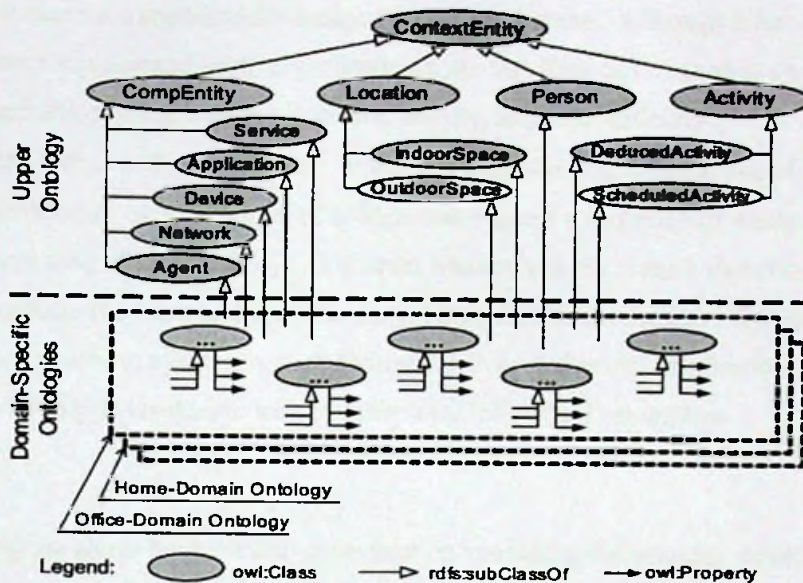


Figure 2.6: Domain-specific ontologies and upper ontologies in CONON [31]

Due to its ability to model intelligent environments in any domain to the framework this ontology is selected for context modelling in perception.js.

### 2.2.3.2. Context reasoning

Although context can be represented using a context model while providing extra benefits presented in the previous chapter, context models alone doesn't have the ability to provide reasoning features for context-aware systems. To address that issue different methods of reasoning can be implemented in context-aware applications. According to B.Y Lim [32] The following methods of reasoning can be implemented in a context-aware application;

- Rules: Rules are popular in domains such as activity recondition, adaptation, awareness, monitoring, and location guides. It is the most popular approach used in context-aware applications [32].
- Decision Trees: Which is a machine learning approach that learns a tree from a dataset which decides on its output by deciding on a specific input feature at each node as it traverses down and returns once it reaches a leaf node. This approach is popular in use cases such as recognizing identity/ability, interruptibility, and mobility.
- Naïve Bayes: Which is a probabilistic approach that applies Bayes theorem to model the probability of the output of a system given the inputs which applies a naive assumption



that features are conditionally independent of one another. Although it has a low level of accuracy training and runtime performance are fast. This can be used in domains such as recognizing physical activity, domestic activity, and interruptibility.

- Hidden Markov Models (HMM): Which is a Bayesian probabilistic classifier that model the probability of a sequence of hidden states given a sequence of observations (input features with respect to time). First order Markov models assume that only the previous state affects the next, and only the current state influences the current observation. This method is used to model physical, domestic activity and several applications also combine rules for higher level logic with classifiers for lower level recognition.

Taking the above methods into consideration and taking the ontology modeling approach presented previously, the best suited approach for the solution framework is a rule based approach. Rules can also be implemented using any programming language compared to other methods. In addition, taking the performance factor of the framework into consideration, rules are the best option. Other advantage of using rules is that, since ontologies are good at defining the domain model of a given context, using rules with ontologies enable the developers to easily separate the behavior from the domain model [33]. Hence this approach also provides more maintainability in source code.

Hamadache et.al. [34] has presented a middleware known as SOCAM (Service-Oriented Context-Aware Middleware) which uses OWL (Web Ontology Language) ontologies to model the context, and SWRL (Semantic Web Rule Language) for context processing. The rules fire a specific action according to the context change.

SOCAM also uses OWL to define our context ontologies for the following reasons [34]:

- much expressive compared to other ontology languages.
- has the capability to exchange and share context knowledge between different systems.
- OWL is a W3C standard.

Since Ontologies and rules are effective for reasoning, the for the solution framework these methods can be adopted. In addition to that ontologies and rules are also works well with the extensibility mechanisms that are implemented in the solution framework.



#### 2.2.4. Context presentation

Context presentation is required after the context has been identified and required the application to modify its behavior accordingly. Da et al. [21] presents three context aware features that context-aware applications may support which are;

1. Presentation of information and services to a user.
2. Automatic execution of a service.
3. Tagging of context to information for later retrieval.

As mentioned in introduction to make proactivity or invisibility to be effective Adaptation strategies can be implemented [1]. According to Satyanarayanan [10] there are three alternative strategies for adaptation in Pervasive Computing;

1. A client can guide applications in changing their behavior so that they use less of a scarce resource.
2. A client can ask the environment to guarantee a certain level of a resource.
3. A client can suggest a corrective action to the user.

According to Chefrour [35] there are three types of adaptations which are;

1. Reactive adaptation changes the behavior of the application according to the environment changes due to due to context or user preference change.
2. Evaluative adaptation aims to extend the functionality of the application such as correcting its errors, to increase its performance, and to provide QOS changes.
3. Adaptation for integration addresses the problem of integration incompatible services or components such as different hardware or software interfaces, or different protocols. This type of adaptation is done after any previously defined adaptation has been done for example due to the execution of this adaptation, a component can no more communicate with another one because they have different security protocol. In this case, the system will launch an adaptation for integration.



Since the solution framework follows a context server approach, the context presenting client will require to access the context that is in the context server. Baldauf [24] presents two different ways which the client can access the information which are synchronous and asynchronous. In the synchronous manner, the client is polling the server for changes via remote method calls. In synchronous mode, the client pulls messages from the server. The asynchronous mode works via subscriptions. In asynchronous mode whenever a change occurs the server pushes messages to the client.

Taking the synchronous and asynchronous paradigm further, Da et al. [21] presents three methods which client can communicate with the context server which are;

1. Event based: Communicates through events which can be filtrated, set up timeouts and can be classified by topics. Event-based mechanism decouples the communication from the time-space and the control flow [21].
2. Tuple Space: allows communicating with shared memory similar to a black board which clients poll the Tuple Space in intervals.
3. Connector based: allows communication with connectors which provide unified I/O interfaces for components so applications components are not concerned by remote communications.

In the solution framework when efficiency is taken into consideration event based communication is the best suited method which can be adopted due to its asynchronous nature. Synchronous communication imposes more overhead and it doesn't provide real-time context presentation features. Therefore, approaches of tuple space and connector based are the most effective solutions for the solution framework.

Taking the communication application layer protocol into consideration different clients may use different protocols a web based client may use a web socket based communication mechanism, a mobile client may use a TCP based communication. Therefore, extensibility must be focused on extending the framework with multiple application layer protocols.



## 2.3. Related Work

### 2.3.1. Context-aware frameworks

Currently most of the context-aware frameworks are built using the Java platform. In the current implementations of Java frameworks do not use non-blocking I/O library such as netty, the current frameworks built using Java would have a limitation of handling a higher concurrency because integrating multiple sensors would require an application to be optimized for IO bound tasks which is not available as a built-in feature of Java. JavaScript on the other hand has built in support for non-blocking IO.

Aware framework is one such framework written using Java which has its core feature for data gathering, abstracting out higher-level context, and explain context [36]. This framework also has extensibility features which allows the application to be extended using plugins. The framework has built in data mining and machine learning techniques to abstract out lower level sensor data to higher level context information [36]. Although it has good context acquisition capabilities it doesn't have built in context modeling capabilities. However, it has a support for multiple mobile platforms, and context server approach.

Ambient Dynamix is another context-aware framework written in Java which supports developing web and mobile applications [37]. This framework also doesn't support the context server approach. Ambient Dynamix also support plugins, and supports installing plugins on demand. The plugins enable the developer to integrate various sensing mechanisms, and to provide access to rich contextual information [37]. This framework also allows the developer to express context data using Java instead of using XML format [37]. However, this framework also doesn't provide built in context reasoning capabilities, and preprocessing capabilities.

Java Context Awareness Framework is another Java based service oriented, distributed, event based system used for context awareness [38]. This framework also supports plugins for context acquisition, and functionality for reasoning [38]. This framework provides the developer to model the context using an object-oriented approach, also it enables the developers to specify actuators to respond to contextual changes [38]. This framework doesn't have any built-in machine learning capabilities; however, it provides the developer to transform lower level context into higher level context.



Hydra introduces a middleware based approach for context-aware application which consists of a SOA based framework for the application developers [39]. Hydra also consists of a rule engine for inferencing context, a context acquisition component for acquiring contextual data, and a component to create actuators [39]. Hydra is also extensible and supports rule based configurations for collection, processing and combining contextual information [39]. However, the only limitation is that it doesn't have any capabilities to be ported to mobile and web browsers.

Currently, Hubiquitous is the only a JavaScript framework available for context awareness. This framework is based on the actor model, and operates through passing messages among actors [40]. This framework is can be scaled to bigger sizes, which an actor group runs on containers. Each container runs as a separate node process [40]. When a sensor is required to be integrated, the application developer should create a separate actor for each sensor [40]. The advantage of this approach is that sensors can also be in a distributed environment which makes the application more scalable. However, since it lacks context modeling capabilities, and data transformation capabilities, later the application built using the framework could face maintainability problems.

### 2.3.2. Context-aware systems

[41] Aura focuses on running a set of personal applications used by humans known as the personal aura in different computation devices. It is a task oriented system which manages user's tasks across all the devices smoothly in a context-aware fashion. Aura enables a user to preserve continuity in his work when moving between different environments, and to adapt to on-going computation of a environment. Aura consists of four major components: context observer (collects context and send it to task and environment managers), task manager (also called prism, four different kinds of changes: user moves to another environment, environment, task, and context), environment manager (handles context suppliers and related service), and context suppliers (provides context information).

Gaia [42] is a distributed context infrastructure uncertainty based reasoning. Ontologies are used to represented context information. Gaia has employed a Prolog based probabilistic reasoning framework. The architecture of Gaia consists of six key components: context



provider (data acquisition from sensors or other data sources), context consumer (different parties who are interest in context), context synthesizer (generate high-level context information using raw low-level context), context provider lookup service (maintains a detailed registry of context providers so the appropriate context providers can be found based on their capabilities when required), context history service (stores history of context), and ontology server (maintains different ontologies).

e-SENSE [43] enables ambient intelligence using wireless multi-sensor networks for making context-rich information available to applications and services. e-SENSE combines body sensor networks, object sensor networks, and environment sensor networks to capture context in the IoT paradigm. The features required by context-aware IoT middleware solutions are identified as sensor data capturing, data pre-filtering, context abstraction data source integration, context extraction, rule engine, and adaptation.

COSMOS [44] is middleware that enables the processing of context information in ubiquitous environments. COSMOS consists of three layers: context collector (collects information from the sensors), context processing (derives high level information from raw sensor data), and context adaptation (provides access to the processed context for the applications). In contrast to the other context solutions, the components of COSMOS are context nodes. In COSMOS, each piece of context information is defined as a context node. COSMOS can support any number of context nodes which are organized into hierarchies. Context node is an independently operated module that consists of its own activity manager, context processor, context reasoner, context configurator, and message managers. Therefore, COSMOS follows distributed architecture which increases the scalability of the middleware

Octopus [45] is an open-source, dynamically extensible system that supports data management and fusion for IoT applications. Octopus develops middleware abstractions and programming models for the IoT. It enables non-specialized developers to deploy sensors and applications without detailed knowledge of the underlying technologies and network. Octopus is focused on the smart home/office domain and its main component is solver. Solver is a module that performs sensor data fusion operations. Solvers can be added and removed from the system at any time based on requirements. Further solvers can be combined dynamically to build complex operations.



DMS-CA [46] (Data Management System-Context Architecture) is based on smart building domain. XML is used to define rules, contexts, and services. Further, an event driven rule checking technique is used to reason context. Rules can be configured by mobile devices and push them to the server to be used by the rule checking engine. Providing a mobile interface to build rules and queries is important in a dynamic and mobile environment such as the IoT.

## 2.4. Software Reusability and Software Frameworks

The main quality attribute of a software framework is reusability which is a software development strategy where the development process is geared to reusing existing software [136 p-440]. Software reusability demands for lower software production and maintenance cost, faster delivery of systems and increases software quality. Framework reusability leverages the domain knowledge and prior effort of experienced developers to avoid re-creating and re-validating common solutions to recurring application requirements and software design challenges. Reuse of framework components can yield substantial improvements in programmer productivity, as well as enhance the quality, performance, reliability and interoperability of software. In addition to that it provides the following benefits [136 p-441];

1. Increased dependability:
2. Reduced process risk
3. Effective use of specialists
4. Standards Compliance
5. Accelerated Development

According to Sommerville [136 p-450], object oriented development suggested that objects were the most appropriate abstraction of reuse. However, since objects are more fine-grained and too specialized for a particular application, software frameworks were introduced by creating a higher level of abstraction in order to handle more coarse grained scenarios.

Sommerville [136 p-451] discusses three classes of framework.



1. System infrastructure framework – Support the development of system infrastructures such as communications, user interfaces, and compilers.
2. Middleware integration frameworks – Consists of a set of standards and associated object classes that support component communication and information exchange.
3. Enterprise application frameworks – Concerned with specific application domains such as telecommunications or financial systems. These embed application domain knowledge and support the development of end user applications.

The primary benefits of OO application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers, as described below:

- Modularity: Frameworks enhance modularity by encapsulating volatile implementation details behind stable interfaces.
- Reusability: The interfaces provided by the framework enables the reuse of generic components that can be reapplied when creating new applications by avoiding re-creating and re-validating common solutions.
- Extensibility: A framework enhances extensibility by providing explicit hook methods that allow applications to extend its stable interfaces.
- Inversion of control: This architecture enables canonical application processing steps to be customized by event handler objects that are invoked via the framework's reactive dispatching mechanism. Inversion of control allows the framework (rather than each application) to determine which set of application-specific methods to invoke in response to external events.

#### **2.4.1. Usability of frameworks**

Taking the developer's perspective into consideration, API usability will enable the developers to effectively and efficiently use the solution framework in their context-aware applications. As mentioned in the problem statement developers face issues due the complexity of context-aware applications, increased development time and bug rate due to the complexity. Therefore, simplicity is another area that must be taken into consideration. Even though the solution framework simplifies the process of context-aware application development by hiding the complexities of heterogeneity, performance, and its encapsulated context awareness functionality, if the solution is not usable the developers would not yield



the best results of it. This section of the report aims to identify problematic areas that the developers face when using APIs, and tactics that aims to solve such issues.

Pane et al. [48] had constructed a programming language known as HANDS that will enable children to develop software which provides good points for usability which can be adopted for the solution framework. Pane et al. [48] Further mentions that the von-Neumann computation model which is the most commonly used computation model is an obstacle for beginners because it is unfamiliar and has no real-world counterpart [49]. Therefore, usability could be improved by providing a different model for the computation that is concrete and familiar [50]. Taking that into consideration the solution framework should not expose much technical details for the developer and should enable him to easily specify the context acquisition, context processing and context presentation logic.

Cwalina and Abrams [51] describes the qualities of a well-designed framework from a software engineering perspective and states that frameworks must adhere to quality attributes such as performance, security reliability and dependency management. Because framework provide reusable APIs which enables the developers to develop their applications absence of quality attributes will result in poor quality software. Cwalina and Abrams [51] describes the following quality attributes of a framework;

1. Well-designed frameworks are simple: Frameworks should define its scope to avoid pressure, feature creep, or to satisfy every little corner case scenario in the development process.
2. Well-designed frameworks are expensive to design: Best framework designs are done by the people who are solely responsible for its construction which require a special skill.
3. Well-designed frameworks are full of tradeoffs: Tradeoffs must be made focusing more on the problem the framework aims to solve as well as to understand the alternative options, their benefits and drawbacks.
4. Well-designed frameworks borrow from the past: Appropriate research must be conducted to identify proven designs and best practices in order to construct a solid design.
5. Well-designed frameworks are designed to evolve: Careful consideration can save from degrading overtime, and not being able to preserve backward compatibility. When making tradeoffs focus should be made to determine how the decision will affect the ability to further evolve.
6. Well-designed frameworks are integrated: Integrates well with a large eco-system of different development tools, programming languages, application models.



7. Well-designed frameworks are consistent: Most important factors affecting productivity. Allows for transfer of knowledge between the parts developer is trying to learn. Enables to quickly recognize which parts are truly unique and need special attention.

When designing the solution framework, the above factors must be taken as a guideline to create a framework that is simple, robust, and effective.

When learning and using a framework Ko et al. [52] describes six barriers as follows. Eliminating these barriers would increase the usability of the solution framework.

1. Design barriers: Defined as cognitive difficulties of a programming problem, separate from the notation used to represent a solution. This barrier occurs if the framework is designed in a way that is too complicated or requires dealing with too much of technical aspects. To overcome this barrier, the solution framework should be designed in a way that enables the developer to focus more on context awareness logic rather than focusing on technology.
2. Election barriers: Prevents a developer from finding the best suited interface to achieve a particular behavior. These emerged when learners could not determine which programming interfaces were capable of a particular behavior. To overcome this barrier, the programming interfaces should be given meaningful names and should be well documented.
3. Coordination Barriers: Prevents a developer from combining a programming interface with a programming system to achieve complex behaviors. This barrier can be overcome by the extensibility because it enables the developer to extend the solution framework by using the programming systems' language constructs.
4. Understanding barriers: are properties of a program's external behavior (including compile and run-time errors) that obscure what a program did or did not do at compile or runtime. These emerged when learners could not evaluate their program's behavior relative to their expectations.
5. Use barriers: Are properties of a programming interface that obscure in what ways it can be used, how to use it, and what effect such uses will have. To solve this issue proper documentation should be available for the developer.
6. Information barriers: Prevents a developer from acquiring information about a program's internal behavior. In contrast to the approach of the solution framework which uses JavaScript which is an interpreted language, compiled languages face this barrier. Because



JavaScript is being used to develop the framework it enables the developer to acquire information about the framework's internal behavior.

Even though each barrier is presented individually those barriers interrelate with each other. The following graph shows common path of failures learning Visual Basic [52];

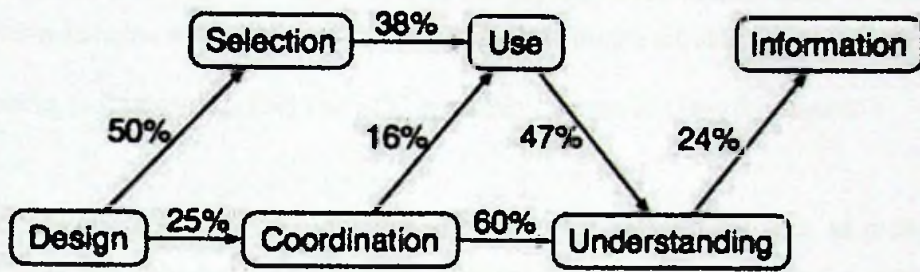


Figure 2.7: Learning barriers for APIs and how they relate each other [52]



#### 2.4.2. Architectural pattern for the framework

To provide the context acquisition, processing and presentation logic in the framework which should be built with an architecture that supports the functionality, as well as its quality attributes such as extensibility and usability which is relevant. The architectural pattern that is selected to construct the framework is Sense/Compute/Control. According to Taylor et al. [53] SCC application is one that interacts with the physical environment which is suited for pervasive domains which makes this pattern applicable for the solution framework.

According to Cassou et al. [54] The SCC application pattern involves four layers;

- Sensors: Entities that retrieve information from the environment such as probes and entities that store previously collected information from the environment (i.e. databases). Sensors send information sensed from the environment to the context operator layer through data sources. Sensors can both push data to context operators and respond to context operator requests.
- Context operators: Refine (aggregate and interpret) the information given by the sensors. Context operators can push data to other context operators and to control operators. Context operators can also respond to requests from parent context operators.
- Control operators: Transform the information given by the context operators into orders for the actuators.
- Actuators: Trigger actions on the environment. Sensors are proactive or reactive components whereas context operators, control operators and actuators are always reactive. These properties ensure that SCC applications are reactive to the environment state that is, all computation is initiated by a publish/subscribe interaction with a sensor



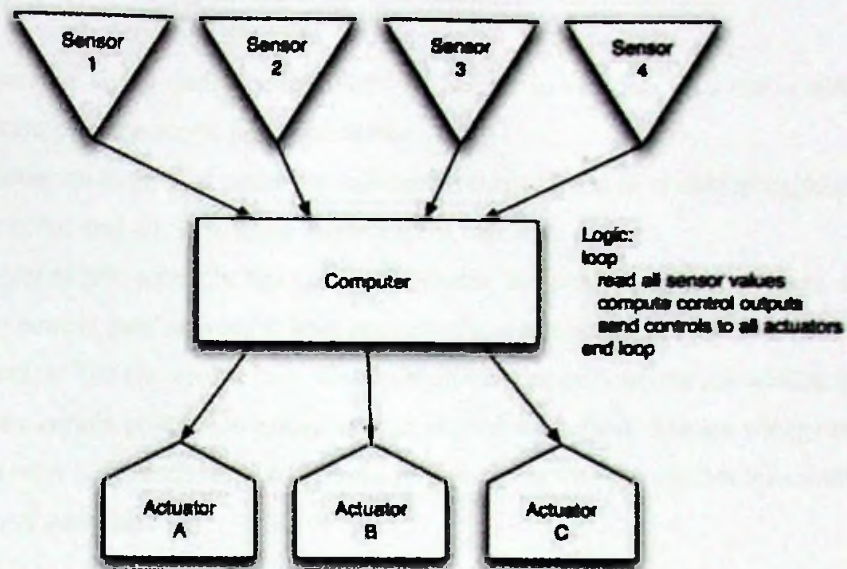


Figure 2.8: Sense compute control architectural pattern [54]

Fitzpatrick et al. [55] present Sentient Object Model which shares similar characteristics to SCC architectural pattern which is defined as a mobile, intelligent software component that can sense its environment via sensors and react to sensed information via actuators. Sentient objects are context-aware, aware of both their internal state and the state of their surrounding local environment. Following figure illustrates a sentient object.

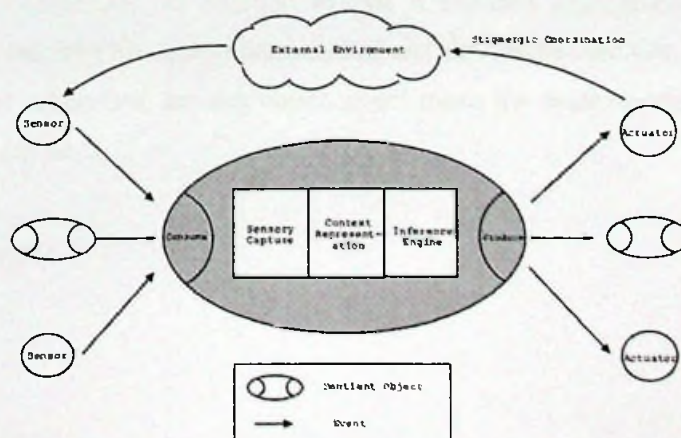


Figure 2.9: Sentient object model architecture [55]



The sentient object model consists of the following components;

1. Sensor: an entity that produces software events in reaction to a real-world stimulus detected by some world hardware device
2. Actuator: an entity that consumes software events, and reacts by attempting to change the state of the real world in some way via some hardware device
3. Sentient object: an entity that can both consume and produce software events, and lies in some control path between at least one sensor and one actuator
4. Inference: The knowledge base of an inference engine contains the knowledge required to solve a certain problem, encoded as a set of production rules. The knowledge encoded in such rules is generally captured from a human expert who can express his expertise in the form of such rules.

Taking the issues and challenges mentioned in pervasive computing which was mentioned previously which are invisibility, integration, heterogeneity, and scalability sentient object model effectively addresses this problem. Invisibility is implemented inside the sentient object using the inference engine, integration occurs in producer and consumer end by communicating with various sensors and actuators, and scalability is achieved by being able to use one sentient object as a producer or consumer to another sentient object. With this architecture extensibility can also be introduced that will enable to extend previously mentioned areas such as extending sensors, communication protocols, data storage, and data preprocessing mechanisms. In addition to that it supports asynchronous, event based communication required for context presentation, and also can be used with ontologies in the inference engine. Therefore, sentient object model meets the requirements for the solution framework architecture.



## **2.5. JavaScript and its Related Technologies**

### **2.5.1. JavaScript programming paradigm**

To develop the solution framework its implementation language which is JavaScript must be evaluated to make the best use of its language features to meet its requirements. Taking the programming paradigm into consideration JavaScript is an imperative, dynamic and object-oriented language [56].

In contrast to other object-oriented languages like Java, C#, or C++ JavaScript consists of a prototype-based object system which each object has a prototype field which refers to another object. If a new object is to be created it clones its prototype by calling its function as the constructor. Property lookup involves searching the current object, then its parent, and its parent until the property is found. This contrast with a class-based object orientation of the above mentioned languages. In addition to that in JavaScript and object can be represented in JSON literal.

The JavaScript object system is extremely flexible. As a result, it is difficult to constrain the behavior of any given object. For example, it is possible to modify the contents of any prototype at any time or to replace a prototype field altogether. This characteristic makes JavaScript a dynamic language.

Taking the imperative characteristic into consideration JavaScript shares the same control structures such as if, for, while that are in the above mentioned programming languages. JavaScript consists of only 5 data types which are String, Number, Boolean, Array and Object.

### **2.5.2. JavaScript design patterns**

To provide the functionality of the framework in terms of functionality, reusability, extensibility and API usability design patterns can be adopted. However, since JavaScript follows a prototype based object orientation paradigm design patterns that are designed for class based object orientation must be implemented by taking that into consideration. The following JavaScript design patterns can be adopted in order to meet the above mentioned criteria of the framework;



### *2.5.2.1. Closure*

The objective of a closure is to define private variables in order provide encapsulation property of object-orientation. Since JavaScript defines objects as functions, a function can return an object which exposes public members while private members can be kept within the function [57]. This pattern is known as a closure. In our solution framework closures can be used to create various components that encapsulates various functionality and connectors can be defined as public methods. In addition to that this approach will enable to structure the code in the solution framework.

### *2.5.2.2. Callback*

This pattern enables to deal with discontinuous events. Since asynchronous communication is used intensively for scenarios such as responding to contextual changes, and to collect sensor data [57]. This approach enables to implement the sentient object architecture in JavaScript.

### *2.5.2.3. Cascade*

If a method returns the same object which the method is being called, the cascade pattern can be implemented. In a cascade many methods on the same object is being called in a sequence in a single statement [57]. Cascade pattern enables to increase API usability by making it easier to use the framework by enabling them to specify the context acquisition logic, define ontologies, and to define context presentation logic. This approach also eliminates some of the previously mentioned API usability barriers such as design and election. In addition to that the developers could focus on simplicity and in the meantime, they could focus more on the context awareness logic. An example of the cascade pattern is as follows;

```
getElement('myBoxDiv').move(350,150).width(100).height(100).color('red').border('10px outset').padding('4px').appendText("Please stand by").
```

### *2.5.2.4. Observer*

This pattern is used to handle events which is also known as the subscriber/publish pattern which the subscriber is known as the observer while the object which is being observed for the event is called publisher or the subject [58]. This pattern can be used for context presentation



because the client or the publisher can subscribe to a particular context change and respond accordingly.

#### *2.5.2.5. Mediator*

This pattern enables to loosely couple many objects as well as to improve the maintenance of the code by defining a mediator object that prevents objects from communicating with each other directly [58]. When one of the object change its state, it notifies the mediator. Taking the solution framework into consideration this pattern can be used to construct the sentient object which will consist of the context model, which will be used for reasoning by responding to events which occur from the environment.

### **2.5.3. JavaScript Development Stack**

#### *2.5.3.1. Node.js*

Node.js is a platform that is built on Chrome's V8 JavaScript runtime for building scalable network applications [59]. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, which makes it perfect for distributed and real-time applications which meets our criteria of the solution framework.

The advantage of non-blocking I/O in Node.js is that it enables to handle large amounts of concurrency [60]. In traditional model of concurrency, a new thread is created to address the issue of blocking one thread when it waits for I/O, the drawback of this approach is that due to the resource constraint threads cannot be created for every connection. Node.JS addresses this problem by providing an asynchronous model to prevent blocking I/O operations.

Although high equipped servers with high-end hardware are available with the traditional model of concurrency the system would not meet the real-time performance requirements [59]. In addition to that Node.JS provides following benefits;

- Asynchronous - JavaScript is naturally asynchronous with event model well suited for building highly scalable web applications through callbacks.
- Less Learning curve - A huge base of developers is already familiar with both JavaScript and asynchronous programming from years, developing JavaScript in web browsers



because the client or the publisher can subscribe to a particular context change and respond accordingly.

#### *2.5.2.5. Mediator*

This pattern enables to loosely couple many objects as well as to improve the maintenance of the code by defining a mediator object that prevents objects from communicating with each other directly [58]. When one of the object change its state, it notifies the mediator. Taking the solution framework into consideration this pattern can be used to construct the sentient object which will consist of the context model, which will be used for reasoning by responding to events which occur from the environment.

### **2.5.3. JavaScript Development Stack**

#### *2.5.3.1. Node.js*

Node.js is a platform that is built on Chrome's V8 JavaScript runtime for building scalable network applications [59]. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, which makes it perfect for distributed and real-time applications which meets our criteria of the solution framework.

The advantage of non-blocking I/O in Node.js is that it enables to handle large amounts of concurrency [60]. In traditional model of concurrency, a new thread is created to address the issue of blocking one thread when it waits for I/O, the drawback of this approach is that due to the resource constraint threads cannot be created for every connection. Node.JS addresses this problem by providing an asynchronous model to prevent blocking I/O operations.

Although high equipped servers with high-end hardware are available with the traditional model of concurrency the system would not meet the real-time performance requirements [59]. In addition to that Node.JS provides following benefits;

- Asynchronous - JavaScript is naturally asynchronous with event model well suited for building highly scalable web applications through callbacks.
- Less Learning curve - A huge base of developers is already familiar with both JavaScript and asynchronous programming from years, developing JavaScript in web browsers



- Fast Script engine – Huge advances in execution speed has made it practical to write server-side software entirely in JavaScript.
- Code Transformation - JavaScript is a compilation target and there are many languages that have compiled to it already.
- Support for NoSQL - JavaScript is the language used in various NoSQL databases (i.e. CouchDB, MongoDB) so interfacing with them is a natural fit.

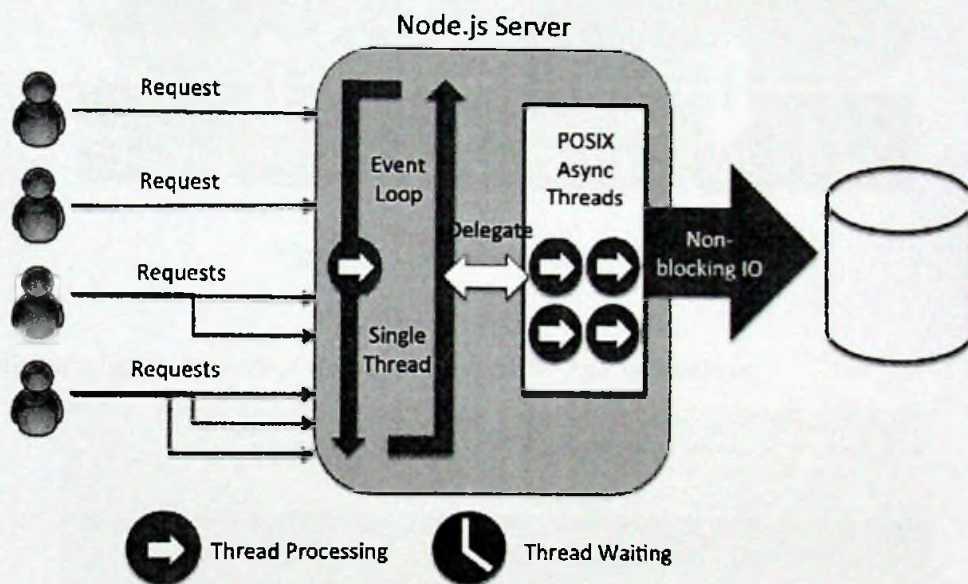


Figure 2.10: Non-blocking I/O feature of Node.JS [59]

### 2.5.3.2. Apache Cordova

Apache Cordova enables a developer to develop his mobile applications using HTML and JavaScript and cross-compile to any mobile platform such as Android, or Windows Mobile. Apache Cordova is also known as PhoneGap. Apache Cordova consists of interfaces that allows the mobile app developer to access native device function such as the camera or accelerometer from JavaScript [61]. Apps built using Cordova are still packaged as apps using the platform SDKs, and can be made available for installation from each device's app store.

The following figure illustrates the architecture of Apache Cordova;



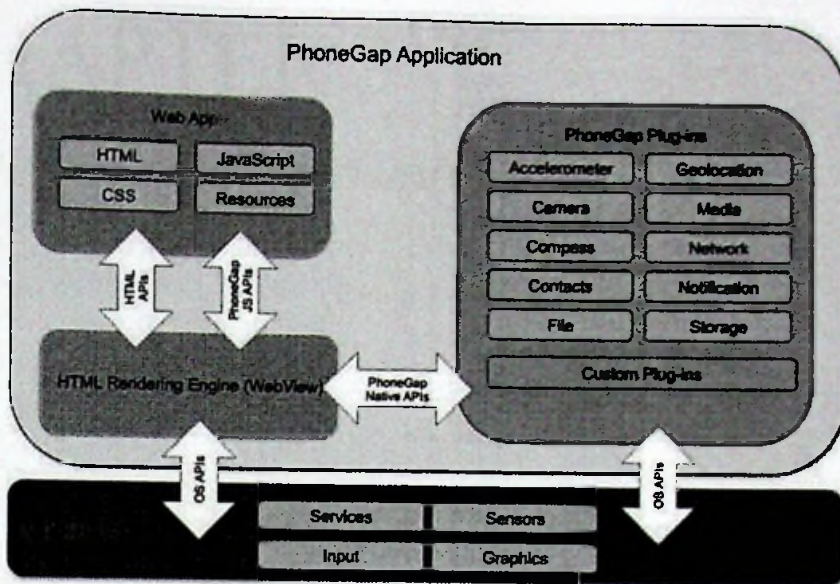


Figure 2.11: PhoneGap architecture [61]

Following figure illustrates the platform support for Apache Cordova;

	amazon- fireos	android	blackberry10	Firefox OS	ios	Ubuntu	wp8 (Windows Phone 8)	windows (8.0, 8.1, Phone 8.1)	tizen
Platform APIs									
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
BatteryStatus	✓	✓	✓	✓	✓	✗	✓	✗	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture	✓	✓	✓	✗	✓	✓	✓	✓	✗
Compass	✓	✓	✓	✗	✓ (GSM)	✓	✓	✓	✓
Connection	✓	✓	✓	✗	✓	✓	✓	✓	✓
Contacts	✓	✓	✓	✗	✓	✓	✓	partially	✗
Device	✓	✓	✓	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✗	✓	✓	✓	✓	✓
File	✓	✓	✓	✗	✓	✓	✓	✓	✗
File Transfer	✓	✓	✗ Do not support onprogress nor abort	✗	✓	✗	✗ Do not support onprogress nor abort	✗ Do not support onprogress nor abort	✗
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Globalization	✓	✓	✓	✗	✓	✓	✓	✗	✗
InAppBrowser	✓	✓	✓	✗	✓	✓	✓	uses iframe	✗
Media	✓	✓	✓	✗	✓	✓	✓	✓	✓
Notification	✓	✓	✓	✗	✓	✓	✓	✓	✓
Splashscreen	✓	✓	✓	✗	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✗	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓
Vibration	✓	✓	✓	✓	✓	✗	✓	✗	✗

Figure 2.12: Platform support for Apache Cordova [61]



# CHAPTER 3

## DESIGN AND IMPLEMENTATION



### **3.1. Introduction**

The objectives of this chapter are to present the requirements, architecture, and detailed design of the framework. The requirements are based on the concepts of context awareness, extensibility, portability and usability of the framework as presented in the literature review.

The first section of the chapter presents the requirements that are identified in the framework which includes functional and non functional requirements such as portability, extensibility and usability. In addition, it presents the functionalities that are implemented in the framework.

Next section focuses on the architectural design based on the requirements presented in the previous section. The section also presents the design decisions behind the architecture. In addition to that it also presents an implementation of the architectural baseline which consists of various components that facilitates the development of the framework. The detailed design is also presented in the following section which includes more fine-grained components that are available in the framework. The section also presents various mechanisms that are implemented to achieve the functional and non-functional requirements of the framework.

Finally, this chapter presents the coding syntaxes that are available for the developer that provides the functionality presented in the beginning of the chapter.

### **3.2. Requirement Analysis**

#### **3.2.1. Requirements of the framework**

Taking the problem background of this research into consideration the framework should consist of the following scope;

- The solution framework should consist a framework with a mechanism for the developers to easily map real world entities into the framework, define rules for inferencing context, and to integrate technical components such as sensors, transport mechanisms, and storage mechanisms as extension.
- The framework should not include of any pre-built extension.



- The framework should not implement any domain specific logic.
- The framework should implement the 'invisibility' characteristic of pervasive computing.
- In addition to that to make the framework usable it should implement the characteristic 'effective use of smart spaces' in context awareness, and to make the framework capable of integrating with various sensing mechanisms the characteristic 'heterogeneity' should be implemented.

In order to implement the framework within the above scope, the framework should consist of the following functional and non-functional requirements;

### **Functional Requirements**

- The framework should enable the developers to model real world entities using ontologies.
- The framework should enable the developers to specify rules to inference context.
- The framework should enable the developers to map sensors to ontologies.
- The framework should enable the developers to respond to contextual changes.
- The framework should enable the developers to preprocess raw sensing information in to higher level contextual information.
- The framework should allow the developers to integrate various technologies (i.e. sensors, transport mechanisms, and storage mechanisms) through extensions.

### **Non-Functional Requirements**

#### **Portability**

- The framework should be portable among heterogeneous platforms such as node.js, apache Cordova, and the web browser.

#### **Usability**

- The interfaces provided by the framework should be consistent, and meaningful.
- The framework should be easy to learn by the developers.





- The framework should reduce the effort for the developers to develop context acquisition, context process, and context presentation mechanisms for their applications.

### **Performance**

- The framework should be optimized for I/O bound tasks to make it capable of integrating with multiple sensors.
- The framework should be able to handle a higher concurrency for multiple users and sensors.

### **Maintainability**

- The developers should be able to monitor the internal workings of the framework.
- The developers should be able to apply industry standard patterns and practices such as SOLID principles with the framework.

### **Extensibility**

- The framework should enable the developer to develop extensions for sensors, transport mechanisms, and storage mechanisms.
- The framework should enable the developers to integrate extensions developed by other developers.
- The framework should enable the developers to loosely couple business logic and technical components.



### 3.2.2. Use case scenarios

Since the end user of perception.js are developers who develop context-aware applications the functionality that the framework should provide features for the developers to address the previously described factors. The following use case diagram describes the functionality of the framework;

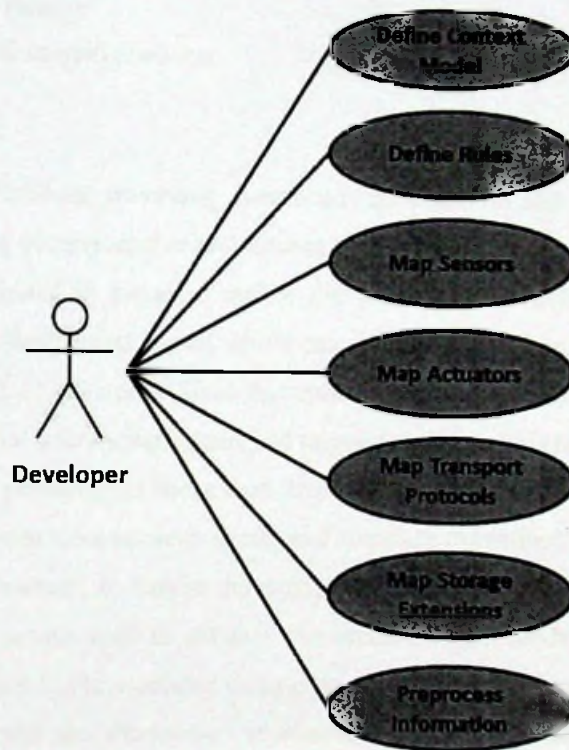


Figure 3.1: Use case diagram of Perception.js

- **Define context model:** Enables the developers to define and model the context using ontologies.
- **Define Rules:** Enables the developers to define rules for the ontologies in order to deduce contexts which can be used for context presentation.
- **Map Sensors:** Enables the developer to integrate sensors to the framework as extensions and map such extensions to the ontology.
- **Map Actuators:** Enables the developer to integrate actuators and map such actuators to the ontology.



- **Map Transform Protocols:** Enables the developer to integrate transport protocols to communicate between client and context servers and map such extensions to ontologies.
- **Map Storage Extensions:** Enables the developer to integrate storage mechanisms to store contextual information and map such extensions to ontologies.
- **Preprocess Information:** Enables the developer to preprocess the information obtained from the sensors in order to handle sensor imperfections.

### 3.3. Architectural Design

#### 3.2.1. Architectural considerations

In order to fulfill the previously mentioned functionalities and goals, this chapter is focused on creating a comprehensive architecture that satisfies the core concepts of context awareness. As mentioned in literature review the reference architecture selected for this framework is the sentient object model which can address requirements related to the core functionalities of context awareness. Since the sentient object model can interact with sensors, process with its internal inferencing engine, and respond to contextual changes using actuators this suits well for the perception.js framework. In addition to that this architecture can be used to communicate between more sentient objects and distribute the context acquisition logic and processing logics. However, to handle the extensibility mechanisms of the framework the layered architecture is also used to enhance the sentient object model because it aims to identify the areas which can be extended using plug ins. The resulting architecture is a hybrid of sentient object model and the layered architecture which is illustrated in the following diagram.

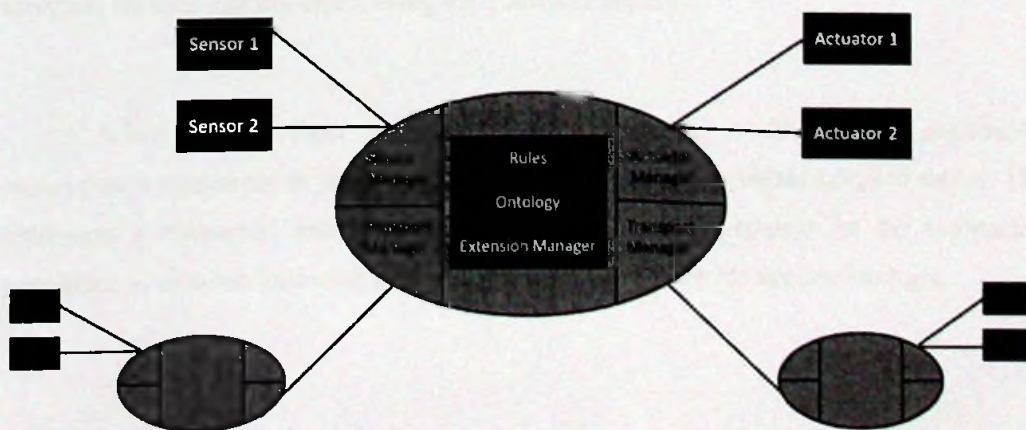


Figure 3.2: Architecture of Perception.js



A single hybrid sentient object in perception.js consist of the following components;

- Actuator Manager which will trigger related events based on contextual changes.
- Ontology Manager which enables the developer to model the context and represent as an ontology
- Rule Manager which will define rules for various entities in the context model which enables to deduce a context when a set of rules are satisfied.
- Extension Manager which enables integrating and bootstrapping of extensions.
- Sensor Manager which instantiates sensors of a specific sensor plug-in type and map such instances to ontologies.
- Transport Manager which enables to communicate between sentient objects and map such transport extensions to ontologies.
- Storage Manager which will store contextual information in every value update of a particular ontology property.

When a developer develops an application using perception.js he could transform his application to a sentient object. First the framework enables the developer to define and model the context as an ontology which can be done using the Ontology Manager, and define the rules using the Rule Manager which determines the rules on how to respond to various contexts. Next the developer could integrate sensors using the Sensor Manager and map those sensors to various properties in the ontology class. Finally, the developer could specify how to respond to contextual changes. If the developer prefers to use a context server approach he could integrate another sentient object using the Transport Manager.

When the developer uses the above components to develop his application, perception.js represents its internal information to the following object-oriented model. This represents a conceptual model when a developer models his context for the application, perception.js uses the following internal representation to keep his application logic.



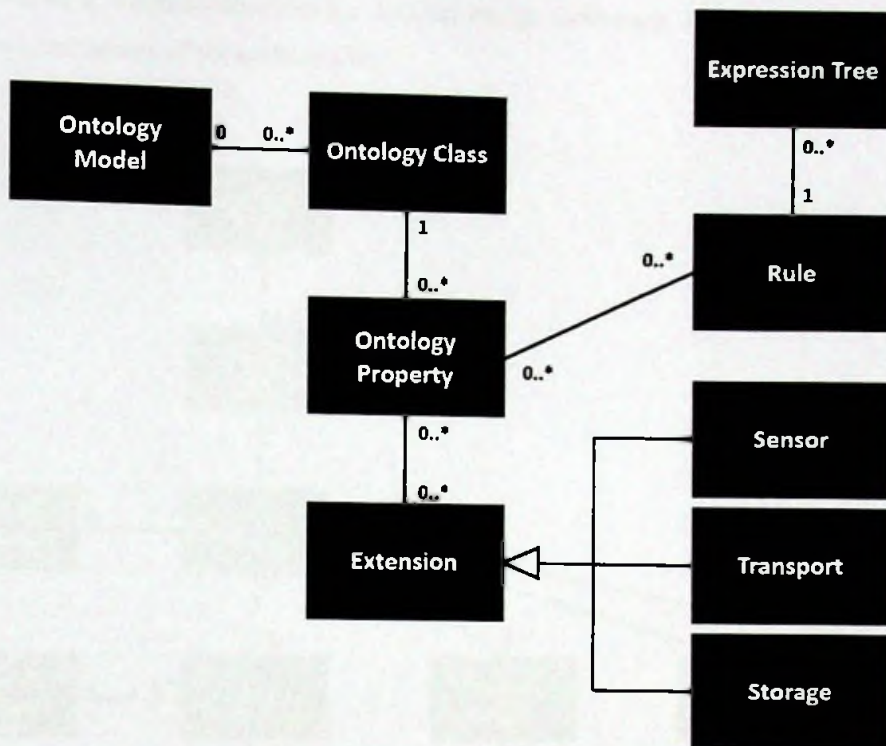


Figure 3.3: Internal representations of Perception.js

Once the ontology, its classes and its properties are defined using the Ontology Manager, it will create an instance of Ontology Model. When the rules are defined using the Rule Manager it will map properties of ontologies to Expression Trees. In addition to that each extension can be mapped to a property of an ontology. Sensors extensions update the values of properties of ontologies, whereas storage extensions stores data when a value of a mapped property changes, and the transport extension can be configured to transport data if a value of an ontology property changes or update an ontology property of it receives values.



The following diagram illustrates the detailed design combining the internal representation and the components of the architecture;

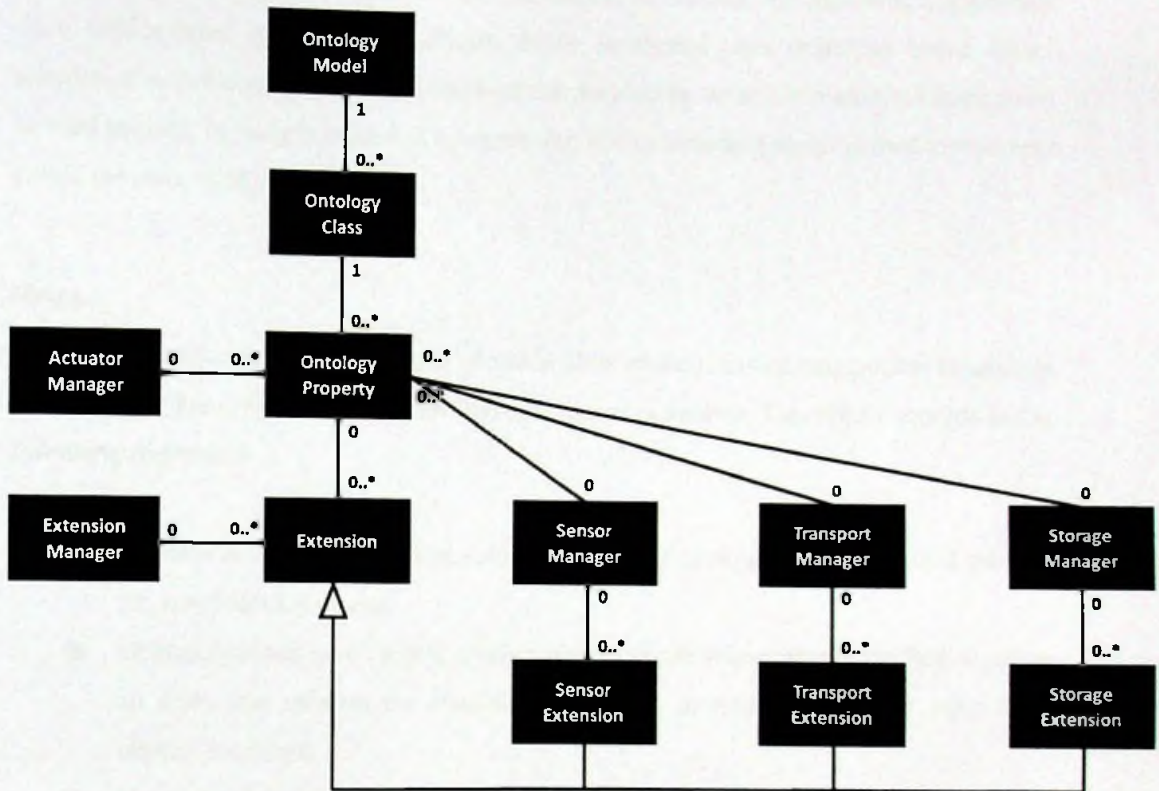


Figure 3.4: Class diagram of Perception.js



### 3.2.2. Implementation of the helper functions

To facilitate the development of the core functionalities of the framework a collection of helper functions was developed for the framework. As a result, the following components were implemented as JavaScript objects. Since JavaScript uses prototype based object orientation which doesn't provide classes for the developers, as an alternative a function can be used as class, its body is treated as a constructor, and its returning object is used to represent public properties [58].

#### Helper

The helper JavaScript object consists of reusable library functions that manipulates JavaScript programming features to make the development more convenient. This object consists of the following methods;

- **FindByField:** given a key and value, returns the first object in an array that satisfies the condition key=value
- **DefaultByField:** given a key, a value, and a default value, returns the first object in an array that satisfies the condition key=value, or return the default value if no objects are found.
- **NewAsyncIterator:** creates a new instance of async iterator class

#### Async Iterator

This JavaScript object enables the developer to iterate through an array and perform a custom defined operation in each iteration which is similar to an asynchronous version of the foreach loop.

#### Logger

Enables the developer to add debugging logs for the application which enables the developer to keep log traces of internal working of perception.js.



## **Event Manager**

As mentioned in Literature review, perception.js uses event driven architecture to coordinate between sensors and actuators, the EventManager object implements handles all the events in the perception.js framework. EventManager object implements the observer pattern which uses a pub/sub style event management.

## **Configuration Manager**

Configuration Manager enables the framework to store configurations for each component, or extension (i.e. sensors, transport protocols, or storage mechanisms). The application developer could use this class to manage the configuration in the application level.

## **3.4. Detailed Design and Implementation**

### **3.4.1. Portability**

Perception.js is capable of being embedding in multiple execution environments such as a web browser, a mobile application, or a node.js server. Since a context-aware application would require running in multiple platforms coding in different languages or platforms is a counterproductive approach when developer's perspective is taken into consideration. For example, web and mobile platforms would provide good sensing capabilities while node.js provide good context processing capabilities in the backend. Taking this fact into consideration perception.js framework was written as a single JavaScript file that can be ported among different platforms which eliminates the need to using multiple technologies for development. As a result, the development complexity is reduced.

Since the framework codebase is written in a single JavaScript file, in runtime the framework requires to identify its execution environment when initializing. The execution environment is identified in the following ways;

1. In node.js the variable 'module' is available in runtime, if it's available perception.js identifies the execution environment as node.js.
2. In web browser, the variable 'document.URL' is available.
3. If any of the above conditions are not satisfied perception.js identifies its execution environment as Apache Cordova.



### 3.4.2. Bootstrapping process

The bootstrapping process in perception.js framework is used to initialize the framework and its extensions when the application completes loading. In perception.js each extension is implemented in a separate file. In addition to that perception.js can be deployed in different platforms such as node.js, apache Cordova, and in the web browser. for different platform, different types of bootstrapping mechanisms are required.

In the web browser, perception.js is imported through the script tag in a particular web page. Similarly, the extensions of perception.js are required to be imported using the script tags. When perception.js is loaded in a web page it creates a top-level variable in DOM (in window object) known as perception. When each plug in is imported through the script tag it will use the top level 'perception' object's register method to integrate with perception.js. The same bootstrapping approach is used in when perception.js is used by a mobile application that runs on Apache Cordova.

In node.js the framework uses a different approach. node.js has the ability to keep modules a separate file which can be imported using a method known as 'require'. This approach is known as CommonJs module format. In the bootstrap process for node.js the framework uses that feature to import the module, once it's imported it will register in perception.js framework.



### 3.4.3. Extensibility

As mentioned in literature review, perception.js uses extensibility to address the heterogeneity in pervasive computing which benefits a context-aware application because it enables the developer to extend the framework with multiple technologies. In addition to that the layered architecture model was selected to identify the areas that can be extended using extensions, which are sensors, storage, and transport protocols.

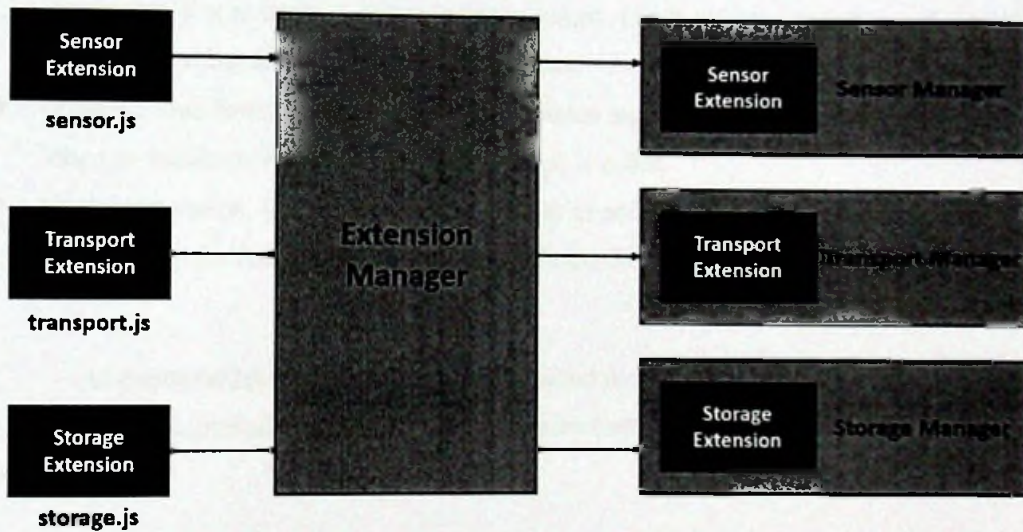


Figure 3.5: Integrating extensions

when each extension calls the register method in the Extension Manager JavaScript object which handles the lifecycle of each extension. Once each extension is registered, each extension is allocated to its relevant JavaScript controller object. This way sensor extensions are allocated to the SensorManager object, storage extensions are allocated to StorageManager object, and transport extensions are allocated to TransportManager object.



### 3.4.2.1. Writing custom extensions

To write a custom extension for Perception.js framework, first the developer must write the code to register the extension in perception.js. As mentioned previously in node.js the developer can use the 'perception' object that is available in the 'global' object whereas in web/mobile the developer could use the 'perception' object in available in the 'window' object. For the registration process the developer must specify the following fields in the input object required for the register () method;

- category: transport/sensor/or storage which is based on the type of the extension.
- class: represents a unique name that can be used to instantiate extensions based on the class name. For example, a sensor instance named 'OutdoorTemperatureSensor' can be instantiated using the 'TemperatureSensor' class.
- create (): This function should handle the creation logic of the extension. Every time when an extension is instantiated, this function is called.
- name, description, author: these fields consists of additional information which doesn't contribute to the functionality but to keep information about the extension.

As mentioned previously, the create () method is used to instantiate extensions. When an extension is instantiated as an object, the extension instance should consist of the following functions;

- play (): Activates an instance of the extension. For example, sensor extensions will start getting inputs, transport extensions would open a port or establish connections, whereas storage sensors would create a connection to a database.
- pause (): Temporarily stop the activity of the extension.
- stop (): Destroys the instance of the extension.
- onPlay (): An event listener that enables the outside object to track if the sensor has been started.
- onPause (): An event listener that enables the outside object to track if the sensor has been paused:
- onStop (): An event listener that enables the outside object to track if the sensor has been stopped.



For sensor extensions, the instance should consist of the following callback method;

- `onResult()`: An event listener that enables the outside objects to obtain inputs from the sensor extension.

For transport extensions, the instance should consist of the following fields;

- `onRecieve ()`: An event listener that is used to obtain the received data for the transport extension.
- `send ()`: Used to send information for the client or the server
- `type`: server/or client

#### *3.4.2.2. Messaging format for transport extensions*

To communicate between different sentient objects, `perception.js` uses a specific messaging format in transport extensions. This approach is useful when the sensor information is gathered from the client side and being sent to the context server for processing. the message format is JSON and consists of the following fields;

- `fullname`: consists of the full name of the ontology class or the property
- `value`: value that is required to be set in an ontology property
- `timeout`: If the information requires to be expired after a particular value, `timeout` field can be used.
- `revert`: Optionally this field is used to once the timeout is reached, and requires the value of the ontology property to be changed back to a specific value



### 3.4.4. Context representation

Perception.js enables the developer to model the context through the framework. As identified in literature review the most effective approach is to use ontologies to model the context because ontologies also make it easier for the developer to map real world to the framework. Therefore, ontologies were selected to model the context using the framework. Wang et. al [31] has presented an ontology that can be used for context-aware applications known as CONON.

In addition to that this approach also enables the developer to easily map extension instances to elements of the ontology model (i.e. a temperature sensor instance can be mapped to update the temperature field of the room class).

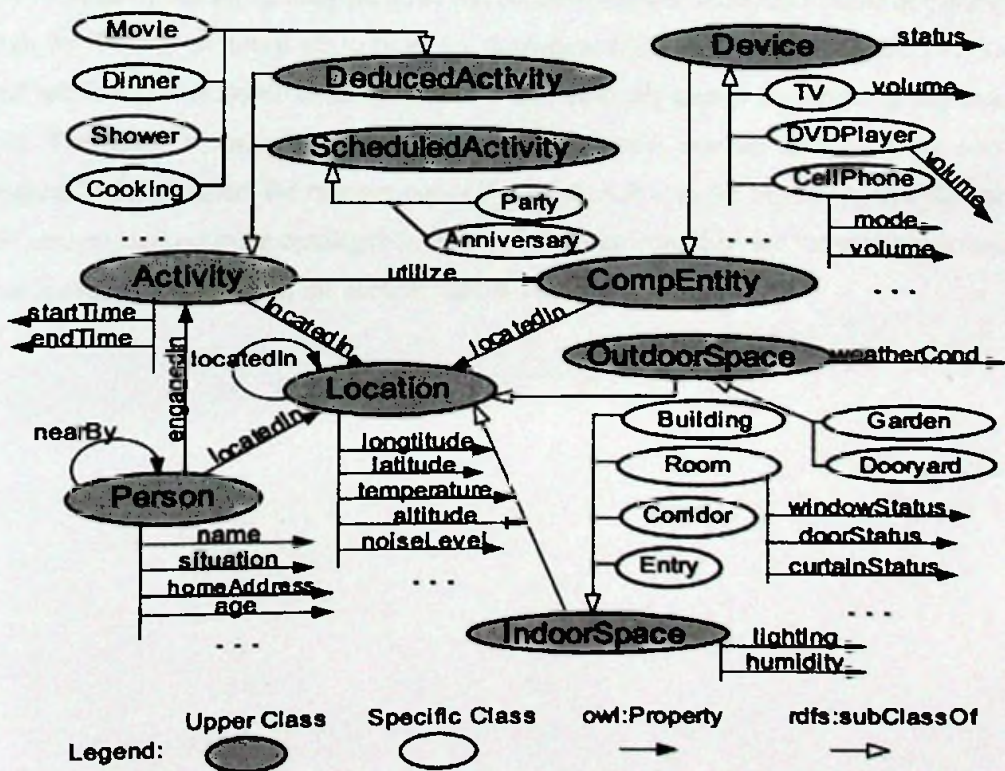


Figure 3.6: Default ontology in Perception.js [31]

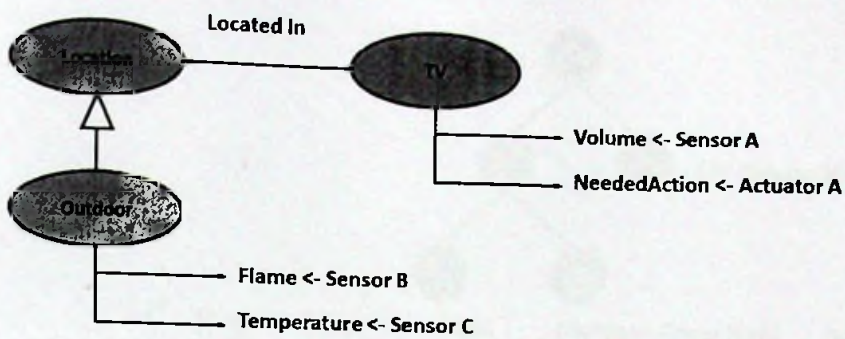


### 3.4.5. Context processing

When a context model is available as an ontology to deduce the context, a mechanism should be available in the framework for the developer to specify rules. From the coding perspective, the RuleManager JavaScript object serves this purpose. In the code, a single rule can be specified in a method chaining mechanism, which the last method should consist of the action that must be taken if all the rules in the chain are satisfied.

Once the rules are defined by the developer internally perception.js creates one or more expression trees based on the context model. The following example illustrates such a scenario. In the following ontology, the TV is located in a particular indoor location and the volume of the TV is obtained by Sensor A (assuming that the TV is a smart TV that could provide its volume through an API), and the outdoor flame measured by Sensor B, and temperature by Sensor A. Suppose a fire has occurred outdoor while the volume of the TV is high the TV can be turned off to avoid the distraction of the person watching the TV. Once that rule is set using the RuleManager perception.js internally creates the following expression tree. The sensor instance is mapped to the expression trees relevant node when the sensor updates its information, the relevant node (The letters A,B,C in the expression tree represent the sensors mapped to the ontology) in the expression tree is notified and the whole expression tree is evaluated to check if the context logic is satisfied.





$(\text{Outdoor Temperature High}) \wedge (\text{Outdoor Flame High}) \wedge (\text{TV Volume High}) \rightarrow (\text{TV NeededAction TurnOff})$

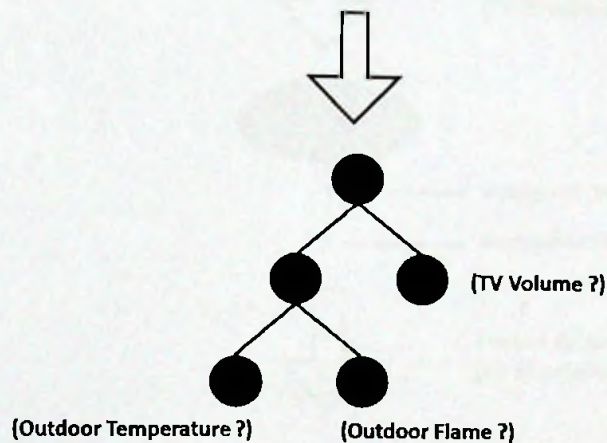


Figure 3.7: Internal representation of contexts using expression trees

When the expression tree is evaluated to true (all the nodes are evaluated to true), a value of an ontology property is updated. In perception.js an actuator can be set to monitor the value of an ontology property. In the above example, the Actuator A monitors the 'NeededAction' property of the TV class. When its value is set to off, the relevant actuator is triggered which will turn off the TV. This mechanism is illustrated in the following figure;



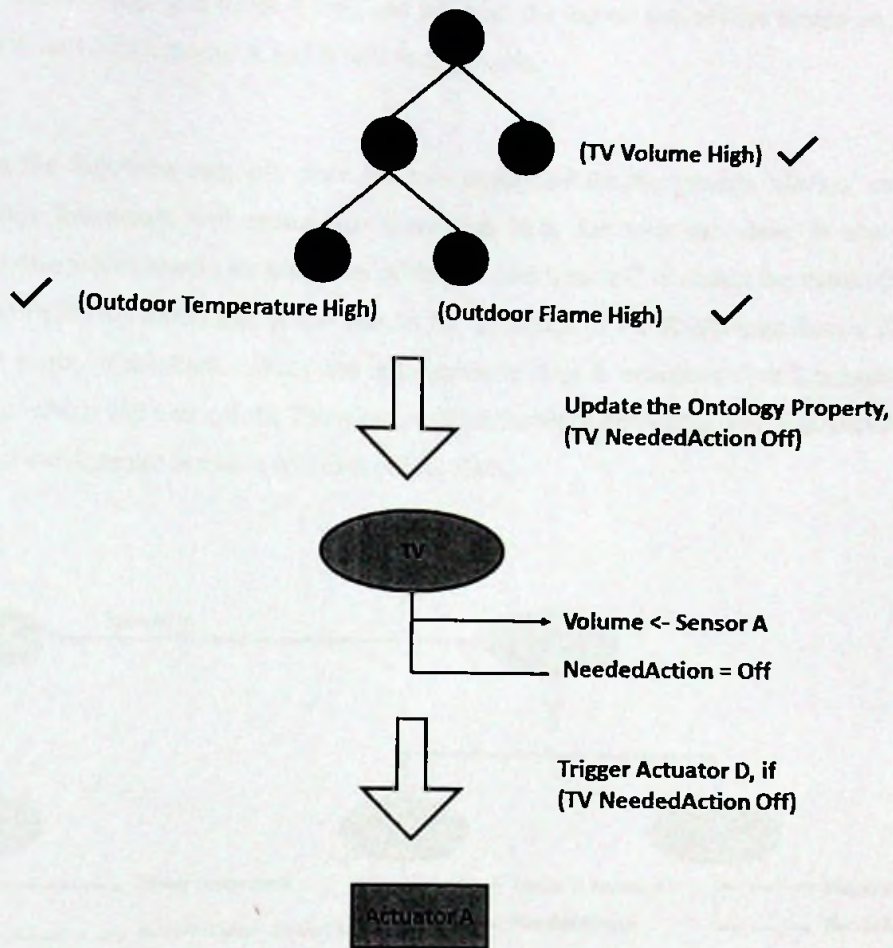


Figure 3.8: How actuators are triggered

Since ontologies consist of subclasses, and if a rule is created for a parent class which has two sub classes it will spawn two expression trees for each sub class. This approach enables two sensor instances to be mapped for each expression tree. When sensors provide information, the expression tree will evaluate once the mapped ontology is changed, and once the expression tree is evaluated to true it will trigger the required action.

The above mechanism is illustrated in the following example which modifies the previous example by replacing the TV ontology class by a more generic 'device' class which is located in the indoor space. Its subclasses TV, and Radio are monitored for their statuses using sensors (assuming the devices are smart devices). In the situation of an outdoor fire when



the temperature is high and flame is high and when all the indoor devices are turned on, the devices will turn off (Actuator A and B will be triggered).

In the following example since the rule is applied for the generic 'device' class, perception.js framework will create two expression trees for each sub class. In the left expression tree which checks for the status of the TV uses Sensor C to obtain the status of the TV. In the right expression tree which checks for the status of the Radio uses Sensor D to obtain the status of the Radio. When the left expression tree is evaluated it will trigger the Actuator A which will turn off the TV whereas when the right expression tree is evaluated it will trigger the Actuator B which will turn off the Radio.

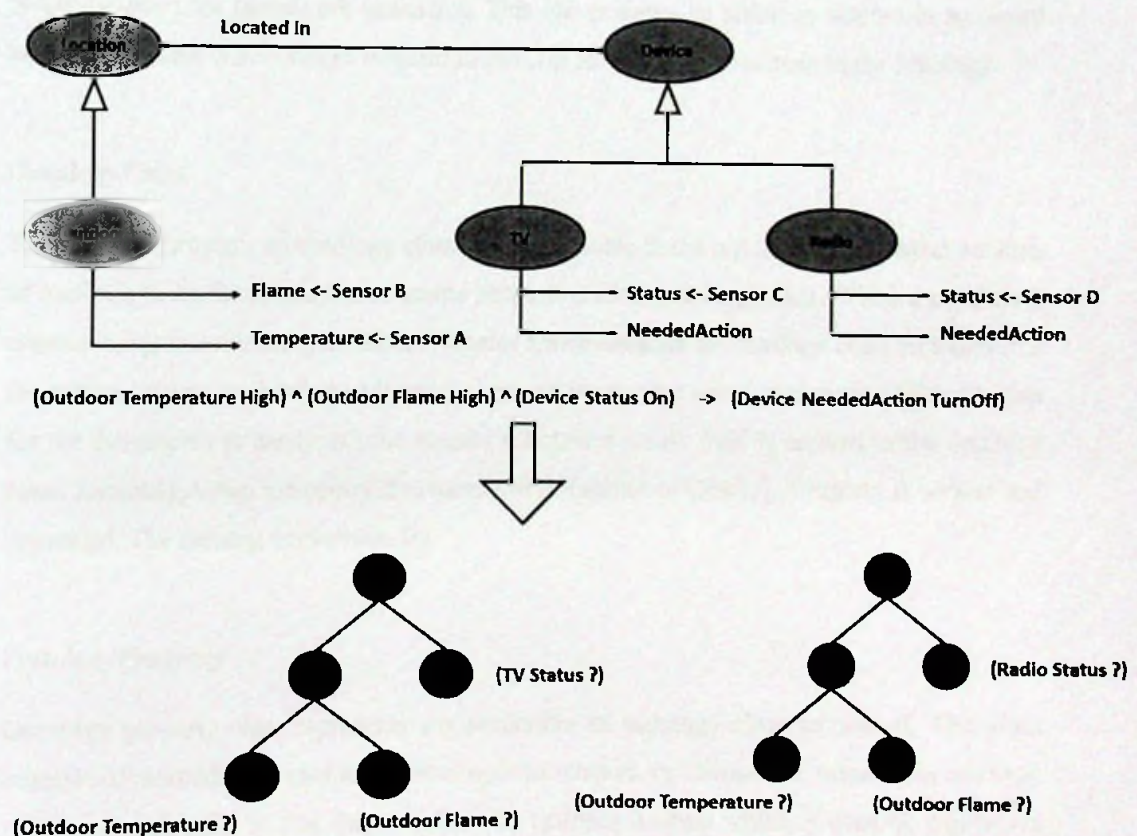


Figure 3.9: How expression trees are created for subclasses



### *3.4.5.1. Internal Implementation of Context Processing*

#### **3.4.5.1.1. Internal Representation of the context model**

To represent the context internally the following three JavaScript objects are used in perception.js;

#### **OntologyManager**

The ontology Manager enables the developer to model the context based on the CONON ontology and enables the developer to lookup for any class available in the ontology. Since CONON ontology includes default classes, the Ontology Manager class will create a default ontology when the framework initializes. This object keeps its ontology classes in an object known as classes which makes it easier to look up for any class available in the ontology.

#### **OntologyClass**

This object represents an ontology class that is available in the ontology. This object consists of methods to create subclasses or create properties of an ontology class. When a subclass is created using its subclass () method, it creates a new instance of Ontology class and adds it to the 'classes' object in Ontology Manager class and returns the new instance. In addition to that for the developers to easily use the created subclasses a new field is created in the ontology class. Similarly, when a property is created a new instance of OntologyProperty is created and appended. The naming convention for

#### **OntologyProperty**

Ontology property class represents the properties an ontology class consist of. This class consists of methods that enable the developer to retrieve, or change the value of an ontology property. In addition to that this consists of a callback method which is used by expression trees to identify whether the value of the property has been changed by a sensor input.

#### **Scheduler**

Since contextual information has a timeliness property which ensures that contextual information is valid only for a particular amount of time, when setting a value for an ontology property, an expiry time could be set in order to ensure that information is valid for a particular



amount of time. the Scheduler class provides features to set a value of a particular ontology property and expire its value and revert to a defined value after a specific amount of time.

#### 3.4.5.1.2. Internal Representation of context processing

##### **RuleManager**

This JavaScript object enables the developer to specify rules that can be used for inferencing purposes to identify the required action to be taken for a particular context. this class consists of methods to use logical operators such as 'and', and 'or'. The rules can be specified in a method chain in JavaScript. When the developer creates a single rule set, initially an expression tree is created, and as the developer keeps on adding rules the expression tree will get expanded or cloned depending on the ontology classes a developer would use. The last method in the method chain should consist of a 'then' method which will consist of the required action a use should take in order to respond to the contextual change if all the rules in the chain are satisfied.

##### **ExpressionTree**

This class is used to represent an instance of an expression tree. This class consists of methods that enables to add binary expressions, to clone the same expression tree to an identical one, and callback methods that will enable the developer to identify whether the expression tree has been evaluated to true.

##### **BinaryExpression**

The BinaryExpression object consists of a binary node in an expression tree which has a logical operator (i.e. and/or/not), and left and right child nodes. Each child node could consist of either a unary operator or a binary operator. This consists of methods to evaluate a binary expression, clone the same expression object, and to evaluate whether the binary expression is set to true. When the binary expression is being evaluated, it will evaluate all its child nodes to check whether the condition satisfies to true. In addition to that each expression tree has its root node as a binary expression.





## **UnaryExpression**

This object represents a unary expression which consists of a single value. This is often a child node of a binary tree. This has callback method to trigger if its value has been changed. The value of a unary expression object is always gets its input from an ontology property object.

### **3.4.7. Context acquisition and presentation**

As mentioned previously the sensor extensions are responsible of acquiring contextual information. Once the context is modelled using an ontology, sensor extensions can be mapped to a relevant ontology class's property. SensorManager class in perception.js is responsible for this feature. Sensor manager enables the developer to create an instance from a sensor extension and map its inputs to an ontology property. Since sensors provide raw data mapping such data into higher level contextual data would require preprocessing. Perception.js framework enables the developer to provide a transformation function that maps low level sensor data into high level contextual information.

The create () method is responsible for creating an instance of a sensor, the map() method is responsible for mapping the created instance to an ontology property, whereas the preprocess function is responsible for preprocessing raw sensor data. These methods can be used as a method chain.

Context presentation in perception.js can be handled using actuators. Actuators in perception.js always listen to changes for an ontology property. When a developer specifies a rule, he would typically change the ontology property 'needed action' of a person class. When a developer requires to track whether the 'needed action' property has been changed, he could create an actuator to listen changes of that ontology property. The class ActuatorMapper enables serves this purpose. it consists of two methods when () which will listen for a particular condition, and then() when will have a callback function to respond to any contextual changes.



### **3.5. Integrating with Existing Technologies**

Perception.js enables the developers to integrate with existing and widely used tools and technologies to the context-aware application. Existing tools and technologies can be used for acquiring, storing, and processing contextual information.

#### **3.5.1. Integrating with technologies to acquire contextual information**

In a context-aware application contextual information is required to be collected from multiple sources. As mentioned previously Perception.js uses sensors to capture contextual information from various mechanisms such as hard sensing (i.e. temperature sensors, humidity sensors, or GPS) or from soft sensing (i.e. which gather data from Systems, Databases, or Log Files).

##### *3.5.1.1. Integrating technologies as soft sensors*

A soft sensor in a context-aware application is a software based sensor which can acquire contextual information from a system, a database or a file. In Perception.js a sensor should be developed by encapsulating the logic that acquires contextual information from the source.

To receive messages from systems databases or files, Enterprise Integration patterns recommend two integration patterns which are Polling Consumer [62] and Event Driven, Consumer [63].

In a Polling Consumer the receiver polls for the message processes it and polls for the next message which can occur periodically [62]. In this pattern the system or the application could not notify the consumer about any changes in data. Instead the consumer periodically polls for the information from the system which is also known as a pull based communication [62]. Usually enterprise or cloud systems expose their functionality and data as SOAP or REST endpoints. Also, most database systems implement a request/reply communication mechanism to query data from the



database. Therefore, in Perception.js this pattern can be used to create sensors for such scenarios.

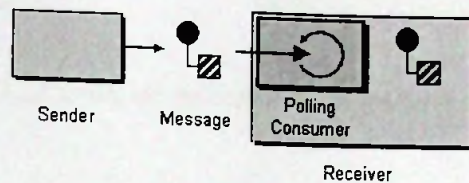


Figure 3.10: Polling consumer integration pattern [62]

The following architectural diagram illustrates how external systems can be connected using polling consumer approach;

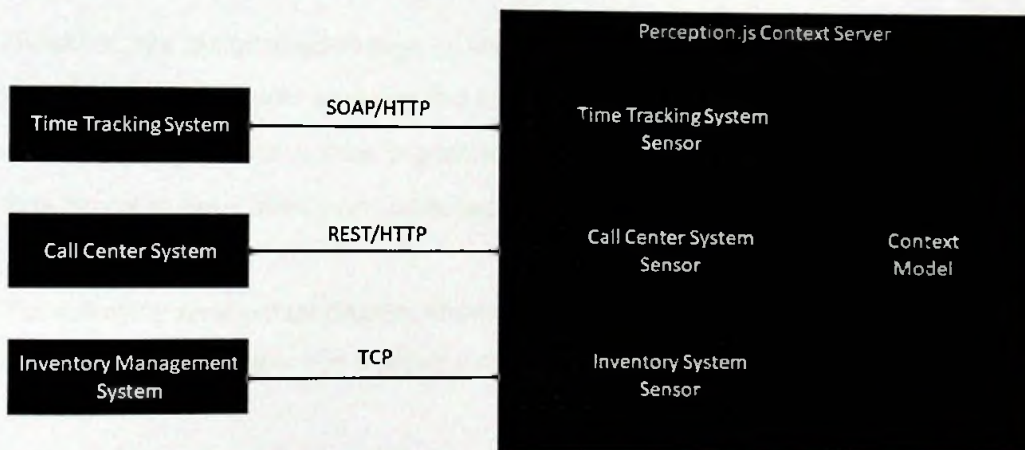


Figure 3.11: How polling consumer integration pattern can be used

In the above example three systems are integrated to Perception.js as three different sensor extensions which uses three different communication protocols to poll the system. When the data is received it can be used to update the context model.

An event driven consumer can react to the messages or events that are sent by the server to the client which the client has been subscribed to [63]. One approach to implement this pattern is that client must establish a long-lived connection to the server and should listen for incoming events. The disadvantage compared to the polling consumer approach is that the application should handle connection losses, reconnection mechanisms, and reliable message delivery mechanisms.



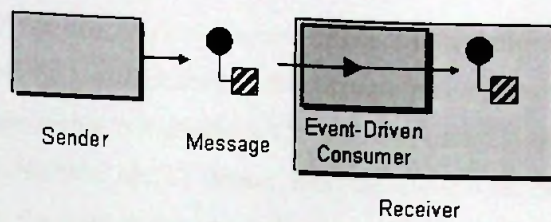


Figure 3.12: Event driven consumer integration pattern [63]

Another approach to implement this pattern is to use a message broker such as RabbitMQ, Kafka, or ActiveMQ which addresses the previously mentioned problems in the first approach. In this approach IoT devices such as ESP8266 or Arduino can use a MQTT message broker such as ActiveMQ that sends messages sent by IoT devices to the Perception.js context-aware application.

However, the major disadvantage of both approaches is that the sending application should either implement a mechanism to send data to a message broker or to maintain a connection pool which is an expensive approach. The major advantage is that new data becomes immediately available upon the changes.

The following architectural diagram illustrates how external system can be connected using an event driven consumer with multiple communication paradigms;

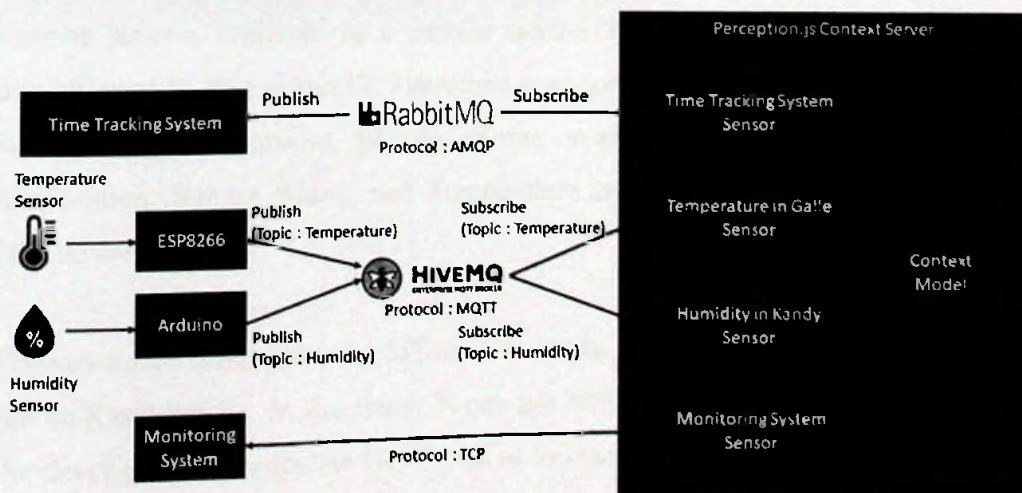


Figure 3.13: How event driven consumer integration pattern can be used



In the above diagram the time tracking system publishes information about employee finger print readings to RabbitMQ, and the time tracking system sensor in Perception.js which listens to the relevant message queue in RabbitMQ. The ESP8266 device located in Galle gets sensor readings and sends to HiveMQ MQTT Broker. Also, the Arduino device located in Kandy gets sensor readings from the humidity sensor and sends to HiveMQ. In perception.js two sensor extensions can be created for the temperate and humidity sensors by subscribing to the relevant topics in HiveMQ. Also, the Monitoring System that monitors employee activity which has a two-way TCP connection is also integrated as a sensor extension.

Modern NoSQL databases such as Firebase [64] and RethinkDB [65] consist of this feature which enables the context-aware application to respond to any data changes that occur in the database system.

#### *3.5.1.2. Integrating technologies as hard sensors*

To integrate physical sensors to Perception.js context-aware application the platform which the context-aware application is developed should be taken into consideration. As mentioned in the thesis Perception.js is portable which can be used to develop applications in platforms such as the web browser, apache Cordova, and node.js. In a Perception.js physical sensors can be integrated when the application or a component is built using Cordova or Node.js platforms. Cordova also uses plug-ins to integrate various sensors available in a mobile device. However, such plug-ins cannot be directly used by Perception.JS. Therefore, a sensor extension should be developed by encapsulating a Cordova plug-in. Some available Cordova plug-ins include Geolocation, Battery Status, and Temperature sensor which are available in their official website [66].

Context-aware applications for IoT devices can be written using Perception.JS can be run on Raspberry Pi. In Raspberry Pi can use NPM to download libraries that enable the developers to manipulate GPIO pins or to read data from various sensors [67]. A well-known library to use Raspberry PI sensors with Node.JS is known as raspisensors [67]. As mentioned previously sensors integrated to devices such as Arduino or ESP8266 can use protocols such as MQTT to transport messages to a context server written in node.js.



### 3.5.2. Integrating with technologies to store contextual information

Perception.js uses storage extensions to store contextual information gathered from a sensor. When a sensor updates a value of an ontology property the value is stored in a database or can be sent to a message queue for further processing. Storage plugins can be developed to support various storage mechanisms such as NoSQL or relational databases to store contextual information or RabbitMQ or Kafka to send the information for the server for further processing. When contextual information is streamed to a remote location using RabbitMQ or Kafka real time machine learning for classification can be integrated with the context-aware application.

The following diagram illustrates how various storage mechanisms can be encapsulated using storage extensions in Perception.js;

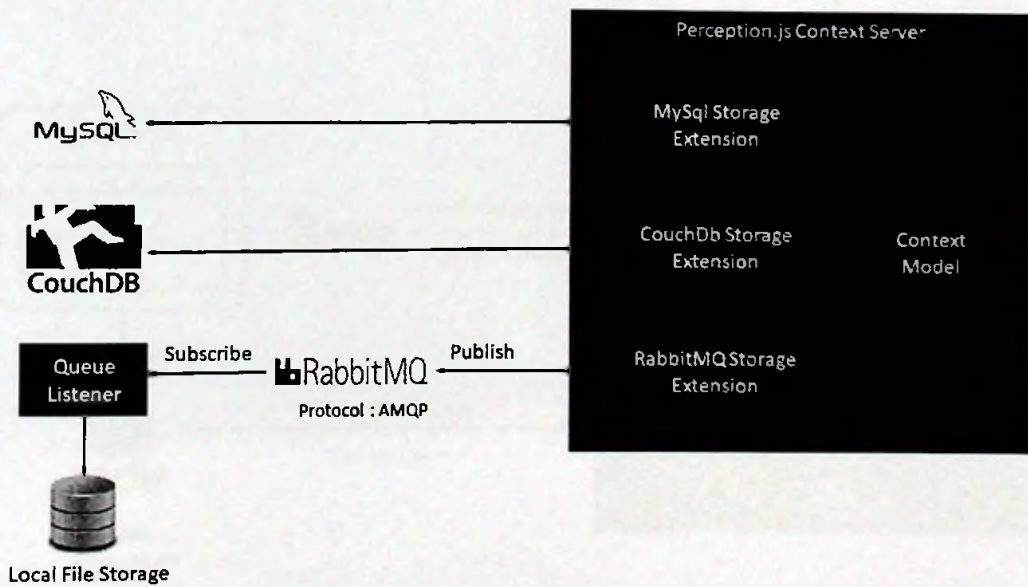


Figure 3.14: How storage extensions can be used to integrate different storage mechanisms



### 3.5.3. Integrating with technologies to process contextual information

In the current implementation of Perception.js one limitation is that when rules for context processing has been specified by the developer, the framework cannot dynamically update its context model. For example, in a context-aware smart home when a person comes home the context-aware system identifies the current mood based on the person's context and plays a music of a genre (i.e. Jazz, Rock). Even though the system identifies person's context and plays a music track, the person would change the music track manually to suit his mood. Therefore, the context-aware system should learn from the context that makes him trigger the context-aware functionality manually and update the context model accordingly. As a result, the system could identify his context more accurately and play the correct music track next time. However, in Perception.js such features are not available, instead it could use existing machine learning tools such as TensorFlow to incrementally learn from the user's context.

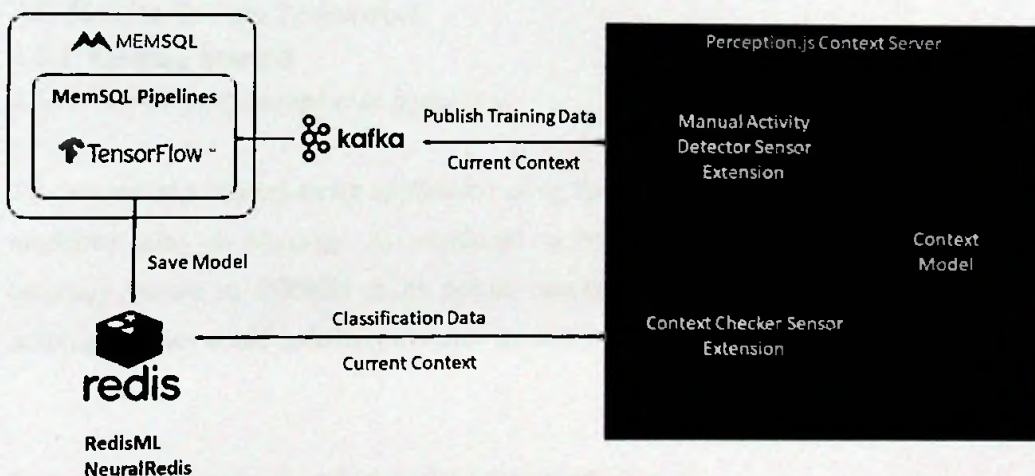


Figure 3.15: Integrating external context processing systems

When the above example is implemented using Perception.js, the context-aware application should have two sensor extensions. First sensor extension detects the person's manual activity in this example, changing the music track. Once the activity has been done the sensor extension should publish the data which includes the current context (his heart rate) and the genre of the music track (which is the label that is required the classification algorithm) to Apache Kafka topic to train the machine learning model. MemSQL Pipelines is used to subscribe for the topic



to listen to the incoming training dataset [68]. When a new item for the training data is being received by MemSQL Pipelines, it stores the new entry in MemSQL (if it is required to train the model with a new algorithm later), and train the machine learning model incrementally using TensorFlow. For the above example, a classification algorithm available in TensorFlow such as Deep Neural Networks or Logistic Regression can be used [69]. When the model has been updated the model can be stored in Redis. In Redis extensions are available such as RedisML or Neural Redis to store machine learning models as data types [70] [71].

The second sensor extension is used to classify listen to context changes in the Perception.js context model. When a context change has been detected, the sensor will send a relevant machine learning command to Redis and will attempt to classify which music genre the person would like to listen to.

## **3.6. How to Use the Framework**

### **3.6.1. Getting Started**

#### *3.6.1.1. Developing a client-side application*

To develop any context-aware application using Perception.js the context model should be modelled using an ontology. As mentioned in literature review, the framework uses an ontology known as CONON as its default ontology. Developers can extend the default ontology or they could optionally override the ontology.

To create an ontology perception.model() method can be used;

```
perception.model(function(context){
    context.activity.deduced.subclasses ("eating");
});
```

The method accepts a function with a single parameter which gives the developer the context model which can be extended using domain-specific ontologies.

When the context model has been developed, rules can be added to the ontology programmatically using the framework. The method perception.rules can be used to create rules for the ontology;



```

perception.rules(function(rules, context) {
  rules.logic(context.person.locatedIn,
    context.bedroom.and(context.person.engagedIn,
    context.sleeping).then(context.person.neededaction , "TURN_OFF_TV")
  });
};

```

The methods `logic()`, `and()` are used to specify conditions to a rule. The `then()` function is used to update a property of an ontology to a specific value.

Next the sensors should be mapped to relevant ontology classes and their properties. To do the task `perception.sensors()` method can be used which accepts a function with two arguments which are `sensorManager`, and `context`.

```

perception.sensors(function(sensorManager, context) {
  sensorManager.create("activityDetector",
    "fakeActivityDetector",
    {}).map(context.supun.engagedIn).preprocess(function(mapping, data) {
    context.supun.engagedIn.setValue(context.sleeping);
  });
});

```

`sensorManager` object can be used to create an instance of the sensor and map it to a property of an ontology class. Once the data is received by the sensor instance the data can be optionally preprocessed using the `preprocess()` function which is used to handle sensor imperfections. Once the data is being preprocessed the value of the ontology property can be updated.

To respond to contextual changes the framework enables the developers to create actuators. Actuators listen to changes in context model. When the context change has been detected it will trigger a particular callback function. The method `perception.actuators()` can be used to create an actuator which has two arguments which are `actuatorManager`, and `context`. An example is as follows;

```

perception.actuators(function(actuatorManager, context){
  actuatorManager.when(context.person.neededaction,
    "TURN_OFF_TV").then(function(){
    console.log ("Turning off TV");
  });
};

```



```
});  
});
```

Finally to start the context-aware application the framework should be initialized using the `initialize()` method and the framework activity should be started using the `play()` method.

```
perception.initialize();  
perception.play();
```

### *3.6.1.2. Developing a context server application*

In a context server context processing takes place in the server, and contextual information is passed to the context server from the connected devices. To accomplish that there should be a transport mechanism to transport context from a client to a context server. Perception.js enables the developers to use transport extensions to facilitate the communication between a client and a context server.

For the server side which runs on node.js the ontology model should be created and rules must be specified to identify a context. To identify contextual changes a transport extension should be created. When the transport extension receives contextual information from the clients it will update the values of relevant ontology properties accordingly. The sensor extension should be mapped to a client-side actuator to notify the client about the context changes. The following example shows how an instance of WebSockets server is started in port 4000, and updates an ontology property in the client side.

```
perception.transport(function(transportManager, context){  
    transportManager.create("WebSockets", "wsServer",  
    {port:4000}).map(context.person.neededaction);  
});
```

In the client-side same ontology model that is defined in the server should be created. However, the client side shouldn't contain any rules instead it should transport the contextual information gathered by sensors to the context server. To accomplish the task an instance of a transport extension which has the client should be created and mapped to ontology properties. The properties which has their values changed will send the updated values to the server using



the transport client extension. The following example shows how to send the updated values to the context server;

```
perception.transport(function(transportManager, context) {  
    transportManager.create("WebSockets", "wsWebClient",  
    {host:"localhost", port:4000}).map(context.tv.status,  
    context.supun.engageIn , context.supun.locatedIn);  
});
```

### 3.6.1.3. Developing extensions

To develop an extension for Perception.js it is recommended to use the closure design pattern in JavaScript because it enables to keep the extension scope in JavaScript isolated from the rest of the source code. First a JavaScript file should be created which has the sensor logic. Next, in the JavaScript file a closure should be created as follows;

```
(function(base) {  
})(window)
```

Extensions can also be created for multiple platforms such as Cordova, web browser, or Node.js. The above example works only for the web browser and Cordova. To make the extension work in Node.js 'window' object should be replaced by the 'global' object in Node.js.

Next the extension should be registered to make it available for the context-aware application. To accomplish that `perception.register()` method should be called with the relevant parameters. It should pass the mandatory parameters class to enable the context-aware application to create an instance of the extension class, category to indicate the type of the extension which could be sensor, transport or storage, and the `create()` method which handles the create logic of the extension. The following code example shows the structure of a Perception.js extension.

```
(function(base) {  
    base.perception.register ({  
        category: "sensor",  
        class: "fakeActivityDetector",  
        name: "Fake Activity Sensor",  
        description: "Fake Activity Sensor",  
        author: "Supun",
```





```
    });
    create: function(){return {}}
  })(window)
```

Extension logic can be added within the closure.

When developing sensor extensions the create() function should return a set of callback functions as follows;

```
create: function(){
  return {
    play: function(){},
    pause: function(){},
    stop: function(){},
    onPlay: function(func){},
    onPause: function(func){},
    onStop: function(func){},
    onResult: function(func){}
  }
}
```

Similarly for transport extensions the create() method should return the following object with following parameters;

```
create: function(){
  return {
    play: function(){},
    pause: function(){},
    stop: function(){},
    onPlay: function(){},
    onPause: function(){},
    onStop: function(){},
    onRecieve: function(f){},
    send: function(obj){},
    type: "server"
  }
}
```

For storage extensions the method should return the following object with following parameters;

```
create: function(){
  return {
    play: function(){},
    pause: function(){},
    stop: function(){},
    onPlay: function(){},
    onPause: function(){},
    onStop: function(){},
    store: function(obj){},
  }
}
```



Detailed descriptions of the above methods for each extension type can be found under extensibility in detailed design section.

### 3.6.2. Using Framework Features

Taking the usability requirements of the framework into consideration, the syntaxes were designed to maintain consistency in code, reduce the learning curve, and to efficiently code the context awareness logic. Also, the programming paradigm of JavaScript was taken into consideration.

#### 3.6.2.1. Creating a context model

In order to create a context model using the framework, the developers can use the `model()` function available in the framework. The method accepts a function which has a parameter called 'context'. Using the 'context' parameter the developer can define the context model as follows;

```
perception.model(function(context){
    context.<MAINCLASS>.subclasses("subclass1", "subclass2");
    context.subclass1.property("locatedIn", context.location);
});
```

#### 3.6.2.2. Defining rules to infer a context

To define rules for the ontology which was already modelled, the developer can use the `rules()` function which has two parameters 'rules' and 'context'. Using the 'rules' parameter the developer can create rules to deduce a context. The 'context' parameter can be used to map the ontology defined by the developer to the rules.

```
perception.rules(function(rules, context){
    rules.logic(context.person.locatedIn,
    context.bedroom).and(context.tv.status,
    "ON").and(context.person.engagedIn,
    context.sleeping).then(context.person.neededaction, "TURN_OFF_TV");
});
```



### 3.6.2.3. Mapping sensors to the context model

In order to map sensors to a context model the `sensors()` method can be used. Using the `sensor.create()` method, an instance of a sensor extension can be created. Using the `map()` method the created instance can be mapped to an entity in the context model. The `preprocess()` method can be used to map lower level contextual data into higher level contextual data.

```
perception.sensors(function(sensor, context){
    sensor.create("activityDetector", "fakeActivityDetector",
    {}).map(context.supun.engagedIn).preprocess(function(mapping, data){
        context.supun.engagedIn.setValue(context.sleeping);
    });
});
```

### 3.6.2.4. Mapping actuators to the context model

In order to create actuators for the context model the `actuators()` method can be used which accepts a function consists of two parameters. The 'actuator' parameter can be used to respond to a change in any entity of the context model. If the value of an entity in the context model is updated for a particular value, `then()` method can be used by the developers to write the context presentation logic.

```
perception.actuators(function(actuator, context){
    actuator.when(context.person.neededaction,
    "TURN_OFF_TV").then(function(){
        alert("Turning OFF TV");
        context.tv.status.setValue("OFF");
    });
});
```

### 3.6.2.5. Mapping transport extensions to the context model

In order to integrate transport protocol extensions to the framework the `transport()` method can be used which accepts a function with two parameters 'protocol', and 'context'. Using the 'protocol' parameter an instance of a transport plugin can be created, and using the `map()` method the a list of ontology entities using the 'context' parameter can be specified to be updated through the transport extension.

```
perception.transport(function(protocol, context){
    protocol.create("WebSockets", "wsWebClient", {host:"localhost",
    port:4000}).map(context.tv.status, context.supun.engagedIn ,
    context.supun.locatedIn);
});
```



### 3.6.2.6. Mapping storage extensions to the context model

In order to integrate storage extensions to the framework the `storage()` function can be used which accepts a function with two parameters 'storage', and 'context'. Using the 'storage' parameter an instance of the storage plugin can be created, and using the `map()` method the a list of ontology entities using the 'context' parameter can be specified to be updated through the transport extension.

```
perception.storage(function(storage, context) {
  storage.create("MySQL", "mysqlClient", {host:"localhost",
  user:"root", password:"root",
  database:"test"}).map(context.tv.status, context.supun.engagedIn ,
  context.supun.locatedIn);
});
```

### 3.6.2.7. Listening to internal events of the framework

Using the `on()` method in `perception.js` internal events of the framework can be monitored. The `on()` method has two overload functions;

The first function accepts a string which should be the event the developer should listen to and a function which the developer should include his logic to respond to the event.

```
perception.on("perception.plugin.output", function(data) {
  log (data.text);
});
```

If there are multiple events a developer needs to listen to, the developer could pass an object to the `on()` function with a key value pair which the key is the event name and the value is a function. This enables the developers to keep their code clean.

```
var eventMap = {
  "perception.plugin.output": function(data) {
    log (data.text);
  },
  "perception.core.output": function(data) {
    log (data.text);
  }
}
perception.on(eventMap);
```



### 3.6.3 Best Practices

#### 3.6.3.1. Best practices for developing context-aware applications

##### Always Use Method Chaining

Perception.js supports method chaining to improve productivity and speed up development, and increase readability of code. Method chaining also promotes a declarative style of coding which is more human friendly. Following is an example of method chaining.

```
rules.logic(c.person.locatedIn,c.bedroom).and(c.tv.status,
"ON").and(c.person.engagedIn,c.sleeping).then(c.person.neededaction,
"TURN_OFF_TV");
```

an alternative counterproductive way of writing the above code is as follows;

```
var s1 = rules.logic(c.person.locatedIn, c.bedroom);
var s2 = s1.and(c.tv.status, "ON");
var s3 = s2.and(c.person.engagedIn, c.sleeping)
s3.then(c.person.neededaction , "TURN_OFF_TV");
```

##### Always Listen to Framework Events to Monitor Internal Workings

Perception.js enables the developer to identity internal workings of the framework using events. This can be used for debugging purposes and to identify errors in runtime. Perception.js also allows the developers to specify an event map which is an object of a key-value pair which has an event name as the key and a callback function as a value. This approach also reduces the number of lines required for coding.

```
var eventMap = {
  "perception.plugin.output": function(data){
    log (data.text);
  },
  "perception.core.output": function(data){
    log (data.text);
  }
}
perception.on(eventMap);
```



### **Always Create Multiple Classes and Properties in Single Method Call**

Perception.js enables the developer to specify the context model as an ontology. Since ontologies consist of classes and properties and a since class can have multiple subclasses and properties, using the method chaining approach multiple classes and properties can be created using a single method call. An example of creating multiple subclasses is as follows;

```
context.activity.subclasses ("eating", "reading", "cooking",  
"sleeping");
```

### **In the Web Browser Always Import Extensions Before Importing the Application Script**

When using Perception.js to develop a component or a context-aware application, extensions should be loaded manually. When loading extensions manually the corresponding script file should be imported before loading script file which has the application logic.

```
<script type="text/javascript"  
src="plugins/sensors/fakeActivityDetector.js"></script>  
<script type="text/javascript"  
src="plugins/sensors/fakeLocationDetector.js"></script>  
<script type="text/javascript"  
src="plugins/sensors/fakeTvSensor.js"></script>  
<!-- Always load plugins first before loading the application logic  
-->  
<script type="text/javascript"  
src="contextserver.client.js"></script>
```



### 3.6.3.2. Best practices for developing Perception.js extensions

#### **Always Create a New Instance of the Sensors in the create() method**

In Perception.js it is required to register a plugin for the context-aware application to use it. For the perception.register() method it is required to pass a callback function which is called every time when a new instance of a particular extension is created. As a good practice it is recommended to create a new instance of an object and return. An alternative approach would be to always return a singleton object. However, the creating a new object enables the developers to keep the state of each instance or the plugin isolated from one another. Following is an example of using this best practice;

```
function Sensor(){
    return {
    }
}

window.perception.register ({
    create: function(){return new Sensor()}
});
```

#### **Always Start Extension Activity Inside the play() method**

To start a Perception.js application the play() method must be called. When the method is being called, the framework calls the play() method in every plugin instance. Starting the extension activity outside of the play() method would make the application unstable.

#### **Don't Leave the Controller Methods Blank**

It is not recommended to keep controller methods such as pause(), stop(), onPlay(), onPause(),etc. as a blank function. Keeping it in such a way could cause context-aware application unstable because the plugin doesn't allow the framework to take control over its functions.



# CHAPTER 5

## TESTING AND ANALYSIS



## 4.1. Introduction

The objective of this chapter is to present the testing strategy, and test results of the framework which includes the results of functional and non-functional testing which was mentioned in the previous chapter. To verify the requirements, unit testing was performance to test the functionality from the component level where as system testing was performed to verify the requirements using various scenarios.

The first scenario is focused on testing the core features by creating an application with mock/fake sensors which aims to verify the context acquisition, context processing, and context presentation features, as well as the extensibility features of perception.js framework. The next scenario is focused on creating a context server with the framework to verify whether the framework supports context server approach. In the last scenario, a usability test was performed with actual developers within a given timeframe to verify whether the framework solves the problems presented in the solution outline section.

## 4.2. Unit Testing

This chapter focuses on presenting the test results of the unit test cases of each component and their methods. The previous chapter consists of detailed descriptions about each method.

### Component: Helpers

Table 4.1: Unit testing results of the component: Helpers

Method Name	Test Case	Expected Result	Actual Result	Status
findByField()	Check when the specified field is not available in the JSON object the method should return undefined.	undefined	undefined	Pass
findByField()	Check when the specified field is available in the JSON object the method should return the value of the field.	value	value	Pass



defaultByField()	Check when the field is available in the JSON object the method should return the value of the field.	value	value	Pass
defaultByField()	Check when the field is unavailable in the JSON object the method should return the default, value specified as a method parameter.	Returns default value	Returns default value	Pass
concatArray()	Check when two arrays are specified the method should return a concatenated array consisting of the arrays passed as method parameters.	Returns concatenated array	Returns concatenated array	Pass

### Component: AsyncIterator

Table 4.2: Unit testing results of the component: AsyncIterator

Method Name	Test Case	Expected Result	Actual Result	Status
onCompleteAll()	When the iterator completes processing all the elements in an array asynchronously, the function passed as a parameter should invoke.	The function should be invoked upon completion.	The function invoked upon completion.	Pass
onComplete()	When a single item completes processing asynchronously, the function passed as a parameter should invoke.	The function should be invoked upon completion of a single item.	The function triggered upon completion of a single item.	Pass
onError()	When an error occurs when an item is being processed, the function passed as a parameter should be invoked.	The function should be invoked when an error occurs.	The function invoked when an error occurred.	Pass
method()	Upon each iteration, the function passed as a parameter should be invoked.	The function should be invoked in each iteration.	The function invoked in each iteration.	Pass



**Component: EventManager**

Table 4.3: Unit testing results of the component: EventManager

Method Name	Test Case	Expected Result	Actual Result	Status
on()	Upon registering for an event, a function should be returned to unregister an event.	A function should be returned to unregister an event.	A function returned to unhandled the registered event.	Pass
on()	Multiple handlers should be able to register for a single event.	When triggering an event, all the handler functions should call	All the handler functions called.	Pass
remove()	The event should be unregistered upon calling this function.	When an event is being triggered, the callback method should not call.	The callback method didn't call after unregistering an event.	Pass

**Component: ConfigurationManager**

Table 4.4: Unit testing results of the component: ConfigurationManager

Method Name	Test Case	Expected Result	Actual Result	Status
set()	The method should be able to set a configuration with a key value pair.	The configuration should include the newly created key value pair.	The configuration included the newly created key value pair.	Pass



get()	The method should return a configuration value if its available.	The value of the key should be returned.	The value of the key was returned.	Pass
getPlugin()	The method should return a plugin specific configuration value given a key and a plugin name.	The method should return a value.	The method returned a value.	Pass
setPlugin()	The method should be able to set a plugin specific configuration key and a value.	The plugin configuration should include the newly created key value pair.	The plugin configuration included the newly created key value pair	Pass

### Component: RuleManager

Table 4.5: Unit testing results of the component: RuleManager

Method Name	Test Case	Expected Result	Actual Result	Status
and()	The method should be able to add an 'and' rule to the expression tree.	The expression tree should include the new 'and' rule.	The expression tree included the newly created 'and' rule.	Pass
or()	The method should be able to add an 'or' rule to the expression tree.	The expression tree should include the new 'or' rule.	The expression tree included the newly created 'or' rule.	Pass
then()	The method should be able to specify a property that should be changed when the rule chain is satisfied.	The specified ontology property should change when the rule chain satisfies.	The specified ontology property changed when rule chain was satisfied.	Pass
else()	The method should be able to specify a property that should be changed when the rule chain is not satisfied.	The specified ontology property should change when the rule	The specified ontology property changed when rule	Pass





		chain does not satisfy.	chain was not satisfied.	
--	--	-------------------------	--------------------------	--

**Component: TransportManager**

Table 4.6: Unit testing results of the component: TransportManager

Method Name	Test Case	Expected Result	Actual Result	Status
setInstance()	ability to register an instance of a plugin to transport manager	The plugin instance should be available in TransportManager	The plugin instance was available in TransportManager	Pass
start()	The method should start all the transport plugin instances.	All the transportation extensions should start.	All the transportation extensions were started.	Pass
stop()	The method should stop all the transport plugin instances.	All the transportation extensions should stop.	All the transportations extensions were stopped.	Pass

**Component: OntologyManager**

Table 4.7: Unit testing results of the component: OntologyManager

Method Name	Test Case	Expected Result	Actual Result	Status
setValueByTransportData() ()	When a transport extension sends information that should be updated in the ontology, the ontology should be updated.	The ontology should be updated.	The ontology updated.	Pass
userRules()	The method should return an instance of RuleManager to set rules for an ontology.	The method should return an instance of RuleManager	The method returned an instance of RuleManager	Pass
findClass()	The method should return an ontology	The method should return an instance of	The method returned an instance of	Pass



	class if the class is available	the ontology class.	the ontology class.	
defineDefault ()	The default ontology should be created when the method is called.	The default ontology should create.	The default ontology was created.	Pass

**Component: OntologyClass**

Table 4.8: Unit testing results of the component: OntologyClass

Method Name	Test Case	Expected Result	Actual Result	Status
subclass()	Check whether the method should be able to create a subclass in an ontology class.	A subclass should be created in the ontology class.	A subclass was created in the ontology class.	Pass
subclass()	Check whether the method should be able to create multiple subclasses in an ontology class	Multiple subclasses should be created in the ontology class.	Multiple subclasses were created.	Pass
property()	Check whether the method should be able to create a new property in an ontology class.	A property should be created in the ontology class	A property was created in the ontology class.	Pass
property()	Check whether the method should be able to create multiple properties in an ontology class	Multiple properties should be created in the ontology class.	Multiple properties were created.	Pass
getChildren()	Check whether the method should return all the subclasses in an ontology class.	The method should return the subclasses.	The method returned subclasses.	Pass



**Component: OntologyProperty**

Table 4.9: Unit testing results of the component: OntologyProperty

Method Name	Test Case	Expected Result	Actual Result	Status
onChange()	Check when the value of the property changes, the callback method should trigger.	The callback method should trigger.	The callback method was triggered.	Pass
setValue()	Check whether a value of an ontology property should be updated.	The ontology property should have the newly updated value.	The ontology property was updated with the new value.	Pass
setValue()	Check whether a value of an ontology property should be updated with an expiry time and a revert value.	The ontology property should have reverted value after the expiry time.	The ontology property had the reverted value after the expiry time.	Pass
getValue()	Check whether the method should return the value of the ontology property.	The value should be returned.	The value was returned.	Pass

**Component: Scheduler**

Table 4.10: Unit testing results of the component: Scheduler

Method Name	Test Case	Expected Result	Actual Result	Status
schedule()	The method should be able to set a schedule with a reference object, expiry time, and a revert value.	The reference object's value should change to the revert value after the expiry time.	The reference object's value changed to the revert value after the expiry time.	Pass
start()	Check whether the scheduler is getting started.	The scheduler should start.	The scheduler started.	Pass



stop()	Check whether the scheduler is getting stopped.	The scheduler should stop.	The scheduler stopped.	Pass
--------	---	----------------------------	------------------------	------

**Component: ExpressionTree**

Table 4.11: Unit testing results of the component: ExpressionTree

Method Name	Test Case	Expected Result	Actual Result	Status
binary()	Check whether the expression tree can create a binary expression	The new binary expression should be created in the root	The new binary expression was created in the root.	Pass
setRoot()	Check whether the root of the expression tree can be changed.	The expression tree should replace its root.	The root of the expression tree was replaced.	Pass
getRoot()	Check whether the root of the expression tree can be retrieved.	The method should return the root node of the expression tree.	The method return the root node of the expression tree.	Pass
onChange()	Check whether the callback method was triggered when the expression tree is evaluated.	The callback function should be invoked.	The callback function was invoked.	Pass
clone()	Check whether the expression tree can be cloned.	The new expression tree should be returned by the function.	The new expression tree was returned.	Pass

**Component: BinaryExpression**

Table 4.12: Unit testing results of the component: BinaryExpression

Method Name	Test Case	Expected Result	Actual Result	Status
getParent()	Check whether the parent node can be returned in a binary expression	The method should return the parent node.	The method returned the parent node.	Pass



setParent()	Check whether the parent node can be set in a binary expression.	The method should return the modified binary expression.	The method returned the modified binary expression.	Pass
setLeft()	Check whether the left child of the binary expression can be set.	The method should return the modified binary expression.	The method returned the modified binary expression.	Pass
setRight()	Check whether the right child of the binary expression can be set.	The method should return the modified binary expression.	The method returned the modified binary expression.	Pass
clone()	Check whether the binary expression can be cloned including its child nodes.	The method should return the modified binary expression.	The method returned the modified binary expression.	Pass
evaluate()	Check whether the binary tree can be evaluated.	The method should return 'true' if evaluated.	The method returns true when it was evaluated.	Pass

### Component: UnaryExpression

Table 4.13: Unit testing results of the component: UnaryExpression

Method Name	Test Case	Expected Result	Actual Result	Status
triggerChange()	Check whether the callback function is getting triggered when the unary expression value is being changed.	The callback function should trigger when the unary expression changes.	The callback function triggered.	Pass
clone()	Check whether the unary expression should be able to clone.	The method should return a new identical instance.	The method returned a new identical instance.	Pass



## 4.2. System Testing

### 4.2.1. Testing of functionality, and extensibility

#### 4.2.1.1. Test goals

- Verify whether the framework can provide the core functionality of context acquisition, context processing, and context presentation.
- Verify whether the framework facilitates application development with the sentient object architecture.
- Verify whether a context can be modelled using ontologies in the framework.
- Verify whether sensor extensions can be integrated to the framework.
- Verify whether sensor extensions can be developed.
- Verify whether data preprocessing can be map raw sensor data to higher level contextual information.
- Verify whether subclasses can be created using ontology classes.
- Verify whether properties can be created for ontology classes.

#### 4.2.1.2. Test scenario

The application developed for the test scenario monitors whether a person falls asleep while watching the TV, the context-aware application will turn off the TV. To track whether the person is sleeping, a mock/fake sensor that simulates the heart rate tracking of a fitness band was created. Also, to check if the TV is turned on another mock/fake sensor that simulates a Smart TV was created. In addition, another mock/fake sensor that tracks the location of that person is also created which simulates the location tracking of a fitness band.



The architecture of the context-aware application is as follows;

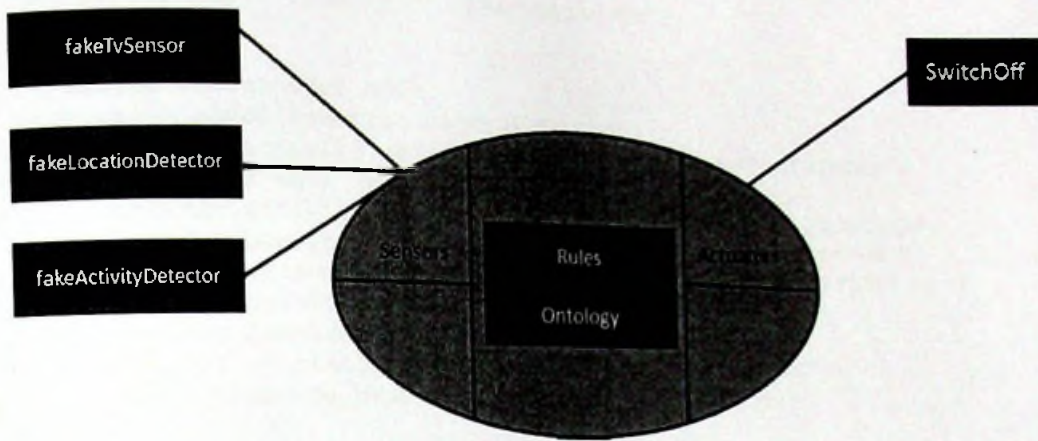
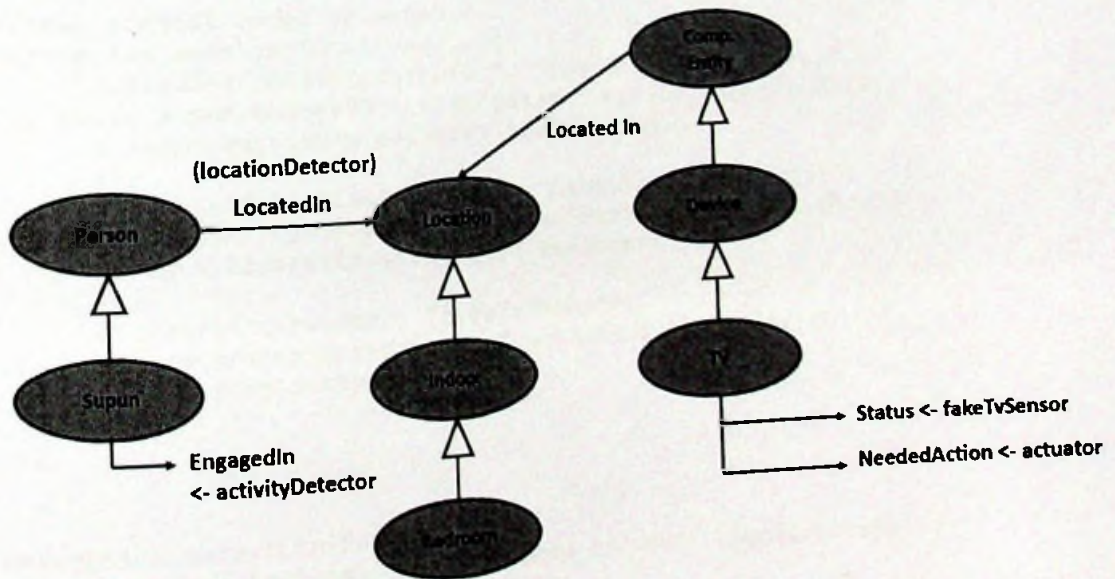


Figure 4.1: Architecture of the first sample application

To represent this scenario, the following ontology and the following rules were created;



$(\text{Person LocatedIn Bedroom}) \wedge (\text{Person EngagedIn Sleeping}) \wedge (\text{TV Status On}) \rightarrow (\text{TV NeededAction TurnOff})$

Figure 4.2: Ontology of the first sample application





#### 4.3.1.3. Source code of the context-aware application

```
var perception = require ("./perception.js");

//define the context model
perception.model(function(context){
  context.activity.deduced.subclasses ("movie", "dinner",
"shower", "cooking", "sleeping");
  context.activity.property("locatedIn", context.location);
  context.person.property("locatedIn", context.location);
  context.person.property("engagedIn", context.activity);
  context.person.subclasses("supun");
  context.location.indoor.subclass("bedroom");
  context.compEntity.subclass("device").subclass("tv");
  context.device.property("status");
});

//define the rules
perception.rules(function(rules,c){
  rules.logic(c.person.locatedIn, c.bedroom).and(c.tv.status,
"ON").and(c.person.engagedIn, c.sleeping).then(c.tv.neededaction ,
"TURN_OFF_TV");
});

//map context model to sensors
perception.sensors(function(s,c){
  s.create("activityDetector", "fakeActivityDetector",
{}).map(c.supun.engagedIn).preprocess(function(mapping,data){
  c.supun.engagedIn.setValue(c.sleeping);
});
  s.create("locationDetector", "fakeLocationDetector",
{}).map(c.supun.locatedIn).preprocess(function(mapping,data){
  c.supun.locatedIn.setValue(c.bedroom);
});
  s.create("tvSensor", "fakeTvSensor",
{}).map(c.tv.status).preprocess(function(mapping,data){
  c.tv.status.setValue("ON");
});
});

perception.actuators(function(a, c){
  a.when(c.tv.neededaction, "TURN_OFF_TV").then(function(){
  console.log ("Turning off TV");
});
});

perception.initialize();
perception.play();
```



#### 4.3.1.4. Test results and analysis

Table 4.14: Test results of the first test scenario

Test Case	Status
Sensor extensions can be developed by 3rd party developers	Pass
Context can be modelled using the framework	Pass
Rules can be applied to ontologies, and their properties	Pass
Actuators can be applied to the context model	Pass
Subclasses can be created for ontology classes	Pass
Properties can be created for ontology classes	Pass
Multiple subclasses can be created for an ontology class	Pass
When a rule set satisfies its condition a property in an ontology gets changed	Pass
When an ontology property gets changed an actuator gets triggered	Pass
Sensors can be mapped to ontology properties	Pass
Rules can be applied via a chain	Pass
Multiple sensors can be used with a single ontology model	Pass
Preprocessing can be used for sensor data	Pass

Taking the test results in to consideration, the test has verified that the core framework provides the core functionality of context awareness. Using the framework context can be acquired via sensor extensions, context can be processed using rules and ontologies, and the application can respond to contextual changes for context processing via actuators. Following a context modeling approach such as ontologies has also enabled the developers to easily model a real-world context.

Taking the sentient model reference architecture into consideration, the framework exposes interfaces to use sensors, ontologies, rules, and actuators. It is also noticed that the number of lines required to build the scenario applications is 40 lines excluding white spaces and comments. Therefore, it can be concluded that the applications developed using the framework are lightweight. Taking the programming paradigm into consideration, it uses JavaScript design patterns such as callbacks to use sensors, and actuators, and to create rules and ontologies. In addition to that it uses cascading design pattern to improve the productivity of the developer.



Taking the sensor extensions into consideration, sensor extensions are implemented in separate JavaScript files which uses the closure design pattern. In addition to that sensors can easily be implemented via the bootstrapping process in the framework. In order to preprocess the raw sensor data a callback method can be passed to the sensor extension. In this example the preprocessing function sets a predefined value for the particular property of an ontology.

## 4.2.2. Testing of context server approach, and portability

### 4.2.2.1. Test goals

- Verify whether the framework can be used to create a context server.
- Verify whether transport extensions can be used with the framework to communicate between the context server and the client.
- Verify whether the inferencing occurs in the context server while sensing occurs in the client.
- Verify whether the actuator is getting triggered in the client side when the proper context has been inferenced in the context server.
- Verify whether the framework can be ported among the applications developed for the web browser, and node.js

### 4.2.2.1. Test scenario

The objective of this test scenario is to present a context server based approach to context processing and to present the portability features that are available in perception.js framework. For this example, the previous scenario was further expanded with a context server approach. The client collects sensor data in a web browser using the previously created mock/fake sensors, and the context server runs in a node.js backend hosted in a Linux server. The ontology is defined in both the client and context server. In the client, the ontology model is used to map the sensors and actuators, and to map transport extensions to context server. In the server, the ontology model is used to create the rules and to notify the client when contextual change occurs. To transport the sensor data to the context server, and to notify the



client when the context changes, a transport extension was created which uses web sockets as the transportation mechanism.

The architecture of the context-aware application is as follows;

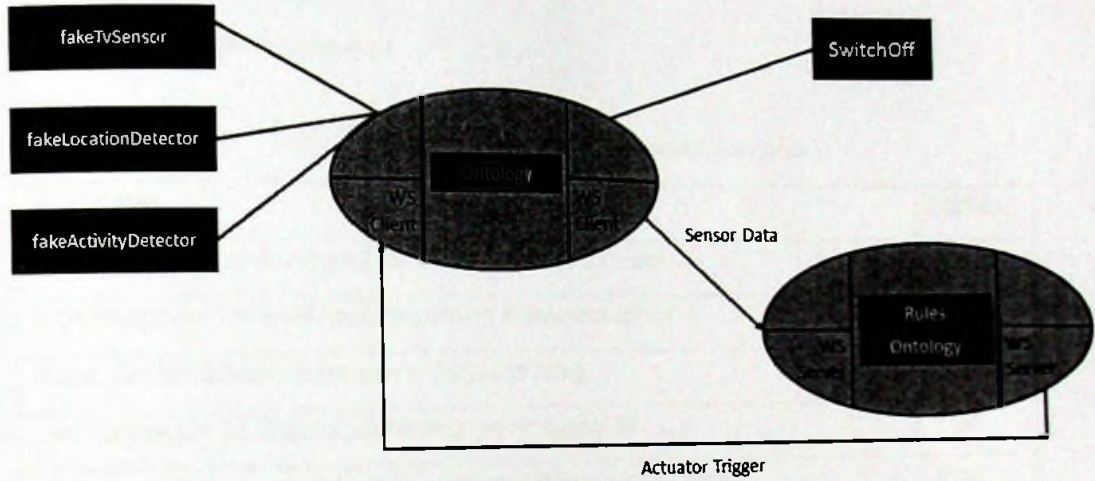


Figure 4.3: Architecture of the second sample application

Since the application consists of a client-side component, and a server-side component, two JavaScript files were created which runs on the client side and the server side. In both the files the following additional code segments were added to integrate the transport extensions to the ontology model;

In the context awareness client, the following code segment was added which accepts parameters to connect to the context server, and to specify which sensor information should be sent to the context server for processing;

```
perception.transport(function(t,c){
    t.create("WebSockets", "wsWebClient", {host:"localhost",
    port:4000}).map(c.tv.status,
    c.supun.engagedIn ,c.supun.locatedIn);
});
```



In the context awareness server, the following code segment was added to expose a web sockets endpoint with a port number and map the ontology property to the extension which will notify the clients of any contextual changes.

```
perception.transport(function(t,c){
    t.create("WebSockets","wsServer",
        {port:4000}).map(c.tv.neededaction);
});
```

#### 4.2.1.2. Test results and analysis

Table 4.15: Test results of the first second scenario

Test Case	Status
Extensions can be developed for context awareness client	Pass
Extensions can be developed for context awareness server	Pass
Rules can be defined in the server for inferencing	Pass
Ontologies can be defined for context server approach	Pass
Changes in context model can be replicated from client to server	Pass
Changes in context model can be replicated from server to client	Pass
Actuators can be triggered in the client after reasoning is happened in server	Pass
Transport client extensions can be configured	Pass
Transport server extensions can be configured	Pass
The developer can define transport extensions	Pass

Taking the test results into consideration, it was verified that that the framework can be used to develop a fully functional context server application and context client. In addition to that the client-side application had sent sensing data to the server for processing. When the context server had identified the context, an actuator was trigged in the client.

Taking the portability requirement into consideration, the framework was able to be used in a node.js context server applications and a browser based client application. Through the bootstrapping processes the sensor extensions were successfully loaded in both platforms.



Taking the transport extensions into consideration, a client-side web sockets transportation extension could connect to the context server after when it was configured use a specific port and a specific host. On the other hand, the context server used a web sockets server extension to expose a port using a specific configuration. Therefore, each extension can be configured to have their own settings.

### **4.3. Usability Testing**

#### **4.3.1. Test goals**

- Verify whether the framework can be used to get the expected results for the developers without focusing on technical details.
- Verify whether the framework can be used to get the expected outcome with less effort, time, and bug rate.
- Verify whether the application built using the framework is easy to debug and test.
- Verify whether the framework has a less learning curve.
- Verify whether the framework can be used with existing best practices in the industry.

#### **4.3.2. Test scenario**

This test scenario was focused on verifying whether the framework meets the usability requirements. To achieve that objective a context-aware application with a less number of features were given for a team of two developers, Deleepa and Rifhan. The developers didn't have any previous knowledge in context-aware application development. However, they had knowledge and previous experience in developing node.js, and Cordova applications. Therefore, they successfully installed and configured the development environment in their laptops. Also, the two developers have been in the industry for nearly two years. The application was developed on a Saturday apart from their work. instructions were given to them on how to solve their various problems using the framework.



### 4.3.3. Test process

At the beginning 2 hours of training was given for the developers which covered the concepts of context awareness, and how the framework enables the developers to easily use context awareness features in their applications. The topics covered included on how to use its features, how to develop their own extensions, and how to architecture their applications using the framework. Then the developers were asked to come up with a hypothetical scenario for the application which they plan to build. Once they established their idea they were given the opportunity to come up with an architecture for their applications which is based on the sentient object model. Next, they have come up with a work breakdown structure which consisted of all the tasks they have worked on. While they were working on the tasks the time was recorded for each task. At the completion, a questionnaire was given and feedback was taken for analysis.

#### 4.3.3.1. Overview of the application being developed

The application that is being developed for this application was an automated employee health tracking system which operates in a hypothetical scenario. In a hypothetical organization, employees are required to work under pressure which weakens their health. In addition to that when employees work under stress for longer periods their productivity could go down and ultimately, they could end up in burn outs. If the management could find out such conditions at the early stages both the businesses and employees will benefit.

In a situation when an employee is under stress or anxiety his heart rate can increase which can be measured using the sensor in his smart watch. In situations when an employee is busy his number of allocated tasks in his task list (i.e. tasks in Jira) increases which can be tracked using a virtual sensor which connects to a task management system such as Jira. Also, if a person doesn't have much physical activity he would spend almost all his time at his cubicle which can be tracked using the GPS sensor in his smart watch. If an employee has a higher heart rate, a higher number of tasks in Jira, and if he stays in his cubicle, an email is sent to his line manager indicating that specific employee could be under pressure and the management should take action to help the employee to maintain his health.

In this application, in the client side the smart watch of an employee collects all his personal data such as his heart rate, his location, and his task list in Jira using sensors. Context



processing occurs in the server side. Therefore, the rules and the actuator is implemented in the server side. The actuator will send an email to the manager.

#### 4.3.3.2. Requirements of the application

- The client-side application should collect location data from the smart watch.
- The client-side application should collect heart rate data from the smart watch.
- The client-side application should collect his number of Jira tasks from the smart watch.
- The client-side application should send data to the context server for processing.
- The context server should maintain information about all the employees.
- The context server should respond to contextual changes of all the employees.
- the context server should send an email to his line manager if an employee is under pressure.

#### 4.3.3.3. Architecture of the application being developed

The developers have developed the following architecture for their application which is based on the sentient object model. The sensor extensions are developed for the client-side application. Also, the client-side application uses the web sockets client extension to communicate with the context server. The ontology is modelled in the client side because an ontology is required by the framework to integrate sensors. In the context server, an actuator is implemented to send email. Context processing takes places in the context server since rules are defined in the server.

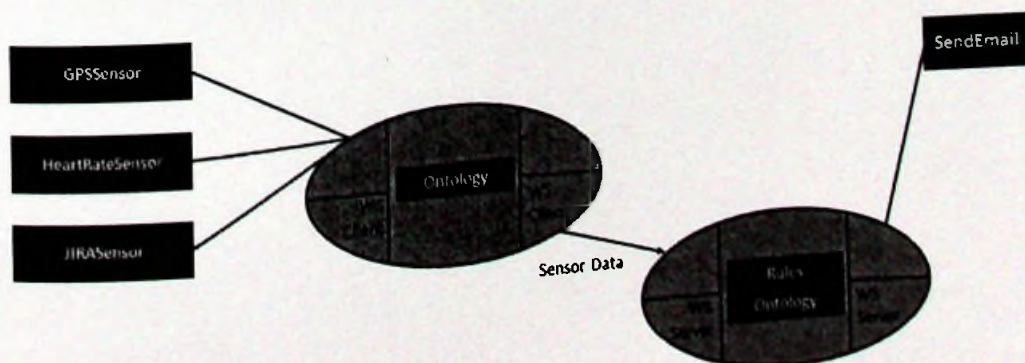


Figure 4.4: Architecture of the third sample application



#### 4.3.3.4. Detailed design of the application being developed

In the client application, the following JavaScript objects were created;

- SmartWatchApp: this object consists of the core business logic of the client-side application
- OntologyMapping: this object consists of a reusable component which consist of the context model.
- SensorMapping: this object consists of the sensor integrations for the framework.
- TransportMapping: this object consists of the web socket client extension integration.

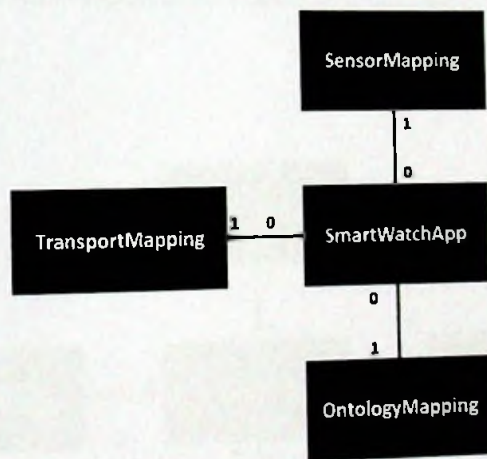


Figure 4.5: Class diagram of the client application of the third sample application



In the context server application, the following JavaScript objects were created;

- ContextServer: this object consists of the main app object which consists of the core business logic of the application
- OntologyMapping this object consists of a reusable component which consist of the context model. The same object that was used for the client application was used in the server as well.
- TransportMapping: this object consists of the web sockets server extension integration with the application.
- RuleMapping: this object consists of the rules which are used to deduce the context which sends an email
- EmailSender: this object consists of the logic to send an email.

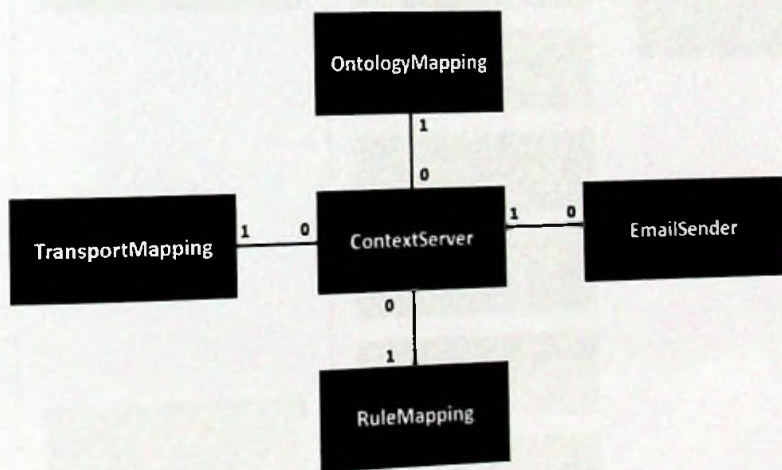


Figure 4.6: Class diagram of the context server of the third sample application



4.3.3.5. Breakdown of tasks

To develop the above architecture, the following work breakdown structure was created.

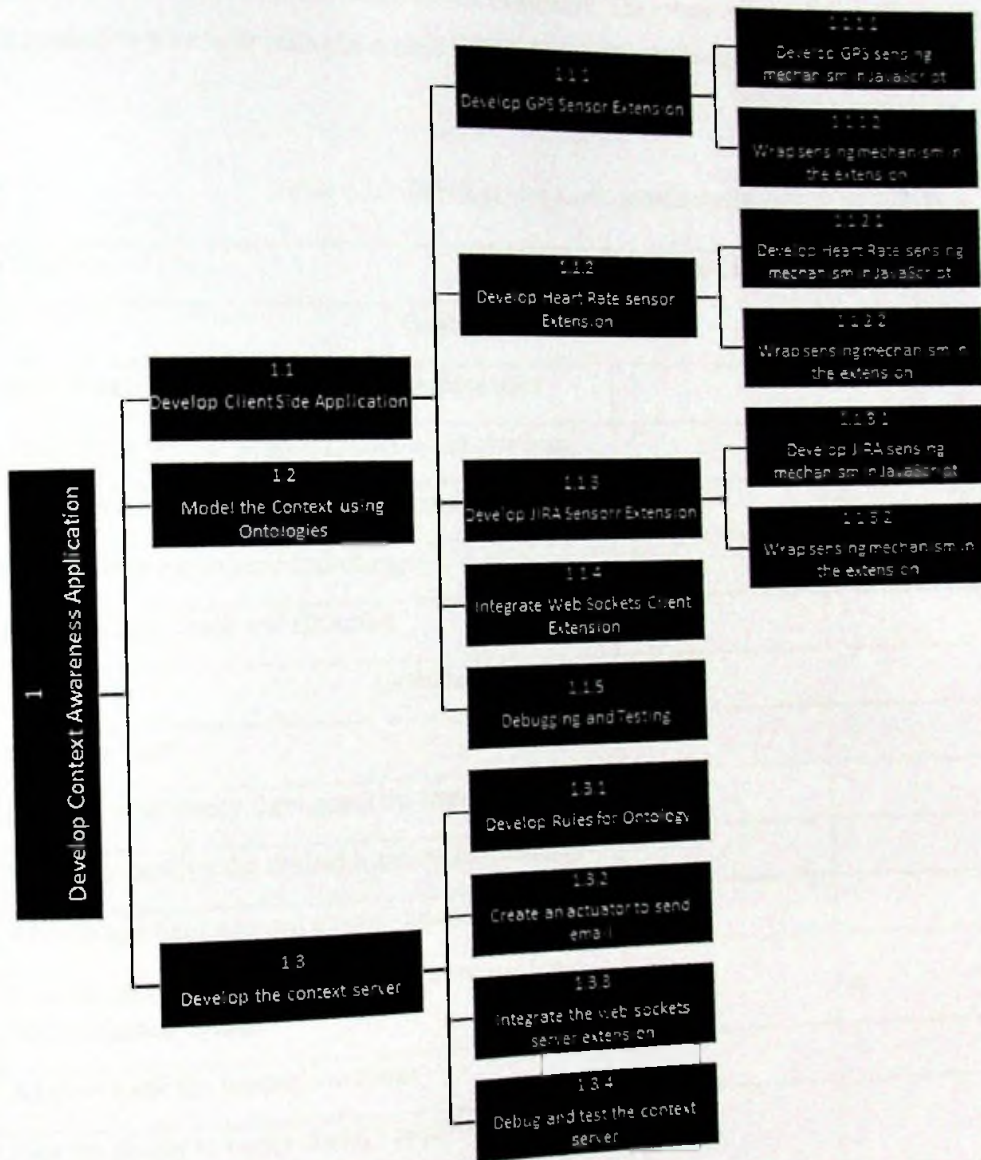


Figure 4.7: Work breakdown structure of the third sample application

While the developers were working on the tasks they were advised to track time for the duration it took to complete each item in the work breakdown structure.



#### 4.3.4. Test findings

##### 4.3.4.1. Developer feedback questionnaire

When the development had completed a questionnaire was given to the two developers to identify their experiences they had with the framework. They were asked a series of questions and asked to give their rating in a scale of 5 given that 1 is the lowest and 5 is the highest.

Table 4.16: Developer feedback questionnaire

Question	Delepa's Score	Rifhan's Score
<b>General Development</b>		
Able to use with existing tools and technologies	5	5
Able to use with established patterns and practices	4	5
Able to easily respond to requirement changes	5	4
Able to respond to technical changes	4	4
Able to easily track and fix errors	3	4
<b>Usability of the Framework</b>		
Easier to learn	4	4
Has the consistency throughout the framework	5	5
Easier to develop the desired business functionality	4	4
Able to use hard and soft sensing mechanisms	5	5
Has the ability to focus on context awareness logic than technical details	4	4
Able to track the internal workings	5	5
Has the ability to easily identify required methods to perform the desired action	5	5
Easier to model any domain or environment using ontologies	5	5
Easier to specify inferencing rules for the context model	4	4
Easier to develop extensions for the framework	3	3
Easier to develop sensing extensions	3	3
Easier to develop transport extensions	3	3



4.3.4.2. Time taken to develop each task

Table 4.4: Time taken complete each task

TaskID	Task Name	Parent Task ID	Assignee	Duration (hours)
1	Develop context-aware application	N/A		8.75
1.1	Develop client-side application	1		6.75
1.1.1	Develop GPS sensor extension	1.1		0.75
1.1.1.1	Develop GPS sensing mechanism in JavaScript	1.1.1	Deleepa	0.5
1.1.1.2	Wrap sensing mechanism in the extension	1.1.1	Deleepa	0.25
1.1.2	Develop Heartrate sensor extension	1.1		1.25
1.1.2.1	Develop Heartrate sensing mechanism in JavaScript	1.1.2	Deleepa	1
1.1.2.2	Wrap sensing mechanism in the extension	1.1.2	Deleepa	0.25
1.1.3	Develop Jira sensor extension	1.1		3.25
1.1.3.1	Develop Jira sensing mechanism in JavaScript	1.1.3	Rifhan	3
1.1.3.2	Wrap sensing mechanism in the extension	1.1.3	Rifhan	0.25
1.1.4	Integrate the web socket client extension for mobile	1.1	Deleepa	0.5
1.1.5	Debug and test the client application	1.1		1
1.2	Model the context using ontologies	1		0.25
1.3	Develop the context server	1		2.75
1.3.1	Develop rules for the ontology	1.3	Rifhan	0.25
1.3.2	Create an actuator to send email	1.3	Rifhan	1
1.3.3	Integrate the web sockets server extension	1.3	Deleepa	0.5
1.3.4	Debug and test the context server	1.3		1



#### 4.3.4.3. Developer feedbacks

##### **Deleepa's Feedback**

It was a good learning for me to use this framework since I haven't developed context-aware applications before. The concept behind the framework is straightforward which makes it easier for new developers like us to understand and learn. However, developing extensions for *perception.js* has been a bit difficult to me because it uses complex JavaScript patterns such as closures. However, because of the extensions the framework becomes more agile that helps us to respond to technical changes.

##### **Refhan's Feedback**

This has been a whole new experience for me. Being a backend developer, I find this framework helpful for my future work. This framework can also be used to develop applications in small teams so managing software projects also becomes easier with this framework. I think this framework would be more better if rules can be generated dynamically with the data gathered by sensors. That way the system can become more intelligent. Also, it would be better if latest trends such as Neural Networks and Deep Learning can be adopted to the framework.

#### 4.3.5. Analysis of usability

As mentioned in Literature Review, when the developer uses or learn a framework and if the framework is poorly designed, he would encounter six barriers [52]. The framework was designed to avoid such barriers to ensure that the framework is usable by developers. Although use barriers can be overcome by creating appropriate documentation, the framework was designed to address the following barriers;

##### **Avoidance of design barriers**

Design barriers defined as cognitive difficulties of a programming problem, separate from the notation used to represent a solution [52]. Since *perception.js* enables the developer to model real world context using ontologies and define rules, this enables the developer to easily represent a real-world scenario using the framework. Following such a modelling approach has enabled to overcome this barrier.



According to the responses of the questionnaire, both the developers have scored 4 points when it comes to making it easier to use the desired business functionality of the framework. In addition to that both the developers have given a point of 5 when ease of modeling a context is concerned, and a point of 4 when ease of specifying rules is concerned. When the times are taken into consideration which are 15 mins each to model the context and to specify the rules, solving design barriers is more reinforced. Also, the developers have agreed that using the framework the context awareness logic can be focused more than focusing on the technical details.

### **Avoidance of election barriers**

Election barriers prevents a developer from finding the best suited interface to achieve a particular behavior [52]. Perception.js consists of a global object known as 'perception' which can be used in any platform such as node.js, Cordova, or a web browser. Through portability the underlying technical complexity of using different methods/interfaces for different platforms were eliminated. Instead generic methods were provided to developers to map sensors (the sensors () method), map actuators (the actuators () method), create rules (the ontology () method), and model ontology (the model () method). Such method names were selected to enable the developers to easily find the required method to perform his task.

Taking the responses of the questionnaire, the developers have agreed that it is easier to develop the business functionality by giving a score of 4 points in the questionnaire. Also, developers have agreed that the consistency is maintained throughout the framework.

However, ease of developing extensions is taken into consideration, the both developers have given a score of 3. Taking the time duration to develop each extension is taken into consideration, the Jira extension had taken 3 hours and 15 minutes due to the complexity of integration, whereas the developing the GPS sensor has taken 45 minutes to develop, whereas the heart rate sensor had taken 1 hour and 15 minutes to develop. However, to create a wrapper around the sensor logic each developer has logged 15 minutes. Therefore, it can be concluded that developing extensions involve additional technical complexity which is outside of the scope of the framework.

### **Avoidance of coordination barriers**

Prevents a developer from combining a programming interface with a programming system to achieve complex behaviors [52]. Using extensibility features this barrier was avoided which



enables the developer to extend the perception.js framework by creating an extension using the programming interfaces provided by perception.js to utilize the features of the underlying platform (i.e. node.js, Cordova or the web browser).

Taking the responses of the questionnaire into consideration, both the developers have given a score of 5 points when ease of using the framework together with existing tools is taken into consideration. In addition to that when the ease of using the framework with existing best practices and patterns is taken into consideration the developers have scored 4 and 5 respectively. However, when the time taken to debug the client and server applications each the developers have logged 1 hour for each task. However, it can be concluded that the design decisions taken to avoid coordination barriers are effective.,

#### **Avoidance of information barriers**

Prevents a developer from acquiring information about a program's internal behavior [52]. When the architectural baseline was implemented, an eventing mechanism was implemented in order to identify the internal workings of the framework which enables the developers to know events such as an expression tree is created, an expression tree is evaluated, etc...

Taking the responses of the questionnaire into consideration, both the developers have agreed on a point of 5 when the ability of tracking internal workings of the framework is concerned. Therefore, it can be concluded that the framework is suitable of preventing information barriers.

In addition to the barriers of use the developers the questionnaire included questions related to the agility (being able to respond to changes). The developers have agreed on scores of 5 and 4 respectively when addressing requirement changes is taken into consideration. In addition, the developers have agreed on scores of 4 and 4 when facing technical changes is taken into consideration. Therefore, it can be concluded that the framework can also be used to develop real-world business applications.

The developers have also given scores higher than 4 for the functionality related questions in the questionnaire such as the ease of modelling the context, ease of specifying the rules, etc.... Therefore, it can be concluded that the framework meets the usability goals.



# CHAPTER 4

## CONCLUSION





## 5.1. Summary of the Developed Framework

The developed framework enables the developer to easily develop context awareness features related to context acquisition, context processing and context presentation. The framework has extensibility features which enable the developers to integrate various sensing technologies (i.e. hard sensors, and soft sensors), integrate various transport protocols (i.e. TCP, WebSockets), and integrate various storage mechanisms (Relational databases, NoSQL databases). In addition, the framework has portability features which enable the developers to embed the framework in applications developed for platforms such as node.js, Cordova, and the web browser. The developers can transform their applications into a 'Sentient Object' which is the reference architecture the framework is based on.

Context pre-processing can also be performed using the framework to handle sensor imperfections by providing a callback function to the framework for each sensor integration. Context processing in the framework is handled using ontologies and rules which is internally represented using expression trees. In addition to that developing context servers for context processing can be done using the framework. Context presentation is handled using actuators which can be created using the framework.

## 5.2. Advantages of the Framework

- The framework integrates well with widely used technologies in the industry such as databases (i.e. MySQL, Redis, MemSQL), message queues (RabbitMQ, Kafka, or HiveMQ), machine learning tools (i.e. Tensor Flow)
- The Framework integrates well with good JavaScript coding practices such as separation of concerns, DRY (Don't Repeat Yourself) Principle, SOLID Principles.
- The framework core can easily be ported among different platforms such as Node.js, Cordova, and Web Browser.
- The Framework is designed with good usability features that enables the developers for rapid development and to reduce learning curve.
- The framework can deal with multiple heterogeneous devices, systems and technologies using extensions.
- The framework supports localized scalability through the sentient object model architecture.



### 5.3. Limitations

Upon conducting testing, the test results have identified the following limitations in the framework;

- In the context server approach currently, the context model is required to be defined in both the client and the server, therefore inconsistencies between context models can occur. As a result, the application could become unstable. As a future work, an automated mechanism can be implemented to check the consistencies in the context models to make the application more stable.
- Developers have found out that developing extensions for perception.js is comparatively difficult that using the framework to develop context-aware applications. The developer interviews have revealed that to develop as extension the 'closure' design pattern should be implemented. Since this approach is counterproductive by the developers, a new design pattern can be introduced for the developers to develop extensions.

### 5.4. Future Enhancements

Even though the framework facilitates ease of context-aware application development, it could further improve its capabilities if the following features are added in the future;

- If a feature is available to import an ontology from an OWL file and rules from SWRL file, the ontology and the rules can be separated from the source code which improves the maintainability of the application.
- If context model can be partially defined in each sentient object (i.e. sensor acquisition part in the client side, and reasoning part on server side) the context model would be more maintainable.
- Developer productivity can be further improved if the framework can be wrapped using a language such as TypeScript because it enables the developer to follow good practices such as SOLID principles to improve maintainability.
- The framework can be modified to dynamically create inferencing rules in runtime which enables the context-aware application to learn from user's behavior and contextual information and optimize its adaptability.



- A graphical user interface can be developed to make it easier to model and create rules without using code.
- A mechanism can be implemented to offload processing to the context server if the client (i.e. mobile) runs out of battery or processing power.
- If features for aggregating sensors are provided through the framework, uncertainties in contextual information can be reduced.

## 5.5. Conclusion

In summary, this research is focused on developing a framework for software developers to mitigate the technical complexity in developing context-aware applications to reduce the development time and improve the overall quality of the application.

Moreover, this research was conducted in the academia related to context acquisition, context processing and presentation to identify their challenges, and approaches that have been presented to overcome such challenges. To improve the usability of the framework research was conducted on API usability. In addition to that various JavaScript design patterns were also presented to implement quality attributes such as extensibility, and portability.

In Design and Implementation, the requirements, architecture and detailed design and the internal implementations of the framework were presented based on the findings of literature review. The chapter also included detailed and in-depth descriptions of mechanisms related to context modelling with ontologies, context processing and inferencing using expression trees, mapping extensions (sensors, transport, and storage) to ontologies, and source code syntaxes on how to perform functionalities of context awareness.

In Testing and Analysis, the framework was tested against three different scenarios to verify whether the framework meets the functionality, usability, extensibility, and portability requirements. For all the scenarios sample applications were developed using the framework. In the last scenario, a usability test was conducted to verify whether the framework can be used by real-world software developers to develop a real-world context-aware application.



In addition to that to integrate existing technologies to the framework the thesis has presented methods on how to accomplish this task for the areas of acquiring contextual information, and storing contextual information.



## REFERENCES

- [1] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century", *Computer*, vol. 36, no. 3, pp. 25-31, 2003.
- [2] M. Weiser, "The Computer for the 21st Century", *Sci Am*, vol. 265, no. 3, pp. 94-104, 1991.
- [3] Pervasive computing Definition and Meaning, 2015. [Online]. Available: <http://www.dictionarofengineering.com/definition/pervasive-computing.html>.
- [4] V. Basili, "The Role of Experimentation: Past, Present, Future (keynote presentation)," *International Conference on Software Engineering*, 1996.
- [5] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721-733, 2002.
- [7] A.K. Dey, G.D. Abowd, "Towards a Better Understanding of Context and Context-Awareness", 2002.
- [8] P. Mehra, "Context-Aware Computing: Beyond Search and Location-Based Services", *IEEE Internet Comput.*, vol. 16, no. 2, pp. 12-16, 2012.
- [9] B. Schilit and M. Theimer, "Disseminating active map information to mobile hosts", *IEEE Network*, vol. 8, no. 5, pp. 22-32, 1994.
- [10] M. Satyanarayanan, "Pervasive computing: vision and challenges", *IEEE Personal Communications*, vol. 8, no. 4, pp. 10-17, 2001.
- [11] A. Holzinger, A. Nischelwitzer, S. Friedl and B. Hu, "Towards lifelong learning: three models for ubiquitous applications", *Wireless Communications and Mobile Computing*, vol. 10, no. 10, pp. 1350-1365, 2008.
- [12] A. Cockburn, *Agile Software Development*. Reading, Massachusetts: Addison Wesley Longman, 2001.
- [13] L. Williams, W. Krebs, L. Layman, A. Ann and P. Abrahamsson, "Toward a Framework for Evaluating Extreme Programming", 2015.
- [14] Ranganathan, A., McGrath, R.E., Campbell, R.H., Mickunas, M.D. "Use of ontologies in a pervasive computing environment. Knowl". *Eng. Rev.* 18, 209-220 (Sep 2003)
- [15] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *IEEE Computer*, vol. 31, no. 5, pp. 23-31, May 1998.
- [16] Chen, H., Finin, T. and Joshi, A. (2004) "An ontology for context-aware pervasive computing environments", *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, Vol. 18, No. 3, pp.197-207.
- [17] Indulska, J. and Sutton, P. (2003) "Location management in pervasive systems", *CRPITS'03: Proceedings of the Australasian Information Security Workshop*, pp.143-151.
- [18] P.D. Hagighi, S. Krishnaswamy, A. Zaslavsky, M.M. Gaber, "Reasoning About Context in Uncertain Pervasive Computing Environments", 2008



- [19] S Bobek, G. J. Nalepa, "Incomplete and Uncertain Data Handling in Context-Aware Rule-Based Systems with Modified Certainty Factors Algebra", 2014
- [20] D. Kramer, S. Oussena, T. Clark, P. Komisarczuk, "An extensible, self-contained, layered approach to context acquisition", 2011
- [21] K. Da, M. Dalmau and P. Roose, "A Survey of adaptation systems", *International Journal on Internet and Distributed Computing Systems*, vol. 2, no. 1, pp. 1-18, 2015.
- [22] P. Fortier and H. Michel, "Computer systems performance evaluation and prediction". Burlington, MA: Digital Press, 2003.
- [23] K. Henriksen, J. Indulska, A. Rakotonirainy "Modeling Context Information in Pervasive Computing Systems", 2008.
- [24] M. Baldauf, S. Dustdar and F. Rosenberg, "A survey on context-aware systems", *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, p. 263, 2007.
- [25] T. Strang, C. Linnhoff-Popien, "A Context Modeling Survey", 2006.
- [26] Strang, T., Popien, C. "A context modeling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management", *UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing (2004)*
- [27] S Amulowitz, M., M Ichahahelles, F., and L Innhoff - P Opie , C. Capeus: An architecture for context-aware selection and execution of services. In *New developments in distributed applications and interoperable systems (Krakow, Poland, September 17-19 2001)*, Kluwer Academic Publishers, pp. 23–39.
- [28] Chen, P.-S. The entity-relationship model - toward a unified view of data. *ACM Transaction on Database Systems* 1, 1 (March 1976), 9–36.
- [29] U Schold, M., and G R Uninger, M. *Ontologies: Principles, methods, and applications*. *Knowledge Engineering Review* 11, 2 (1996), 93–155.
- [30] DE B Ruijn, J. *Using Ontologies - Enabling Knowledge Sharing and Reuse on the Semantic Web*. Tech. Rep. Technical Report DERI-2003-10-29, Digital Enterprise Research Institute (DERI), Austria, October 2003.
- [31] X.H. Wang, D.Q. Zhang, T. Gu, HK. Pung. *Ontology Based Context Modelling and Reasoning using OWL*. School of Computing, National University of Singapore, Singapore.
- [32] B.Y Lim, A.K Dey, "Toolkit to Support Intelligibility in Context-Aware Applications", (2010)
- [33] S Bobek, G. J. Nalepa, "Incomplete and Uncertain Data Handling in Context-Aware Rule-Based Systems with Modified Certainty Factors Algebra", 2014
- [34] K. Hamadache, E. Bertin, A. Bouchacout, I. Benyahia, "Context-Aware Communications Services: an Ontology Based Approach", 2007
- [35] Chefrou, D. "Developing component based adaptive applications in mobile environments". *Proceedings of the 2005 ACM symposium on Applied computing*. p. 1146–1150. SAC '05, ACM, New York, NY, USA (2005)
- [36] Context – AWARE, 2017. [Online]. Available: <http://www.awareframework.com/context/>



- [37] Ambient Dynamix – The plug-and-play context framework, 2017. [Online]. Available: <http://ambientdynamix.org/>
- [38] J.E. Bardram, “The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications”, 2005
- [39] A. Badii, M. Crouch, C. Lallah, “A Context-Awareness Framework for Intelligent Networked Embedded Systems”, 2010
- [40] GitHub - hubiquitus/hubiquitus-core: Hubiquitus core system. Actor oriented framework massively distributed, 2017. [Online]. Available: <https://github.com/hubiquitus/hubiquitus-core>
- [41] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, “Project aura: Toward distraction-free pervasive computing,” *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 22–31, Apr. 2002. [Online]. Available: <http://dx.doi.org/10.1109/MPRV.2002.1012334>
- [42] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, “A middleware infrastructure for active spaces,” *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74–83, Oct. 2002. [Online]. Available: <http://dx.doi.org/10.1109/MPRV.2002.1158281>
- [43] A. Gluhak and W. Schott, “A wsn system architecture to capture context information for beyond 3g communication systems,” in *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, dec. 2007, pp. 49–54. [Online]. Available: <http://dx.doi.org/10.1109/ISSNIP.2007.4496818>
- [44] D. Conan, R. Rouvoy, and L. Seinturier, “Scalable processing of context information with cosmos,” in *Proc. 7th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, ser. DAIS’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 210–224. [Online]. Available: <http://hal.inria.fr/docs/00/15/50/45/PDF/article.pdf>
- [45] B. Firmer, R. S. Moore, R. Howard, R. P. Martin, and Y. Zhang, “Poster: Smart buildings, sensor networks, and the internet of things,” in *Proc. 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys ’11. New York, NY, USA: ACM, 2011, pp. 337–338. [Online]. Available: <http://doi.acm.org/10.1145/2070942.2070978>
- [46] J. Herbert, J. O’Donoghue, and X. Chen, “A context-sensitive rule-based architecture for a smart building environment,” in *Future Generation Communication and Networking, 2008. FGCN ’08. Second International Conference on*, vol. 2, dec. 2008, pp. 437–440. [Online]. Available: <http://dx.doi.org/10.1109/FGCN.2008.169>
- [47] I. Sommerville, *Software engineering*. Harlow, England: Addison-Wesley, 2007.
- [48] J.F. Pane, B.A. Myers “Using HCI Techniques to Design a More Usable Programming System”, 2007.
- [49] B. du Boulay, T. O’Shea, and J. Monk, “The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices,” in *Studying the Novice Programmer*, E. Soloway and J. C. Spohrer, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1989, pp. 431–446.
- [50] D. C. Smith, A. Cypher, and J. Spohrer, “KidSim: Programming Agents without a Programming Language,” *Communications of the ACM*, vol. 37, pp. 54–67, 1994.
- [51] K. Cwalina and B. Abrams, *Framework design guidelines*. Upper Saddle River, NJ: Addison-Wesley, 2006.



- [52] A.J. Ko, B.A. Myers, H.H. Aung, "Six Learning Barriers in End-User Programming Systems", 2005.
- [53] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.
- [54] B.A. Myers. "Leveraging Software Architectures to Guide and Verify the Development of Sense-Compute-Control Applications", 2015.
- [55] A. Fitzpatrick, G. Biegel, S. Clarke, V. Cahill, "Towards a Sentient Object Model", 2001.
- [56] G. Richards, S. Lebresne, B. Burg and J. Vitek, "An analysis of the dynamic behavior of JavaScript programs", ACM SIGPLAN Notices, vol. 45, no. 6, p. 1, 2010.
- [57] D. Crockford, JavaScript. Beijing: O'Reilly, 2008.
- [58] S. Stefanov, JavaScript patterns. Sebastopol, CA: O'Reilly, 2010.
- [59] A.K. Hota, D. Madan Prabhu. "NODE.JS - Lightweight, Event driven IO web development", 2014.
- [60] I. Roth, "StrongLoop | What Makes Node.js Faster Than Java?", Strongloop.com, 2015. [Online]. Available: <http://strongloop.com/strongblog/node-js-is-faster-than-java/>. [Accessed: 24- Feb- 2015].
- [61] Cordova, 2015. [Online]. Available: <http://cordova.apache.org/>. [Accessed: 24- Feb- 2015].
- [62] Enterprise Integration Patterns – Polling Consumer, 2018. [Online]. Available: <http://www.enterpriseintegrationpatterns.com/patterns/messaging/PollingConsumer.html>
- [63] Enterprise Integration Patterns – Event Driven Consumer, 2018. [Online]. Available: <http://www.enterpriseintegrationpatterns.com/patterns/messaging/EventDrivenConsumer.html>
- [64] Firebase, 2017. [Online]. Available: <https://firebase.google.com>
- [65] RethinkDB the open-source database for the realtime web, 2018. [Online]. Available: <https://www.rethinkdb.com/>
- [66] Plugin Search – Apache Cordova, 2018. [Online]. Available: <https://cordova.apache.org/plugins/>
- [67] raspisensors - npm, 2018. [Online]. Available: <https://www.npmjs.com/package/raspisensors>
- [68] Real-time machine learning with TensorFlow, Kafka, and MemSQL | InfoWorld, 2018. [Online]. Available: <https://www.infoworld.com/article/3237166/machine-learning/real-time-machine-learning-with-tensorflow-kafka-and-memsql.html>
- [69] TensorFlow Wide & Deep Learning Tutorial | TensorFlow, 2018. [Online]. Available: [https://www.tensorflow.org/tutorials/wide\\_and\\_deep](https://www.tensorflow.org/tutorials/wide_and_deep)
- [70] GitHub - RedisLabsModules/redis-ml: Machine Learning Model Server, 2018. [Online]. Available: <https://github.com/RedisLabsModules/redis-ml>
- [71] GitHub - antirez/neural-redis: Neural networks module for Redis, 2018. [Online]. Available: <https://github.com/antirez/neural-redis>

