CARL VON OSSIETZKY universität OLDENBURG

FAKULTÄT II – INFORMATIK, WIRTSCHAFTS- UND RECHTSWISSENSCHAFTEN
DEPARTMENT FÜR INFORMATIK

Reference Architecture for Smart Environmental Information Systems

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels eines

Doktors der Ingenieurswissenschaften (Dr.-Ing.)

angenommene Dissertation

von Frau Ruthbetha Kateule

geboren am 15.03.1987 in Tanzania

Oldenburg, September 2019

Gutachter:

Prof. Dr. Andreas Winter

Prof. Dr. Jorge Marx Gómez

Tag der Disputation:

17.09.2019

This thesis is dedicated to my beloved late father Beatus Ngalela Kateule for his unconditional love and support. Thank you so much dad.

Acknowledgements

This thesis represents the critical point of my career, and it incorporates knowledge from different sources, professionals and academics. During this research, I was privileged to get support, guidance and encouragement from many people and I would like to take this opportunity to acknowledge their contributions.

Firstly, I would express my profound gratitude to my supervisor Prof. Dr. Andreas Winter for initiating the research and his patience, understanding, immense knowledge, support and guidance during the past four years. I could not have imagined having a better supervisor and mentor for my PhD study. I extend my special thanks to Prof. Dr. Jorge Marx Gómez (my second supervisor) for his valuable inputs and support throughout my research.

I would like to express my sincere gratitude and appreciation to my colleagues; Dr. Christian Schönberg, Dr. Dilshodbek Kuryazov, Johannes Meier, Jan Jelschen, and Muzaffar Artikov for their valuable inputs, time and encouragements that motivated me to complete my thesis. I would also like to thank my mother Alwina Unami, whole family and friends for your continuous support throughout this journey. A special thanks to my friends Dr. Tupokigwe Isagah and Ola Mustafa for their support.

Finally, but most importantly, I would like to thank the German Academic Exchange Service (DAAD) for their financial support throughout the research and also University of Dar es Salaam (UDSM) for granting the study leave.

Abstract

Smart environmental information systems (SEISs) are playing a more prominent role in modern society as the utilization of SEISs has dramatically increased for effectively monitoring and controlling environmental events. This is achieved by firstly, monitoring and displaying the environmental conditions. Afterwards, the environmental conditions are analysed and manipulated with the intent of preventing or reducing the effects of various environmental phenomena, i.e. air pollution, fire, flood, road traffic congestion etc. The wireless sensor-actuator networks (WSANs) enable SEISs to monitor the environment in both non-real-time and real-time on remote and inaccessible places, diagnostics, and finally protection of further environmental severe phenomena. The intention of SEISs is not restricted to only gathering and manipulating data from various locations, but also to disseminate information required by practitioners such as researchers, planners, and policy-makers for making crucial decisions regarding the management and improvement of the environment.

Although there is a high potential, the growth of SEISs is coupled with the existence of various hardware devices, extensive use of off-the-shelf components, growth of the size, various stakeholders and multiple programming languages. The inevitable complexity of such systems makes the *development and maintenance of the SEISs to be difficult, time-consuming and require extensive knowledge of the domain*. Dealing with such complexity can be challenging for even most experienced software engineers such as architects, developers and maintainers if start from scratch. Since, in spite of the considerable relevance and established architectural knowledge of the existing SEISs, there is a lack of a *reference architecture for SEISs* that supports the development of SEISs. This can be achieved by replacing the ad-hoc solutions (which focus on single-solution development) with reuse-driven and knowledge-based best practices for SEISs.

A conventional method for maximizing the reusability of established knowledge and best-practices of systems that belong to a particular domain is a *Reference Architecture*. The reference architecture encompasses systems common sets of requirements, architecture designs, and software elements from the previous software developments mainly existing concrete software architectures that can be applied to a number of concrete systems in a

particular domain. The elements of these software architectures which are being reused in the reference architecture are proven to be good. It is argued that the employment of reference architecture in SEISs would facilitate the development of SEISs by reducing the SEISs development time and efforts. Therefore, the *main objective* of this thesis is to propose a reference architecture for SEISs that would serve as a baseline for creating concrete SEISs.

An effective approach to describe an understandable and easier maintainable software architectural description is by partitioning the software architecture into a number of separate views. Each view addresses one aspect of the architecture using a specific set of models from a specific viewpoint. Therefore this thesis proposes a required *set of Viewpoints for SEISs*, some of these viewpoints are adopted from Siemens view model. Essential viewpoints for SEISs include *conceptual*, *module*, *execution*, *code*, *topology*, and *data* viewpoints. Furthermore, these viewpoints have been integrated to ensure conceptual integrity and consistency in the construction of a well-integrated system. Such viewpoints are applied in the development of the *reference architecture for SEISs (RefSEISs)*. The description of the RefSEISs follows the ISO/IEC/IEEE 42010:2011 standard for the architectural description and adopts the best practices established in the existing SEISs. Thus, the proposed RefSEISs encompasses various software artefacts and knowledge such as the sets of stakeholders, concerns, requirements and architectural designs that could be reused in the construction of concrete SEISs architectures.

As proof of concept, the proposed RefSEISs is applied to several applications. First, the RefSEIS is applied in the *construction of forest fire detection system* as new concrete SEIS to demonstrate the applicability of the proposed RefSEISs in real scenarios and verification of the design with the requirements. Secondly, a conceptual validation is presented in which the proposed RefSEISs is used for *mapping some of the existing SEISs*; *Forest Fire Monitoring System (IPNAS)*, *Urban Air Quality Monitoring System*, *Flood Risk Assessment System* and *Indoor Air Quality Monitoring System*. These mappings demonstrated the proposed RefSEISs provides insights into how to improve the existing SEISs since the similarities and differences of existing SEISs with respect to RefSEISs have been described.

An analysis of these experiences showed that the proposed RefSEISs effectively supports the constructions of new concrete SEISs since the time and efforts required for developing and maintaining forest fire detection system have been substantially reduced. And also the mappings have demonstrated that the proposed RefSEISs can be used to compare and improve the existing SEISs by describing the similarities, differences, and how such SEISs can be improved. Results indicated that the RefSEISs has a positive influence in achieving the requirements of SEISs. This RefSEIS presents, therefore, good perspectives to be adopted and contribute to the development and maintenance of SEISs.

Kurzfassung

Intelligente Umweltinformationssysteme (SEISs) spielen in der modernen Gesellschaft eine immer herausragendere Rolle, da die Nutzung von SEISs zur wirksamen Überwachung und Kontrolle von Umweltereignissen stark zugenommen hat. Dies wird erreicht, indem zum einen Umweltereignisse überwacht und angezeigt werden. Zum anderen werden Umweltereignisse analysiert und manipuliert, um die Auswirkungen verschiedener Umweltphänomene, z.B. Luftverschmutzung, Feuer, Erdrutsche, Verkehrsstaus usw., zu verhindern oder zu reduzieren. Drahtlose Sensor-Aktuator-Netzwerke (WSANs) ermöglichen es einem SEIS, die Umgebung sowohl verzögert als auch in Echtzeit an abgelegenen und schwer zugänglichen Orten zu überwachen und flexibel vor schwerwiegenden Folgen zu schützen. Ziele eines SEIS sind nicht nur das Sammeln und Manipulieren von Daten von verschiedenen Standorten sondern auch die Verbreitung von Informationen an Praktiker wie Forscher, Planer und Entscheidungsträger, um ihnen Entscheidungshilfen zum Management und zur Verbesserung der Umwelt an die Hand zu geben.

Obwohl es ein hohes Potenzial gibt, ist das Wachstum von SEISs mit dem Vorhandensein verschiedener Hardwaregeräte, der umfangreichen Nutzung von off-the-shelf-Komponenten, dem Wachstum der Größe, verschiedenen Interessengruppen und mehreren Programmiersprachen verbunden. Die unvermeidliche Komplexität solcher Systeme macht die *Entwicklung und Wartung von SEISs komplex, zeitaufwändig und erfordert umfangreiche Kenntnisse der Domäne*. Der Umgang mit dieser Komplexität kann selbst für erfahrene Software-Entwickler wie Architekten, Entwickler und Wartungspersonal eine Herausforderung sein, wenn sie ein SEIS von Grund auf neu entwickeln. Obwohl sie eine erhebliche Relevanz besitzt und die Architektur vieler bestehender SEISs gut bekannt ist, fehlt eine *Referenzarchitektur für SEISs*, die die Entwicklung von SEIS unterstützt. Dies kann erreicht werden, indem die Ad-hoc-Lösungen (die sich auf die individuelle Entwicklung genau eines einzigen SEIS konzentrieren) durch bewährte und wissensbasierte Verfahren insbesondere mit dem Ziel der Wiederverwendung ersetzt werden.

Eine bewährte Methode zur Erhöhung der Wiederverwendbarkeit von etabliertem Wissen und Bestpraktiken innerhalb von Systemen, die zu der gleichen Domäne gehören, ist

die Nutzung einer *Referenzarchitektur*. Die Referenzarchitektur beschreibt allgemeine Anforderungen, Architekturentwürfe und Softwareelemente aus früher entwickelten Systemen dieser Domäne, die auf neue konkreter Systeme innerhalb der gleichen Domäne angewendet werden können. Es wird argumentiert, dass die Verwendung einer Referenzarchitektur in SEISs die Entwicklung von SEISs erleichtern würde, indem die Entwicklungszeit und der Entwicklungsaufwand der SEIS verringert würde. Das Hauptziel dieser Arbeit ist es daher, eine Referenzarchitektur für SEISs vorzuschlagen, die als Basis für die Erstellung konkreter SEIS dienen soll.

Ein effektiverer Ansatz zum verständlichen, übersichtlichen und leichter wartbaren Beschreiben von Softwarearchitekturen besteht darin, die Softwarearchitektur in mehrere separate Sichten zu unterteilen. Jede Sicht zeigt für einen Aspekt der Architektur aus einem bestimmten Blickwinkel die dazu passenden Modelle. In dieser Arbeit wird daher zunächst eine *Menge von Sichten für SEIS vorgeschlagen*. Einige dieser Sichten werden aus dem Siemens-View-Modell übernommen. In den zentralen Sichten für SEIS werden u.a. Komponenten, Module, Ausführung, Code, Topologie und Daten beschrieben. Darüber hinaus wurden diese Gesichtspunkte integriert, um die konzeptionelle Integrität und Konsistenz beim Aufbau eines gut integrierten Systems sicherzustellen. Solche Standpunkte werden bei der Entwicklung der *Referenzarchitektur für SEISs (RefSEISE)* verwendet. Die Beschreibung der RefSEISs folgt dem ISO/IEC/IEEE 42010:2011 Standard für die Architekturbeschreibung und übernimmt die best-practices, die in den bestehenden SEISs eingeführt wurden. So umfassen die vorgeschlagenen RefSEISs verschiedene Software-Artefakte und Kenntnisse, wie z.B. die Mengen an Interessengruppen, Bedenken, Anforderungen und architektonische Entwürfe, die für den Bau von konkreten SEIS-Architekturen wiederverwendet werden können.

Als Proof of Concept werden die vorgeschlagenen RefSEIS auf mehrere Anwendungen angewendet. Erstens wird das RefSEIS beim Bau eines Waldbrandmeldesystems als neues konkretes SEIS angewendet, um die Anwendbarkeit der vorgeschlagenen RefSEIS in realen Szenarien und die Überprüfung des Designs mit den Anforderungen zu demonstrieren. Zweitens wird eine konzeptionelle Validierung vorgestellt, in der die vorgeschlagenen RefSEISs zur Abbildung der vorhandenen SEISs verwendet werden; *Waldbrandüberwachungssystem (IPNAS)*, *System zur Überwachung der Luftqualität in Städten*, *Hochwasserrisikobewertungssystem* und *System zur Überwachung der Luftqualität in Innenräumen*. Diese Zuordnungen haben gezeigt, dass die vorgeschlagenen RefSEIS Einblicke in SEIS bieten, da die Ähnlichkeiten und Unterschiede der bestehenden SEIS in Bezug auf RefSEIS beschrieben wurde.

Eine Analyse dieser Erfahrungen hat gezeigt, dass die vorgeschlagenen RefSEIS die Entwicklung von konkreten SEIS effektiv unterstützen, da die Zeit und der Aufwand für die Entwicklung und Weiterentwicklung von Waldbranderkennungssystemen erheblich reduziert wurden. Auch die Zuordnungen haben gezeigt, dass die vorgeschlagenen RefSEIS zum Vergleich mit und zur Verbesserung der vorhandenen SEIS verwendet werden können, indem Ähnlichkeiten, Unterschiede und Möglichkeiten zur Verbesserung dieser SEIS beschrieben werden. Die Ergebnisse zeigen, dass die RefSEIS einen positiven Einfluss auf die Erfüllung der Anforderungen für SEIS haben. Diese RefSEIS bieten daher gute Möglichkeiten, die zur Entwicklung und Aufrechterhaltung von SEIS beitragen und die zukünftig weiterentwickelt werden können.

Table of contents

List of figures

List of tables

Part I

Challenges

This thesis takes the step towards improving the development and maintenance of SEISs by developing a reference architecture for SEISs. The thesis is divided into five parts; Part I: Challenges, Part II: Background and Related Work, Part III: Approach, Part IV: Evaluation and Part V: Conclusion. The thesis commences with Part I, in which the motivation, research questions, and methodology are presented in Chapter 1.

Chapter 1

Introduction

This chapter introduces the main areas in this thesis, which are linked to the smart environmental information systems (SEISs) and software architecture paradigms. In which the motivation, research questions, and methodology are presented.

1.1 Motivation

Environmental information systems manage data about the soil, water, air, and other things such as vehicles in the world around us [Günther, 1997]. These systems intend to support high-level decisions regarding the ecosystem, natural resources and other external factors that would affect human life by performing the following tasks; environmental phenomena description and response, environmental monitoring, environmental reporting, data storage and access, modeling and decision-making [Athanasiadis and Mitkas, 2004]. This is achieved through environmental data collection, analysis, storage and meta-data management. Monitoring and controlling of environmental parameters such as humidity, temperature, smoke, pressure, gases, etc., enable the more in-depth understanding of environmental processes so as to overcome the devastating effects of environmental phenomena mainly fire, flood, air pollution, air quality deterioration, freshwater shortages, threats to drinking water quality, landslides, etc. The actual access to up-to-date and high-quality information is critical in minimizing the impacts of environmental phenomena.

As the requirements for accurate and timely information in environmental information systems are increasing, the need for incorporating advanced, and intelligent features in these systems is revealed. In this context, the emergence of sensing-actuating techniques, information technologies and wireless networks led to the proliferation of *low-power sensors and wireless sensor-actuator networks* (WSANs) [Massinissa et al., 2016, Warneke and

Pister, 2002] which are promising to satisfy these requirements. These inventions allow the environmental information system to monitor and control the environmental parameters in both non-real-time and real-time on remote and inaccessible places [Arslan et al., 2014]. This initiated the formation of smart environmental information systems (SEISs). The SEIS incorporates the functions of sensing, actuation and control to analyze the current status of the environment and are associated with the rising of an automatic alarm if some event such as forest fires, flood, air pollution, landslides etc. occurred. The use of SEISs facilitates the establishment of self-monitoring and self-protecting environment. The road traffic congestion, landslides air pollution, and forest fires detection systems are among a range of SEISs. The intention of SEISs is not restricted to only gathering data from various locations, but also to disseminate information required practitioners such as scientists, researchers, planners, and policy-makers for making crucial decisions regarding the management and improvement of the environment.

Various researchers have proposed a number of SEISs as described in Section 3. For instance, [Zhang et al., 2008, Sunkpho and Ootamakorn, 2011, Lozano et al., 2012, Nguyen et al., 2015, Ranjini et al., 2011] proposed forest fire detection, real-time flood monitoring and warning, indoor air quality monitoring, rainfall-induced landslide detection and adaptive road traffic control systems respectively. The diversity of these systems is observed on the application level, and technically they possess similar three subsystems mainly *sensor-actuator*, *information control centre* and *communication subsystems* [Kateule and Winter, 2016]. The *Sensors-Actuators Subsystem* measures physical environment events, i.e., temperature, humidity etc., pre-processing the collected data and alters the state of environmental phenomena to satisfy the intended objectives of a system. The *Information Control Centre Subsystem* integrates, analyzes, processes, stores and visualizes the information as received from the sensors and generates alarms and events via actuators based on the objectives of a particular system. The *Communication Subsystem* facilitates the interactions between the subsystems mentioned above.

Although there is a high potential, the growth of SEISs is coupled with the existence of various hardware platforms, extensive use of off-the-shelf components, growth of the size, various stakeholders and multiple programming languages. This increases the systems' development learning curve and hence makes the *development, maintenance and comparison of SEISs to be difficult, time-consuming and require extensive knowledge of the domain*. Dealing with such complexity can be challenging for even most experienced software engineers such as architects, developers and maintainers if it starts from scratch. In spite of the considerable relevance and established architectural knowledge of the existing SEISs, a *reference architecture for SEISs* that could support a more systematic development of SEISs

while dealing with those recurring challenges, i.e. energy efficiency, maintainability and interoperability in a holistic way is missing, and hence these systems are built from scratch which consumes more time and efforts.

To overcome the challenges mentioned above, automated approaches need to be adopted to facilitate the development and maintenance of SEISs. This can be achieved by replacing the ad-hoc solutions (which focus on single-solution development) with reuse-driven and knowledge-based practices for SEISs. Effective reuse of requirements, architecture designs, knowledge and software elements from the previous software developments facilitate the development and quality of systems [Lin et al., 2009]. In this context, a conventional method for maximizing system reusability is a *development of the reference architecture* which provides reusable solutions or artifacts and best practices from the previous software developments that can be applied to a number of concrete systems in a particular domain [Affonso and Nakagawa, 2013]. Currently, the SEISs have already accumulated enough experience to be comprehended by a reference architecture.

A reference architecture for SEISs is considered as the first and necessary foundation to facilitate the correct understanding of the stakeholders, concerns, requirements, comparisons of SEISs, how the elements of the SEISs interact, the underlying principle of the design, and determine the quality of the SEISs. Therefore, this work proposes a *reference architecture for smart environmental information systems (RefSEISs)*. The proposed RefSEISs can be used as the blueprint or baseline for designing concrete SEISs. Such reference architecture manifests best practices which are derived from the predefined existing architectures and associated with both common and variable parts as described in Section 2.2. The variable parts refer to those parts of the architecture which can be changed or adapted to satisfy the needs of concrete SEISs, while the common elements refer to those elements which are used as they are without being altered or replaced. During this work, the RefSEISs is iteratively built following the best practices principles of recommended practice for describing architecture based on the *ISO/IEC/IEEE 42010:2011 Standard* [ISO/IEC/IEEE, 2011], and *Siemens View Model* [Hofmeister et al., 2000].

1.2 Research Questions

As just discussed, a reference architecture is a baseline for the software systems belongs to a particular domain. The use of reference architecture allows systematically reuse of knowledge and elements when developing concrete software architectures for a specific domain. This thesis aims at facilitating the rapid development and maintenance of SEISs through the design of a reference architecture for SEISs.

The core objective of this thesis is to introduce a reference architecture for Smart Environmental Information Systems (RefSEISs) to facilitate the development and maintenance of SEISs.

The proposed RefSEISs intends to be independent of technologies, abstract, complete and applicable. This main objective is achieved through the investigation of the following research questions:

1. Which are essential aspects or perspectives that are required to describe SEISs?

   The use of a single software architecture view which is overloaded with all-encompassing model makes the resulted software architecture description to be difficult to understand, maintain and also quickly becomes irrelevant in developing a system [Rozanski and Woods, 2005]. A common and more effective approach to describe an understandable and easier maintainable software architectural description is by partitioning the software architecture into a number of separate views. Each view addresses one aspect of the software architecture using a specific set of models from a particular viewpoint. Therefore this research question concerns with an investigation of the current state of SEISs regarding the approaches and technologies used in developing SEISs. This will sketch out the state-of-the-art in SEISs to identify the crucial aspects or perspectives required to frame the concerns of the stakeholders of SEISs. As a result, a required set of viewpoints for describing essential aspects of SEISs is defined. Such viewpoints contain the conventions for constructing, interpreting and analyzing the views of the reference architecture for SEISs.

2. How to describe a reference architecture for SEISs (RefSEISs)?

   The second research question deals with the actual development of the RefSEISs. In which the specified set of viewpoints from the *Research Question 1* will be applied in describing the RefSEISs, and hence the RefSEISs validates the description of the proposed viewpoints. The resulted RefSEISs manifests the best practices of the existing SEISs, and its description follows the ISO/IEC/IEEE 42010:2011 standard [ISO/IEC/IEEE, 2011]. Thus, the proposed RefSEISs will encompass various software artefacts and knowledge such as the sets of stakeholders, concerns, requirements and architectural designs that could be reused in the construction of concrete SEISs architectures.

3. How to apply the proposed reference architecture for SEISs?

   As a proof of concept, the proposed RefSEISs from *Research Question 2* will be applied to several applications to demonstrate its applicability. First, the RefSEIS

will be employed in the construction of forest fire detection system to illustrate the applicability of the proposed RefSEISs in the construction of new concrete SEIS. Then to demonstrate the applicability of the proposed RefSEISs on the existing SEISs. The Forest Fire Monitoring System (IPNAS) [Stipanicev et al., 2018], Urban Air Quality Monitoring System [Ujang et al., 2013], Flood Risk Assessment System [Amirebrahimi et al., 2016] and Indoor Air Quality Monitoring System [Abraham and Li, 2016] will be mapped onto the RefSEISs.

4. How to evaluate the proposed reference architecture for SEISs?

   To demonstrate the effectiveness of the proposed RefSEISs, the resulted architecture of forest fire detection system from *Research Question 3* will be implemented to verify the fulfilment of specified requirements of the system. Similarly, the adherence of the RefSEISs to the identified quality attributes will be demonstrated.

A design-oriented research approach is adopted to address the aforementioned research questions as described in the next section.

1.3 Research Methodology

This thesis follows a *design-oriented research approach* for the design and development of reference architecture for SEISs and is associated with a case study for evaluation. Usually, a design science research approach consists of six-steps; problem identification and motivation, design and development, demonstration of the product, evaluation and communication through publishing the results [Peffers et al., 2008].

Four artefacts will be created following the guidelines of the design-oriented research approach. The required *set of viewpoints* for describing essential aspects of SEISs are formed from the analysis of the existing SEISs during the literature review. Some of these viewpoints are adopted from the Siemens View Model [Hofmeister et al., 2000]. The *RefSEISs* is created from the requirements based on the research work done in the literature review. The *forest fire detection system software architecture and its implementation* artefacts are generated from the specified requirements while referring to the proposed RefSEISs. Finally, these artefacts *will be evaluated* by demonstrating their adheres to the requirements.

Figure 1.1 depicts the steps undergone in this thesis while answering the research questions identified in Section 1.2. Each step will act as a basis for the following step as further elaborated below:

- **Research Background:** This involves the exploration of the literature to gain a clear understanding of software architecture and their essential concepts. Besides, the related work sections include the architectures of the existing SEISs, description of software architectures and state of the art of reference architectures. This step establishes the requirements of SEISs.

- **Design and Development:** This involves the design and development of the viewpoints and RefSEISs and hence fulfil the research question 1 and research question 2. First, the specified requirements from the research background step are used to define a set of viewpoints that is required to describe essential aspects of SEISs. Then the established set of viewpoint is applied in the construction of the RefSEISs. The description of the RefSEISs follows ISO/IEC/IEEE 42010:2011 [ISO/IEC/IEEE, 2011] which starts with an identification of stakeholders and their concerns. Followed by the derivation of requirements of the RefSEISs. Then the architecture views of the RefSEISs are created using the established set of viewpoints and their corresponding architecture models while encompassing best practices of the existing SEISs (design decisions). The approach does not focus on a specific type of technology or content. The main intention of the proposed RefSEISs is to provide a high-level description of SEISs which can assist the actual implementation of concrete SEISs.

- **Case Study:** This demonstrates the applicability of the proposed RefSEISs on the construction of the concrete architecture of SEISs. A forest fire detection systems is used as a case of study. In which the requirements of such a system are identified while referring to the defined requirements of the RefSEISs. Then to fulfil such requirements, an architecture of forest fire detection system is constructed as referred to the proposed RefSEISs. Similarly, to demonstrate the applicability of the proposed RefSEISs on the existing SEISs. Some of the existing systems such as the Forest Fire Monitoring System (IPNAS) [Stipanicev et al., 2018], Urban Air Quality Monitoring System [Ujang et al., 2013], Flood Risk Assessment System [Amirebrahimi et al., 2016] and Indoor Air Quality Monitoring System [Abraham and Li, 2016] are mapped onto the RefSEISs. This step answers the research question 3.

- **Evaluation:** The prototype implementation of the resulted architecture of the forest fire detection system from the previous phase verify the fulfilment of the specified requirements proposed RefSEISs. This phase answers research question 4.

- **Conclusion:** Finally, a conclusion is written, lessons learned and future research areas identified.

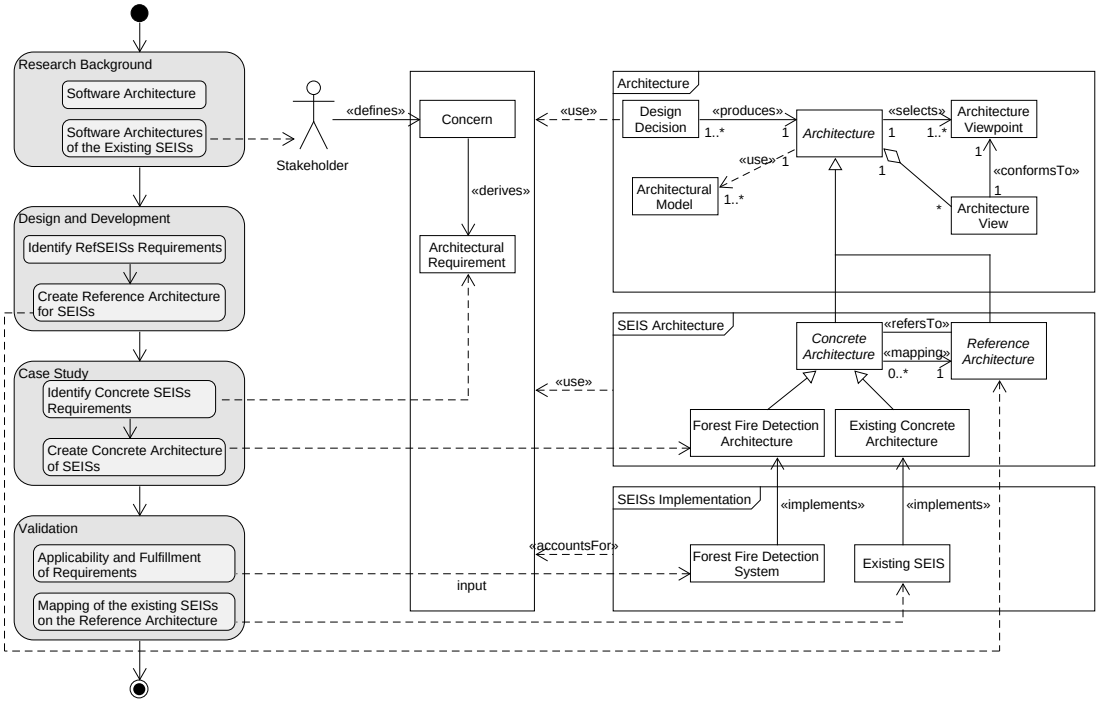


Figure 1.1: Research Methodology

1.4 Thesis Outline

This thesis is divided into five parts; Challenges, Background and Related Work, Approach, Evaluation and Conclusion. The thesis starts with Part I: Challenges which include Chapter 1. In which the motivation of the thesis, research questions, and methodology are presented.

In Part II: Background and Related Works, the fundamental concepts related to software architecture and an overview of the current state of smart environmental information systems (SEISs) are presented. This part includes Chapter 2, Chapter 3 and Chapter 4. Chapter 2 outlines the essential foundation concepts of this thesis mainly software architecture, reference architecture and documentation of software architecture. Chapter 3 presents the research synthesis in which the existing software architectures of SEISs are reviewed, examined and presented. Chapter 4 provides an outline of the related work on the existing reference architectures that will be considered reference while developing the proposed RefSEISs.

Part III: Approach is the core part of the thesis which describes the proposed RefSEISs. This part is divided into three chapters; Chapter 5, 6 and Chapter 7. Chapter 5 demonstrates the establishment of the requirements of the proposed RefSEISs which include the set of stakeholders, their concerns and architectural requirements. Chapter 6 proceeds with the identification and definition of the set of required viewpoints in describing essential aspects

of SEISs. Chapter 7 presents the views of the proposed RefSEISs which resulted from the application of the specified viewpoints into ReFSEISs.

Part IV: Evaluation demonstrates the evaluation of the proposed approach which is partitioned in three chapters; Chapter 8, Chapter 9 and Chapter 10. Chapter 8 demonstrates a case study for evaluating the proposed RefSEISs through a detailed description of the software architecture and prototype implementation of the forest fire detection system as concrete SEISs. Chapter 9 provides the conceptual validation of the proposed RefSEISs by mapping it to some of the existing architectures of SEISs. Chapter 10 presents how the specified requirements of the proposed RefSEISs are fulfilled.

Part V: Conclusion presents the conclusion of the dissertation in Chapter 11 with the thesis summary, main contributions of the thesis and discusses future work that can be accomplished to improve the proposed RefSEISs.

1.5 Summary

In this chapter, the motivation and research problem to be addressed have been summarized and provide context for the research objective. Then the essential research questions to fulfil the specified objective of the thesis have been described. Finally, the overall research methodology and outline of the thesis have been presented.

Part II

Background and Related Work

This part presents basic concepts related to software architecture, overviews of both the current state of smart environmental information systems (SEISs) and existing reference architectures within the area of environmental monitoring and controlling information systems. Such that Chapter 2 gives the reader brief introductions to software architecture, reference architecture, and documentation of software architecture since they are core concepts which have been employed in this thesis. Chapter 3 presents the research synthesis in which some of the existing software architectures of SEISs, i.e. forest fire detection systems, air pollution detection systems, flood detection systems, landslide detection systems and road traffic control systems are reviewed, examined and presented. Such review is important to identify a commonality of SEISs so as to identify and make use of the abstractions that are common to all SEISs in the development of a reference architecture for SEISs. Chapter 4 describes and analyzes some of the existing reference architectures within the area of environmental monitoring and controlling information systems so as to be used as a reference or basis in developing the reference architecture for SEISs.

Chapter 2

Foundations

The chapter introduces significant concepts that technically guide the development of the proposed approach of this research. Thus to fully comprehend the topics discussed in this thesis, basic knowledge of some related fields of software architecture has to be known beforehand. This chapter aims to give the reader brief introductions to software architecture in Section 2.1, reference architecture in Section 2.2, and documentation of software architecture in Section 2.3 since they are core concepts which have been employed in this thesis. To acquire a deeper understanding of such theories, please refer to the quoted literature for more details. Finally, a summary of the chapter is presented in Section 2.4.

2.1 Software Architecture

The recent research in the field results in abundant definitions of software architecture. This section discusses some of the essential definitions that have been applied in this thesis. According to the early book of software architecture written by Shaw and Garlan, the following definition of the software architecture was provided;

> **Definition 2.1.** *"The software architecture of a system as a collection of computational components with a description of the interaction among these components which is facilitated by connectors"* [Shaw and Garlan, 1996].

The definition 2.1 represents software architectures informally in the form of the box and line diagrams with strong emphasize on run-time mechanism via connectors. Some software engineers somehow criticize this definition, since the static structures of the system are left out [Bass et al., 2003]. Nevertheless, the latest version of the software architecture book by

Shaw and Garlan handles this misconduct by proposing the following description:

> **Definition 2.2.** *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them"* [Bass et al., 2012].

Such a definition has some interesting aspects which are highly accepted by the software engineering research community. For instance; (1) a system may consist of multiple structures which are expressed using various architectural views as will be discussed in section 2.3.1 (2) an architecture comprises externally visible properties of components, this means some component properties of the system may not be part of the architecture. Finally, the ISO/IEC Standard 42010 for Systems and Software Engineering-Architecture description provide the following definition of software architecture:

> **Definition 2.3.** The software architecture refers to the fundamental properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution [ISO/IEC/IEEE, 2011].

This definition highlights the following influential concepts; (1) elements (2) relationships between elements and (3) principles in the design and evolution of these interrelated elements. Software architecture presents the earliest software design decisions which are critical in the system development life cycle. Alternatively, software architecture is regarded as a set of architectural decisions which can be documented in an architecture description as explained in Section 2.3.1. Software architectures guarantee the prosperity of software (systems) by defining sets of concepts and principles that guide the analysis of specifications, designs, implementation, maintenance and evolution of software systems. All these definitions highlight the architecture of the single system which is regarded as concrete software architecture used for the development of specific software application. In this thesis, the term "concrete software architecture" is widely utilized and referred as a structure or structures of the system which is expressed using multiple views for its particular set of stakeholders, concerns and requirements. The description of software architecture is further provided in Section 2.3.

Software architectures are critical design artefacts for the successful development and evolution of software-intensive systems [Kruchten et al., 2006, Brown and McDermid, 2007]. However, the increased complexity of software systems and demands for shorter time to market have motivated the system engineers to utilize a higher level of abstractions mainly the use of reference architectures in handling the issues of concrete software architectures.

Reference architectures systematize the experience of developing concrete software architectures by establishing a common conceptual basis that is associated with best practices in describing concrete software architectures of a particular domain.

2.2 Reference Architecture

A reference architecture is an abstraction of 'real' architectures. Enterprise reference architectures, solution reference architectures, information systems reference architectures, etc. represent various forms of reference architectures. This section provides the essential concepts of the solution reference architectures. Starting with definitions of reference architectures in Section 2.2.1, quality criteria in Section 2.2.2, introduction of architectural styles and patterns used in construction of concrete architecture in Section 2.2.3 and finally the evaluation techniques are presented in Section 2.2.4.

2.2.1 Definitions

Reference architectures and reference models have long traditions in both software engineering and information science research [Stricker et al., 2010]. The commonly accepted definition of "reference architecture" is missing in the literature. Different research defines and classifies reference architectures and reference models in slightly different ways, but they all share a common notion of knowledge reuse and the principle of generalizations in the creation of concrete systems. Some of the significant definitions of reference architectures include;

Definition 2.4. Reference architectures are special types of software architectures which can be applied to create concrete architectures and validated based on the derivations from the other architectures [Müller et al., 2010].

Concrete software architecture deals with concrete business goals of the stakeholders of a specific system [Angelov et al., 2012]. This implies concrete architectures are designed based on the required set of functionalities and quality attributes of systems as defined by the stakeholders. The knowledge about how to design such concrete software architectures of a particular application domain is encompassed by a reference architecture [Nakagawa et al., 2011]. A reference architecture can be defined in two ways [Angelov et al., 2008]; (1) After the existence of practical experiences (Bottom-up approach). In which concrete architectures play a crucial role in the design of a reference architecture. (2) Before the existence of concrete architectures (Top-Down Approach).

Generally, reference architectures are considered as special architectures for the specific domain (class of systems) based on best practices [Angelov et al., 2008]. While [Amirat, 2012] defines reference architectures as software architectures which provide a frame of reference for a particular field of interests or domain with a common vocabulary, industry best practices and reusable designs.

Definition 2.5. According to Nakagawa et al. [Nakagawa et al., 2014], reference architectures are regarded as abstract architectures that encompass experiences and knowledge in a given application domain to facilitate and guide the development, evolution and interoperability of software system in such domain.

Similar to Definition 2.5, reference architectures improve effectiveness by providing guidance (best practices, architectural principles), architectural baseline, and blueprint as well as managing synergy and capturing and sharing architectural patterns [Müller et al., 2010]. The use of reference architectures allows systematically reuse of knowledge and elements when developing concrete software architectures in a particular domain. Although syntactically different, all these definitions imply the same essence: the reuse of architectural knowledge or artifacts of software development in a specific domain. Such reusable software assets include requirements, architectural designs, code, design patterns, test results, etc. The generic nature of reference architectures facilitates their applicability in multiple and different contexts while addressing the requirements of various stakeholders in those contexts. Thus, a software architecture is regarded as a reference if it is abstract, independent from a particular technology or any other context and hence can be used in different contexts [Martínez-Fernández et al., 2014]. Reference architectures and models are widely used in assisting decision-makers and other stakeholders to make crucial design decisions.

Despite the considerable relevance of SEISs as described in Chapter 3, the reference architecture for SEISs is missing. SEISs are still centred on ad-hoc solutions associated with concrete architectures, rather than reuse-driven and knowledge-based practices. This implies the concrete software architectures of SEISs are described from scratch make the actual development of these systems to be difficult, intensive and time-consuming. This could be avoided or reduced through the use of reference architectures in the generation of concrete architectures based on reusability of architecture knowledge rather than reinventing the wheel. The SEISs have already accumulated enough experience to be comprehended by a *reference architecture*. For this purpose, this research focuses on the description of reference architecture for SEISs as proposed in Chapter 5, Chapter 7 and Chapter 6. Such proposed reference architecture is envisioned to be a baseline for designing concrete

SEISs. Establishment of the reference architecture for SEISs is crucial for describing essential components and design decisions while fulfilling both functional and non-functional requirements.

2.2.2 Quality Criteria

According to [Bass et al., 2003, Reidt et al., 2018], reference architectures should possess the following characteristics:

- Abstraction

  A reference architecture represents a family of abstract architectures which highlights the established common components and other components namely variants which are not common [Bosch, 2000, Clements et al., 2001]. The established components are abstract in the sense that the implementation details are not included or specified. The chosen level of abstraction for the reference architectures is important [Winter, 2000] to facilitate the re-use of knowledge in the construction of concrete models [Hars, 1994]. Describing the reference architecture in more details limits the broad use of the reference architecture in the domain and therefore limits its generality while representing only a few very abstract parts reduces the benefit of reusing the reference architecture for concrete architectures [Winter, 2000].

- Independence

  Since the domain is made of various systems that incorporated multiple technologies, then, the reference architecture should not be tied to any specific standards, technologies or implementation details to represent common abstractions of the whole domain. With higher abstractions, there will be no implementation or technology restrictions [Reidt et al., 2018]. As a result, the reference architecture allows the encapsulation of the entire domain in a technology independent way through merging all the domain best practices, standards and technologies.

- Completeness

  The architecture is considered to be complete if all the relevant aspects of the system are covered based on the goal of the architecture and if the architecture does not describes parts which are beyond the specified purpose [Winter, 2000]. However, with high abstractions, the reference architectures are considered to be incomplete. Since reference architectures should cover the main parts of systems within one domain and understandably describe them, reference architectures cannot include all

individual cases. Despite the importance of these criteria in software architecture, the incompleteness of reference architectures does not imply low-quality of them. An incomplete reference architecture is expected to be extensible regarding upcoming changes and new objectives.

- Applicability

  The reference architecture should apply to a specific group of systems within a particular domain (universal applicability), not only one specific, concrete system. The applicability refers to how easy is the application of reference architectures. The developers, implementers or other stakeholders are expected to use reference architectures as a basis or guide for developing or comparing concrete systems. Applicability is significant for reference architectures because they do not describe concrete architectures completely [Gamma et al., 1994]. Applicability can be supported by an understandable reference model regarding its complexity in terms of its size and structure [Winter, 2000], which decreases the initial understanding, and by an easy adaption, for example with defined variants or configurations [Rosemann and van der Aalst, 2007]. This criterion is fundamental because architectures have to be proven as reference architecture by at least two applications, which is supported by easy applicability. This implies at least two different applications of the reference architecture are required to prove the applicability of the proposed reference architecture.

2.2.3 Architectural Styles and Patterns

The development of concrete architectures involves a series of transformation from reference architectures (abstract architectures) to more concrete form. Such conversions should be done iteratively while ensuring the properties of the reference architectures are maintained throughout [Moriconi et al., 1995]. This could be achieved through various mechanisms of transformations including imposing an architectural style, imposing an architectural pattern, using a design pattern, converting non-functional requirements to functionality and by distributing non-functional requirements on transformations [Bosch and Molin, 1999].

Architectural styles and patterns refer to the way of organizing the elements of the system to support the construction of a complete system which achieves the specified requirements of stakeholders [Dobrica and Niemelä, 2002]. There are several architectural styles and patterns available in the software industry, some of these are categorized as follows; shared memory systems (e.g. pipes and filters style), distributed systems (e.g. client-server style), messaging systems (e.g. publish-subscribe style), adaptable systems (e.g. microkernel style), structural systems (e.g layered style) and modern systems (e.g. multi-tenancy style) [Dobrica

and Niemelä, 2002]. Architectural styles and patterns are very similar and can even possess the same names, though architectural patterns are more problem-oriented than architectural styles by providing a solution on a particular recurring design problem that affects only a limited number of classes in architecture, while architectural style describes overall structural frameworks of a system [Buschmann et al., 1996].

This work utilizes the concept of strategies in the development of reference architectures for SEISs which are induced systematically in the constructions of Concrete architectures of SEISs. Such strategies encompass architectural styles and patterns including client-server, Model View Controller (MVC), and layering as described in Chapter 7.

2.2.4 Evaluation of software Architecture

The successful development of reference architectures must be associated with the demonstration of the reusability of the proposed reference architecture on that domain [Harrington, 2012]. Various methods can be employed to assess the effectiveness of such reference architecture which is classified into; (1) Numerical based methods [Vanek et al., 2008] and (2) Scenario-based methods [Muskens et al., 2002].

Numerical based methods utilize metrics to assess the effectiveness of software architectures, for instance, standard coupling metrics, i.e. fan-in or fan-out that relate with the conventional views of a system architecture description [Muskens et al., 2004]. This method is highly applicable in concrete architectures rather than reference architecture since it is dependent on the architecture description and development languages while the reference architectures are supposed to be independent. Stochastic methods determine the architecture robustness or change and error based on change propagation probability on software architecture and error propagation probability introduced in [Abdelmoez et al., 2005] and [Popic et al., 2005] respectively. In the context of higher-level abstractions, e.g. reference architecture, these stochastic methods tend to increase uncertainty [Harrington, 2012].

Scenario-based methods include Software Architecture Analysis Method (SAAM) and Architecture Trade-off Analysis Method (ATAM) as the most common architecture assessment methods in the software industry [Muskens et al., 2002]. These methods require sets of scenarios against the tested system architecture. These scenarios should be derived from the concerns of key stakeholders which are analysed concerning both software performance (implementation specific) and experienced based analysis [Del Rosso, 2006]. Most of the existing reference architectures have been evaluated using scenario-based methods, e.g. automotive [Sanz and Zalewski, 2003], manufacturing planning [Howard et al., 1996] and hybrid vehicle control systems reference architectures [Larsen et al., 2002].

Therefore a scenario-based method is the most appropriate method for the assessment of the proposed reference architecture for SEISs. Such evaluation demonstrates the reusability of the proposed reference architecture over the scope of SEISs and is conducted in two phases; the First phase involves the building of an instance architectural prototype of the reference architecture based on a set of scenarios that can be used to prove the predefined objective of the thesis are defined in Chapter 8. In the second phase, the proposed reference architecture is used to map the existing SEISs in Chapter 9.

2.3 Software Architecture Documentation

Software architecture documentation is very crucial in the life-cycle of the system to meet the specified objectives, as it provides high-level perspectives of a system to a variety of stakeholders. According to the ISO/IEC/IEEE 42010 Standard, an architecture description is: *a work product used to express an architecture* [ISO/IEC/IEEE, 2011]. Also [Rozanski and Woods, 2005], define an architectural description as *set of products that documents an architecture in a way its stakeholders can understand and demonstrate that the architecture has met their concerns* . This allows the clear understanding of the architecture which in turn facilitates the selection of right and effective decisions on both technical and business levels during system definition as well as in later phases of systems development life-cycle, such as system evolution. Such an architectural description describes or specifies all the architectural elements, diagrams, rationale, model decisions and others related to software architecture [Clements et al., 2005].

The software architectural description provides a comprehensive description of software structure and behavior of the software and tends to be both prescriptive (it prescribes some rules and limitations to be considered by stakeholder while taking crucial architectural decisions), descriptive and abstract to be understood by new employee of the company but is so complete with enough information for system analysis. Such description serves as [Clements et al., 2005]; (1) means of introducing and educating users, i.e. team, new architects and others to the system. (2) Means of communication between various stakeholders (3) the bases for system analysis for instance for analyzing the quality attributes of the system. In this thesis, the description of the reference architecture for SEISs follows the IEEE 42010:2011 Recommended Practice for Architectural Description Standard of Software-Intensive Systems. This section introduces important concepts that technically guide the description of a software architecture using ISO/IEC/IEEE 42010:2011 standard in Section 2.3.1. Section 2.3.2 describes some of the essential viewpoints that can be considered in developing a

software architecture based on the existing software architectural viewpoints models. Then architecture description languages are presented in Section 2.3.3.

2.3.1 ISO/IEC/IEEE 42010:2011 Architectural Description

The ISO/IEC/IEEE 42010 is the international standard of Systems and Software Engineering-Architecture description [ISO/IEC/IEEE, 2011]. This standard specifies the conventions, principles and practices required for the description of architectures in a particular domain. It defines how the architectural description of software-intensive systems should be expressed and organized, the essential constructs of ISO 42010 conceptual framework and type of information found in any ISO 42010-compliant architecture description [ISO/IEC/IEEE, 2011]. This standard was developed based on the consensus of current practices with emphasis on the use of multiple views, and reusable models within views of the architecture.

The architectural description template is depicted in Figure 2.1 in terms of Unified Modeling Language (UML) class diagrams which provide an overview of different concepts and their relationship. Every *System-of-interest* (a system that will be developed based on the described architecture) has one *Architecture*. An *Architecture* is expressed (described) by an *Architecture Description*. *Architectural description* consists of a set of related *Architectural View*s which addresses one or more *concerns* of the *stakeholders* [ISO/IEC/IEEE, 2011]. An *Architectural Description* supports the *correspondence*s between views in terms of architectural elements which defined by *Correspondence Rule*s and documents the architectural decisions with their *Architecture Rationale*. An *Architecture Description* provides one or more model kinds in order to frame some concerns of its stakeholders. Some of the essential concepts are further described;

Stakeholders

Stakeholders are defined as people or organization or things (other systems) that have requirements or expectations about a system [Bennett, 1997]. Similarly, IEEE 1471-2000 defines;

> **Definition 2.6.** A stakeholder is defined as *an individual, team or organization (or classes thereof) with interests in or concerns relative to a system* [Group, 2000].

Stakeholders are key players of the successful SEISs and possess a strong influence in the evolution of such systems. These stakeholders include both technical and non-technical people who drive the conception and development of SEISs. Each stakeholder has interests and concerns that need to be addressed. Multiple stakeholders are involved in SEISs. To

mention a few; stakeholders of SEISs include residents, government agencies, researchers, designers, developers, etc.. These stakeholders possess different interests (concerns) which are mapped into different requirements. The identified stakeholder needs or interests could be addressed by an architect while designing architecture. An architect is supposed to find out the requirements of these stakeholders, analyse them and then construct an architecture that fulfils the specified requirements (or demands of the stakeholders).

Concerns

The construction of software architecture for software-based systems must consider the requirements (concerns) that fulfil the needs of various stakeholders for the system [Baida, 2001]. Such concerns in the context of SEISs include the functionalities of SEIS fulfilled, feasibility of constructing the SEIS, etc.

Definition 2.7. Concerns refer to an area of interests of a system which could be a requirement, objective, an interest, aspiration or intention of a stakeholder has for a system [Rozanski and Woods, 2005].

Viewpoints

Software architectural viewpoints consist of a set of guidelines and principles that describe the whole system uniformly [Babar and Gorton, 2011].

Definition 2.8. A viewpoint is a subdivision of the specification of a complete system, established to integrate those pieces of information based on specified concerns [Vallecillo and Informática, 2001].

Viewpoints composed of a collection of patterns, templates and conventions for the construction of software architectural views. Such viewpoints define aims, stakeholders and their concerns [Rozanski and Woods, 2011].

Definition 2.9. The IEEE 42010 defines viewpoints as *established conventions for constructing, interpreting and analyzing the views to address concerns framed by that viewpoint* [ISO/IEC/IEEE, 2011].

Based on the Definition 2.9, the viewpoints frame one or more concerns. Even though the IEEE 42010 standard does not specify a particular viewpoint to be considered explicitly, a set of viewpoints should be included while describing an architecture. As each view is supposed to have a viewpoint explaining the conventions that have been employed in that

view. Definition 2.9 will be used throughout this thesis; some of the viewpoints used in the reference architecture for SEISs include conceptual, module, execution, etc.

Views

The system architecture is presented through a set of architecture views [ISO/IEC/IEEE, 2011]. Each view is made up of architectural system models and described based on the convention specified by its governing viewpoint.

> **Definition 2.10.** A view is a particular representation of the entire system using a set of models from the specific perspective of a viewpoint [ISO/IEC/IEEE, 2011].

Architecture Models

An architecture view consists of one or more architecture models. Architecture models employ appropriate modelling conventions in addressing the concerns of stakeholders. Such conventions are specified using model kinds governing those models which are optionally organized into viewpoints.

Model Kinds

Model kind defines conventions for a given type of modelling by describing rules, operations, metamodels and templates that used to represent information and relationships in an architecture description. Such model kinds include class diagrams [OMG, 2011], data flow diagrams [Gane and Sarson, 1977], statecharts [Harel, 1987], etc.

In general, an architecture description is considered as the main artefact in expressing software architectures. Thus, the IEEE 42010 standard is adapted to enable the construction of an adequate architecture description that can be effectively used during the life-cycle of software systems. This section presented essential elements of the architecture description of software systems based on the IEEE 42010 standard. However, the IEEE 42010 standard does not specify a particular set of viewpoints to be used. Hence Section 2.3.2 describes some of the essential viewpoints that can be considered in developing a software architecture based on the existing software architectural viewpoints models.

2.3.2 Software Architectural Viewpoints Models

As described in section 2.3.1, a common approach to defining software architecture is by using viewpoints. Based on the Definition 2.9, architectural viewpoints represent perspectives

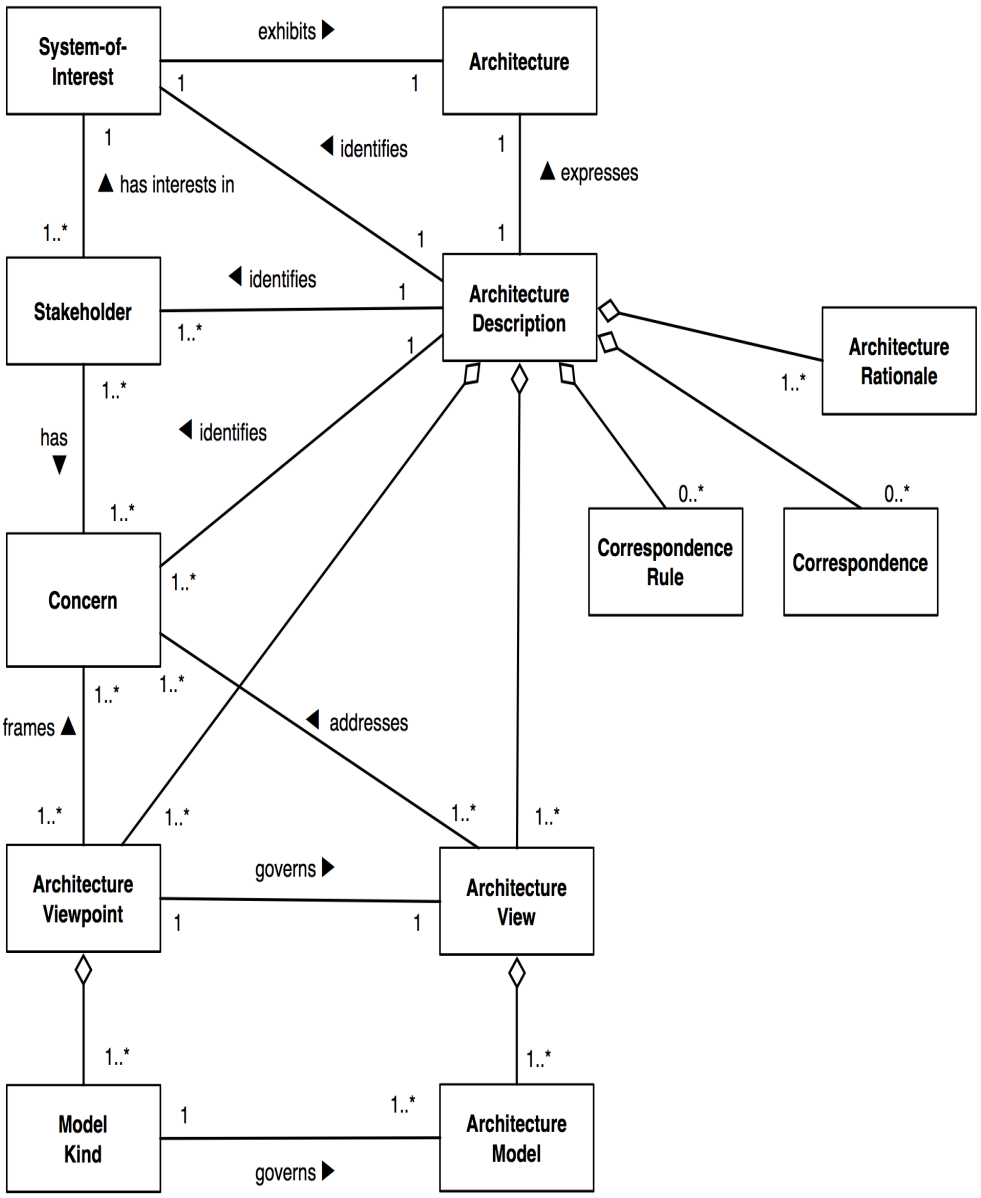


Figure 2.1: Conceptual model of an architecture description Figure copied from [ISO/IEC/IEEE, 2011]

of a system based on a related set of concerns. Multiple software architecture views are essential to deal with a diverse set of stakeholders, i.e. users, developers, maintainers, architects, testers, etc. that need to both understand and use the software architecture from their viewpoints [Babar and Gorton, 2010].

Many architectural viewpoints models are sometimes referred to as architectural frameworks have been proposed based on practical experience to capture various aspects of

software architecture. Such models emphasize describing software architecture using multiple views based on the principle of separation of concerns to manage systems complexity. These models provide *essential viewpoints taxonomies* that should be considered during the description of software architecture while covering various software architecture domains associated with both organization, business and technological environments demands. Some of the existing architectural viewpoints models are associated with the support design rationale. In this section, a number of *useful viewpoints models* are reviewed and described to assist the determination of an optimum set of viewpoints required for SEISs. This section provides a short review of some predominant models for software architecture development; ISO Reference Model for Open-Distributed Processing (RM-ODP) [Vallecillo and Informática, 2001], Siemens four View Model [Hofmeister et al., 2000], Kruchten's "4+1" View Model [Kruchten et al., 2006], and viewpoints Model for Information Systems Architecture [Rozanski and Woods, 2011].

ISO Reference Model of Open Distributed Processing (RM-ODP)

The ISO Reference Model of Open Distributed Processing (RM-ODP) provides a coordination framework for the standardization of *open distributed processing (ODP)* applications [Vallecillo and Informática, 2001]. The model intends to integrate a wide range of current and future ODP standards for distributed systems and maintain consistency among them. RM-ODP provides all stakeholders (from managers to users, from designers to developers) with common vocabulary and semantics as well as emphasizes the use of formal notations on architecture description. RM-ODP has five generic and complementary viewpoints which address stakeholder concerns in a particular set of aspects of the system as described below. Each viewpoint is associated with viewpoint language.

- *Enterprise viewpoint* describes the business requirements with the main focus on the scope, purpose and policies of the system and how to fulfil such requirements.
- *Information viewpoint* concerns with the semantics and processing of the information. This viewpoint describes how the system manages the data, structure and content type of the supporting data.
- *Computation viewpoint* describes system functionality and its functional decomposition on the system into objects which interact at interfaces.
- *Engineering viewpoint* describes the distribution of processes to provide the functionality of the system and manage the information. This viewpoint supports the distribution interactions between objects in the system.

- *Technology viewpoint* describes the technology of the system used in processing, functionality and presentation of information.

RM-ODP provides an architecture that assures mutual consistency among viewpoints and use the standard object model to bind all the viewpoints together [Vallecillo and Informática, 2001]. Although the RM-ODP is intended for distributed software domain, its coverage shows that it is also applicable to other domains. Adhering to RM-ODP viewpoints is not easy as it seems, such that readings of the standard may lead to different interpretations. Since the model does not prescribe any method in guiding the development of software architecture [Vallecillo and Informática, 2001]. Additionally, some concepts in RM-ODP are not found in Unified Modeling Language (UML), e.g. an object with multiple interfaces and multiple types of object in RM-ODP, while the interface can not be instantiated directly and an object has a single type in UML [Putman, 2001]. As presented in Section 2.3.3, UML is used in describing both the reference architecture for SEISs and concrete architectures of SEISs. Therefore RM-ODP viewpoints are not suitable in the description of SEISs.

Siemens Four View Model

The Siemens four view model is developed at Siemens Corporate Research which resulted from the industrial practices of software architecture [Hofmeister et al., 2000]. The model has a strong emphasize on the re-use and reconfiguration of software and hence led to the reduction of implementation complexity of the system. This model is associated with a design approach (model) for the software architect and consists of four views, i.e. conceptual, module, execution and code as described below. Each view addresses different stakeholder concerns.

- *Conceptual view* concerns with the issues relating to the application domain mainly how the system fulfils its requirements. This view describes how the functionality of the system is partitioned to the conceptual components.

- *Module view* describes how the conceptual components are mapped to the actual software entities, i.e. subsystems and modules. In this view, the conceptual view of a system is realized with today's software platforms and technologies.

- *Execution view* describes the runtime interactions of a software application. Such a view is also deal with how the subsystems and modules are allocated to the hardware platforms.

- *Code view* concerns with the mapping of runtime entities into the deployment entities, i.e. executables, libraries etc.

The view mappings are explicitly defined. Starting from the conceptual view to the code view, the design flow as the information passed between views. Such that the conceptual structures are implemented by module structures and assigned-to execution structures. Module structures can be located in or implemented by code structures. Execution structures can be configured by code structures. Each view is an input to another view and hence helps the software architect to analyze the trade-offs as they receive feedback results from the testing of views for the conformance with non-functional requirements of the system. The software architecture development using this model posses feedback loops with both source code development and hardware architecture. This model provides a design approach model that guides software architect in constructing software architectures hence reduce the occurrence of misinterpretations. Additionally, the Siemens four view model provides essential viewpoints required by the SEISs, however, these viewpoints are not enough to cover all the essential aspects of SEISs i.e. topology and data concerns. Therefore the Siemens four view model is adopted with an addition of two viewpoints; topology and data viewpoints.

Kruchten's "4+1" View Model

Kruchten proposed a 4+1 viewpoints model associated with an iterative process for architecture design which starts with the description of concrete scenario [Kruchten, 1995]. Such viewpoints model is used for software architectural analysis and modeling via UML notations. Also, it consists of multiple, concurrent views, that allow the different stakeholder concerns to be addressed separately. Such a viewpoints model includes five interrelated views;

- *Scenario view* presents use cases to discover and verify the architectural design. This view depicts narrative use cases which show how the other four views can work together and hence provide an overall picture of the whole system to all stakeholders.
- *Logical view* presents how the functional requirements of the system are fulfilled through the static structural layout of the software system from the software developer perspective.
- *Process view* partitions the software into independent software tasks to represent running processes and their inter-process communication while fulfilling non-functional requirements, i.e. performance, availability, system integrity, fault tolerance, concurrency, etc.

- *Development view* concerns with the organization of software modules. The software is divided into small chunks belongs into subsystems organized in a hierarchy of various layers.

- *Physical view* describes hardware-software configurations at a platform and deals with non-functional requirements, i.e. availability, scalability, performance, etc. This view involves the mapping software units into hardware nodes based on administrator and developers perspectives.

These multiple views are developed concurrently and not independent. Hence it would be challenging to model the views of SEISs separately without following the design flow between views such as starting with scenarios view proceeding to logical view and then other views. The model relies on a feedback loop in the development process. Although this viewpoint model is popular and has great relevance, a design model for the construction of software architectures is missing and does not explicitly address data or operational concerns [Omrani and Ebrahimi, 2013]. Hence this model restricts the expressiveness of architecture documentation of SEISs.

SEI Viewpoints (Views and Beyond)

Views and Beyond (V & B) is a collection of techniques that should be helpful to the people who depend on it to accomplish their work [Omrani and Ebrahimi, 2013]. Such techniques are classified into a few categories; (1) Determining the stakeholders' needs. (2) Satisfying those needs by providing the information via recording design decisions based on a variety of views plus the beyond-view information. (3) Checking the resulted documentation to verify the fulfilment of the needs. (4) Packaging the information in a useful and understandable form to its stakeholders. Category 3 and 4 concern with the document-centric activities while category 1 and 2 refer to the activities that focus on the architecture design. The SEI model utilizes styles to determine the views of the system that can be generated. V & B consists of three views [Clements et al., 2005];

- *Module View* describes the principle implementation units or modules of a system along with their relations among these units. The architectural styles included in the module view are; uses, generalization, decomposition and layered.

- *Component and Connector View* demonstrates elements that have some runtime presence. Such view includes pipe-and-filter, shared-data, publish-subscribe, client-server, peer-to-peer and communicating process architectural styles.

- *Allocation View* describes a mapping between non=software elements in the software's environment and software elements from either a module or component and connector views. Such view includes deployment, implementation and work assignment styles.

The SEI viewpoints model specifies useful views that can be used to describe many systems. However, this set of views is not enough in describing SEISs. Also, Siemens four view model provides more than the set of views generated by SEI viewpoints model. Thus SEI is not suitable in describing SEISs.

Viewpoints Model for Information Systems Architecture

Nick Rozanski and Eoin Woods proposed a *viewpoints catalogue* for describing software architectures of information systems such as financial systems, etc. [Rozanski and Woods, 2011]. The recommended viewpoints are as described below;

- *Functional viewpoint* describes how the functional requirements of the system are fulfilled in terms of functional elements, their interfaces and interactions. This viewpoint influences the shape of other system structures for instance information structure, deployment structure and other structures. Besides, this viewpoint posses a significant impact on the system quality, i.e. modifiability, security and performance.

- *Information viewpoint* describes the way the information is being stored, managed, distributed and manipulated by architecture. This viewpoint provides a high-level view of static data structure and information flow.

- *Concurrency viewpoint* maps functional elements to concurrency units which refer to parts of the system which can execute concurrently and how the overall process is coordinated and controlled. This viewpoint involves the creation of models associated with process and thread structures that will be utilized by the system and interprocess communication mechanisms for coordinating their operations.

- *Development viewpoint* concerns with the actual software development process. This viewpoint guides the construction of the development view that will facilitate the communication between stakeholders in building, testing, maintaining and enhancing the system.

- *Deployment viewpoint* describes the environment into which the system will be deployed as well as capturing the dependencies of the system during the runtime environment. The resulted view captures both hardware environment required by the system

including processing nodes, network infrastructures, the technical background needed for each identified element and also a mapping of the software elements to the runtime environment.

- *Operational viewpoint* concerns with how the system will be operated, administered and supported during the operational time in its production environment.

The Rozanski and Woods viewpoints catalogue is similar to Kruchten's with an addition of two other viewpoints, i.e., Information and Operational viewpoint. Hence managed to address the data and operational concerns. However, the architectural description process is missing; it becomes unclear what the software architect is required to do to define suitable architectural description [Kheir et al., 2013].

Lesson Learned The use of viewpoints facilitates the management of architecture views. Since they present general aspects in the construction of architecture views which are easier to reuse and do not need to be redefined for every system in a particular domain. Architectural viewpoint models cover relevant architecture views that enable software architect to select a set of viewpoints rather than creating the viewpoints from scratch. The section served as the roadmap towards a selection of the right and appropriate viewpoint model based on stakeholder and concerns of SEISs. The set of viewpoints provided by Siemens view model is best choice viewpoint model for the description of SEISs. Since most of the existing SEISs architecture views are described using the views that belong to this model as described in Chapter 3 and importantly this viewpoint model is associated with an architectural description process model which will assist an architect during the designing phase. However, these viewpoints are not enough to cover all the aspects of SEISs as defined in Chapter 3. Therefore two more viewpoints topology and data viewpoints are added. As a result, the optimum set of viewpoints for SEISs with the most significant coverage includes a conceptual, module, execution, code, data and topology viewpoints as presented in Chapter 6.

2.3.3 Architectural Description Languages (ADLs)

Many researchers believed that software architecture must be provided with its own body of specification languages and analysis techniques to obtain the benefits of an explicit architectural focus [Medvidovic et al., 2002, Wolf, 1997, Allen and Garlan, 1996]. Such languages termed as *Architecture Description Languages (ADL)* which is defined as *a language (graphical, textual, or both) for describing a software system in terms of its architectural elements and the relationship among them* [Varadharajulu and Sridharan, 2009].

This is required to define and analyze the properties of the system. A good ADL must provide abstractions which are adequate for modeling a large system. ADLs are capable of formalizing, describing, specifying, modeling and reasoning on software architectures [Varadharajulu and Sridharan, 2009]. An infinite number of ADLs have been proposed [Medvidovic, 1996, Moriconi et al., 1995, Magee and Kramer, 1996, Luckham and Vera, 1995, Garlan et al., 1994, Allen and Garlan, 1997]. Each ADL has a particular approach towards the specification and evolution of an architecture [Medvidovic et al., 2002]. In this thesis, a Unified Modeling Language (UML) is used as an emerging standard software design language whose ADL supports architectural building blocks. Since UML provides diagrammatic notations which are easily understandable by everyone dealing with software development.

Unified Modeling Language (UML)

A Unified Modeling Language (UML) is a graphical language defined by Object Management Group (OMG) and is considered as the de facto standard for general-purpose systems which can be used to specify and document the artefacts of the system throughout the system development life-cycle [Varadharajulu and Sridharan, 2009]. UML is a high-level modeling language with well defined semi-formal syntax and semantics, useful and extensible set of predefined constructs, the potential for robust tool and is based on the experience with mainstream development methods [Medvidovic et al., 2002, Varadharajulu and Sridharan, 2009].

UML appears to be well suited for describing software architectures of SEISs because of its expressive architectural power in modeling software architectures. Additionally, UML allows different stakeholders or contributors associated with diverse perspectives, i.e. analysts, developers, architects and end-users to communicate on the common ground in regards to specify, visualize and document a system [Kaur and Singh, 2009]. Also, UML supports multiple architecture views via a collection of diagrams for depicting both structural information (class, object and package diagrams) and behaviour information (state machine, use case and interaction diagrams) of a system. This thesis focused on structural information, in which the viewpoints are described using class diagrams, while the views are described using either class, object or package diagrams.

2.4 Summary

This chapter presents related work on software architecture and its description. Such that the essential foundation concepts of this thesis have been examined mainly, software architecture, reference architecture, and documentation of software architecture. A consistent set of terminologies for both software architecture and reference architecture is essential for preventing any confusion. Various definitions of both software architectures and reference architectures have been discussed. In general, software architecture is a structure (/structures) of the systems made up of software elements and their external properties. Well-defined software architectures are crucial for successful system development and evolution while a reference architecture represents special kind of software architecture which encompasses software elements and architectural knowledge that can be reused in the construction of concrete architectures of systems that belongs to a particular domain. The IEEE 42010 standard [ISO/IEC/IEEE, 2011] as the main standard for software architecture documentation has been described. Various architectural viewpoints models have been discussed. These models provide sets of viewpoints to be considered in describing software architecture rather than starting from scratch. It is followed by a brief description of architectural style and patterns. Finally, the architectural description language is discussed. The next chapter continues with the discussion on the foundations by focusing on the actual review of the existing SEISs and reference architectures.

Chapter 3

Related Work for SEIS

This chapter presents the research synthesis in which some of the existing software architectures of SEISs are reviewed, examined and presented. It is worth mentioning that the topic regarding architectures of SEISs is a broad discipline of research, hence the comprehensive survey of the available researches is outside the scope of this thesis. The focus is on the software architectures of SEISs, namely software systems that collect, analyze, process, control and visualize environmental conditions (parameters) for effective monitoring and controlling environmental phenomena. This review describes a set of SEISs with clear descriptions of software architectures as representatives of SEISs. Such review is important to identify a commonality of SEISs so as to identify and make use of the abstractions that are common to all SEISs in the development of a reference architecture for SEISs.

3.1 Overview

The ultimate objective of this review is to identify and describe existing SEISs while paying special attention to their developmental needs and structures. There are many SEIS architectures based on applications that have been proposed by individual researchers, academicians, and companies from various countries [Milke and McAvoy, 1995]. However, some SEISs which serve as representatives of SEISs are briefly discussed; Forest Fire Detection Systems in Section 3.2, Flood Detection Systems in Section 3.3, Air Pollution Detection Systems in Section 3.4, Landslides Detection Systems in Section 3.5, and Road Traffic Control Systems in Section 3.6. Such description includes various systems for each group of SEISs, and the fundamental architectural description concepts, i.e. stakeholders and their concerns, systems' functionalities, architectures, and quality attributes as deduced from IEEE 42010 standard. These constructs facilitate the description of essential architectural elements, commonality and challenges of SEISs that should be considered in developing the reference architecture

for SEISs. The "underline" emphasizes architecture views and styles. The chapter concludes with an outline of ongoing research directions for SEISs.

3.2 Forest Fire Detection Systems

Forests are essential resources for social development and survival of living organisms. However, because of uncontrolled conditions, the life of the forest is threatened by fires. This led to the growth of gas emissions in the planet reaching about 20 per cent of Carbon-dioxide (CO2) emissions in the atmosphere [Zhang et al., 2008]. The occurrence of forest fires is increasing considerably due to human activities, climate changes and other factors. These forest fires are among the most severe disasters to the infrastructure, environment, economy and most importantly, the lives of humans and other living organisms. Many countries in the world are subjected to forest fires, about 13 million hectares of forest are destroyed each year [Bouabdellah et al., 2013]. To mention a few, some of the highly vulnerable countries include India [Sharma, 2016], Europe (the forest fires burn on average of about 500000 ha every year) [Karafilovski et al., 2014] and other countries.

Early detection of forest fires can drastically reduce or prevent the consequences of fire. Hence monitoring, controlling and prevention of forest fires became global concerns in environmental conservation. Traditionally, forest fires were detected using human-operated observation towers such as towers at fire high points [Hefeeda and Bagheri, 2009] and Osborne fire finder [Bahrepour et al., 2008]. However, such techniques were unreliable, inefficient and partially effective since they employ both camera surveillance systems and satellite imaging technologies in monitoring forests which are costly and highly affected by certain weather conditions and (/or) physical obstacles [Kletnikov et al., 2017]. The revolution of sensing-actuating techniques paves the way for the introduction of conventional techniques based on wireless sensor-actuator networks and others [Hartung et al., 2006, Son and Kim, 2006, Bernardo et al., 2007]. The descriptions of the existing software architectures of forest fire detection systems are based on the following categories: stakeholders and their concerns in Section 3.2.1, core functional features of forest fire detection systems are described in Section 3.2.2, their architectures in Section 3.2.3, and supported quality attributes in Section 3.2.4.

3.2.1 Stakeholders and their concerns

A comparative analysis of the forest fire situation in central-eastern Europe presented in [Albers, 2012] defines a set of stakeholders who are crucial for the success of the forest

fire detection systems. Such stakeholders include general public (citizens), operators, fire agencies, non-governmental organizations, forest management groups, landowners, wild-land fire researchers, government and national authorities [Alkhatib, 2014]. The firefighters are in charge of controlling the situation when fires occur to lower the consequences [Hartung et al., 2006, Bernardo et al., 2007]. Hence they should be equipped with a suitable tool or system to view or forecast the occurrence of fire. The major concerns of these stakeholders include system feasibility, decomposition, integration, distribution and management of the nodes, functionalities, and supported quality attributes.

3.2.2 Functional Features

The forest fire detection system proposed in [Canada, 2018] provides public (citizens) with the information regarding the daily danger and conditions for the occurrence of fire in Canada. Such a system uses the Canadian Forest Fire Weather Index (FWI) for modeling and displaying the prediction of fire occurrence in forests via daily monitoring of temperature, wind speed, rainfall and relative humidity.

The FireWxNet [Hartung et al., 2006] provides the firefighters with the ability to measure and view fire and weather conditions over a wide range of locations. FireWxNet detects and locates the actual focus of fire and use images to verify the resulted measurements and then predict fire behaviour, heightening safety considerations. Also, an alarm application based on Telos B motes proposed in [Bernardo et al., 2007]. Such a system facilitates firefighting operations. A combination of both temperature, light and humidity sensors are utilized in difficult access environments. While a fire rescue application named FireNet is proposed in [Sha et al., 2006]. FireNet utilizes wireless sensor networks to fulfil the following requirements; real-time monitoring, resource allocation, accountability of firefighters, and web-enabled services.

The Forest-fires Surveillance System (FFSS) has been deployed to gather measurements of temperature, humidity and illumination from the environment [Son and Kim, 2006]. FFSS consists of a middleware program that determines the forest-fire risk-level by a formula from the Forestry Office. In which, if a fire is detected, an alarm is automatically activated to facilitate an early extinguishing of the fire. Similar to [Son and Kim, 2006], the forest fire detection based on the Fire Weather Index was proposed in North America [Hefeeda and Bagheri, 2009]. This system determines the risk of propagation of fire based on a collected set of index sensors parameters.

An advanced system for monitoring forest area and early detection of forest fires using wireless sensor networks is described in [Kletnikov et al., 2017], is associated with the

following features; capturing and storing information from various sensors based on the line of sight, the collected data is then transmitted through the local wireless network using Zigbee, processing the received information based on models of prediction and approximation algorithms, presenting centralized alarms, captured data and predictions via a base station. Also, a video-based early fire warning system presented in [Dimitropoulos et al., 2012] provides an automatic early warning of fire and extreme weather conditions by remotely monitoring areas of archaeological and cultural. Finally, FireWatch is produced by German Aerospace Institute (DLR) to provide an automatic smoke detection system capable of identifying smoke within a range of 10 – 40km [Alasia, 2013].

To summarize, the main features of forest fire detection systems include monitoring of forests parameters using a collection of sensors, the sensed data is sent to the information system for storage and further processing to predict the occurrence and risk of propagation of forest fires, issuance of alarms in case of the (or high probability) of fire, viewing both collected and processed information.

3.2.3 Architectures

In general, forest fire detection systems are made up of the monitoring nodes, base stations, communication systems, internet access and the information control centre. Sensor nodes are randomly deployed in a forest and construct wireless sensor networks to monitor the forest parameters such as humidity, temperature, smoke, gases and others. Such data is periodically collected and sent to the sink (base station) then to the information control centre for further processing, i.e., alerting citizens or residents and dispatching firefighting crews. Different views have been used to describe various forest fire detection systems in satisfying the stakeholders' concerns while fulfilling functional features as mentioned in Section 3.2.1 and Section 3.2.2.

The decomposition and components integration of forest fire detection systems were described using module views with layered architecture styles. For instance, Jadhav et al in [Jadhav and Deshmukh, 2012], used a module view which was associated with a layered architecture style to describe a sensor node. The sensor node is divided into two layers; Operating System (OS) kernel and Application Programming Interface (API). The OS kernel layer provides a low-level node driver of all hardware devices. The kernel layer consists of task debugging module for controlling the control flow throughout the operating system, power management module for supporting radio frequency transceiver, processor, sensors and other parts of the state control of energy consumption and energy management module for waking up the node at the right time to maximize the energy utilization. The API

layer provides sensor acquisition via sensor modular and radio frequency communication modules.

Also, European Forest Fire Information System (EFFIS) that has been implemented by the Joint Research Centre (JRC) and the General Environmental Directory (ENV) was described using a module view with a layered architecture style [Commission, 1998]. In such a view, the system consists of web-based modules, the data processing part and spatial database for collecting and presenting the information of forest fires on the pan-European scale. Additionally, the system consists of web-based mapping tools of which the EFFIS layers are published and allow the user to access the system by searching and analyzing the information in a web browser. EFFIS layers include wide range of formats; INSPIRE and Open Geospatial Consortium (OGC) standards, such as Web Map Service (WMS), that generates maps in an image format online, and Web Coverage Service (WCS) that offers raster or grid data, and Web Feature Service (WFS), which generates vector data using Geographic Mark-Up Language (GML).

A fire management system installed in South Africa named FireHawk used a module view with a layered architecture style to describe the system's decomposition [Alasia, 2013]. The architecture consists of three layers: Imaging layer which represents installed cameras in suitable places. The Communication layer used to set up the connection through the wireless link. The Machine vision layer is the core layer of FireHawk which employ ForestWatch software and Geographical Information System (GIS) to estimate the actual location and shortest path to the fire. Also, Hartung et al. presented FireWxNet which has been designed as a multi-tiered architecture [Hartung et al., 2006]. This system used web cameras to provide images for the fire and other sensors with small GPS forming wireless sensor networks to provide weather information. FireWxNet is equipped with directional antennas on the top of mountains and ends with multi-hope sensor network to monitor the specified environmental parameters.

Similarly, the forest fire monitoring system called IPNAS introduced and supported by the ministry of science, education and sport of Republic Croatia. The IPNAS was described using module view [Stipanicev et al., 2018]. IPNAS is automated surveillance associated with automatic detection of forest fires. The IPNAS is based on the fields units and a central processing unit. The field units consist of video cameras and min meteorological station which are connected through wireless LAN to a central processing unit for further processing. The data collected by the field units is stored in the databases (GIS database, SQL database and Data warehouse). With the use of data interface, IPNAS offers various services including user-friendly camera control, automatic fire detection and archival retrieval of both data and video. The IPNAS is web-based with a user interface displayed in a standard web browser. A

user can access the system through tunneled SSL (Secure Socket Layer) and VPN (Virtual Private Network).

The distribution and management of nodes in the forest were described using a topology view with a network cluster tree-topology view in which the nodes in the network are organized based on three categories; ordinary bottom nodes, cluster heads and network coordinators [Zhang et al., 2008]. Data is collected by sensor nodes (ordinary bottom nodes) deployed in the forest then transmitted to its cluster head. A cluster head performs both fusion and transmission of data packets to the centralized servers for further processing if there is a direct connection. Otherwise, the information is sent to the network coordinators. The Network coordinators are responsible with the network management functions mainly access control, configurations and equipment registrations.

Also, a topology view with network clustering was used by this system to describe the network topology of sensor nodes [Jadhav and Deshmukh, 2012]. In which, nodes are classified into common and cluster-head nodes. All sensor nodes are allocated to some clusters. The common nodes collect the data and then transmit it to the cluster-head node which stores the data to the database. At the same time, the expert decision support system uses such data for further processing and analysis.

Finally, the description of the main functionalities and how all the components of forest fire detection system fit together was presented using a conceptual view that utilizes client/server architecture style. Such a view described a forest fire detection system proposed in [Zhang et al., 2008].

3.2.4 Quality Attributes

The forest fire detection system described in [Aslan et al., 2012] supports; *energy efficiency*, *accuracy* and *adaptability*. *Energy Efficiency*, since batteries power the sensor nodes deployed in the forest, then the system was designed with low consumption of energy as replacement of batteries in the forest is too costly, and impractical or infeasible. *Accuracy*, the early detection of forest fires with an accurate estimation of the fire location is crucial for the control of fire. This system estimates the accurate fire location to send the firefighters to the correct spots with minimal duration. *Adaptability*, this system is highly flexible and hence capable of operating in harsh conditions, i.e. high temperature, humidity pressure and even with node damages and link errors.

Yu et al. proposed *energy efficient* real-time forest fire detection system [Liyang et al., 2005]. Such system prolongs the lifetime of a sensor network by using neural wireless sensor

network and an efficient clustering algorithm for routing the data, in which the nodes used to send the sensed data to the cluster head and then are sent to the base station. Also, the communication between sensors was reduced through in-network processing mechanisms.

The fire monitoring system presented in [Kumar and Kishore, 2017] posses *low maintenance* and *low power consumption*. This is achieved through the utilization of a wireless sensor network that has included both short-range communication via Zigbee and long-range communication via GPRS. The short-range communication used for data exchange locally in the forest, and processing of the collected data while long-term communication is used for sending alerts. Also, Gao et al. proposed forest fire monitoring and early warning system which offers the *flexible* structure, *easy maintained* and *low cost* of operations based on multi-sensor and multi-level data fusion [Gao and Huang, 2015].

3.3 Flood Detection Systems

Flood is also one of the major problems in many countries around the world. According to the United Nations Educational Scientific and Cultural Organizations (UNESCO), fifty per cent of water-related natural disasters worldwide are caused by flood [Organization, WMO]. In most cases, a flood is regarded as a natural disaster caused by the changing weather conditions, i.e. global warming which led to heavy rainfall, strong winds, unusually high tides or tsunamis etc. [Nuhu et al., 2016]. Various experts have researched flooding on different perspectives [Sunkpho and Ootamakorn, 2011], in which flood monitoring, prediction or forecasting and alerting or warning were identified as crucial aspects of managing the flood.

SEISs facilitate flood management by using sensors in monitoring environmental parameters, i.e. precipitation, flow or water bodies levels in real-time. The collected data is sent to the information control centre, in which models are used to predict the occurrence of the flood to make reasonable decisions in preventing or reducing the effects that could be imposed by flood [Sunkpho and Ootamakorn, 2011]. The descriptions of the existing software architectures of flood detection systems are based on the following categories: stakeholders and their concerns in Section 3.3.1, core functional features of flood detection systems are described in Section 3.3.2, their architectures in Section 3.3.3, and supported quality attributes in Section 3.3.4.

3.3.1 Stakeholders and their Concerns

To prevent or reduce the effects of floods, several stakeholders are required to participate in flood detection systems to make crucial decisions. These stakeholders are as defined in

[Sari et al., 2013] include; (1) Governmental agencies such as regional planning agency, local disaster management agency, environmental offices, a regional parliament and health department. All these agencies are responsible for managing water resources, and coordinate the activities of environmental, disaster and emergency management. (2) Police help the people in the emergency during evacuation processes and also advise the local government to curb the building in flood areas. (3) Universities, research institutes and NGOs participate in both the prevention and reduction of consequences of the flood. (4) Local community or citizens (5) Technical people such as developers, analysts, operators, architects, maintainers ensure the system works as required. In order to accomplish their objectives, these stakeholders are interested to know the feasibility of flood detection systems, how are they decomposed and implemented, their nodes interconnected, functionalities and supported quality attributes.

3.3.2 Functional Features

The flash-flood alerting system proposed in [Castillo-Effen et al., 2004] aims at aiding the population of the Andean region of Venezuela, where large amounts of property damage and residents casualties occur due to flash-floods. The proposed system fulfils the following requirements; collection of relevant meteorological, and hydrological data from the threatened regions in which flash-flood and landslides originate, data transmission from the sensed fields to the urban areas, extraction of data, and graphical display of relevant information for assisting authorities to make crucial decisions, registration and storage of collected data, generation of alerting signals and allowance of users using mobile devices to interact with the system.

Also, Sunkpho et al. presented a real-time flood monitoring and warning system for a specific region of the southern part of Thailand [Sunkpho and Ootamakorn, 2011]. Such a system provides an information channel for responsible authorities and also a web-based information source for the public which used to respond to their requests regarding the information of water conditions and flood. Also, the flood forecasting and warning response submit reports to various government agencies along with public media sources to distribute the alert [Chowdhury, 2005].

3.3.3 Architectures

The flood detection systems use module views with layered architecture styles to describe the decomposition and main components of such systems. For instance, the real-time flood monitoring and warning system in [Sunkpho and Ootamakorn, 2011] was described using a module view with three modules; Monitoring Remote Sensor module, Database and Web

Application Server module, and Data Processing and Transmission modules in both remote site and control centre. The monitoring remote sensor module includes customized sensors for collecting data; precipitation, water level and flow measurements. Such data passes through mobile General Packet Radio Service (GPRS) tunnel via a Data Processing and transmission module in the remote site to the data processing and transmission module in the control centre. The data processing and transmission module involve a GPRS Data Unit (GDU) installed at a remote site while the GPRS gateway server is implemented at the control centre. The database and web application server module processes the collected information in real-time and makes them accessible to the user via a web-based application.

Also, The flood risk assessment system proposed in [Amirebrahimi et al., 2016] uses module view of this system using layered architecture style consists of four layers; i.e. Data Layer (DL), Data Access Layer (DAL), Business Layer (BL), and the Presentation Layer (PL). The data layer (DL) contains both the required spatial and non-spatial data to undertake the damage/risk assessment and visualization processes. The data access layer (DAL) is an intermediate layer between the data and business layers as well as providing simplified access and retrieval mechanisms for data stored in the data storage, e.g. files or databases. The business layer (BL) concerns with execution and maintenance of the logical processes and rules that required for calculations. The functionalities of this layer include water infiltration modeling, damage and costs assessment, etc. The output of these functionalities is transmitted to Presentation Layer for the presentation to the user, which in turn the data is sent back to Data Access Layer for the export and storage. The presentation layer (PL) consists of the user interface (UI) which allows the interactions of the users to the system, i.e. capturing users requests and presenting responses.

Similarly, the flood monitoring and detection system (FMDS) in [Udo and Isong, 2014], uses a module view with three modules; sensor field, surveillance centre and mobile phone. The sensor field module consists of sensors for sensing and transmitting the observed parameters to the gateway. The surveillance centre module includes a database which holds the phone numbers and the monitored parameters in the host computer (for the GUI and to initiate alert SMS to the occupants of the flood area), printer and a broadband modem to enable the sending of SMS. The mobile phone module presents the residents of the flood-prone region who will receive the alerts through SMS.

The implementation of weather monitoring in flood disaster management system in [Nithya and Vanamala, 2018] was described using an execution view with a client/server architecture style. Such a view included a person, controller and sensor nodes are used. The person node specifies time and date and request access to the controller. The controller node validates the user (person) to grant the permission of accessing the system. The

sensor node consists of two components for measuring and displaying the data. Finally, the flood detection systems proposed in [Greenwood et al., 2006, Seal et al., 2012] utilizes a hierarchical architecture with sensing nodes, computation nodes and monitoring stations. The computation node concerns with data aggregation and transmission. The system in [Greenwood et al., 2006] is different from the one in [Seal et al., 2012] since [Greenwood et al., 2006] uses a mini-grid based computational model of which part of the computation is done at the monitoring station while the rest is at the nodes.

3.3.4 Quality Attributes

The flood monitoring system described in [Degrossi et al., 2013] supports; *interoperability* and *adaptability*. The *interoperability* is achieved through the adoption of the Sensor Web Enablement standards defined by the Open Geospatial Consortium (OGC) [Botts et al., 2008]. The OGC defines the Sensor Observation Service (SOS) which handles data originating from sensors in an interoperable way. While *adaptability* is achieved by using specifically the SOS framework provided by 52 North German initiative for Geospatial OpenSource software [Jirka et al., 2012]. Such software is easily *adapted* since it is open source code distributed under the terms of GNU license.

The proposed real-time flood monitoring and warning system offer accurate and reliable information for the southern provinces of Thailand to avoid unnecessary destructions and losses [Sunkpho and Ootamakorn, 2011]. The flash-flood alerting system [Castillo-Effen et al., 2004], ensures *energy efficiency*, *flexibility* and *extendability*. The system is supposed to remain functional for a longer period of time. Hence, sensor networks are required to be *energy efficient* to possess long lifetime. This makes the energy optimization to be considered explicitly in designing sensor networks. Also, this system is required to be *flexible* and *extendable* to accommodate the growth of sensor network and topological changes. The node discovery strategies and self-forming capability need to be included in the network routing protocols to facilitate the accommodation of new nodes in joining the network.

The real-time flood monitoring and warning system proposed in [Marin Perez et al., 2012] is *cost effective*, *reliable*, *energy efficient*, *heterogeneous* and *maintainable*. The system is *energy efficient* with robust communication capability and also provides accurate measurements in real-time. The system is flexible to accommodate many applications for autonomous data collection with reliable transmission over large areas.

3.4 Air Pollution Detection Systems

Air pollution refers to the presence of contaminants (i.e. hazardous gases, dust particles and chemical substances) in the air which results into dangerous damages to the health of living organisms mainly humans and harmful environmental effects. Air pollution is one of the major factors that affect the health of humans [Alesheikh et al., 2005] and ecological balance. The World Health Organization reported that 2.4 million people die each year from causes directly attributable to air pollution, so clean air is a crucial requirement for good health [WHO, 1999]. Many activities including industrial activities, burning of forests and (or) trees, transportation, and the explosion of gases pollute the air. Such pollution led to the reduction of air quality which will eventually harm the health of humans and other living organisms [WHO, 2000]. Some of the primary air pollutants in the atmosphere which cause the reduction of air quality include ozone (O3), nitrogen oxides (NO2), sulphur dioxide (SO2), carbon monoxide (CO), aromatic compounds and particulate matter. O3 and NO2 are originating from photochemical reaction while NO, aromatic compounds and CO are from traffic [Lee et al., 1999]. Therefore measuring the level of air pollution is highly required to raise the level of clean air.

Even though monitoring of air pollution is being a very crucial task but is very difficult [Khedo and Chikhooreeah, 2017]. Initially, data loggers were utilized to collect data periodically. This process consumed more time and too costly. The invention of sensors simplifies the process and allows the obtainment of more instantaneous readings [Ma et al., 2008]. The data collected by sensors is sent to the information control centre for further processing, in which models are utilized to determine the quality of air. The following sections provide the descriptions of the existing software architectures of air pollution detection systems are based on stakeholders and their concerns in Section 3.4.1, core functional features of air pollution detection systems are described in Section 3.4.2, their architectures in Section 3.4.3, and supported quality attributes in Section 3.4.4.

3.4.1 Stakeholders and their Concerns

According to the air quality monitoring workshops; SECURE workshop [FP7, 2016] and Research to Practice workshop [Clements et al., 2017], some stakeholders were identified and classified in two groups; the first group focuses on collecting and using air quality for creating policies and awareness about the sources and control of air pollution [Clements et al., 2017] for the betterment of the environment and public health improvement. Such group includes government, regulatory agencies, private companies, citizens, environmental experts, academia and research institutes. This second group concerns with the actual

development of the system (the technical aspect of the system) with regards to how data can be collected, integrated and communicated to other stakeholders. Such group includes IT related experts (developers, maintainers, analysts, architects, operators). The main concerns of these stakeholders include the feasibility of air quality monitoring to meet the intended objectives of the system, system decomposition and implementation, data management, functionalities and supported quality attributes.

3.4.2 Functional Features

Samadzadegan et al. developed an air quality monitoring web-based system that utilizes the internet to help disaster managers to reduce the impacts of air pollution through distributed component system [Samadzadegan et al., 2013]. While in Baralis et al. [2016], an air monitoring system is proposed to help people (citizens) to realize their activities concerning the deterioration of air quality. This system creates awareness of the public on how various factors like pollutants and toxic gases affect the quality of air and analyze air pollution based on meteorological data, pollutants and traffic data.

The CO pollution warning system [Purnamasari et al., 2013] monitors the conditions of air pollution in the covered parking area and warnings to humans for creating an awareness of pollution nearby. Such system displays CO in the form of graph and numbers for ppm level and three pollution conditions: safe (AMAN), alert (WASPADA) and dangerous (BAHAYA). The pollution conditions are indicated in three colours of the LED; green, yellow and red for safe, alert and dangerous respectively. The system activates a buzzer and fan in dangerous pollution conditions.

Air quality monitoring and analysis system presented in [Yaacoub et al., 2013] assists decision makers in making appropriate measures to reduce or mitigate the impacts of air pollution by providing reliable information regarding air pollution. This system handles data cleaning and filtering hence it processes and presents the air quality measurements in real-time and adequate formats depending on specific end-user groups, e.g. the general public or environmental experts.

In [Ustad et al., 2014], Ustad et al. introduced an air pollution monitoring system that monitors the concentration level of air pollutants, i.e. CO, SO2, and dust concentration in the environment due to the industrial processes. Similarly, Kgoptjo et al. presented an air quality monitoring system (AQMS) based on IEEE/ISO/IEC 21451 standard [Zheng et al., 2016]. AQMS uses electrochemical and infrared sensors to measure the concentrations of CO, CO2, SO2, and NO2 and store the collected information in the data server.

3.4.3 Architectures

The indoor air quality monitoring system presented in [Lozano et al., 2012] uses module architecture to describe the decomposition of a system. In which a sensor node is decomposed into four modules: Communication and Processors Module, Conditional Signal and Conversion Module, Sensor Module, and Power Supply Module. The communication and processors module uses XBee and XBee pro modules from Maxstream. The conditional signal and conversion modules are electronic circuits and instrumentation used for adapting sensor signals to the range of the conversion modules. The sensor module includes temperature sensor (LM35DZ from National Semiconductor), humidity sensor (HIH–4000 from Honeywell), light sensor (light dependant Resistance NSL-19M51 from SILONEX) and quality air sensor (TGS 2600 from FIGARO with High sensitivity to gaseous air contaminants i.e. hydrogen, ethanol, iso-butane and carbon monoxide). The power supply module is used for energizing the entire sensor node and is designed using commercial electronic devices.

Also, an air quality monitoring systems [Raju, 2014] uses a module view with a layered architecture style to describe the decomposition of the system into two layers: physical layer and application layer. The physical layer concerns with the acquisition of real-time data from sensors together with physical location, date and time through the GPS module. The application layer is responsible for the provision of services relating to the monitoring of air pollution through a website. This layer enables users to have access to the system by log on to the system and view the concentration of pollutants that have been measured.

Similarly, a noise and air pollution system proposed in [Guthi, 2007], uses module view to describe system decomposition into 4-tier architecture. Tier 1 is the environment, provides the collected information regarding the environmental parameters of a particular region required for noise and air pollution control. Tier 2 belongs to sensor data acquisition, and this concerns with suitable sensor devices based on specified characteristics, i.e. the range of sensing and sensitivity. Tier 3 is for decision making. This tiered process the data originated from tier 2 before making crucial decisions regarding the control of noise and air pollution. The intelligent environment belongs to tier 4 which is specifically for the identification of variations in sensor data and fixation of the threshold value based on the observed level of CO and noise.

The implementation of an air pollution monitoring system was described using execution view with a client/server architecture style [Al-Ali et al., 2010]. Such view include Data-Acquisition Unit (Mobile-DAQ) and a fixed Internet-Enabled Pollution monitoring Server (Pollution-Server).

Finally, Ujang et al. uses a data view to conceptualize the data that are either processed inside the process or exchanged between the processes of air pollution dispersion system [Ujang et al., 2013]. Such a system is used to improve visual analysis of air quality monitoring in the event of dealing with urban air pollution. In which, the Operational Street Pollution Model (OSPM) is adopted to implement the 3D spatial city model. Figure 3.1 depicts the attributes of CityGML and OSPM parameters for the amalgamation. The urban air pollution dispersion models involve Operational Street Pollution Model (OSPM), meteorological (wind flow, wind speed, vertical turbulence, and canyon Ventilation), physical data (receptor height, vortex, recirculation zone, and trapeze slant edge length) and other geometrical information. The OSPM includes plume model for calculating vehicle emission, box model for recirculating pollutant and background pollutant information.

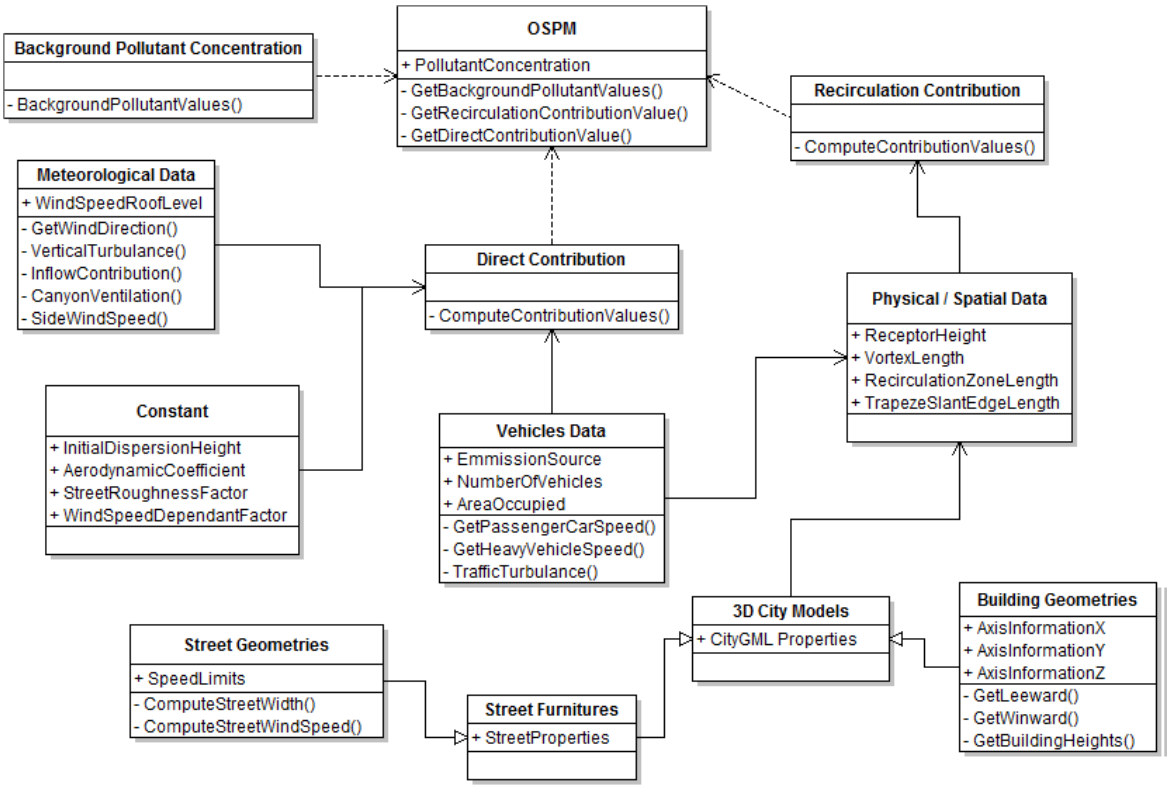


Figure 3.1: Data Model of Air Pollution Dispersion figure taken from [Ujang et al., 2013]

3.4.4 Quality Attributes

Wireless Sensor Network Air Pollution Monitoring System (WAPMS) attained the improved *energy efficiency* through a data aggregation algorithm (Recursive Converging Quartiles (RCQ)) and a hierarchical routing protocol [Khedo et al., 2010]. Such algorithm significantly

reduces the amount of data to transmit by filtering out invalid readings, merging data to eliminate duplicates and summarizing the data to be sent. A hierarchical routing protocol was used for managing power by making nodes to sleep during idle time. Similarly, an Indoor Air Quality monitoring (IAQ) system possess *energy efficiency* via an adaptive duty cycling mechanisms in sensors [Jeličić et al., 2011].

The air quality monitoring system in [Ahuja et al., 2016] uses the Internet of things (IoT) to provide a *low-cost portable* system that creates awareness to the public regarding the air quality and hence make better decisions of either travelling routes or house purchase. F. Samadzadegan et al. proposed a *flexible* and *interoperable* air pollution monitoring system [Samadzadegan et al., 2013]. The system is web-based which uses Sensor Web Enablement (SWE) framework of the Open Geospatial Consortium (OGC) standard in addressing interoperability and flexibility challenges in monitoring and alerting air quality information for supporting the management of air quality. Similarly, Kumar et al. presented an air quality monitoring system in [Kumar and Jasuja, 2017]. This offers *low cost*, *low power*, *high accurate* system for monitoring the environment using dedicated remote sensors through a single board minicomputer Raspberry Pi.

3.5 Landslide Detection Systems

Landslides are one of the major catastrophic disasters that happen around the world. This has been a major problem in many countries including India which faces landslides every year Sri Lanka [W.A.S.R. et al., 2017], Malaysia [Yunus et al., 2015], etc. Landslide involves mass failures of slope, i.e. movement in rock, ice, soil or organic materials due to the influence of gravity and which eventually led to considerable damage to the environment, economy, natural habitat and others [Dakave and Gaikwad, 2015]. The occurrence of landslides originated from either geological, human activities, morphological or physical effects.

The continuous monitoring and early warnings of landslides may reduce or prevent the loss of human, other living organisms and the destruction of properties. However, the design of an early warning system is complicated due to the increased degree of unpredictability of landslides and rapid change of contributing conditions [Ramesh, 2014]. Also, conventional landslide monitoring systems, i.e. traversing inclinometer system are not efficient and very expensive [Hass et al., 2008]. Innovative mechanisms or techniques, i.e. sensing techniques are highly required to capture relevant signals at the right time with minimal monitoring delays. Sensing techniques are one of the leading technologies capable of capturing quickly and processing data in real-time even under harsh or remote environment [Ramesh, 2014],

and WSN has become a most commonly used method for landslide detection. The data collected by sensors is sent to the information control centre for further processing, in which models are utilized to determine the likelihood of landslides. The description of the existing software architectures for landslides detection systems are based on the following categories: stakeholders in Section 3.5.1, core functional features of landslides detection systems are described in Section 3.5.2, their architectures in Section 3.5.3, and supported quality attributes in Section 3.5.4.

3.5.1 Stakeholders and their Concerns

Stakeholders participation in landslide detection systems is needed to ensure the effectiveness of early warnings to prevent or reduce the consequences of landslides. Such stakeholders include public authorities and other institutions, experts, residents, operators, the general public and others [Preuner et al., 2017, Picarelli, 2014]. Public authorities such as national authorities, police, etc. are responsible for the promotion of responsibility sharing and local participation. Hence they are required to interact with the system to determine response time, information distribution mechanisms and receivers. Residents need to protect themselves; thus they expect to be provided with information regularly. Experts possess detailed knowledge of the hazard (landslide) hence are responsible for providing crucial design parameters that can be integrated into the system with the intent of collecting and analyzing data. Additional technical experts develop and maintain the systems. The main concerns of these stakeholders include the feasibility of landslides detection system to meet the intended objectives of the system, system decomposition and implementation, functionalities and supported quality attributes.

3.5.2 Functional Features

Nguyen et al. proposed a system for detecting rainfall-induced landslides capable of data acquisition, data processing, data storage and wireless data transmission [Nguyen et al., 2015]. The system collects data on soil pore water pressure, the vibration of the earth, soil movement, moisture and temperature of the environment.

A drought forecast and alert system (DFAS) alerts the users through mobile communication [Kung et al., 2006]. While the landslide detection system in [Ramesh, 2014] uses broadband to provide connectivity to a broader audience mainly public while streaming data in real-time to allow experts to analyze data, possible predict the actual occurrence of landslide and disseminate information to the government.

The essential requirements of landslide detection system presented in Kotta et al. [2011] are the identification of required sensors to monitor and detect landslide while ensuring efficient delivery of data in real-time.

3.5.3 Architectures

The landslides detection system in [Nguyen et al., 2015] was described using a module view. Such system was decomposed into with three modules; The data acquisition module allows the collection of data from both digital and analogue signals obtained from accelerometer using digital drivers and directly from soil moisture and temperature sensors respectively. The data processing module is a core component used for processing all the collected data from sensors and outgoing data to the gateway via the following submodules; sensor sampling, health monitoring, and power saving. The data communication module concerns the routing algorithms and methods for synchronizing time in a wireless sensor network.

Similarly, the wireless sensor network for landslides detection in Nusa Tenggara Timur, Indonesia [Kotta et al., 2011], uses a module view with a layered architecture style. Such system was decomposed into the two-layer hierarchy; the lower layer consists of wireless sensor nodes which sample and collects data from the deep earth probe (DEP) then data packets are sent to the upper layer. The upper layer aggregates and integrate the data received from the lower layer before sending it to the gateway (sink node).

Finally, the landslide detection system proposed in [Huang et al., 2015], uses a module view with a layered architecture style. The system was decomposed into three layers: support, service and client layers. The support layer uses ArcGIS server to provide a distributed computing environment to meet the high-efficiency requirements for data analysis and processing. This layer consists of databases; a spatial database for spatial data, i.e. images, vector data and lithology, attribute database provides detailed information regarding landslide flows, i.e. name, location, etc., and monitoring database stores real-time monitored data. The service layer employs ArcGIS Extension for Bing Maps application programming interface (API) and Bing maps Interactive Software Development Kit (SDK) to provide data and map services. Besides, this layer sets the triggering criteria and landslide warning thresholds. The client layer consists of three dimensional Virtual Earth interface, supported by Bing maps 3D and JavaScript programming to allow end-users to access the system.

3.5.4 Quality Attributes

The landslides detection system presented in [Nguyen et al., 2015] is *flexible* and *energy efficient*. *Flexibility* is achieved by implementing flexible switching between star and tree topologies to ensure reliable transmission based on weather conditions. The power management is also employed to improve operational reliability and *energy efficiency*. Similarly, Arnhardt et al. proposed a *easy maintainable* real-time monitoring of landslides using open spatial data infrastructure and wireless sensor networks Hass et al. [2009].

A *heterogeneous* landslide detection system described in [Kung et al., 2006]. This system supports various communication technologies, i.e. ZigBee, GSM, satellite to ensure efficient delivery of data to the management centre in real time. While Jian et al. proposed a *user-friendly* landslide monitoring system [Huang et al., 2015]. Such a system supports a wide range of users that analyze, manage and interpret monitored data and also enhance the public to have a better understanding of landslide.

Ramesh introduced a *scalable*, *reliable* and *energy efficient* sensor based landslide detection system [Ramesh, 2014]. The energy efficient data collection methods, fault tolerant clustering approaches, and threshold based data aggregation techniques are incorporated in the system in order to improve *reliability* and *energy efficiency*.

3.6 Road Traffic Control Systems

Vehicular traffic is increasing tremendously around the world especially in urban and metropolitan areas worldwide [Kafi et al., 2012]. This led to increasing traffic congestion which causes dramatic impacts on human health, environment and economy. The physical approaches, i.e. extension of roads are no longer feasible due to high construction costs and physical limitations [Jo et al., 2014]. The road traffic management systems have appeared to be the more viable and suitable solution towards an effective traffic control.

Despite the involvement of many researchers in the study of innovative road traffic management techniques, traditional solutions relied on traffic lights with fixed cycles or manually controlled by humans [Collotta et al., 2014]. Such systems tend to be more effective and suitable in road sections with lower traffic flows rather than areas with high vehicle density. The sensor-based traffic control systems have been deployed to address the challenges of existing solutions. These systems make use of sensors to monitor road parameters such as vehicle density, etc., such information is sent to an information control system so as to be processed and analysed then the resulted control actions are executed

through actuators i.e. traffic signal to optimize traffic flow. The descriptions of the existing software architectures for road traffic control systems are based on the following categories: stakeholders and their concerns in Section 3.6.1, core functional features of road traffic control systems are described in Section 3.6.2, their architectures in Section 3.6.3, and supported quality attributes in Section 3.6.4.

3.6.1 Stakeholders and their Concerns

The successful introduction or utilization of road traffic control systems requires the mobilization of the following stakeholders; local traffic management agencies, administrators, transportation engineers, regional transport authorities and other researchers involved in planning, design, implementation, operating and maintenance of traffic management and traveler information systems, while police (enforcement agencies), and road users (travellers), are interested in accessing traveller information systems for effective control of the traffic flow [Association, 2018]. The major concerns of these stakeholders include the feasibility of road traffic control systems to meet the intended objectives of the system, decomposition, integration, functionalities, and supported quality attributes.

3.6.2 Functional Features

George et al. introduced a traffic monitoring system using IR sensors in [George et al., 2017]. This system fulfilled the following requirements; vehicle detections and increases or decreases the count based on the vehicle density, detection of traffic congestion, control of traffic flow based on vehicle density, handle the data of registered users of the system and suggestions of alternate paths to the registered users.

Ranjin et al. proposed an adaptive road traffic control system which continuously monitors the traffic flow, and then execute an effective scheme on the controller to determines an efficient traffic flow to reduce or prevent congestion [Ranjini et al., 2011]. Additionally, this system consists of a detector store which stores the details of detectors and the vehicles pass through such detectors and (/ or) wait for the signal.

The [Tavladakis and Voulgaris, 1999] proposed an adaptive traffic control system that reads detectors' states and then computes traffic flow information to allocate timing schedule to traffic lights of a particular intersection based on traffic flow. The system allows the communication between controllers of adjacent nodes to share data and synchronization information, switches on and off traffic lights. Similarly, both Zhou et al. proposed an adaptive traffic light control algorithms which consider various traffic factors, i.e. traffic

volume, vehicle density and waiting time to adjust both the sequence and length of traffic lights based on the collected traffic information in real time [Zhou et al., 2010, 2011].

3.6.3 Architectures

An adaptive road traffic control system in [Ranjini et al., 2011] uses conceptual view to describe the system as the whole. The system is partitioned into the following classes; The traffic controller as the main class of the system coordinates other classes. The detector class reflects the detectors (sensors) hence describes the properties of detectors, i.e. physical location and count of vehicles passing through during a certain period. The optimizer class which optimizes the traffic flow and is the parent of other subclasses; signal Optimizer, time limit optimizer and adjacent signal optimizer. The road class comprises information regarding the roads and their average traffic flow rate.

Similarly, the multi-model intelligent traffic signal system (MMITSS) described in [of Arizona et al., 2013] uses a conceptual view with client/server architecture style to describe the partition of the system. The system was partitioned into a traffic signal controller, field sensors and MMITSS Roadside Processor associated with various physical devices, i.e. a processor (at least one), memory, and physical interfaces (e.g. Ethernet, RS-232, or wireless – 3G/4G, 5.9GHz DSRC, or other such as CAN-bus). The system is intended for two types of travelers; motorized vehicles and non-motorized travelers. Non-motorized travelers do not require licenses to operate on the public roadway including pedestrians, bicyclists and other modes. Motorized vehicle travelers include any vehicle that must be licensed to operate on the public road which is either equipped or unequipped.

A real-time traffic adaptive signal control system (RHODES) in Arizona [Mirchandani and Head, 2001], uses module view with hierarchical architecture to describe the decomposition of the system. In which the global traffic control problem is split into small sub-problems which are handled locally and then hierarchically connected to optimize traffic flow. Similarly, Curiac et al. presented a hierarchical architecture of the urban traffic control system which aims at optimizing the vehicle and pedestrian traffic flow in big cities [Curiac and Volosencu, 2010] with the wireless sensor-actuator networks at the first lower level and zonal control systems at the second level of the hierarchy.

Finally, a traffic control system based on wireless sensor network in [Zhou et al., 2010] uses a module view with layered architecture to describe the decomposition of the system. The system has been decomposed into three layers which consist of a traffic flow policy model and Intersection Control Agents (ICAs). Wireless sensors located on the lanes at the intersection collect traffic information such as the number of vehicles, speed etc. and then

send the collected information to the ICAs which determine the optimal flow model of the intersection based on the data sent by sensors.

3.6.4 Quality Attributes

The road traffic monitoring system presented in [George et al., 2017] is *secured* since the admin specify accessibility of the system, *easy maintained*, *accessible* to all even visually impaired people, and *reliable*. While, the urban traffic control system proposed in [Curiac and Volosencu, 2010], is *flexible* such that accommodates various application changes, i.e. adding or removing monitored entities, integrating other services or applications etc.

Jo et al. proposed a traffic information acquisition system emphasizing on high *accuracy*, *energy efficiency* and low *maintenance* using small ultrasonic sensors and routing protocol with an associated power-saving mechanism [Jo et al., 2014]. Similarly, a sensor-based system for traffic jam avoidance presented in [Jiménez and García, 2015] is *energy efficient* by using a data collection algorithm which reduces the number of transmissions and amount of information.

Finally, the smart traffic management system in [Asensio et al., 2015] has been developed with the intention of handling *heterogeneity* of devices by using u-Clouds. u-Clouds allows the deployment of the infrastructure along with wide spaces in the city.

3.7 Summary

In this chapter, a literature review on the existing SEISs was performed to answer the first research question (RQ1): *Which are essential aspects or perspectives that are required to describe SEISs?*.

To answer the research question 1, this review has retrieved a set of existing solutions for SEISs which provide users and other stakeholders with a wide range of services and features for environmental phenomena monitoring and controlling, i.e. forest fires, flood, air pollution, etc. The *diversity of these systems* is observed on the application level, while technically they possess similar subsystems mainly sensor-actuator, information control centre and communication subsystems. The *sensor-actuator subsystem* describes the solutions which measure physical environment events, i.e., temperature, humidity, etc., pre-processing the collected data and alter the environmental parameters to control particular environmental phenomena. The *information control centre subsystem* refers to a subsystem which concerns with the integration, analysis, storage, and presentation of the information received from the

sensors and then generates alarms and control actions via actuators based on the specified objectives of particular SEIS. The *communication subsystem* facilitates the interactions within and between the sensor-actuator subsystem and information control centre subsystem.

From this review, the following set of common functionalities were identified relating to real-time monitoring and controlling of environmental parameters in the field of interest, visualization, prediction and control of environmental phenomena. For instance, forest fire detection systems described in Section 3.2.2, perform real-time monitoring and view fire and weather conditions of forests, use algorithms and alarms to determine forest-fire risks and control the propagation of fires. The flood detection systems as described in Section 3.3.2, collect and analyse relevant meteorological and hydrological data from threatened fields, graphical display of such information and generation of alerting signals warnings to the public in case of likelihood of a landslide. The air pollution detection systems described in Section 3.4.2, perform real-time monitoring of the pollutants, analyse air pollution based on meteorological data, pollutants and traffic data, visualization of such conditions and issuance of alerts in dangerous pollution conditions. The landslides detection systems described in Section 3.5.2, monitor data on soil pore water pressure, earth vibrations, soil movement, etc., visualize and analyse the collected data to predict the occurrence of landslides, disseminate information to public and issue alerts in the likelihood of landslides. Finally, the road traffic control systems described in Section 3.6.2, perform real-time monitoring and viewing of vehicle density, detect traffic congestion, control traffic flow through actuators i.e. traffic signals.

The most critical and common recurring non-functional features include (i) *Energy Efficiency*: SEISs involve battery-powered sensors which consume energy during sensing, processing, communication and in an idle state. Hence the SEISs should be designed to maximize the lifetime of sensor networks (and (or) sensor nodes). (ii) *Interoperability*: The heterogeneity of SEISs can be demonstrated into two different facets in terms of both; heterogeneous data and network originated from the involvement of various sensor or actuator nodes [Nithya and Vanamala, 2018, Al-Ali et al., 2010]. Therefore the reference architecture for SEISs needs to provide a mechanism of unifying heterogeneous data to support the effortless and seamless integration of different data sources. (iii) *Maintainability*: SEISs are required to monitor and control the environmental phenomena autonomously. The operations of SEISs should be designed in such a way that it does not provide an additional burden of human involvement especially in extreme conditions of remote or inaccessible sites.

Currently, the existing SEISs are stand-alone solutions and their descriptions resulted from the application of a lot of different viewpoints and utilizing various architectural styles or patterns to fulfil specified requirements. For instance, conceptual viewpoint [Ranjini

et al., 2011, Zhang et al., 2008, of Arizona et al., 2013], module viewpoint [Jadhav and Deshmukh, 2012, Alasia, 2013, Hartung et al., 2006, Commission, 1998, Stipanicev et al., 2018, Sunkpho and Ootamakorn, 2011, Amirebrahimi et al., 2016, Udo and Isong, 2014, Greenwood et al., 2006, Seal et al., 2012, Lozano et al., 2012, Raju, 2014, Guthi, 2007, Nguyen et al., 2015, Kotta et al., 2011, Huang et al., 2015, Curiac and Volosencu, 2010, Zhou et al., 2010, Mirchandani and Head, 2001], execution viewpoint , topology [Jadhav and Deshmukh, 2012, Zhang et al., 2008], data viewpoint [Ujang et al., 2013]. Nevertheless, to support maintainability, the module viewpoint was applied in flood risk assessment system with layering architectural style with four layers, i.e. data, data access, business and presentation layers [Amirebrahimi et al., 2016]. For dealing with interoperability, a data viewpoint was applied to construct a unified data view with Key-Value type table architectural style for the environmental monitoring system was used in [Ujang et al., 2013]. The energy efficiency of sensor networks has been achieved by the application of topology viewpoint with network cluster-tree on forest fire detection system [Zhang et al., 2008]. In which the nodes in the SEISs were organized based on three categories; ordinary bottom nodes, cluster heads and network coordinators. Despite effective applications of these viewpoints in constructions of SEISs, there is no clear and standardized understanding of these viewpoints in modeling SEISs software architectures. Many SEISs use their own native conventions (or even worse, no particular convention at all). However, it is extremely useful to have well-defined viewpoints to increase consistency and efficiency in modeling SEISs.

Additionally, this review showed that there is still a technological gap between developed and developing countries. Since most of the SEISs are deployed in developed countries for examples forest fire detection systems in Canada [Canada, 2018], Forest Fire Surveillance System in South Korea [Son and Kim, 2006], flash-flood alerting system in Venezuela [Castillo-Effen et al., 2004], flood monitoring system in Germany [Jirka et al., 2012], etc. this implies that the developing countries are still suffering from various environmental phenomena, i.e. landslides, air pollution, flood, etc. without proper precautions. However, the advancement of technologies promotes the development of cost-effective SEISs which can be afforded by developing countries. Despite the fact that some of the presented SEISs have achieved substantial performance, they possess similar requirements that could be fulfilled holistically to gain the full benefit of these systems. A robust and holistic reference architecture for SEISs is still missing. It is therefore vital to have a reference architecture for SEISs which is associated with established best practices to facilitate the development and maintenance of SEISs instead of reinventing the wheel.

The results of this review serve as a roadmap in defining the viewpoints required by SEISs in Chapter 6 and deducing the common stakeholders and their concerns in Chapter 5.

Chapter 4 proceeds with the related work on the existing reference architecture around the domain SEISs to establish theoretical foundations of the proposed reference architecture for SEISs.

Chapter 4

Existing Reference Architectures

The objective of this chapter is to establish theoretical concepts and rationale of facilitating the development and maintenance of SEISs. This is achieved by describing and analyzing some of the existing reference architectures within the area of environmental monitoring and controlling information systems. The previous chapter presented the analysis of the existing SEISs towards the identification of the common requirements and essential perspectives required in the description of SEISs. Eventually, related work on the existing reference architectures is briefly outlined to be used as a reference or basis in developing the reference architecture for SEISs.

4.1 Introduction

A reference architecture achieves the well-recognized understanding of the specific domain, promotes the reuse of design expertise and facilitates the development, standardization and evolution of software systems. A number of reference architectures for various domains relating to the environmental monitoring applications are found in the literature. However, in this section, some reference architectures which serve as representatives of the most relevant reference architectures for SEISs are presented. The amount of research on SEISs is vast, and the relevant information regarding reference architectures is scattered. Such that the central theme of reference architectures around the SEISs domain revolves across the sensor systems, real-time environmental monitoring and early warning systems. In the following sections, such reference architectures and their potential usage and limitations with regards to the context of this research will be presented.

4.2 Reference Architecture for Sensor Networks Integration and Management

The reference architecture for Sensor Networks Integration and Management (SeNsIM) enables the deployment of applications such as environmental monitoring, based on multiple sensor systems by providing a standardized way to manage, query, and interact with sensors [Casola et al., 2009]. The SeNsIM provides a general *integration platform for heterogeneous sensor systems* by facilitating the; (a) deployment of applications based on multiple sensor systems/networks and (b) a generic user or a application to access data sensed by a network. Moreover, the SeNsIM reference architecture ensures scalability since new networks are easily accommodated or deployed. The architectural model of this reference architecture utilizes wrapper-mediator paradigm to bridge the gap between heterogeneous sensor systems and provide a unique mechanism of managing, querying and interacting with those sensor systems. SeNsIM consists of the mediator component which conceal sensor networks heterogeneity from end-users or applications using ad hoc connectors (wrappers). The mediator is responsible for formatting and forwarding user requests to the different networks as well as organising the information received from the wrapper to satisfy user or application queries. The wrappers are responsible for translating the incoming queries and forwarding them to the underlying sensors and then sends to the mediator appropriate description of the related information based on a common data model.

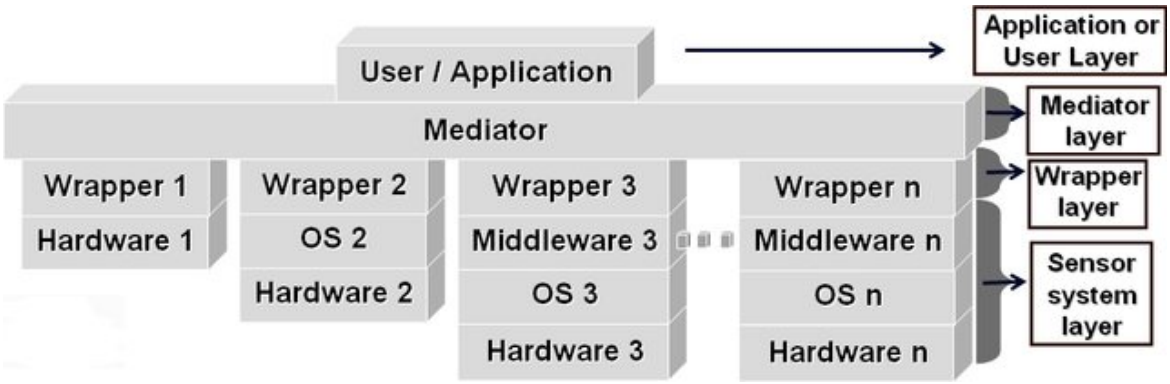


Figure 4.1: Reference Architecture for SeNsIM figure taken from [Casola et al., 2009]

Figure 4.1 illustrates the SeNsIM reference architecture using a module view with a layered pattern. SeNsIM reference architecture is decomposed into four logical layers; Application or User, Mediator, Wrapper and Sensor system layers. The application or user layer allows a user to send requests and elaborate the retrieved data. Also, a generic application can access sensor data through the system API. The mediator layer classifies

network features, format and forward queries to specific wrappers. This layer is associated with a DBMS for storing data related to networks with their sensors, user queries and related results. The wrapper layer extracts and manages information on the underlying network and its sensors. This layer receives data from the mediator and is executed on the local system using its API and the local query language. The sensor system layer extracts the network features and performs the retrieval process. However, this reference architecture focused only on the integration of heterogeneous sensor systems using a unified data model and the description of such reference architecture resulted from the application of a module viewpoint. Therefore this reference architecture does not cover all the essential aspects of SEISs as specified in Chapter 3.

4.3 Reference Architecture for Early Warning System

The reference architecture for Early Warning System (EWS) demonstrates integrated architectural designs of early warnings systems like flood, tsunami, and storms [Athanasiadis and Mitkas, 2004]. The description of such reference architecture includes the system structure, functional behavior and non-functional requirements.

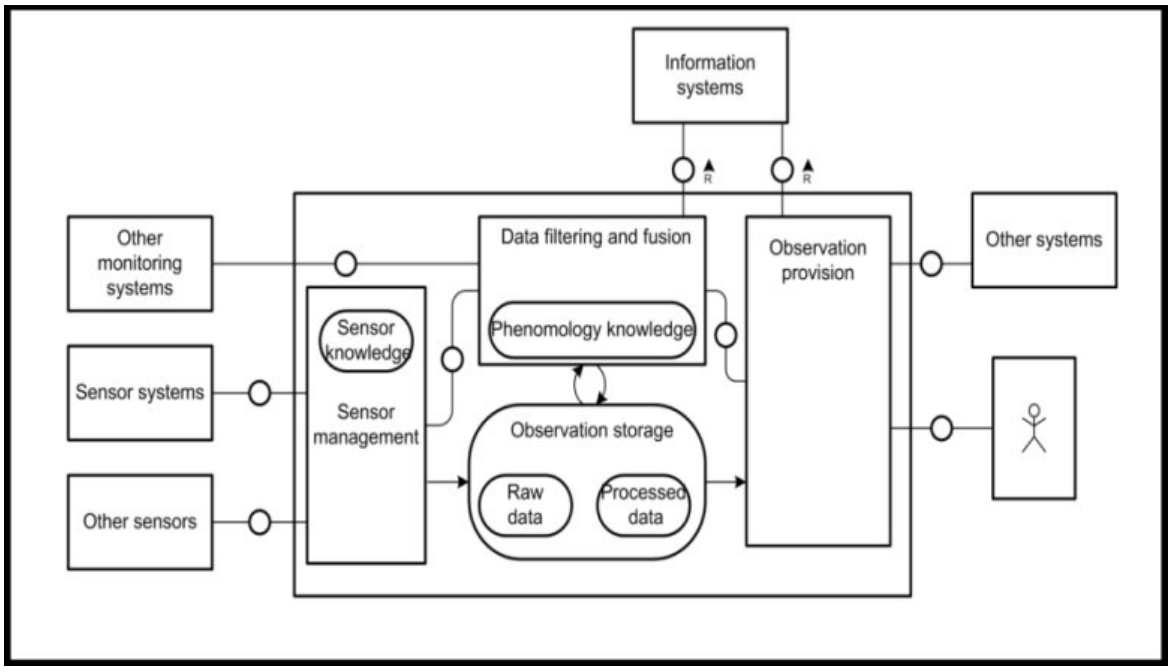


Figure 4.2: An EWS reference architecture figure taken from [Athanasiadis and Mitkas, 2004]

As shown in Figure 4.2, the reference architecture for EWS uses a conceptual view to describe the main components of EWS. The EWS consists of the following main components; sensor management, data filtering and fusion, external information systems, observation information provision, and observation storage. The sensor Management concerns with the sensor management tasks, i.e. sensor communication, sensor control and dynamic addition and removal of sensors. The data filtering and fusion are responsible for the processing of raw sensor data into the output of a monitoring system. The external information systems supply the necessary auxiliary information for data filtering and data fusion components. The observation information provision facilitates the communication to external systems and transforms the output of data fusion/filtering component into the require presentation. The observation storage is responsible for storing raw and processed data and providing access to past observations and increases the robustness of the system [Meissena and Fuchs-Kittowskib, 2014]. It is not clear how this reference architecture handles all the functionality, leading to a rough representation of functional classes. It is also interesting to observe that the components of information systems are ignored. Moreover, the description of this reference architecture did not cover all the essential perspectives of SEISs, since only conceptual viewpoint was applied.

4.4 Distant Early Warning System (DEWS) reference architecture

The Distant Early Warning System (DEWS) reference architecture facilitates interoperable early warning systems based on an open sensor platform [Esbrí et al., 2011]. DEWS gather process and display events and data collected by open sensor platforms to enable operators to quickly decide whether an early warning is necessary and to send warning messages to the authorities and public if desired. This reference architecture is based on *service-oriented architecture (SOA)* concepts and relevant standards (OGC, W3C, OASIS) and is independent such that it can be applied to multiple situations (forest fires, tsunamis, earthquakes, floods, etc.).

As shown in Figure 4.3, the DEWS reference architecture uses a conceptual view to describe the main functionalities of DEWS. In which the DEWS reference architecture follows modular principles of SOA. Typically, the operator works with the Command and Control User Interface (CCUI). The operator initiates the message composition process during the issuance of an early warning. The Information Logistics Component (ILC) generates simple warning messages for each user that must receive the message. User profiles

are usually stored in a separate database which is associated with various parameters for registered users, i.e., language settings, exciting areas, dissemination channels and other settings, which enable personalization of the warning messages to be disseminated. The generated messages are transmitted to the Information Dissemination Component (IDC) converts the messages into channel specific which formats and propagates the messages. Other components like the sensor platform, the simulation and the map servers are connected via standardized OGC services. This reference architecture relies on the application of conceptual viewpoint only, hence other perspectives of SEISs were uncovered.

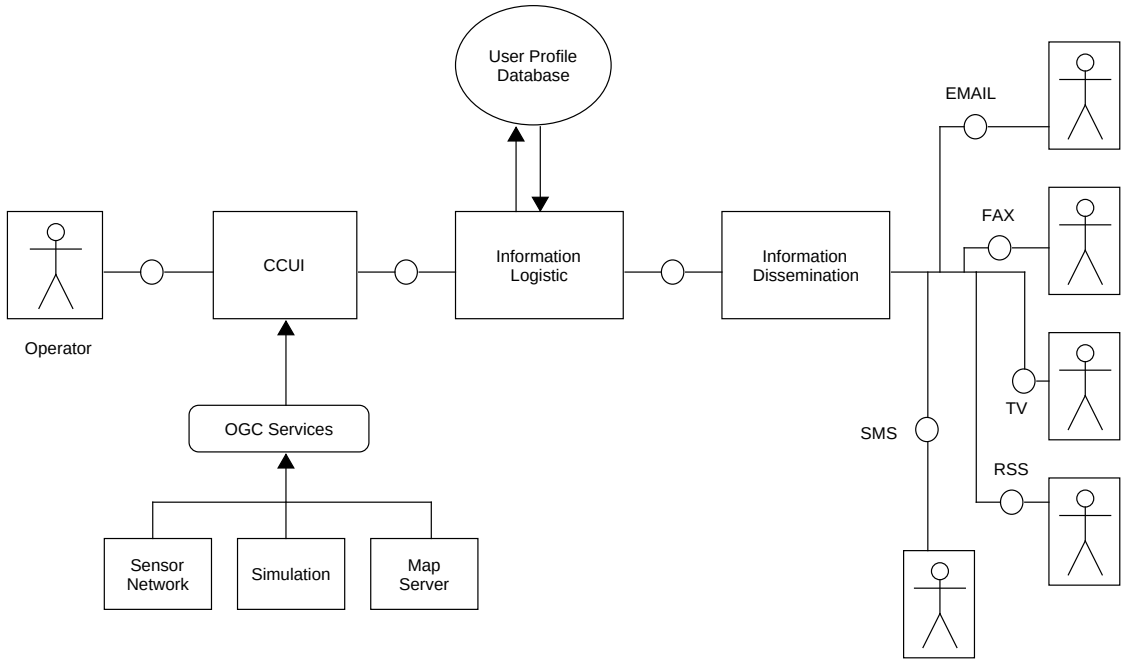


Figure 4.3: DEWS reference architecture from [Esbrí et al., 2011]

4.5 Reference Architecture for Real-time environmental monitoring, early warning and decision support systems (EMEWD)

The reference architecture for Real-time environmental monitoring, early warning and decision support systems (EMEWD) *facilitates the development of large-scale environmental monitoring*, *early warning and decision support systems* [Balis et al., 2017]. Such reference

architecture is associated with the distinction between operational data needed for ongoing assessment and decision-making, and archive data for research and analytical purposes.

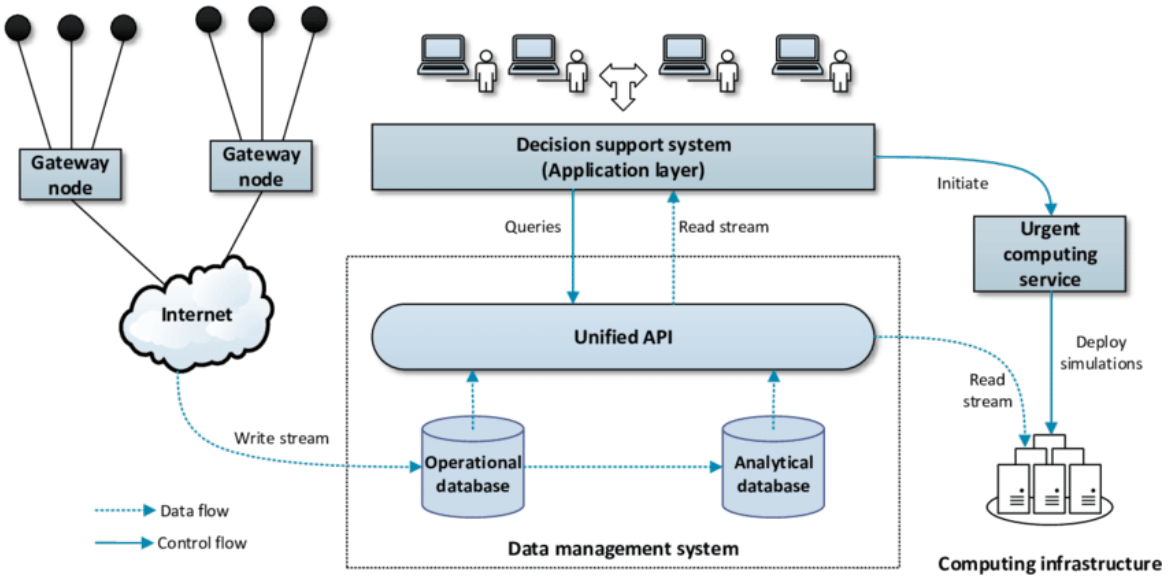


Figure 4.4: Reference Architecture for EMEWD figure taken from [Balis et al., 2017]

Figure 4.4 depicts a reference architecture for EMEWD using a conceptual view with the *focus on the data management system*. The data management system handles the stream of time series data collected from the environmental sensors and allows multiple users to invoke queries through computing services offered by a decision support system. The storage system is divided into an operational database for storing recent data and handling real-time analysis scenarios, and an analytical database for storing archival data. These databases are accessed using a unified programmer's interface (API). This design of databases is crucial for handling the increases sensor measurement frequency, and user queries during the crisis events. This reference architecture lacks guidelines definition for its use and instantiation and ignores other perspectives that required in the description of SEISs, i.e. important architecture views such as execution and topology views were not defined.

4.6 Internet of Things (IoT) Reference Architecture

The IoT reference architecture is designed as a reference for the generation of compliant IoT concrete architectures based on the specific needs [Bauer et al., 2013]. The IoT reference architecture has been kept rather abstract in order to enable the construction of many, potentially different, IoT architectures. This reference architecture is made up of the following views; functional, information, deployment and operational views.

The functional view describes the functional components of IoT systems by first mapping the IoT unified requirements to the different Functionality Groups of the IoT Functional Model. Then clusters of requirements of similar functionality are formed and a Functional Component for these requirements defined. Afterwards, the Functional Components are refined after discussion with the technical work packages. Figure 4.5 depicts the Functional View diagram and shows the nine functional groups of the Functional Model. Such functional groups include; Application, Device, Management, Security, IoT Process Management, Service Organisation, Virtual Entity, IoT Service and Communication. The Management functional group describes the management activities of IoT systems, i.e. configurations, fault, reporting, etc.. The Security functional group concerned with ensuring the security and privacy of IoT-A-compliant systems. The IoT Process Management functional group provides the functional concepts and interfaces necessary to augment traditional (business) processes with the idiosyncrasies of the IoT world. The Service Organisation functional group composes and orchestrates Services of different levels of abstraction and hence acts as a communication hub between several other Functional Groups. The Virtual Entity functional group describes functions for interacting with the IoT System on the basis of virtual entities, and functionalities for discovering and looking up services that can provide information about virtual entities, or which allow the interaction with virtual entities. The IoT Service functional group describes IoT services and functionalities for discovery, look-up, and name resolution of IoT Services. The Communication functional group models the variety of interaction schemes derived from the many technologies belonging to IoT systems and providing a common interface to the IoT Service functional group.

The information view describes how the information to be represented in IoT is defined, structured, stored, manipulated, managed and exchanged through the system by facilitating the generations of static information structure and dynamic information flow. The current version of the information view focuses on the description, handling and life cycle of the information and flow of information through the system and the components involved. Such that the view for modeling the types of virtual entities is provided in more details. The virtual entity is the core element of any IoT system since it models the physical entity or the "Thing" that is the real element of interest. The virtual entity has an identifier (ID), an entityType and a number of attributes for more description of the entity and how it can be used. The modeling of an entityType is of great importance since it determines the attributes that a virtual entity instance can possess by defining its semantics. The IoT services handle the information in the system by providing access to On-Device Resources, e.g. sensor resources which make real-time information about the physical world accessible to the system. Furthermore, other

IoT services may derive additional higher-level information by processing and aggregating the information provided by IoT services or resources.

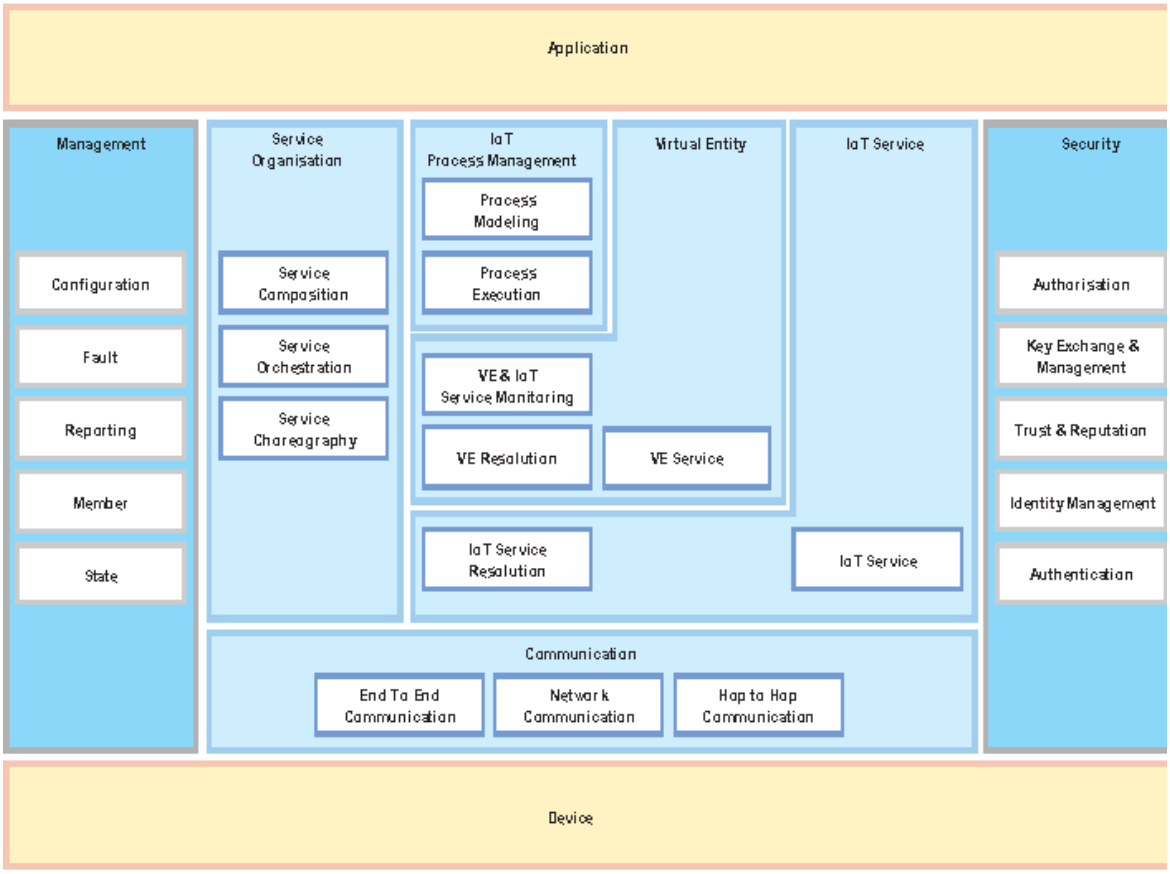


Figure 4.5: Functional View of IoT Reference Architecture figure taken from [Bauer et al., 2013]

The deployment and operation view describes how the IoT system can be realized by selecting technologies and making them communicate and operate in a comprehensive way. The main intention of this view is to provide users of the IoT Reference Models with a set of guidelines for different design choices for designing the actual implementation of their services. The deployment and operation view discusses how to proceed with designing of IoT system from the service description and identification of different functional elements to the selection of technologies in building up the overall networking behaviour for the deployment. As shown in Figure 4.5, Devices, Resource and Services highlighted in red, blue and yellow, are considered as the main elements of the IoT Domain model which possess different deployment concern. The devices in IoT systems are ranging from the simplest of the radiofrequency tags to the most complex servers. Every device is smart even though with different processing capabilities and connected with one another forming part of IoT.

These devices must respect the functionality definitions of the functional model to be fully interoperable in IoT-A compliant system.

The computational complexity for a given device is selected based on the nature of the target application. However, the selection of connectivity types is not direct since different choices offer comparable advantages in different areas and associated with the extraction of communication protocols. Some of the typical technologies in IoT systems include sensor-actuator networks, RFID and smart tags, Wifi, cellular networks, etc.. Then the system designer has to describe where to deploy services and resources as defined in the IoT Service functional group section, i.e. on smart objects, gateways, in the clouds, etc.. This selection has to be made per type of resource and service and depending on the related device. It is also important for the designer to select where to store the information collected by the system, let the sensor networks or users gather the data. The storage choices should take into consideration the sensitiveness, the needed data availability and the degree of redundancy needed for data resiliency, some of the storage options include local only, web only and Local with web cache. Eventually, the designer should choose where to deploy virtual entities, there are two main options; (1) Internal deployment in which the core engine is installed on a server belonging to the system deployment and is dedicated to the target application or shared between different applications of the same provider. (2) External usage in which the core engine is provided by a third party and the system designer has to drive the service development on the third party APIs.

The IoT reference architecture is too abstract and encompasses a wide range of applications and technologies covering various domains such as healthcare, energy management, environmental monitoring, transportation, or any other innovative applications. Hence it can not be easily used as a blueprint for SEISs concrete architecture.

4.7 Industrial Internet Reference Architecture (IIRA)

The Industrial Internet Reference Architecture (IIRA) is a standards-based open architecture that guides the development, documentation and communication about Industrial Internet of Things (IIoT) systems [Lin et al., 2017]. The architecture description of IIRA and representations are generic and at a high level of abstraction for supporting broad industry applicability. The IIRA extracts and abstracts common characteristics, features and patterns from use cases defined in the Industrial Internet Consortium and elsewhere. The Industrial Internet Architecture Framework (IIAF) is at the foundation of the IIRA which adopts the general concepts and constructs in the ISO/IEC/IEEE 42010:2011 architecture specification,

specifically, concern, stakeholder and viewpoint as its architecture frame, and views and models as its architecture representation in describing and analyzing on important common architecture concerns for IIoT systems. The IIRA consists of four viewpoints; business, usage, functional and implementation.

The business viewpoint addresses the business-oriented concerns of business decision-makers, product managers and system engineers that are relating to business value, expected a return on investment, cost of maintenance and product liability in establishing an IIoT system in its business and regulatory context [Lin et al., 2017]. Some essential concepts and relationships between them have been introduced and defined to identify, evaluate and address the stated business concerns as well as mapping to fundamental system capabilities as shown in Figure 4.6. Such a description of the business viewpoint is expressed in terms of a vision and value-driven model. In which the relationship between stakeholders, vision, values, key objectives and fundamental capabilities. Stakeholders are identified as a major stake in the business who drive the conception and development of IIoT systems in an organization, from business decision makers to technical people. Such stakeholders usually develop and present an organization's vision which describes a future state of an organization or an industry. Values reflect how the vision may be perceived by the stakeholders who will fund the implementation of the new system and users of the resulting system. Key objectives are developed by senior business and technical leaders to establish the business outcomes expected of the resultant system in the context of delivering the values. From these key objectives, the stakeholders derive the fundamental capabilities that are required for the system to complete specific major business tasks.

The usage viewpoint addresses the concerns of system engineers, product managers and the other stakeholders including the individuals who are involved in the specification of the IIoT system. The usage viewpoint concerned with how an IIoT system realizes the key capabilities identified in the business viewpoint. The main concerns handled in this viewpoint pertain the expected system usage which is represented as sequences of activities involving human or logical (e.g. system or system components) users that deliver its intended functionality in ultimately achieving its fundamental system capabilities. Such activities describe how the system is used to serve as an input for system requirements, guide the design, implementation, deployment, operations and evolution of the IIoT system. The role, party, activity and task are identified as the usage viewpoint's main concepts. Figure 4.7 depicts the usage viewpoint's main concepts and how they relate to each other. The task is considered as the basic unit of work, i.e. invocation of operation, transfer of data or an action of a party. A task is carried out by a party assuming a role.

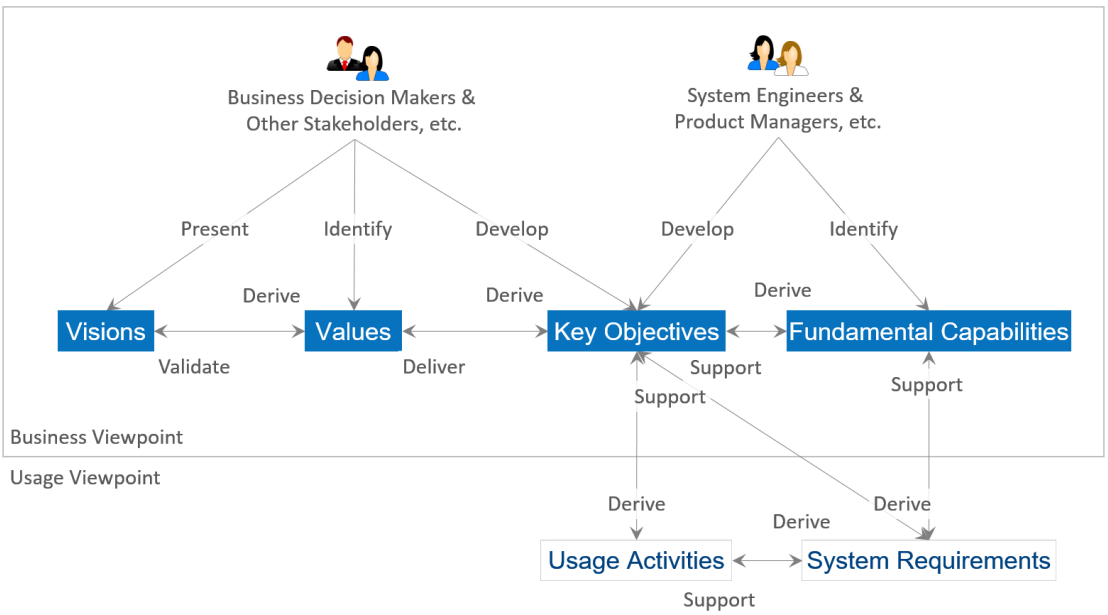


Figure 4.6: A Vision and Value-Driven Model figure taken from [Lin et al., 2017]

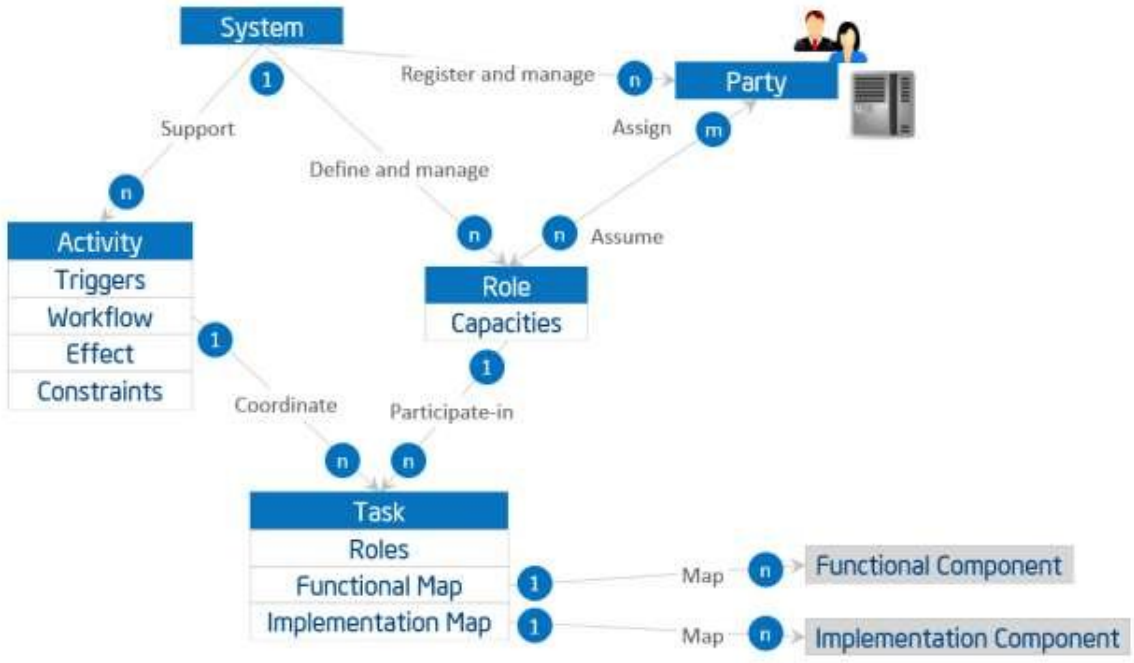


Figure 4.7: Usage Viewpoint's main concepts and how they relate to each other figure taken from [Lin et al., 2017]

The functional viewpoint handles the concerns of architects, developers and integrators pertaining the functional components in an IIoT system, their structure and interrelation,

the interfaces and interactions between them, and the relationships and interactions of the system with external elements in the environment, to support the usages and activities of the overall system. In the functional viewpoint, the functionalities of the IIoT system are decomposed in five functional domains; control, operations, information, application and business as shown in Figure 4.8. In which the data and control flow between functional domains are illustrated to demonstrate how these functional domains relate to each other. Green, red and other horizontal arrows show how data flows circulate across domains, across domains and processing taking place within each domain respectively. The control domain constitutes the collection of functions that are performed by industrial control systems, i.e. reading data from sensors, exercising control over the physical system through actuators, etc.. The operations domain encompasses set of functions responsible for the provisioning, management, monitoring and optimization of the systems in the control domain. The information domain consists of a set of functions for gathering data from various domains mainly the control domain, and transforming, persisting, and modeling or analyzing those data to acquire high-level intelligence about the overall system. The application domain constitutes the collection of functions for implementing application logic that realizes specific business functionalities. The business domain consists of set of functions that enable end-to-end operations of the IIoT systems by integrating them with traditional or new types of IIoT systems specific business functions including those supporting business processes and procedural activities, e.g. Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Product Lifecycle Management (PLM), etc..

The implementation viewpoint deals with the concerns of architects, developers, integrators and system operators pertaining to the technologies needed to implement functional components (functional viewpoint), their communication schemes and their life-cycle procedures. These elements are coordinated by activities from the usage viewpoint and supportive of the system capabilities including cost and go-to-market time constraints, relevant regulation, etc. from the business viewpoint. The implementation viewpoint describes the IIoT system structure and the distribution of components, and the topology by which they are interconnected; technical description of such IIoT system components, including interfaces, protocols, behaviors and other properties; an implementation map of the activities identified in the usage viewpoint to the functional components, and from functional components to the implementation components.

The IIRA covers a wide range of applications and technologies of various domains such as manufacturing, transportation, healthcare, environmental monitoring, etc.. Hence it can not be easily used as a blueprint for SEISs concrete architectures.

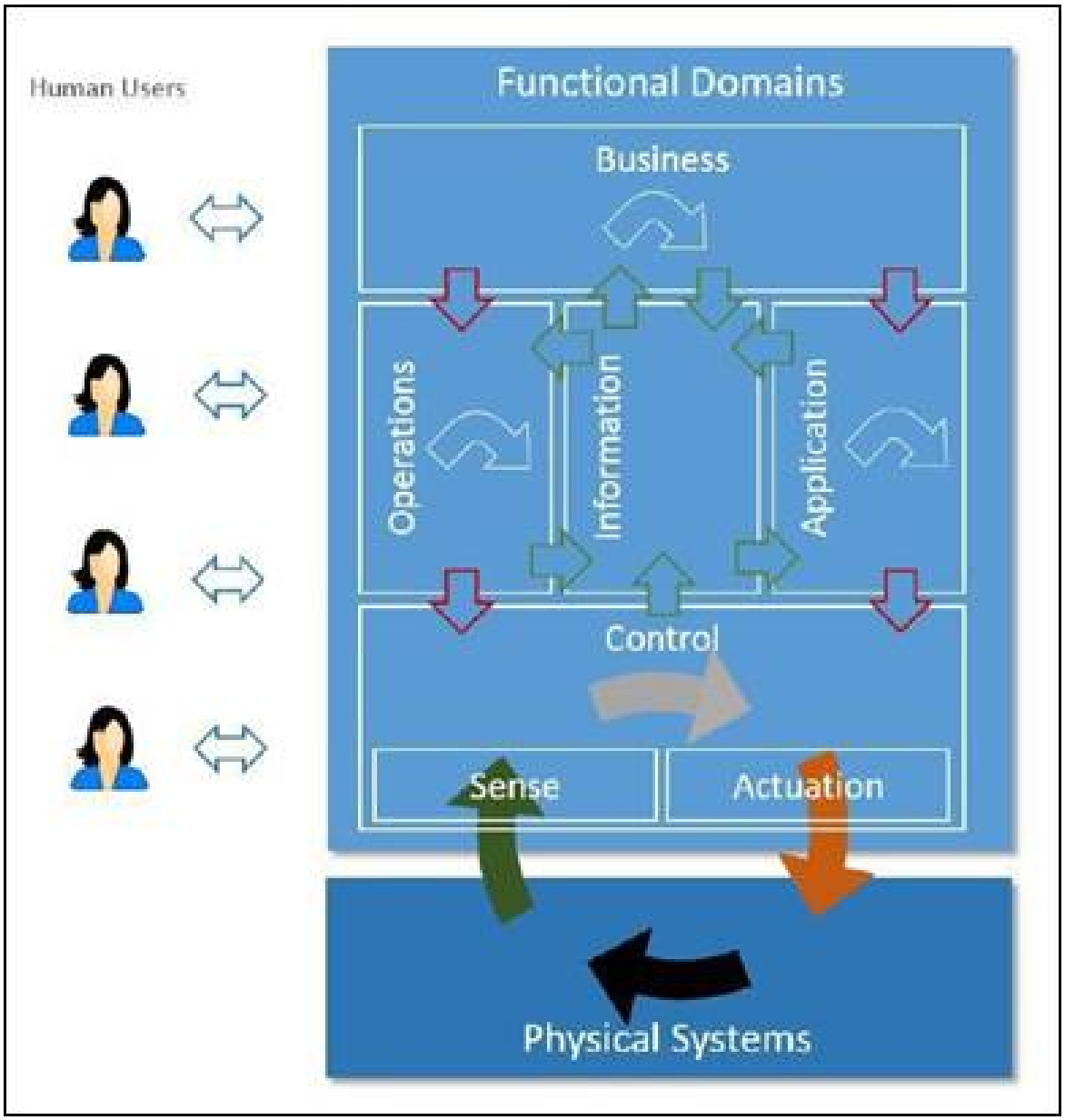


Figure 4.8: Functional Domains of Functional Viewpoint figure taken from [Lin et al., 2017]

4.8 Summary

This chapter proceeds with the related work which focuses on the existing reference architectures in the SEISs domain. Overall, most of the existing approaches attempt to propose reference architectures for facilitating the construction of new environmental monitoring, controlling and early warning systems. However, most of these approaches are; (i) focused in sub-domain applications rather than a whole domain, for example, the reference architecture for EWS focus on the flood, tsunami, and storms as described in Section 4.3. (ii) targeting

significant but specific needs of a particular subsystem, i.e. integration of heterogeneous sensor systems and data management as described by the reference architecture for SeNsIM in Section 4.2 and reference architecture for EMEWD in Section 4.5 respectively rather than dealing with the requirements of all the involved subsystems. (iii) expressed using a single architecture view rather than multi-views causing other essential perspectives of SEISs not to be covered. For instance, the reference architecture for SeNsIM used a module view, a reference architecture for EWS, reference architecture for DEWS and reference architecture for EMEWD used conceptual views. Multiple software architecture views are crucial in the description of SEISs as described in Chapter 3 due to the involvement of diverse set of stakeholders, i.e. users, developers, testers, maintainers, etc. with various concerns. These stakeholders need to understand and use the software architecture description from their respective views. Despite the use of multiple views in IoT reference architecture and IIRA as described in Section 4.6 and Section 4.7 respectively, these reference architectures are too abstract and encompass a wide range of applications and technologies covering various domains which make it too difficult to be used in the construction of concrete SEISs.

In contrast to the reviewed approaches, a reference architecture *covering the SEISs domain as a whole* is highly required to facilitate the rapid development and maintenance of SEISs. Nevertheless, it is clear that a reference architecture for SEISs must take into account existing initiatives and emerging standards for effective environmental monitoring and controlling. This research is supposed to have a strong influence on the elaboration of a useful and practical reference architecture with substantial impacts in both functional and non-functional requirements of SEISs. As a result, this thesis defines a set of viewpoints that is required in describing the essential aspects of SEISs in Chapter 6, proposes a reference architecture for SEISs and presents its description in Part III and demonstrates the applicability of such reference architecture in creation of new concrete SEISs i.e. forest fire detection system in Chapter 8 and mapping of the existing SEISs in Chapter 9.

Part III

Approach: Reference Architecture for SEISs (RefSEISs)

This part presents the core ideas behind this thesis, i.e. the development of the viewpoints and reference architecture for SEISs. First, the set of viewpoints that is required to describe essential aspects of SEISs are defined in Chapter 6. Then the established set of viewpoint is applied in the construction of the RefSEISs. The description of the RefSEISs follows ISO/IEC/IEEE 42010:2011 [ISO/IEC/IEEE, 2011] which starts with an identification of stakeholders and their concerns in Chapter 5. Followed by the derivation of requirements of the RefSEISs. Then the architecture views of the RefSEISs are created using the established set of viewpoints and their corresponding architecture models while encompassing best practices of the existing SEISs (design decisions) in Chapter 7.

Chapter 5

RefSEISs Requirements Establishment

The literature has revealed that a common understanding of how to design SEISs is therefore required to address the problem described in Section 1.1. This is achieved through the development of a *reference architecture for SEISs (RefSEISs)*. As described in Section 2.2, the use of reference architecture allows systematic reuse of knowledge and elements when developing concrete software architectures in a particular domain. Therefore a reference architecture for SEISs would facilitate the development and maintenance of such systems. According to Definition 2.5, reference architectures are regarded as abstract architectures that encompass experiences and knowledge in a given application domain to facilitate and guide the development, evolution and interoperability of software systems in such domain. This definition has been introduced in Chapter 2 and will be used throughout this work.

The architectural description of the RefSEISs contains information about the fundamental architectural constructs which specifies stakeholders, concerns, viewpoints, views, corresponding rules and conditions of applicability. Architects can reuse such prescribed architecture description to identify stakeholders and concerns of the system-of-interests and can further employ the specified architectural representations to clarify, analyze and frame up the identified concerns. Chapter 2 introduced concepts of software architecture from a forward engineering approach while adopting ISO/IEC/IEEE 42010:2011 Standard- Recommended Practice for Architectural Description of Software-Intensive Systems [ISO/IEC/IEEE, 2011]. The ISO/IEC/IEEE 42010 standard is adopted to prescribes the crucial steps in designing software architecture as explained in Section 2.3.1. In the beginning, stakeholders and their concerns should be identified. Hence this chapter demonstrates the establishment of the requirements of RefSEISs which includes the identification of both stakeholders, their concerns and architectural requirements. Then viewpoints are defined to address the identified concerns of stakeholders in Chapter 6. Afterwards, the reference architecture views that conform to the defined viewpoints will be described in Chapter 7.

5.1 Stakeholders and their concerns

Stakeholders and their concerns are essential elements in the description of software architectures as described in Section 2.3.1. According to Definition 2.6, stakeholders are considered as individuals, team or organization with interests in or concerns relative to a system. Stakeholders are key players of the successful SEISs and possess a strong influence in the evolution of such systems. These stakeholders include both technical and non-technical people who drive the conception and development of SEISs. Each stakeholder has interests and concerns that need to be addressed. Multiple stakeholders are involved in SEISs. With a Definition 2.7, a concern involves any influence on a system in its environment including developmental, technological, business, operational, organizational, political, regulatory, or social influences [ISO/IEC/IEEE, 2011]. Hence the concerns of stakeholders should cover all the essential things that an architect must consider while envisioning the system. This section elaborates on various stakeholders, their roles and concerns in SEISs. The method used in such identification is purely based on literature reviews, i.e. SEISs document analysis as described in Chapter 3.

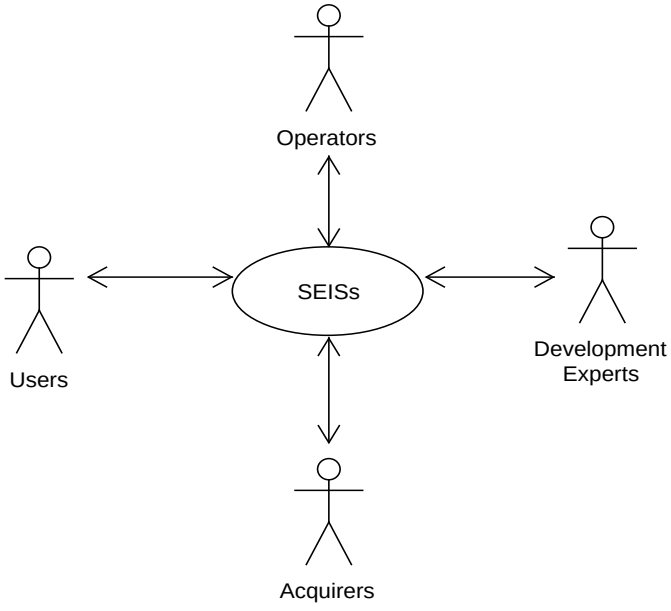


Figure 5.1: Stakeholders of SEISs

Following the extensive literature reviews on concrete SEISs in Chapter 3, four classes of stakeholders of SEISs are derived as illustrated in Figure 5.1. This framework of stakeholders consists of one central circle representing the domain (SEISs) which is surrounded by some

actors, and each actor represents a group of stakeholders, i.e. system acquirers, maintainers, users, and development experts.

Acquirers are the owners or proprietor of SEISs. The SEISs may be provided by various groups of stakeholders: the government agencies, academic and research institutes and Non-Governmental Organisations (NGOs). Some examples of acquirers in the context of SEISs include wild-land fire researchers, government and national authorities as described in Section 3.2.1, regional planning agency, local disaster management agency, environmental offices, a regional parliament and health department as presented in Section 3.3.1. Protection of lives of the people, safeguarding properties and the environment are critical roles of the acquirers mainly the government. This could be achieved through environmental monitoring systems mainly SEISs which monitor the likelihood of environmental phenomena or calamity and plan for mitigation procedures to minimize the impacts of those environmental phenomena. The academics and research institutes include all people who are involved in training people and finding new solutions in preventing or reducing consequences that could be imposed by the occurrence of various environmental phenomena.

These stakeholders initiate, develop, manage and maintain the whole SEISs projects. Therefore they are concerns with constructing discussions and managing agreements with other stakeholders, allocating and coordinating resources, i.e. materials, finances etc. throughout the life cycle of the project with the intent of achieving specified objectives such as time, costs and other requirements. They initiate the system by specifying the needs of the system to be realized by an architecture. These stakeholders concern with tracing the requirements, system development benefits, risks, schedule, and budget. They need to assess the effectiveness of the capabilities of the solution. They are interested to know:

- What is the general idea on the functionality of a SEIS? Acquirers would like to have a big picture of how everything fit together in providing the functionalities of SEISs.
- Is the proposed SEIS economic, operational and technically feasible? These stakeholders would like to know how practicable to build the SEISs in-terms of building and operational costs as well as technical. They are interested to assess the costs and benefits.
- Are the functional and non-functional requirements of SEIS fulfilled? After having a big picture of the system, these stakeholders would like to know how these functional and non-functional requirements are fulfilled.

Development Experts refer to *technical people* from both private companies, individuals or academic research staff responsible for designing, coding, integrating, testing and maintaining SEISs mainly developers, testers, integrators, system engineers, maintainers, and architects as described in Section 3.2.1, Section 3.3.1, Section 3.4.1, Section 3.5.1 and Section 3.6.1. These are experts that could design new services of the system, improve or customize the system based on the requirements of the organization. Therefore it is crucial for them to understand the internal structure of the system to realize components of the systems and their relations as well as identifying parts which could be re-used.

These stakeholders are responsible for the *actual development and maintenance* of SEISs. They are concerns with data modeling and access mechanisms, development, code management and organization, effective topology configurations of field nodes, installation of the system on site, diagnosis of faults and failures, replacement or addition of system elements etc. Concerns that development experts desire to know include:

- How is the SEIS technically realized? Developments experts need to know how the SEIS can be realized, i.e. from the collection of inputs from stakeholders, clarifications of those inputs, conversion of those inputs to technical requirements to be fulfilled by the system.
- How is the SEIS is decomposed? Development experts would like to know how the system is decomposed. Since the system is supposed to be decomposed into software modules or elements to reduce complexity before it is realized.
- What are the components of the systems and their relations? Development experts would like to know the main components of SEISs and their relations which are responsible for fulfilling the specified requirements of SEISs.
- How are the identified components of the SEIS distributed among processing nodes at runtime? After the identification of the main components of the SEIS. The development experts would like to know how the identified components of the SEIS are distributed among processing nodes at runtime. This is important for daily operations and maintenance of the system.
- How are the SEIS software components, distributed in different processing nodes, and do they interact at runtime? It is also important for developing experts to know how the identified SEISs software components that are distributed in different processing nodes communicate or interact with each other.

- How to distribute the nodes within the specified field to achieve the best range and coverage from the sensors? To achieve the desired objective of the SEIS, it is important for the development experts to know how the nodes of SEIS are supposed to be distributed to achieve the best range and coverage.
- How is the SEIS integrated? Development experts are interested in the SEIS integration for improving productivity and efficiency by ensuring subsystems function together as a system.
- How are the functionalities of the SEIS fulfilled? Development experts are interested to know how the functionalities of SEIS are technically realized.
- How is data created, accessed, updated and stored in the system? Development experts are interested to know how data is created, accessed, updated and stored in the system in order to ensure effective management, maintenance and processing of data stored in databases.
- How easy to change the SEIS to correct faults, improve performance, or other attributes, or adapt to a changing environment? for effective maintenance of the SEIS, the development experts would like to know how easy to change the SEIS to correct faults, improve performance, or other attributes or adapt to a changing environment.
- How can the system can be optimized to meet the specified requirements effectively? Development experts are interested in how the specified requirements can be optimized for successful implementation of the SEIS functionality in a cost-effective manner.
- How are the non-functional requirements supported? Development experts would like to know how the crucial non-functional requirements such as maintainability, energy efficiency and interoperability are fulfilled.

Operators include individuals or organizations which are responsible for providing service or product using skills, procedures and knowledge in performing functions of the system. The operators possess different roles in supporting the daily activities of SEISs (SEISs during the operational time) from primary operators of the system, i.e. the field operators, and administrators as described in Section 3.2.1, Section 3.3.1, Section 3.4.1, Section 3.5.1 and Section 3.6.1.

The operators are interested in managing and ensuring the fulfilment of the services offered by SEISs to discover, prevent and reduce the consequences of certain environmental phenomena. Hence they are concerns with the daily operations of the

system, diagnosis of faults and failures, replacement or addition of system elements with minimal impacts on the operations of SEISs. These stakeholders are interested to know:

- How is the SEIS decomposed? In order to manage SEISs, Operators would like to know how the system is decomposed.
- What are the components of the systems and their relations? Operators would like to know the main components of SEISs and their relations that fulfils the specified requirements of SEISs.
- How are the identified components of the SEIS distributed among processing nodes at runtime? After the identification of the main components of the SEIS. The operators would like to know how the identified components of the SEIS are distributed among processing nodes at runtime or implemented. This is important for daily operations and maintenance of the system.
- How are the SEIS software components, distributed in different processing nodes, and do they interact at runtime? It is also important for operators to know how the identified SEISs software components that are distributed in different processing nodes communicate or interact with each other.
- How is the SEIS integrated? Operators are interested in the SEIS integration for improving productivity and efficiency by ensuring subsystems function together as a system.
- How are the functionalities of the SEIS fulfilled? Operators are interested to know how the functionalities of SEIS are technically realized.
- How is data created, accessed, updated and stored in the system? Operators are interested to know how data is created, accessed, updated and stored in the system in order to ensure effective management, maintenance and processing of data stored in databases.
- How easy to change the SEIS to correct faults, improve performance, or other attributes, or adapt to a changing environment? for effective maintenance and optimization of the SEIS, the operators would like to know how easy to change the SEIS to correct faults, improve performance, or other attributes or adapt to a changing environment.
- How are the non-functional requirements supported? Operators would like to know how the crucial non-functional requirements such as maintainability, energy efficiency and interoperability are fulfilled.

Users of the SEISs refer to individuals or organizations which possess different roles of the system, e.g. the organization's staff and also citizens (people) who require timely and high-quality environmental information to be empowered through the knowledge of environmental conditions to act upon that knowledge. This information needs to be understandable and easy-to-access [Peinel et al., 2000]. Some examples of users in the context of SEISs include police as described in Section 3.3.1 and Section 3.5.1, residents as presented in Section 3.2.1, Section 3.3.1, Section 3.4.1 and Section 3.6.1.

The users are interested in the functionalities such as (1) Accessing sensors measurements and their manipulation based on demands. (2) Viewing Maps for locations. (3) Estimation and visualization of environmental phenomena propagation e.g. estimation of fire, air pollution, landslides, etc. propagation. (4) Getting visual and acoustic alarms in case of environmental phenomena. End users are usually interested with easy-access, error-less data processing and intuitive interactions of software [Schmidt, 2013]. Since the SEIS is intended to be used as an assistive tool for these stakeholders, hence SEIS is supposed to be reliable, and user-friendly to achieve the maximum benefit of these systems. Hence, users desire to know:

- What is the general idea on the functionality of a SEIS? Users would like to know how everything fit together in SEISs while providing their functionalities.
- Is the proposed SEIS easy to use, is it easy for users to find their way through different parts of SEISs.

Summary of the concerns

More specifically, this thesis addresses the following concerns which have been summarized from the previous section;

- **C1:** What is the general idea on the functionality of a SEIS?
- **C2:** Is the proposed SEIS economic, operational and technically feasible?
- **C3:** How is the SEIS technically realized?
- **C4:** How are the functionalities of SEIS fulfilled?
- **C5:** What are the components of SEISs and their relations?
- **C6:** How is the SEIS decomposed?

- **C7:** How to distribute the nodes within the specified field to achieve the best range and coverage from the sensors?
- **C8:** How is SEIS integrated?
- **C9:** How is data created, accessed, updated and stored in the SEIS?
- **C10:** What are communication mechanisms and protocols of SEISs?
- **C11:** How are the identified components of SEISs distributed among processing nodes at runtime?
- **C12:** How are the SEIS software components, distributed in different processing nodes, and do they interact at runtime?
- **C13:** How are the identified components of SEISs implemented?
- **C14:** How are the non-functional requirements supported?

A careful examination of these concerns of stakeholders indicates that some identified concerns are possessed by multiple stakeholders, for example; both acquirers and users have a concern on (C1) What is the general idea of functionality of SEIS. Also, these concerns tend to be intertwining and covering the full development life-cycle of the SEISs for instance; (C1) What is the general idea of functionality of SEIS? belongs to SEIS planning, (C2) Is the proposed SEIS economic, operational and technically feasible? belongs to analysis, (C3) How are the functionalities of SEIS fulfilled? belongs to design, (C10) How the identified components of SEISs are implemented? belongs to implementation, (C7) How is SEIS integrated? belongs to testing and integration. This requires various architectural viewpoints to facilitate systematic fulfilment and evaluation of those concerns as defined in Chapter 6.

5.2 RefSEISs Requirements Analysis

This section describes functional and non-functional requirements that should be fulfilled by the RefSEISs. Such requirements are derived from both the existing SEISs as described in Chapter 3 and general requirements of reference architectures as presented in Section 2.2.2.

5.2.1 Functional Requirements

The reference architecture functional requirements refer to the set of requirements of a reference architecture that describes the *common functionalities* of a system domain [Nakagawa

et al., 2014]. The extraction of SEISs from the literature in Chapter 3 is associated with the identification of functional requirements. Such functional requirements are deduced from Section 3.2.2, Section 3.3.2, Section 3.4.2, Section 3.5.2, and Section 3.6.2. Each architectural requirement of a reference architecture could be instantiated into *one or more system requirements* during the design of concrete architectures from the reference architecture [Gamma et al., 1994]. The extraction of SEISs from the literature in Chapter 3 is associated with the identification of functional requirements. Such requirements include:

FN1 Data Acquisition: The reference architecture must allow the development of SEIS that can acquire data (environmental parameters) in both non-real-time and real-time from sensors spread over the field of interest. Such environmental parameters include temperature, light, and humidity as described in [Sha et al., 2006], water conditions [Sunkpho and Ootamakorn, 2011], dust concentrations, and gases [Ustad et al., 2014], soil pore water pressure, vibrations of the earth, and soil movement [Nguyen et al., 2015], vehicle density [George et al., 2017], etc.

FN2 Storage: The reference architecture must support the development of SEISs that able to store the collected data for further processing such as data storage and detector store as presented in [Castillo-Effen et al., 2004, Zheng et al., 2016] and [Ranjini et al., 2011] respectively.

FN3 Phenomena Analysis: The reference architecture must allow the development of SEIS that can establish analysis models i.e. Canadian Forest Fire Weather Index (FWI) model [Canada, 2018] on the processed information to gain an understanding of the environmental phenomena of interest, predict the possible occurrence of such environmental phenomena such as forest fires [Hartung et al., 2006], air pollution [Zheng et al., 2016], flood detection [Chowdhury, 2005], etc. and issue an alert in case of high probability of occurrence of such environmental phenomena.

FN4 Visualization: The reference architecture must enable the development of SEIS that is capable of visualizing collected environmental parameters, predictions and propagation of environmental phenomena, e.g. displaying propagation of fires [Hefeeda and Bagheri, 2009], landslides [Ramesh, 2014], flood forecasting [Chowdhury, 2005], etc.

FN5 Phenomena Control: The reference architecture should support the development of SEIS that can manipulate environmental parameters to control environmental phenomena in the field of interest. This includes the use of adaptive traffic control system which considers various traffic factors, i.e. traffic volume, vehicle density and waiting time to adjust both the sequence and length of traffic lights based on the collected

traffic information in real time traffic lights in controlling traffic flow [Zhou et al., 2010, 2011].

FN6 Sensor Management: The reference architecture should enable the development of SEIS that can coordinate, monitor and control sensors spread over the field of interest.

FN7 Actuator Management: The reference architecture should enable the development of SEIS that can coordinate, monitor and control actuators spread over the field of interest.

5.2.2 Non-Functional Requirements

Apart from functional requirements, the design of the reference architecture is primarily driven by important non-functional requirements. Non-functional requirements are sometimes referred to quality attributes, as measurable or testable properties of a system which indicate how well the system satisfies the concerns of its stakeholders [Bass et al., 2012]. Non-functional requirements judge the operation of a system, rather than specific behaviours. Architectures provide the foundation for the achievement of the qualities of a system rather than achieving the qualities by themselves. Hence, non-functional requirements need to be explicitly considered throughout the system development life-cycle starting from the designing phase. In this thesis, the non-functional requirements of RefSEISs incorporate the requirements in two-levels; domain and reference architecture levels.

Non-Functional Requirements of RefSEISs on the domain level

At the domain level, the most recurring non-functional requirements of SEISs as deduced from Section 3.2.4, Section 3.3.4, Section 3.4.4, Section 3.5.4, and Section 3.6.4 include energy efficiency, interoperability [Degrossi et al., 2013, Samadzadegan et al., 2013, Kung et al., 2006, Asensio et al., 2015] and maintainability [Kumar and Kishore, 2017, Samadzadegan et al., 2013, Hass et al., 2009, George et al., 2017]. These requirements are considered as paramount, essential non-functional requirements which need special attention to be considered explicitly during the designing phase of a SEIS. Therefore the proposed RefSEISs should be able to support such requirements as described below.

NFN1 Energy Efficiency

SEISs consists of sensor nodes with scarce resources (bandwidth, energy source, sensing range, and processing capability) [Mekkaoui and Rahmoun, 2011, Aslan et al., 2012, Marin Perez et al., 2012, Khedo et al., 2010, Nguyen et al., 2015]. The

SEIS is supposed to ensure that the intended objectives of the system are fulfilled regardless of possession of sensors nodes with limited resources, i.e. battery. And the deployment area is usually isolated, remotely and very large with thousands of sensor nodes hence the recharge and replacement of batteries is infeasible, costly and impractical. Therefore the SEIS should consume energy very efficiently.

The development of energy efficient SEISs is among the critical challenges for the researchers in this domain. The involved battery-powered sensors consume power during sensing, processing, communication and in an idle state. It is required to check the energy level of sensor nodes of SEISs mainly during the designing phase before the deployment to avoid incorrect sensor readings by predicting in advance the energy depletion rate of each sensor nodes [Tolle et al., 2005]. This will reduce the earlier failure of the SEISs. The energy consumption of SEISs should be balanced on sensor nodes to maximize the lifetime of sensor networks. Since a large amount of energy is consumed during the communication [Mekkaoui and Rahmoun, 2011], then the employment of suitable routing communication mechanisms should be considered throughout the designing phase to prevent or reduce energy consumption. Therefore the reference architecture must enable the development of energy efficient SEIS.

NFN2 Maintainability

SEISs are required to monitor and control the environmental phenomena autonomously. The operation of SEIS should be designed in such a way that it does do not provide an additional burden of human involvement especially in extreme conditions of remote or inaccessible sites, i,e, forests [Aslan et al., 2012, Liyang et al., 2005], water bodies [Marin Perez et al., 2012], soil [Nguyen et al., 2015], etc. In this context, maintainability is one of the crucial design challenges of SEISs to ensure the long-term success of the software system. Maintainability refers to the ability of the system to accommodate new or change requirements with a degree of ease. In general, *maintainability is the capability of the software product to be modified* [ISO/IEC, 2001]. This includes the addition or manipulation of functionalities, fixation of errors and fulfilment of new raised requirements to meet the demands of the business.

Since it is found that the maintenance phase consumes a large part of a system costs such as between 50 to 80 per cent of the system total costs [Lientz and Swanson, 1980]. Therefore the reference architecture for SEISs must enable the development of SEISs that is easily maintainable SEIS. Such that the resulted SEIS possess the ability to be easily configurable, add, repair and remove both functionalities and (/or) other architectural elements such as new stations, more sensors, actuators or adding mobile

phone numbers to deliver alerts with the purpose of fixing the problem, improvement of system performance or adaption of the needs of various applications.

NFN3 Interoperability

The increased availability and robustness of sensors, the wide-spread use of the internet as a communication environment, as well as intensified research in the area SEISs led to interoperability challenges. *Interoperability is the ability of two or more systems or components to exchange data and use information* [van der Veer and Wiles, 2008]. Since SEISs involve a collection of heterogeneous sensors which operate on different, proprietary hardware platforms and contain multifaceted types of sensor nodes to capture environmental phenomena which cause interoperability problems [Marin Perez et al., 2012, Kung et al., 2006, Asensio et al., 2015]. Such heterogeneity of sensor nodes is demonstrated into two different facets in terms of both; heterogeneous data and network. The heterogeneous networks are demonstrated when multiple sensors with different protocols form a network. This makes communication between those sensors to be difficult. The heterogeneous data resulted from the involvement of a wide range of parameters originated from various sensors required to estimate or creating a particular dataset. As a result, the integration of multiple sensors into the sensor networks becomes difficult. Also, most systems tend to incorporate the sensor resources through proprietary mechanisms rather than developing a system upon a well-defined and established databases. The interoperability does not guarantee by the type of architecture; instead, the architecture helps to achieve it. This is accomplished through the utilization of suitable design choices. Therefore the reference architecture should enable the development of interoperable SEIS, that supports the easy and seamless integration of data as well as allowing effective communication between heterogeneous sensor nodes.

Non-Functional Requirements of RefSEISs on the reference architecture level

In [Bass et al., 2003, Reidt et al., 2018], a number of general requirements to be considered by the reference architectures are presented as described in Section 2.2.2. Inspire by them, and after thorough investigations of reference architectures, the most critical requirements found were generality, completeness, and applicability, as further described below:

NFN4 Generality

Reference architectures describe generic constructs and provide representations for similar systems that belong to a particular domain. Generality is deduced from abstraction and independence quality criteria of reference architecture as described in

Section 2.2.2. Such that the involvement of more details of reference architecture description the less the use of such reference architecture in the domain since it limits its generality. The involvement of very few abstract parts reduces the benefits of reusing such reference architecture in concrete models. Therefore the chosen level of abstraction is important in promoting the generality of the reference architecture by facilitating the re-use of knowledge in the constructions of concrete architectures [Hars, 1994].

Since the domain consists of various systems that incorporate multiple technologies, then, a generic reference architecture is supposed to be independent by not being tied to any specific standards, technologies or implementation details to represent common abstractions of the whole domain. With higher abstractions, there will be no implementation or technology restrictions [Reidt et al., 2018]. Reference architecture of a particular domain is considered to be generic if it can be applied in the defining any concrete system of that domain. Practically, this is achieved by developing an abstract RefSEISs that is independent of underlying technologies, standards and implementation details. Such that the RefSEISs is supposed to encapsulate the entire domain in a technology independent way by merging all the domain best practices, standards and technologies.

NFN5 Completeness

The completeness of models is achieved if the model covers all the relevant aspects of the system [Winter, 2000]. Since the reference architectures are supposed to general, then completeness is not always possible. Preferably, the reference architectures are expected to cover the essential aspects of the systems within a specific domain that advocates the use of context, goal and design dimensions of a reference architecture documentation as described in [Angelov et al., 2012].

NFN6 Applicability

The applicability of models is reflected through how easy is the application of reference architectures. Reference architectures should be applicable to the systems within a specific domain (universal applicability) rather than only a particular concrete system [Pajk et al., 2012]. The reference architectures can be applied for the design of new concrete SEISs as well as analysis of the existing systems. This implies that the RefSEISs is supposed to be easily understandable by both business and other experts or stakeholders to enable the architects to use RefSEISs as a baseline towards the construction of new concrete and existing SEISs architectures. The applicability of reference architectures can be proven by at least two applications [Gamma et al., 1994].

5.3 Summary

The development of RefSEISs commences with the elicitation of requirements from the information sources of the existing SEISs described in Part II (Foundations) of the thesis which provide the essential inputs to the identification of stakeholders, their concerns and requirements of SEISs. Therefore, this chapter presented the stakeholders, their concerns and requirements of SEISs. In which common characteristics and functionalities that should be possessed by a proposed reference architecture are identified. And the non-functional requirements have been introduced that will be used in the analysis of the RefSEISs. Then chapter 6 proceeds with the description of architecture viewpoints for the design of RefSEISs with a strong emphasis on the granularity and separation of concerns to frame up the identified concerns.

Chapter 6

Architectural Viewpoints

At the core of the ISO/IEC/IEEE 42010:2011 standard (Recommended Practice for Architectural Description of Software-Intensive Systems) [ISO/IEC/IEEE, 2011] are viewpoints. Following the Definition 2.9 and description of the viewpoints provided in Section 2.3.1, viewpoints specify the conventions that enable the creation, illustration, and analysis of views. Viewpoints are considered as metamodels representing the basic concepts of the system architecture from a particular perspective. One viewpoint addresses one or more architectural concerns. Hence the analysis of the existing SEISs in Chapter 3, and identification of the relevant stakeholders of SEISs and their concerns in Chapter 5 led to the selection of essential viewpoints required in the description of SEISs which will be presented in this chapter.

The specification of SEISs is structured around the concept of architecture viewpoints of Siemens view model as described in Section 2.3.2 with an addition of two other viewpoints: topology and data viewpoints. Therefore in this thesis, six viewpoints are proposed to cover essential aspects or perspectives of SEISs as will be described in their corresponding sections: conceptual viewpoint described in Section 6.1, module viewpoint in Section 6.2, execution viewpoint in Section 6.3, code viewpoint in Section 6.4, topology viewpoint in Section 6.5, and data viewpoint in Section 6.6 as proposed in [Kateule and Winter, 2016]. The integration of SEISs viewpoints is described in Section 6.7 for ensuring consistency between the proposed viewpoints. The proposed viewpoints are selected and specified based on the separation of concerns, understandability, and principle of granularity. The description of these viewpoints is associated with the structure of architectural viewpoint and conforming notations used in the construction of the view models. Each viewpoint provides the constructs for describing different views of the architecture from these viewpoints. This chapter ends with the summary in Section 6.8

6.1 Conceptual Viewpoint

The conceptual viewpoint presents the architecture design that describes the system realization of main functionalities and their dependencies at a high-level of abstraction in terms of component and connectors. The system functionality represents a service of the system which is provided to either another function of the system or external entity. The conceptual viewpoint is considered as a fundamental viewpoint of software architectures since it serves as an input for other viewpoints which at the end facilitate the understanding of the system as the whole. This viewpoint addresses the following concerns;

- **C1:** What is the general idea behind the SEIS?
- **C2:** Is the proposed SEIS economic, operational and technically feasible?
- **C4:** How are the functionalities of SEISs fulfilled?
- **C5:** What are the components of SEISs and their relation?
- **C6:** How is SEIS decomposed?
- **C14:** How are the non-functional requirements supported?

The stakeholders of these concerns typically consist of acquirers, users, operators and development experts as described in Section 5.1.

6.1.1 Structure

To address the stated concerns, a conceptual viewpoint is needed as demonstrated in the existing SEISs [Ranjini et al., 2011, Zhang et al., 2008, of Arizona et al., 2013]. The proposed conceptual viewpoint of SEISs is based on application domain components necessary to implement the functional requirements of SEISs. The Siemens four view model well-known abstractions of components, connectors and configuration of conceptual viewpoint are adapted [Hofmeister et al., 2000] to address the identified concerns. The conceptual viewpoint is presented in Figure 6.1 which describe essential concepts and their relations required in the description of conceptual views;

- *Conceptual Components:* A *CComponent* represents a functionality of the system, i.e. computation unit or storage unit. The SEIS is decomposed into a set of *CComponents* which reflect the functionalities of SEISs. *CComponents* consist of *CPorts* which define how the component can interact with other elements of the SEISs. When the

component is further decomposed its ports can be bound directly to the port of the resulted components.

- *Conceptual Connectors:* A *CConnector* facilitates the interactions between *CComponents* of the system by specifying the models and rules, e.g. method invocation, pipes and filter, client-server protocols, etc.. The *CConnectors* contain *CRoles* for interaction, and both a port and role obey a specified protocol. The *CConnector* can contain multiple *CRoles*. Both port and role obey a particular *CProtocol* which defines how the incoming and outgoing operations can be ordered.

- *Conceptual Configuration:* A *Configuration* refers to an arrangement of elements and relations of conceptual view that describe the SEISs. The SEISs can be decomposed to a certain extent to fulfil various non-functional requirements.

The conceptual viewpoint facilitates the construction of the conceptual view which provides an overview of the software architecture of SEISs. It is actually the first design which stakeholders would get a big picture of how everything fits together and hence find out how the system will do what it is supposed to do. By doing so, the acquirers will be able to get a general idea behind the SEISs and assess the feasibility of the system economically, operationally and technically. The conceptual viewpoint is adapted from the Siemens four view model [Hofmeister et al., 2000] with some extensions to suit the need of SEISs. For instance, the *Container* is added as a superclass that encloses the resulted conceptual view or configuration by showing the beginning and end of the design.

6.1.2 Notations

The constructed conceptual views shall use the UML profile diagrams for describing the static configurations of the system, state diagrams (or sequence diagrams) for specifying the functional behaviour of SEIS and sequence diagrams for showing a particular sequence of interactions among a group of components. Conceptual views can also be detailed using informal non-technical language that can be easily understood by all stakeholders. The conceptual viewpoint is used for the construction of conceptual views as demonstrated in Section 7.2.1

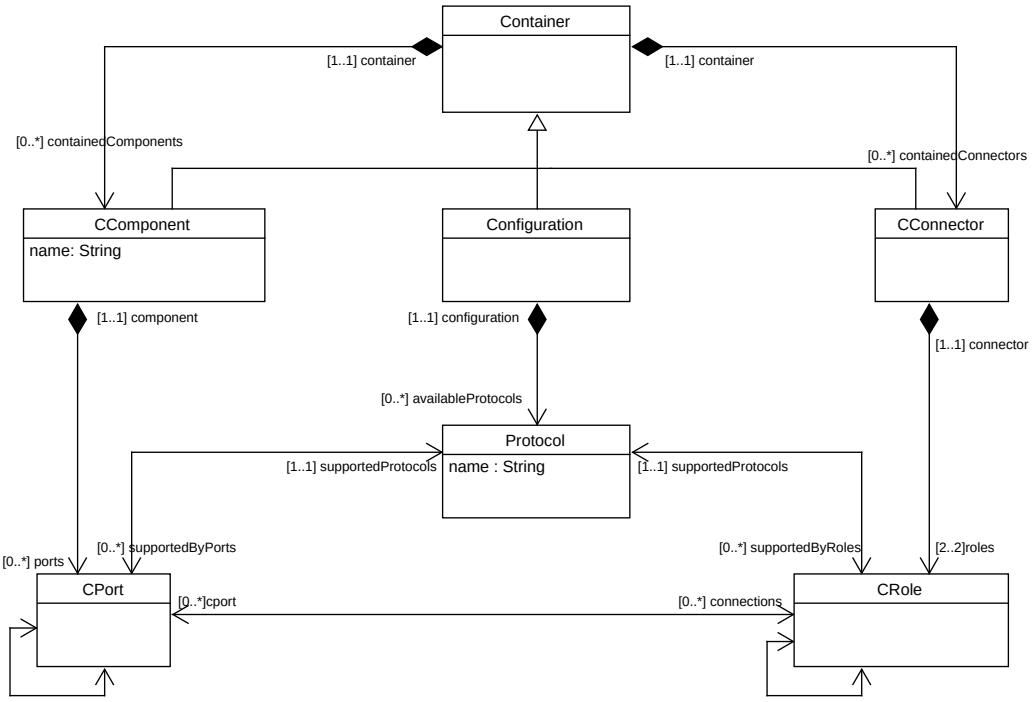


Figure 6.1: Conceptual Viewpoint

6.2 Module Viewpoint

The module viewpoint focuses on describing the software implementation modules in SEISs, their interfaces and interactions in attaining the intended objective of the system. This viewpoint addresses the following concerns;

- **C3:** How is the SEIS technically realized?
- **C4:** How are the functionalities of SEISs fulfilled?
- **C6:** How is the SEIS decomposed?
- **C14:** How are the non-functional requirements supported?

These concerns are of particular interest to system operators and development experts mainly developers, maintainers, architects, and integrators as described in Section 5.1.

6.2.1 Structure

To address these concerns, a module viewpoint is needed as demonstrated by the existing SEISs [Jadhav and Deshmukh, 2012, Hartung et al., 2006, Stipanicev et al., 2018, Sunkpho

and Ootamakorn, 2011, Amirebrahimi et al., 2016, Udo and Isong, 2014, Seal et al., 2012, Lozano et al., 2012, Raju, 2014, Guthi, 2007, Nguyen et al., 2015, Kotta et al., 2011, Huang et al., 2015, Curiac and Volosencu, 2010, Zhou et al., 2010, Mirchandani and Head, 2001]. The proposed module viewpoint of SEISs is depicted in Figure 6.2, in which implementation elements and their relationship required for the construction of module views of SEISs are illustrated. Such elements include;

- *Modules:* The module view should show how the elements of the software are mapped to the implementation modules. The modules are units of software implementation that encapsulate data and operations to fulfil a coherent set of system functionalities. Modules are organized into two orthogonal structures, i.e. decomposition and layers. The decomposition structure demonstrates how the system is decomposed into subsystems and modules, while the layering structure demonstrates how the modules are assigned to layers which are then constrained their dependencies to other modules.

- *Relationship between modules:* Modules interact with each other through the required and provided *Interfaces*, though these interfaces possess no associated implementation. The modules can also be decomposed in other modules, and are organized in terms of subsystems and layers since most of the existing SEISs utilized a layered architectural style [Sunkpho and Ootamakorn, 2011, Udo and Isong, 2014, Nguyen et al., 2015, Lozano et al., 2012]. Subsystems correspond to the conceptual components of higher level (one that is decomposed in other components and connectors), they can contain other subsystems or modules. Each subsystem consists of various modules which interact with each other through module use-dependency. While in a layering view, modules are organized into a partially ordered hierarchy and they use interfaces to declare their services. The interaction of modules across different layers is facilitated by interfaces (both required and provided interfaces). Furthermore, modules in the specific layer have the same abstraction hence they should depend on other modules in the same layer and can communicate with modules in the layer directly underneath through interfaces.

To facilitate easy changing in SEISs and maintainability, the modules should be designed with low coupling and high cohesion such that individual modules are cohesive while unrelated modules are loosely coupled. Also, the power saving module can be encompassed in the module view to facilitate energy efficient SEIS. The module viewpoint is adapted from the Siemens four view model [Hofmeister et al., 2000] with some extensions to suit the need of SEISs. For instance the *Configuration*, and *InterfaceElement* classes are added in the metamodel. The *Configuration* class is for enclosing the resulted module view or

configuration by showing the beginning and end of the design. A *InterfaceElement* to express both a module and layer that use interfaces for communication. The decomposition structure of *Module* modeled using a composite pattern to treat all objects of SEISs, i.e. both individual and composite uniformly.

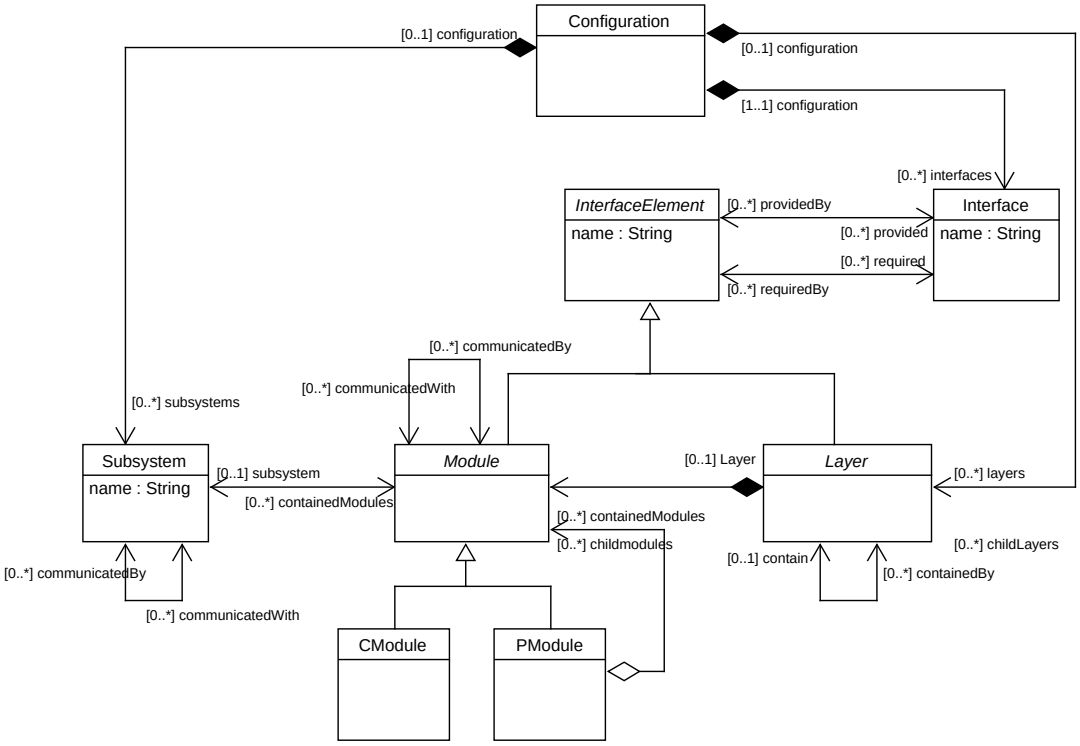


Figure 6.2: Module Viewpoint

6.2.2 Notations

The constructed module views shall use UML class diagrams for describing the use-dependencies between modules, package diagrams for showing both subsystem decomposition dependencies and use-dependencies among layers and assignment of modules to layers. The module viewpoint facilitates the constructions of module views in which the elements of the conceptual view as shown in Section 7.2.1 are mapped into software implementation modules as described in Section 7.2.2.

6.3 Execution Viewpoint

The execution viewpoint assigns the corresponding modules from the module view described in Section 7.2.2 into runtime entities and allocating them to the hardware resources. The execution viewpoint handles the concerns of operators and development experts mainly developers, testers, sensor providers and architects as described in Section 5.1 such as;

- **C10:** What are communication mechanism and protocols of SEISs?
- **C11:** How are the identified components of SEISs distributed among processing nodes at runtime?
- **C12:** How are the SEIS software components, distributed in different processing nodes, and do they interact at runtime?

6.3.1 Structure

All the aforementioned concerns are handled by execution viewpoint which identifies how the SEISs components mainly software modules will be deployed and allocated to the processing nodes. This involves the description of runtime structure and behaviour of SEISs that can be facilitated using an execution viewpoint illustrated in Figure 6.3. This execution viewpoint provides an essential set of elements and relationships that are utilized in execution models. The execution views of the existing SEISs [Nithya and Vanamala, 2018, Al-Ali et al., 2010] indicates that such an execution view of SEISs should describe information on the following aspects;

- *Processing nodes:* The processing nodes in the execution views should describe more consistent and useful information. Such that the execution viewpoint describes that the execution platform elements should include information about the characteristics information of processing nodes and functional elements. Hence the functional elements of SEISs are represented using *RuntimeEntities* which assign modules from the module view as described in Section 6.2. It is also recommended to include the characteristics of processing nodes such as allocated system-specific hardware devices, operating systems, system software through *HardwareResource*, *PlatformResource*, and *SoftwareResource* respectively. The *HardwareResource* refers to the physical device in which provide platform resources. The *PlatformResource* refers to the operating system or program execution environment, or a combination of both, while the *SoftwareResource* refers to the actual software which executes specified runtime entities.

- *Links between processing nodes:* The interaction between processing nodes is facilitated through links. Hence the execution viewpoint should encompass the description of links between processing nodes. The crucial aspects of the links to be included in the description include; the function of the link or link's technological characteristics such as bandwidth or capacity or other resources the system require from the link. All these properties can be expressed through *CommunicationPaths* and *Communication-Mechanisms* which facilitate communication between multiple runtime entities using multiple communication protocols.

This execution viewpoint of SEISs extends the concepts of predefined execution viewpoint of Siemens four view model to address the concerns of SEISs. The particular extensions that have been introduced include the *ExecutionModel*, *BoundedElement* and composite pattern in *SoftwareResource*. The *ExecutionModel* is used for demonstrating the execution configurations by showing the beginning and end of the execution design. The *BoundedElement* allows description of multiple instances of both*PlatformResources* and *RuntimeInstances* The *SoftwareResource* is decomposed using composite pattern to reflect the current technology to treat all object both composite and singular resources uniformly.

6.3.2 Notations

The constructed execution views shall use UML class diagrams for describing the execution configurations of SEISs, UML sequence diagrams for showing the dynamic behaviour of SEISs and UML state diagrams (or sequence diagrams) for demonstrating protocols of the communication path. The execution viewpoint facilitates the constructions of execution views in which the software modules of the module view as described in Section 7.2.2 are assigned into runtime entities and allocating them to the hardware resources as described in Section 7.2.3.

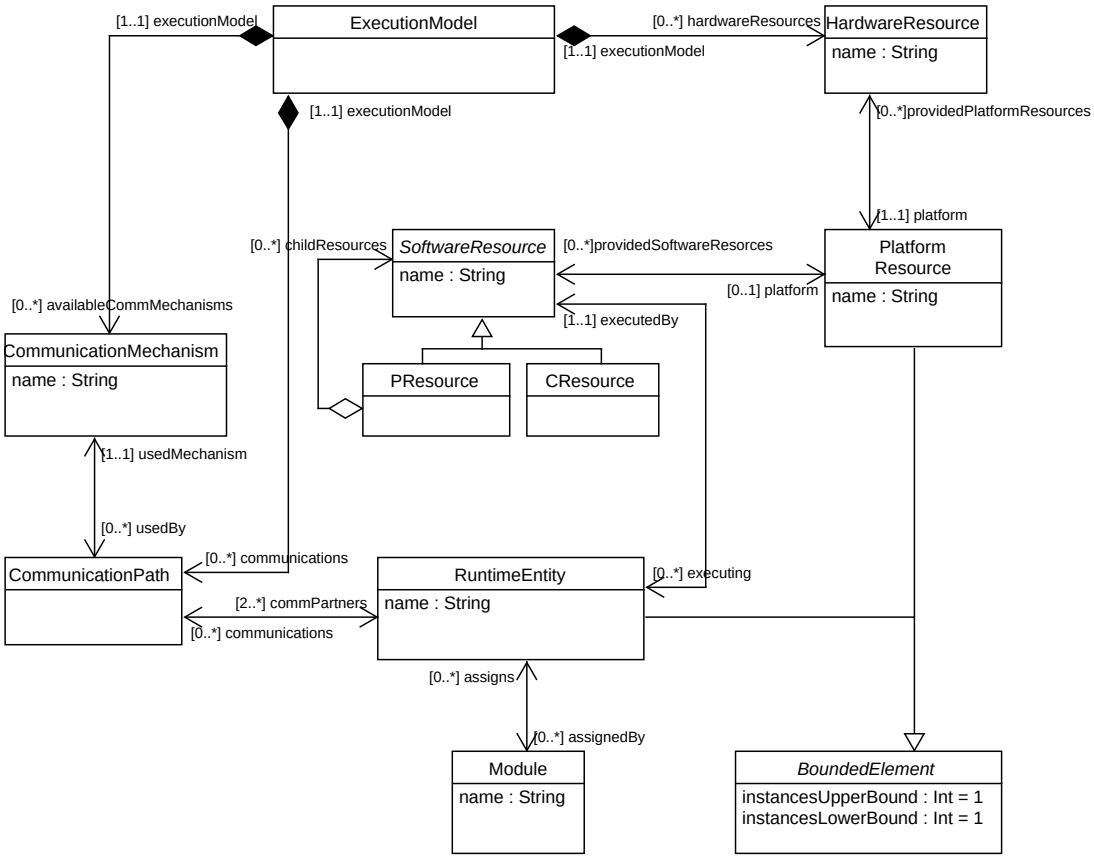


Figure 6.3: Execution Viewpoint

6.4 Code Viewpoint

The code viewpoint describes how the software implementation is organized by mapping the elements from both module and execution views to code components. The code viewpoint handles the **C13:** How are the identified components of SEISs implemented? the concern of operators and development experts mainly developers, maintainers, testers and architects.

6.4.1 Structure

The code viewpoint addresses the stated concern by describing how the software implementing the system is organized into source and deployment components. This is achieved by mapping the software elements from module and execution views as described in Section 7.2.2 and Section 7.2.3 respectively to code components of SEISs. i.e. source files, libraries, configuration files etc and their organizations. Figure 6.4 illustrates essential concepts and

relations that can be utilized in the construction of code views of SEISs. The description of those code components of the SEISs includes the following aspects;

- *Source Components:* The Implementation of software modules led into creation of multiple separate files such as those contain the source codes. *SourceComponents* implement individual software modules. Thus it is important for code views to include the information about *SourceComponents* and they should be organized using storage structures, i.e. files or directories. These *SourceComponents* are related to each other by two kinds of language-specific dependencies; (i) For the successful compilation, a *SourceComponent* may need to import another *SourceComponent*. (ii) A *SourceComponent* may be generated from another *SourceComponent*, e.g. during pre-processing. The *SourceComponents* are associated with *BinaryComponents* which are specific to the implementation language and development environment and in some languages are compiled to one or more program libraries. The *BinaryComponents* are related to their respective *SourceComponents* by link dependency. The *BinaryComponents* are also related to static *Libraries* by link dependency.

- *Deployment Components:* The runtime entities are instantiated through deployment components during runtime. These components include files that have to be included and usually contain definitions of processes and (/or) resources, files that describe how to build an executable, and files that are the result of translating, or compiling, i.e. dynamic *Libraries*, *ConfigurationDescriptions* and *Executables* respectively. *Executables* and dynamic *Libraries* are related to *BinaryComponent* and static library by a link dependency. The link dependency has also bound the relationship between the *Executables* and dynamic *Libraries*. The use of dynamic libraries can reduce the link-time dependencies and offer more flexibility in changing the implementations.

- *Organization of Code Components:* After the identification of source and deployment components, then these components need to be organized for the development so as not to lose control and integrity of the system [Clements et al., 2005]. The configuration management techniques can be employed; in the simplest case, a hierarchy of directories in a file system using *CodeGroups* for grouping. *CodeGroups* are organized such that they contain other *CodeGroups* or various *SourceComponents*, *Libraries*, *Executables*, and *ConfigurationDescriptions*.

The code viewpoint is adapted from the Siemens four view model with some extensions to suit the need of SEISs. For instance the *Component* class is added in the metamodel. The *Component* class is allowing multiple instances of code components in the code views. Also

the decomposition structure of *CodeGroup* is modeled using composite pattern to treat all code groups, i.e. both individual and composite uniformly (*CCodeGroup* and *PCodeGroup*).

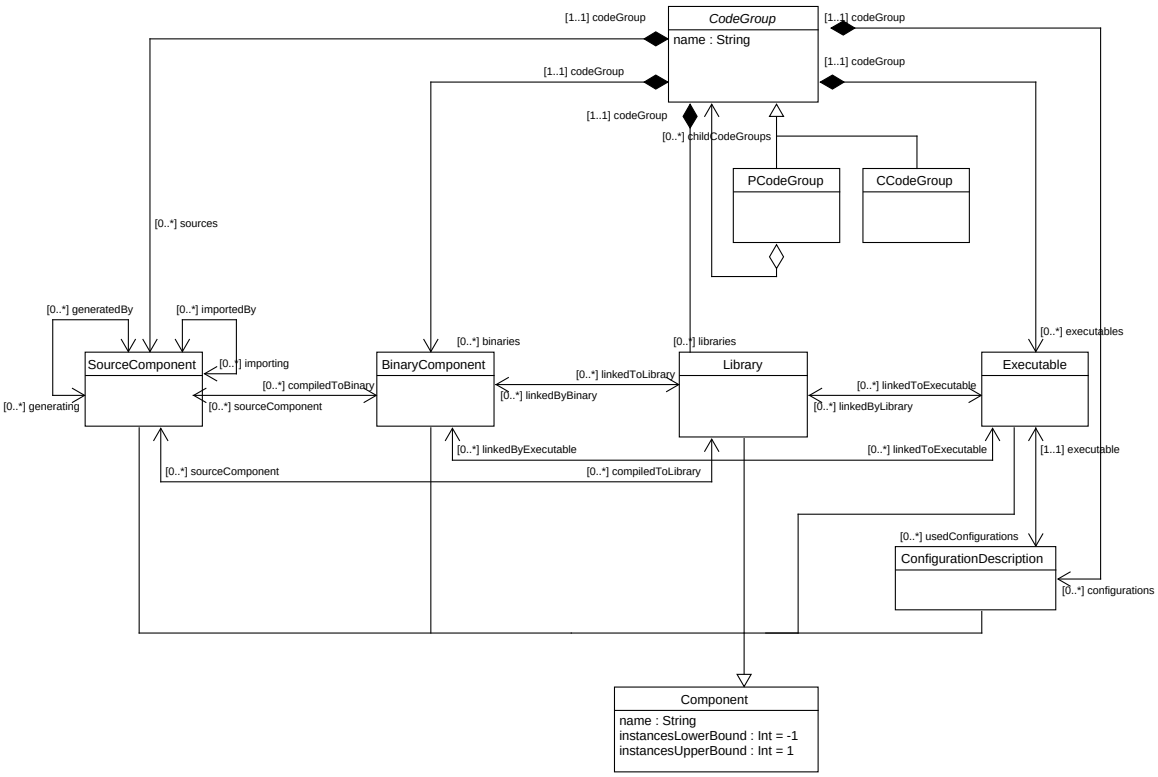


Figure 6.4: Code Viewpoint

6.4.2 Notations

The constructed code views shall use UML component or class diagrams to represent the source, intermediate and executable files. The package notation is used for directories. This viewpoint is selected to facilitate the construction, installation, integration and testing of the SEISs. The resulted code views are particularly useful as a guide for developing experts and maintainers during the construction, integration, installation, and maintenance of a system. The code viewpoint facilitates the constructions of views in software elements from module and execution views as described in Section 7.2.2 and Section 7.2.3 respectively to code components SEISs. i.e. source files, libraries, configuration files etc., and their organizations as described in Section 7.2.4.

6.5 Topology Viewpoint

The topology viewpoint determines the organization of nodes and their interconnecting links in the field of interests. The topology viewpoint is the new viewpoint which is also required in the description of SEISs [Zhang et al., 2008, Jadhav and Deshmukh, 2012]. Such viewpoint addresses the concerns of operators, and development experts mainly developers, sensor-actuator providers, maintainers and architects such as;

- **C7:** How to distribute the nodes within the specified field to achieve the best range and coverage from the sensors?
- **C10:** What are communication mechanism and protocols of SEISs?
- **C14:** How are the non-functional requirements supported?

The topology viewpoint determines the organization of nodes and their interconnecting links in the field of interests. The main intention of this viewpoint is to guide the construction of topology views focusing on the specification of the field nodes network topology. Such a topological structure of the network may be depicted either physically or logically.

6.5.1 Structure

The topology viewpoint addresses the aforementioned concerns by describing the physical layout or network arrangement of multiple nodes and how they communicate with each other in the field of interest. The topology viewpoint is depicted in Figure 6.5 describes the required main entities for the construction of the network topology. The topology views of the existing SEISs [Jadhav and Deshmukh, 2012, Zhang et al., 2008] indicate that; such a topology view of SEISs should describe the information of the following aspects;

- *Nodes:* The *Nodes* in the topology view refer to the devices connected to the network. In the context of SEISs, the nodes reflect the hardware resources of the execution view of SEISs as described in Section 7.2.3. The nodes can be of different type *NodeType*., i.e. sensor nodes, server, actuators, computers, etc..
- *Links:* The *Nodes* in the network can have more than one *Link* connected to other *Nodes* (source and outgoing links) which are denoted by *LinkEnd*. *Links* reflect the communication paths of execution view of SEISs as described in Section 7.2.3.
- *Distribution of Nodes:* The topology view should resemble as much as possible the actual physical and geographical distribution of the nodes. This is particularly required

to make some design decision explicit to support certain non-functional requirements, i.e. energy efficiency, best range and coverage of sensors, security measures, etc.. For instance, the topology view of SEISs can indicate how the sensor nodes are distributed in the field of interest through various topologies i.e. bus, star, mesh, etc..

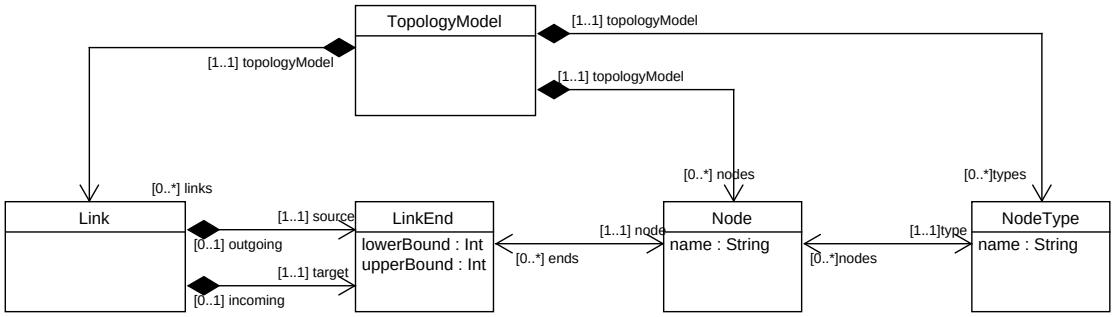


Figure 6.5: Topology Viewpoint

6.5.2 Notations

The constructed topology views shall use UML class diagrams for showing the interconnection among multiple nodes and graph like structures for visualizing the network topologies [Shikhaliyev, 2012]. The topology viewpoint facilitates the constructions of topology views in which the hardware resources and communication paths of execution view as described in Section 7.2.3 are mapped to nodes and links forming topologies as described in Section 7.2.5.

6.6 Data Viewpoint

The data viewpoint describes the high-level view of the static data structure in terms of data entities and their relationships. The data viewpoint is also the new viewpoint which is required in the description of SEISs [Ujang et al., 2013]. Such viewpoint addresses the concerns of operators and development experts mainly developers, testers, maintainers, sensor providers and architects mainly;

- **C9:** How is data created, accessed, updated and stored in the SEIS?
- **C14:** How are the non-functional requirements such as interoperability supported?

6.6.1 Structure

Data viewpoint is one of the core viewpoints required in describing the structure of the data handled in information systems [Merson, 2009]. The data viewpoint facilitates the construction of data views which describes the static information structure in terms of data entities and their relationships. In addition to other practical advantages, the data viewpoint serves as the blueprint for the creation of a physical database, i.e. how data is stored, manipulated, interpreted and accessed. The data viewpoint led into the construction of standardized data view which supports interoperability by enabling smooth data exchange between systems. The data views (or models) are often represented graphically using entity-relationship diagrams (ERDs) or UML class diagrams. Therefore, this data viewpoint is derived from the UML class diagram metamodel as described in [F. Paige et al., 2011] and contains only the relevant elements and properties for data modeling as shown in Figure 6.6. Such viewpoint shows that the data view of SEISs should describe the following aspects;

- *Data Entities:* The data entity is the fundamental entity of this viewpoint. It represents any piece of data involved in the system. The data modeling in SEISs is concerned with data entities and their respective attributes which are relevant to SEISs. Such entities possess names, and list of attributes. Such that the data model of SEISs is bundled in a *Package*. The *Package* can contain various *Classifiers*. The *Classifier* is a super class representing data entities or elements and only contains a *Name* attribute to give each element a name. A *Classifier* is either a *PrimitiveDataType* i.e. String, Int, etc., *Class* or an *Association*. The *Class* can have multiple *Attributes* and *Operations*. The *Attribute*, *Operation* and *Parameter* are associated with the *Classifier*.

- *Relationship between Data Entities:* The description of the relationship between data entities is important for designating a logical association between entities. This is specified using *Association* which is associated with source and target *Classes*. There three allowed relationships *Composition*, *Aggregation* and *Generalization*. The *Composition* represents a part-whole relationship between entities. The *Aggregation* is an abstraction which turns a relationship between entities into an aggregate entity. The *Generalization* indicates an is-a relation between data entities.

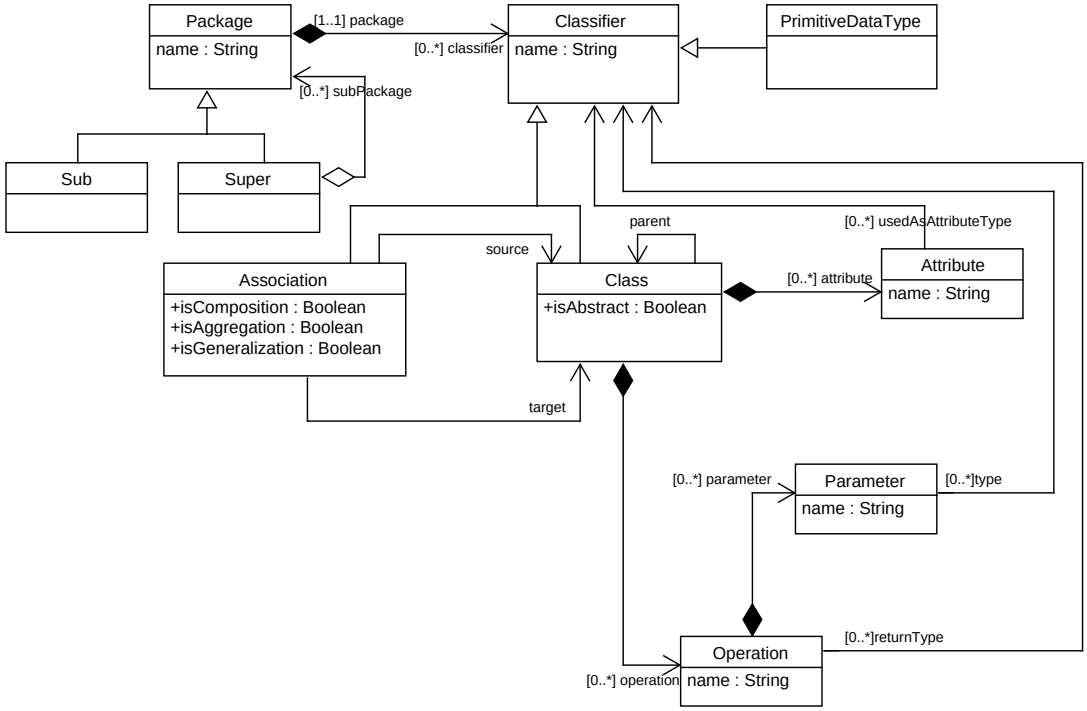


Figure 6.6: Data Viewpoint

6.6.2 Notations

The constructed data views shall use entity-relationship and UML class diagrams for modeling both static and dynamic information structure of SEISs. The data viewpoint facilitates the constructions of data views in which each software module of module view as described in Section 7.2.2 contains data entities as described in Section 7.2.6.

6.7 Viewpoints Integration

The principle of separation of concerns is a well-known and valid principle in software development but it can easily lead to lacking integration between the viewpoints of various stakeholders [Gordijn et al., 2001]. Therefore, it is important to make relations between viewpoints visible and traceable. This can be achieved through the unification of viewpoints, i.e. viewpoints integration. Since the results of modeling a system using a certain viewpoint may be different from the modeling of the same system using another viewpoint. Thus, we have to deal with multiple viewpoints. Despite the fact that these viewpoints are required to be independent with useful semantics to their respective stakeholders, they all represent one architecture. There is a need for integrating viewpoints to demonstrate that the contents of those viewpoints are fully integrated and coherent to facilitate the construction of well-integrated system architecture. This integration addresses the concern of developing experts which pertain to **C8:** How is SEIS integrated?. Thus viewpoints integration is required in SEISs to ensure conceptual integrity and consistency. Such consistency should be done automatically rather than manually by hand since the consistency management by hand is tiresome and error-prone.

The viewpoints integration is associated with maintaining the consistency of viewpoints by establishing the suitable correspondence relations between viewpoints, and, finally applying suitable consistency rules to verify the consistency between the viewpoints [Enders et al., 2002]. In which the correspondence relation describes the relationship between corresponding viewpoints. The semantics of these relations must be specified separately and can partly be done by means of consistency rules [Dijkman et al., 2008]. The consistency rules represent requirements on the relations between the contents of different viewpoints to handle inconsistencies that could take place between the related viewpoints. All consistency rules must evaluate to 'true' in a consistent design [Dijkman et al., 2008]. This enables continuous evaluation in early design stages which avoids problems like unknown feature or design entity hence promotes the verification of the system functional integrity. The correspondence relations and consistency rules have been developed by Ruthbetha kateule and Johannes Meier, a PhD candidate in the Software Engineering Group at the Carl von Ossietzky University of Oldenburg.

6.7.1 Establishing Correspondence Relations and Consistency rules

A suitable correspondence relation is achieved by identifying exactly which parts of the specifications of the viewpoints have to do with the concerns of the system. SEISs are described using six viewpoints, i.e. conceptual, module, execution, code, topology, and data viewpoints.

Recalling objectives of these viewpoints; conceptual viewpoint describes the main system functionalities and their dependencies in terms of components and connectors cf. Section 6.1. Module viewpoint maps the conceptual elements into the software implementation modules cf. Section 6.2. Execution viewpoint defines how the software implementation modules will be assigned, their run-time instances communicate and allocation into physical resources cf. Section 6.3. Code viewpoint describes how the software implementing the system is organized by mapping the elements from both module and execution views to code components cf. Section 6.4. Topology viewpoint describes how the nodes and their interconnecting links from the execution viewpoint are organized. cf. Section 6.5. Data viewpoint describes how the software modules store, manipulates, manages and distribute data cf. Section 6.6. These objectives demonstrate the relations between these viewpoints by keeping tracks of the relation between the basic features or elements of the viewpoints and their mappings to other viewpoints.

An inconsistency check is initiated by a specific viewpoint with respect to a particular feature or element within another viewpoint. Thus, first, a relationship between these viewpoints need to be established in order to check and handle well such inconsistency. Therefore to be able to specify and check these relations, integration of viewpoints is considered as part of the design. As shown in Figure 6.7, the correspondence relations of these viewpoints exists between: conceptual - module, module - data, module - code, module-execution - code, execution - topology. To facilitate the consistency of viewpoints, such relations are associated with the consistency rules (CRs).

Conceptual - Module

The module viewpoint refines the main functions of the system as represented using component and connectors by the conceptual viewpoint in terms of implementation software modules. This relation is associated with the following corresponding rule;

CR1.1: Conceptual elements (components, connectors, ports, roles and protocols) are mapped into software modules in module view.

CR1.2: The complete mapping of conceptual elements into software modules has to be decided manually and cannot be automated. Since an arbitrary number of conceptual elements such as components, connectors, ports, roles and protocols can be realized in a single module.

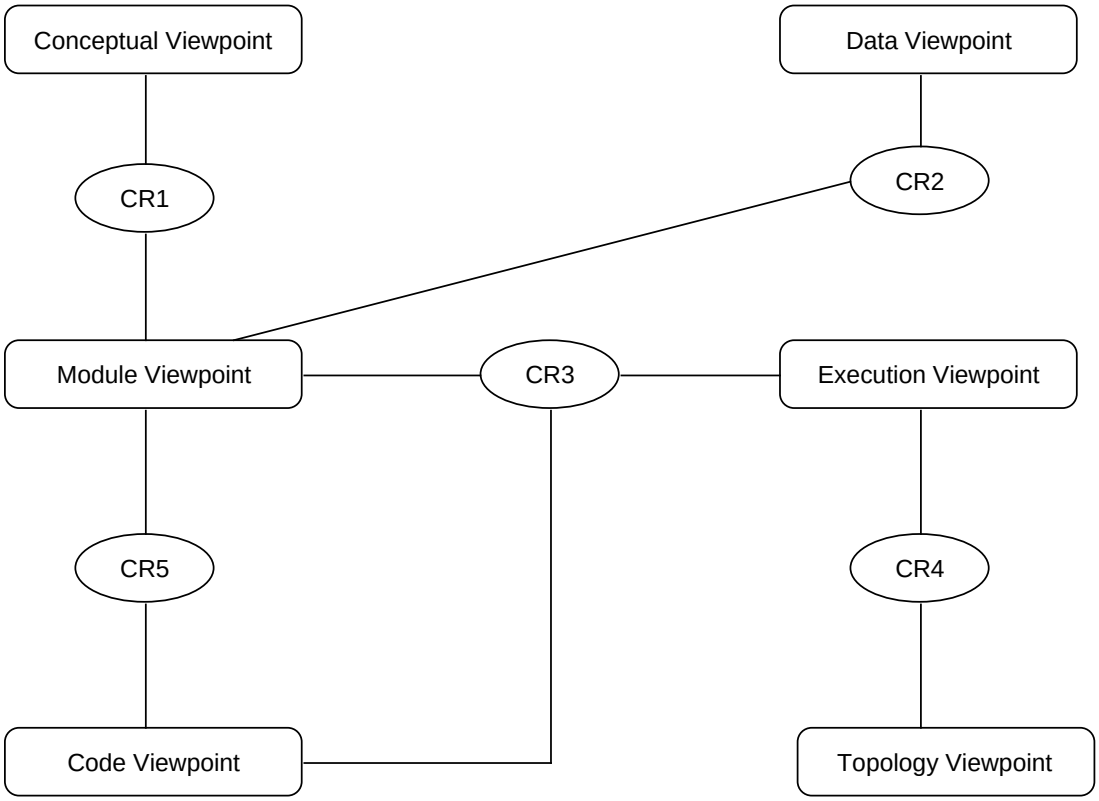


Figure 6.7: Viewpoints Integration

Module - Data

The data viewpoint describes how software modules represented by module viewpoint store, manipulate, manage and distribute data. Each software module and interface defined by module viewpoint can contain multiple data entities. This relation is associated with the following corresponding rule;

CR2: Modules and Interfaces have to be linked with the packages which contain the data structures which are required for them. This has to be done manually since there is no automation for that.

Module - Execution - Code

The execution viewpoint allocates software modules of the module viewpoint to the execution platform elements, i.e. their run-time entities, and allocation into physical resources. Software elements of both module and execution viewpoints are mapped to code components. This relation is associated with the following corresponding rules;

CR3.1: Each module in the module view can be deployed at an arbitrary number of runtime entities in the execution view.

- If a module is deployed at a runtime entity, an arbitrary number of executables can be specified (which are runnable on the hardware resource and is derived from the implementation of the module). The arbitrary number of executables is helpful since there might be several executables which are required to execute one module.
- The complete mapping/deployment has to be decided manually and cannot be automated. Since an arbitrary number of software modules can be deployed in a single module.

CR3.2: The communication paths between runtime entities in the execution view can be derived from the communication between Modules (internal and external) in the module view.

- Communication between Modules inside the Module view can be with and without the use of an Interface, i.e. direct communication without Interface is demonstrated between the Modules inside the same Layer while the indirect communication via an Interface is provided by at maximum one Module and is required/used by an arbitrary number of Modules.
- If two Modules communicate with each other (whether they use an Interface or not), the corresponding RuntimeEntities are connected with a Communication-Path.

Execution - Topology

The topology viewpoint describes how the nodes (mainly physical resources) and their interconnecting links from the execution viewpoint are organized. The execution - topology relation is associated with the following corresponding rules;

CR4.1: Each hardware resource in the Execution view is represented by exactly one node in the Topology view (and vice versa).

- Corresponding instances are identified by the same names.
- Newly created elements in execution view are also added to the topology view (and vice versa).

- Deleted elements in execution view are also deleted in the topology view (and vice versa).

CR4.2: Each communication path in the Execution view is represented by exactly one link in the Topology view (and vice versa). This counts only for communication paths which connect two runtime entities which belong to different hardware resources. (Otherwise, it represents internal communication, which is not shown in the Topology view.)

- In the SEIS domain, there is at maximum one communication path between two hardware resources. There are no cases, where more than one communication path between the same two hardware resources exist.
- New (external) communication paths in the execution view result in new links in the topology view since they will be derived.
- New links in the topology view will be removed/are not allowed, because they cannot be realized in the execution view since information about the related runtime entities is missing.

Module - Code

The code viewpoint describes how the software implementation is organized by mapping the elements from both module view to code components. This relation is associated with the following corresponding rules;

CR5.1: Modules and interfaces are implemented in source components.

- Since modules and interfaces can be implemented for different platforms, each module and interface can have an unlimited number of implementations.
- There might be modules or interfaces without any implementation.
- Normally, each source component realizes only one module or interface. Since there are special some situations, for example, there are various modules in sensor nodes but they are all implemented within a single source component, one source component can be linked to more than one Module or Interface.
- All mappings have to be specified manually since there is no automation.

CR5.2: The implementation of subsystems and layers from the module view and hardware resources from the execution view are contained in code groups from the code view.

- Since subsystems and layers can be implemented for different platforms, each subsystem and layer can have an unlimited number of implementations which can be linked by code groups.
- Since code groups can be used also for hierarchical structuring, there might be code groups without a linked subsystem or layer.
- Additionally to subsystems and layers, also hardware resources (from the Execution View) can be linked by code groups.
- Some mappings can be automated by comparing their names regarding equality.

6.7.2 Realization of Viewpoints Integration

The integration of SEISs viewpoints has been realized by Johannes Meier through a MoConseMI (Model Consistency Ensured by Metamodel Integration). The MoConseMI is a Single Underlying Model (SUM) approach which starts with existing initial models and conforming metamodels and creates a Single Underlying MetaModel (SUM(M)) [Meier et al., 2019]. MoConseMI is a suitable approach for the integration of SEISs viewpoints compared to other approaches,i.e. since this integration involves existing viewpoints (metamodel) which need to be reused [Meier et al., 2019]. The MoConseMI is capable of handling the inconsistencies in the existing models and resolve them during integration. This approach is based on chains of operators which describe the transformations between the initial models and the SUM. These chains of operators are executed to initialize the SUM and ensuring all the initial models and the SUM to be consistent with each other [Meier and Winter, 2018]. Furthermore, operators split long transformation into short and manageable transformations that can be easily debugged and iterative developed while ensuring the models before and after are consistent [Meier and Winter, 2018]. Such operators can be configured regarding their impact on the metamodel and model to fulfil the current consistency rule.

The integration of SEISs viewpoints as illustrated in Figure 6.8 is demonstrated by a chain of operators. Such that the SUM(M) is formed based on the initial metamodel (viewpoint), then the operators are applied to change the current metamodel and model together in a step-wise way (both viewpoints and views). For instance, the integration of module and data viewpoints is associated with the CR2 in which Modules and Interfaces have to be linked with the packages which contain the data structures which are required for them. This has to be done manually since there is no automation for that. These consistency rules have been used to express that the data viewpoint must be the correct refinement of the part of the module viewpoint to which it is related. For the realization of this integration, a new

association has to be created between Packages and their related Modules and Interfaces. This is achieved through the configuration of the following operators in Figure 6.8:

- 11→12 for addClass to create the new Class with their (already existing) subclasses Module and Interface.
- 21→22 addRelation to create the new Association in the metamodel, without adding any links in the model.

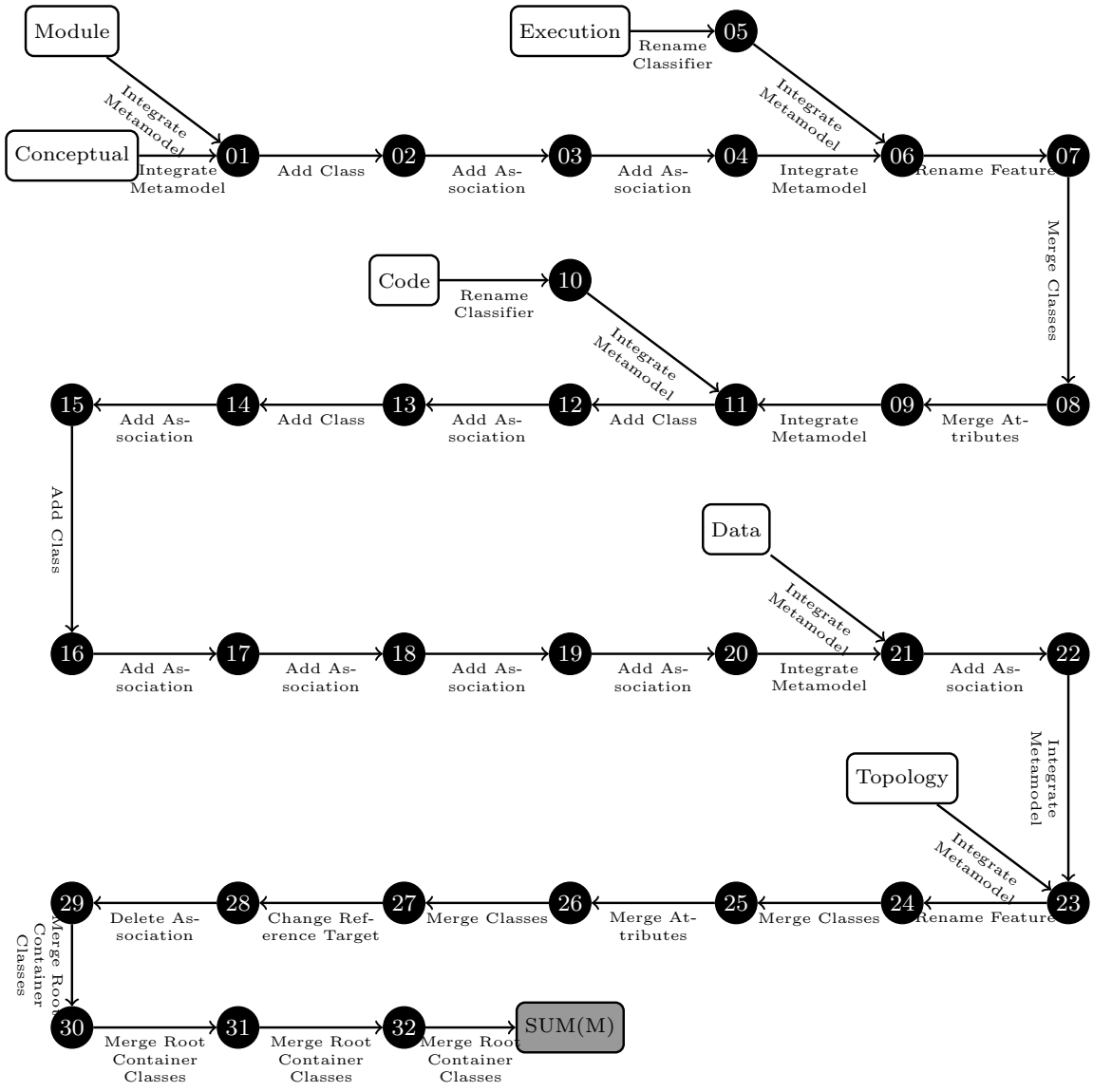


Figure 6.8: Operator Orchestration for the technical Integration of both SEISs viewpoints

6.8 Summary

Capturing the essence and details of the whole architecture in a single model may result in complex architecture which cannot be understandable to the stakeholders. Therefore an efficient way of dealing with such complexity is the use of multiple views which focus on important aspects of SEISs while addressing key concerns of various stakeholders. However, the selection of which views to be used is facilitated by the use of viewpoints which provide standardized definitions of views contents. This chapter proposed viewpoint catalog of SEISs, comprising, conceptual, module, execution, code, topology and data viewpoints. Such viewpoints have been defined by analyzing various SEISs, identifying stakeholders of SEISs and determining the proper framing of identified concerns described in Chapter 5. Furthermore, the SEISs viewpoints have been integrated to ensure consistency between viewpoints. Then Chapter 7 proceeds with the application of these viewpoints into the construction of RefSEISs.

Chapter 7

RefSEISs Architectural Views

The software architecture of the system is organized into one or more architectural views. As noted in Chapter 2.3, the description of RefSEISs adheres to ISO/IEC/IEEE 42010 standard. The standard describes, analyzes and resolves the set of specific concerns in each of the viewpoints are expressed as the architecture views. The combination of both architecture views and their corresponding architecture models are considered as the representations of the software architecture. As stated in Definition 2.10, a view refers to a particular representation of the entire system using a set of models from the specific perspective of a viewpoint. This definition has been introduced in Section 2.3.1 and will be used throughout this work. The separation and modeling of different views for the various concerns of the stakeholders reduce the complexity of the architectural description of a system and specialized notations can be used to describe each view.

This chapter presents the multi-views of RefSEISs which conform to their corresponding viewpoints specified in Chapter 6. These views depict the generic structures of SEISs (concepts and relations). Although the identified views of RefSEISs can be constructed separately, they are somehow related as each view describes the same system. The descriptions of concrete SEISs architectures do not require the description of all the proposed views instead some views can be omitted. The RefSEISs encompasses six views as described in their respective sections (i) Conceptual view in Section 7.2.1, (ii) Module view in Section 7.2.2, (iii) Execution view in Section 7.2.3, (iv) Code view in Section 7.2.4, (v) Topology view in Section 7.2.5, and (vi) Data view in Section 7.2.6. Additionally, the adoption of Siemens view model is associated with the use of global analysis as a starting point. Hence, the description of architecture views of RefSEISs starts with global analysis in Section 7.1.

7.1 Global Analysis

The global analysis intends to identify essential factors that influence the architectural design and then develop strategies for each identified factor. Comprehensive analysis helps software development experts mainly software architects to determine factors that affect the architectural design as well as identifying, making and recording crucial architectural design decisions [Schwanke, 2003]. This is accomplished through the use of factor tables in capturing the information. Such factors represent vital issues that must be satisfied during the development of SEISs. Based on the identified concerns of stakeholders and requirements of SEISs established in Chapter 5, the main factors related to the development of SEISs that are appropriate to the benefits of identified stakeholders are as summarized in Table 7.1.

Table 7.1: Factors influencing architectural design of SEISs

Factor 1	The system must be easily maintainable and capable of evolving by supporting new features.
Factor 2	The system should possess energy efficient sensor networks.
Factor 3	The system must support heterogeneous devices.
Factor 4	The system must be operated by a user.
Factor 5	The system must have a friendly user interface.

The identification of architectural factors proceeds with the identification and selection of architectural strategies that could be used to fulfil the factors mentioned above. Such strategies may include architecture styles, architecture tactics and design patterns. These strategies guide architectural decisions. Those factors discussed above in Table 7.1 are supposed to be covered by at least one strategy. The factors and their corresponding strategies are illustrated in Table 7.2.

Table 7.2: Factors and corresponding strategies

Factors	Corresponding Strategies
1,5	Use of well-known architectural patterns such as Model View Controller (MVC), Client - Server and Layering.
1,4,5	Utilize a central controller component.
1,3	Use of Key-Value type table architectural style.
1,4,5	Decouple the user interaction module.
1,2	Utilize a network cluster tree-topology architecture style.
1,2,4,5	Modules are designed to achieve high cohesion and low coupling.

The RefSEISs consists of three main subsystems that can be combined in different variants to build a concrete SEIS solution; Sensor-Actuator subsystem, information control

centre subsystem and communication subsystem. A complete SEIS must consist of all these three subsystems even if the functionality of some subsystems are reduced to a minimum. The *sensor-actuator subsystem* describes the solutions which measure physical environment events, i.e., temperature, humidity etc., pre-processing the collected data and alter the state of environmental phenomena to satisfy the intended objectives of a system. The *information control centre subsystem* describes the integration, analysis, storage and presentation of the information received from the sensor and then generates alarms and events via actuators based on the objectives of particular SEIS. The *communication subsystem* facilitates the interactions within and between the sensor-actuator subsystem and the information control centre subsystem. Hence the communication subsystem is usually demonstrated implicitly within both sensor-actuator and information control centre subsystems.

7.2 Architecture Views

This section describes the architecture views used to design the RefSEISs which are associated with the elements of the system and how these elements are mapped to one another. Each view results from their respective viewpoint provided in Chapter 6 and also documented based on concerns and requirements described in Chapter 5, global analysis factors and strategies outlined in Section 7.1 that serve as a rationale using a specific architectural style or pattern.

7.2.1 Conceptual View

This conceptual view provides an abstract (or high level) design of essential system functional entities and their relationship with each other in terms of components and connectors. The conceptual view is supposed to be a technology-independent and conforms to the conceptual viewpoint described in Section 6.1. This view employs a Client-Server architecture style, central controller component, and decouple the user interaction model strategy through Module-View Controller (MVC) pattern as adapted from [Zhang et al., 2008, Ramesh, 2014, Sulistyowati et al., 2015]. The conceptual view of RefSEISs addresses the following architectural requirements: FN1: Data Acquisition, FN2: Storage, FN3: Phenomena Analysis, FN4: Visualization, FN5: Phenomena Control, FN6: Sensor Management, and FN7: Actuator Management. The utilization of Client-Server architecture style and decouple the user interaction model strategy through MVC makes the SEISs highly maintainable and adaptable since the changes will be accommodated comfortably. As depicted in Figure 7.1,

the conceptual view of SEISs is expressed in terms of sensor-actuator, information control centre and communication subsystems.

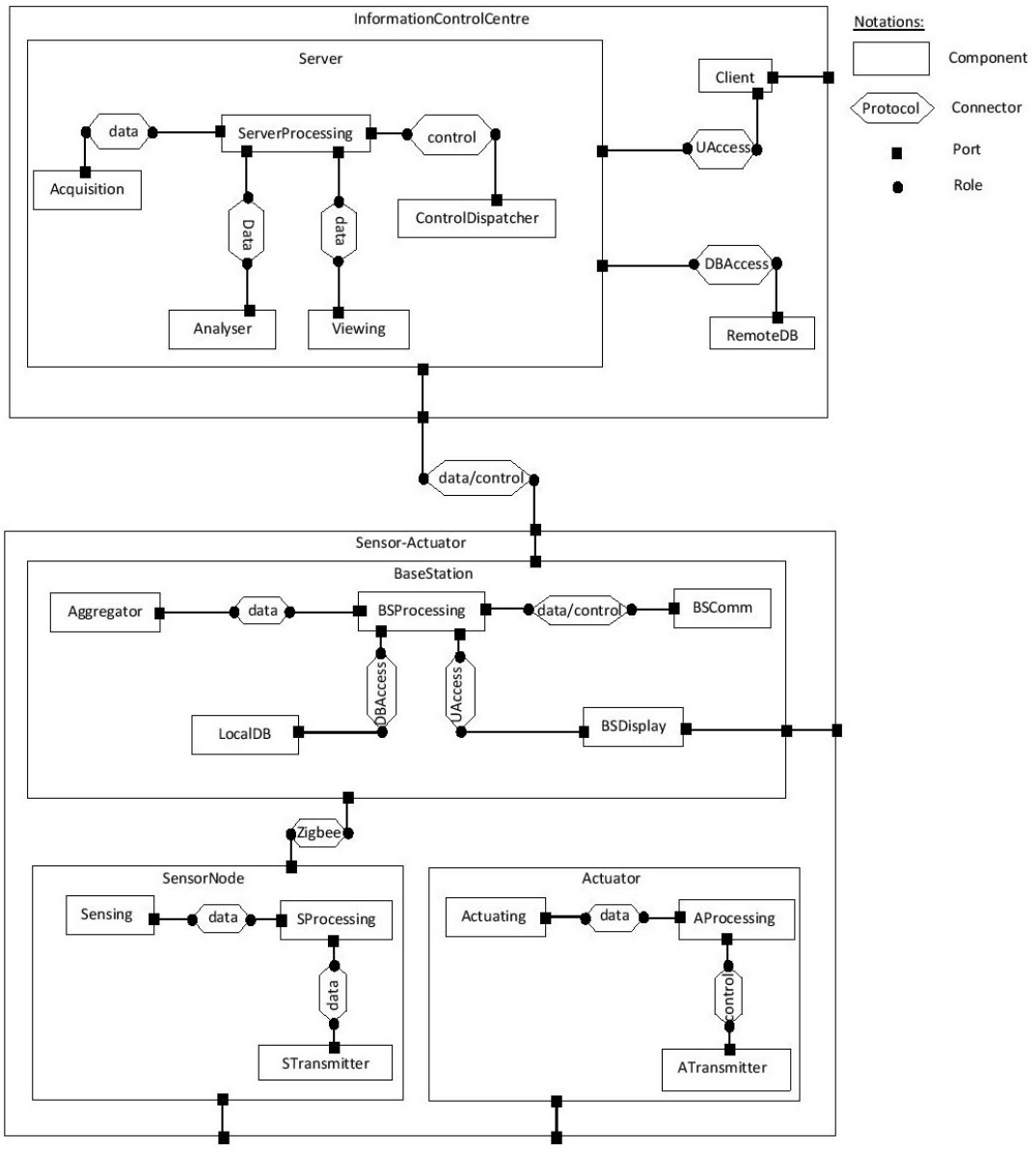


Figure 7.1: Conceptual View of RefSEISs

The sensor-actuator subsystem is decomposed into three main components; *BaseStation*, *SensorNode* and *Actuator*. The *BaseStation* communicates with both *SensorNodes* and *Actuators* through *data* and *control* connectors respectively. The *BaseStation* manages both sensor and actuator nodes in the field of interests. The *BaseStation* integrates data from various *SensorNodes* and then transmits the collected data to the information control centre

subsystem for further processing. The *BaseStation* distributes and executes control actions through *Actuators* as commanded by information control centre subsystem. The *SensorNodes* measures the environmental parameters directly from the field of interest while *Actuators* manipulates the environmental parameters via execution of control actions. With the application of decouple the user interaction model strategy and separation of concerns; the *BaseStation* is further decomposed into *Aggregator* for acquiring data from sensors, *BSProcessing* for local processing through *data* connectors, the local database (*LocalDB*) is used for storing the collected data locally in the field of interest, *BSDisplay* for allowing the user (mainly an operator) to view data, as well as control data acquisition and (or) execution of control actions, and *BSComm* is used for transmission of the collected data or receive information to and from sensor nodes, actuators and information control centre subsystem. The *BaseStation* defines and sets the policies for collecting and processing the data (environmental parameters). The *SensorNode* is further decomposed into *Sensing*, *SProcessing* and *STransmitter* for collecting, processing and transmitting the observed environmental parameters to the *BaseStation*. The *Actuator* is also decomposed into *ATransmitter*, *AProcessing* and *Actuating* for receiving, processing and executing control actions or signals from the *BaseStation*.

Similarly, the information control centre subsystem employs a client-server architecture style, central controller component, and decouple the user interaction model strategy through Module-View Control (MVC) pattern. A user interface is expressed as a *Client* component as an operator uses a port to send requests to the *Server* component as the controller. The interaction between the *Client* and *Server* components is facilitated by *UIAccess* connector. The *Server* component uses another port to access the model in the *RemoteDB* component via *DBAccess* connector while processing the requests of the clients and send them back as views. A *Server* is further decomposed into *Acquisition*, *ServerProcessing*, *ControlDispatcher*, *Analyser* and *Viewing*. The *Acquisition* for receiving the observed information from the field of interests. The *ServerProcessing* processes and analyses the observed information through *Analyser* and send the control actions to the actuators through *ControlDispatcher*. The *Viewing* generates potential views for displaying information as required by clients.

The communication subsystem is demonstrated explicitly within both subsystems (sensor-actuator subsystem and information control centre subsystem) through connectors which encompass protocols to be obeyed by each of the ports or roles. This conceptual configuration is described using UML Class diagram using special graphical symbols for the component, connector, role and port to make the description more clear, easy and natural to read.

7.2.2 Module View

The module view is originated from the application of the module viewpoint described in Section 6.2. The module view of the RefSEISs addresses the following architectural requirements; FN1: Data Acquisition, FN2: Storage, FN3: Phenomena Analysis, FN4: Visualization, FN5: Phenomena Control, FN6: Sensor Management, and FN7: Actuator Management. All these requirements are fulfilled by module view which brings the system closer to the software implementation by mapping the elements of the conceptual view of SEISs described in Section 7.2.1 into the implementation modules as shown in Table 7.3.

Table 7.3: Mapping between Conceptual and Module Architecture Views

Conceptual Element	**Module or Subsystem**
Sensing	MSensing
SProcessing, STransmitter, and data connectors	MSensorManager, MPowerSaving
Actuating	MActuating
AProcessing, ATransmitter and control connectors	MActuatorManager
Aggregator	MAggregator
BSProcessing, BSComm, data and control connectors	MTopology, MFProcessing
Analyser	MAnalysis
Viewing	MViewing
Acquisition	MAcquisition
ControlDispatcher	MControlDispatcher
Acquisition	MAcquisition
ServerProcessing, data and control connectors	MServerProcessing
LocalDB	MLocalDB
RemoteDB	MRemoteDB
BSDisplay	MFDisplay
Client	MUserInterface
UAccess	IUserAccess
DBAccess	IDataAccess

A module is a representation of a set of functionalities of the SEISs that can be realized and provided as service. Figure 7.2 shows the module view of RefSEISs based on modules and dependency relationships. According to global analysis, the module view of RefSEISs is described in terms of sensor-actuator, information control centre and communication subsystems that use layered architectural pattern, in which modules of subsystems are organized hierarchically into a triple-layer classic architecture; *GUI*, *Application*, and

DataAccess. These layers are associated with commonly defined interfaces to facilitate communication between modules. This layered architecture formed based on the following strategies: separate the user interface and database model from the application-specific logic via MVC with a central controller component, and the resulted modules possess low coupling and high cohesion such that individual modules are cohesive while unrelated modules are loosely coupled. All these strategies have been adopted from the architectural knowledge of the successful implementations of the existing SEISs [Huang et al., 2015, Sunkpho and Ootamakorn, 2011, Udo and Isong, 2014, Nguyen et al., 2015, Lozano et al., 2012].

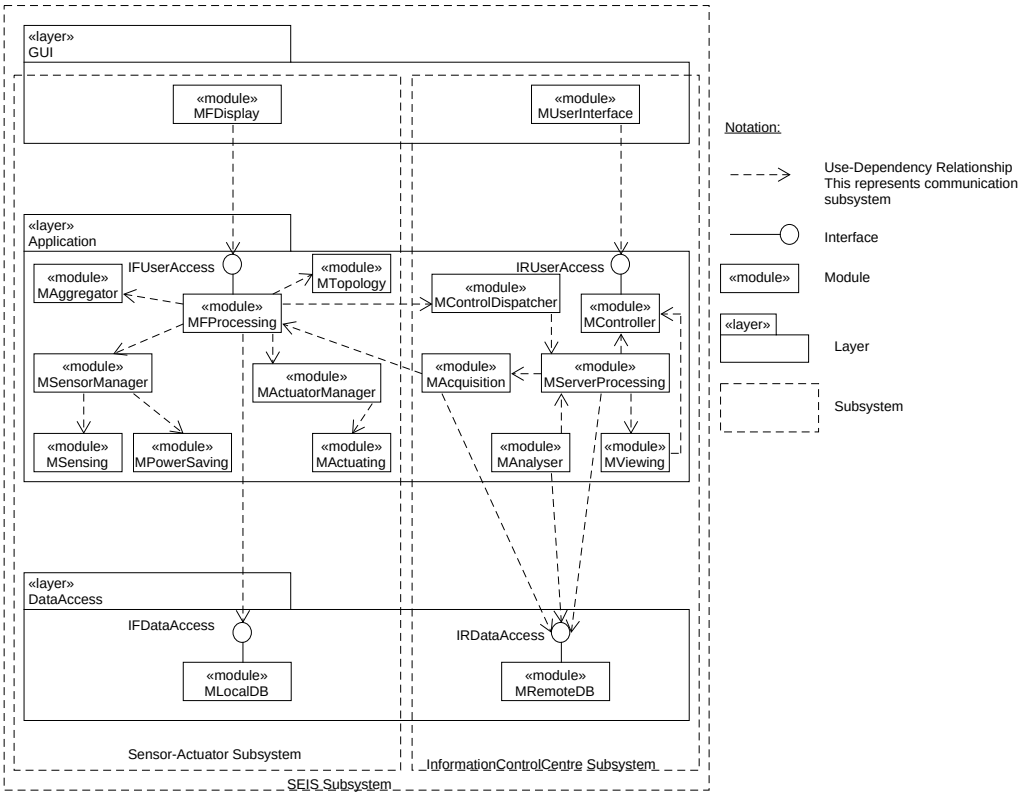


Figure 7.2: Module View of RefSEISs

The SEIS subsystem consists of sensor-actuator, information control centre and communication subsystems. The *GUI Layer* handles the interaction between users and the SEIS that comprise the business logic layer mainly *Application Layer*. This layer consists of *MUserInterface* and *MFDisplay* modules for a user to access both sensor-actuator and information control centre subsystems respectively.

The *Application Layer* defines the sets of modules that implement the business logic of SEISs. This layer encapsulates the system services, data processing units, rules and data.

The *Application Layer* is a user-centric layer which executes various tasks for the users using *MUserInterface* and *MFDisplay* use *IRUserAccess* and *IFUserAccess* interfaces for sending users' requests and accepting responses to and from the *Application Layer*. In this layer, the sensor-actuator subsystem consists of: *MSensing*, *MFProcessing*, *MSensorManager*, *MActuatorManager*, *MActuating*, *MTopology*, *MPowerSaving* and *MAggregator* modules. The *MSensing* module obtains data from the real world by allowing the collection of data from either digital or analog sensors. The *MPowerSaving* module imposes power saving mechanisms to the sensor nodes such as the implementation of state transitions mechanisms on sensor nodes. The *MActuating* module executes control actions as commanded. The *MSensorManager* and *MActuatorManager* modules specify and control the activities of sensor nodes and actuators respectively. The *MFProcessing* module is the core module which processes both incoming and outgoing data in the field of interest. This module implements the sensor sampling and actuator tasks, time synchronization and is distributed on the multiple physical nodes of the sensor-actuator subsystem. Additionally, the *MFProcessing* handles user requests that received through *MFDisplay* that concern with accessing the system in the field of interests. The *MTopology* module handles the configurations of nodes in the field. The *MFProcessing* module is associated with *MAggregator* module to collect, filter and integrate data originated from various sensors and process them for transmission to the information control centre subsystem.

The information control centre subsystem in *Application Layer* is made up of: *MServer-Processing*, *MAnalyser*, *MAcquisition*, *MController*, *MViewing* and *MControlDispatcher* modules. The *MServerProcessing* and *MAnalyser* modules handle the received data from the sensor nodes in the field through *MAcquisition* module, analyzing the environmental phenomena and then issue the control actions and send them to the actuator nodes in the field. The *MControlDispatcher* module dispatches commands, or control actions for controlling certain environmental phenomena or issue alerts in case of occurrence of particular environmental phenomena or the high probability of occurrence of such environmental phenomena is observed. The *MServerProcessing* module receives the requests and sends the responses from and to the users respectively. In processing the requests of users, the *MViewing* module generates views as instructed or required by the *MServerProcessing*. In processing the requests of users, the *MViewing* module generates views as instructed or required by the *MController*. The *MController* handles the control functionality while other modules in this layer handle the application functionality.

The bottom layer is the *DataAccess Layer* which provide the database interfaces of SEISs for storing the data in both remote and local (located in the field of interest) databases via *MRemoteDB* and *MLocalDB* respectively. These modules belong to the sensor-actuator

and information control centre subsystems which are accessed through *IFDataAccess* and *IRDataAccess* interfaces. The communication subsystem is described explicitly through the use-dependency relationship between the modules of different layers and implicitly through the dependencies between the modules belong in the same layer. Modules in the specific layer have the same abstraction hence they should depend on other modules in the same layer and can communicate with modules in the layer directly underneath through interfaces. The separation of these modules is logical rather than physical, such that multiple modules may co-exist on the same physical node and also single module can be distributed on multiple physical nodes. This view helps to achieve maintainable SEISs and utilization of *MPowerSaving* module facilitate the development of energy efficient SEIS. This module configuration is described using packages for layers, while the modules are demonstrated using stereotyped classes. Each module of the module view of RefSEISs can be instantiated during the design of concrete SEISs software architectures. Besides, modules can be omitted depending on the goals of the system. This design helps to reduce system complexity.

7.2.3 Execution View

The execution view assigns the corresponding modules from the module view described in Section 7.2.2 into runtime entities and allocating them to the hardware. This view conforms to the execution viewpoint described in Section 6.3 while addressing the following functional architectural requirements; FN1: Data Acquisition, FN2: Storage, FN3: Phenomena Analysis, FN4: Visualization, FN5: Phenomena Control, FN6: Sensor Management, and FN9: Actuator Management.

According to global analysis, this execution view supports maintainability by utilizing the high cohesion and low coupling strategy for minimizing dependencies among processes and resource utilization. This is reflected through the use of client-server architecture style as inherited from the module view such that a single-server and base station processes are used to serve many clients. This minimizes the number of executable processes and associated context switches. These strategies have been adopted from the successful implementation of the existing SEISs [Greenwood et al., 2006, Seal et al., 2012, Mirchandani and Head, 2001, Curiac and Volosencu, 2010, Jadhav and Deshmukh, 2012, Alasia, 2013, Raju, 2014, Kotta et al., 2011]. As depicted in Figure 7.3, the execution views of the most SEISs possess the same basic structure for their execution configurations.

In sensor-actuator subsystem, the field devices include *SensorNode*, *Actuator*, *BaseStation*, *LocalDBServer* and *FUserComputer*. Such devices contain *SensorNodeManager*, *ActuatorManager*, *GatewayApplicationContainer* and *WebBrowser* processes respectively

which are executed in their corresponding operating systems. Each process is associated with one or more modules as described from the module view in Section 7.2.2. The *SensorNode-Manager* process deploys all the modules of sensor nodes, i.e. *MSensing*, *MSensorManager*, and *MPowerSaving*. The modules of actuators, i.e. *MAProcessing* and *MActuating* are deployed in the *ActuatorManager* process, while that of *BaseStation* i.e. *MAggregator*, *MFProcessing*, and *MTopology* are deployed in the *GatewayApplicationContainer* process and *WebBrowser* process is used for accessing the system locally in the field. Both *SensorNodeManager* and *ActuatorManager* processes communicate to the higher level nodes mainly a *GatewayApplicationContainer* process through wired or wireless communication mechanisms. The *LocalDBServer* process deploys *MLocalDB* module for storing data locally in the field.

The information control centre subsystem is deployed into *ApplicationServer*, *DBServer* and *RUserComputer* devices which consist of one or more *WebBrowser*, one *WebContainer*, *AppContainer* and *DBServer* processes. These processes are also associated with their corresponding modules. The *WebContainer* process handles the requests of users on the server's side by deploying *MController* and *MViewing* modules. The *WebBrowser* process deploys *MUserInterface* module to allow users to access the system. The *AppContainer* process is associated with *MAnalyser*, *MServerProcessing*, and *MControlDispatcher* modules for executing the intended business logic of the system. The *DBServer* process deploys *MRemoteDB* module. The Application Server is connected to field nodes through *BaseStation*. Finally, the communication subsystem is reflected explicitly through the communication paths: *SensorConnection*, *ActuatorConnection*, *BSConnection*, *UserConnection*, and *DB-Connection*. This execution view can be supplemented with other diagrams, i.e. sequence diagrams for describing the sequence of operations or other changes in the configuration.

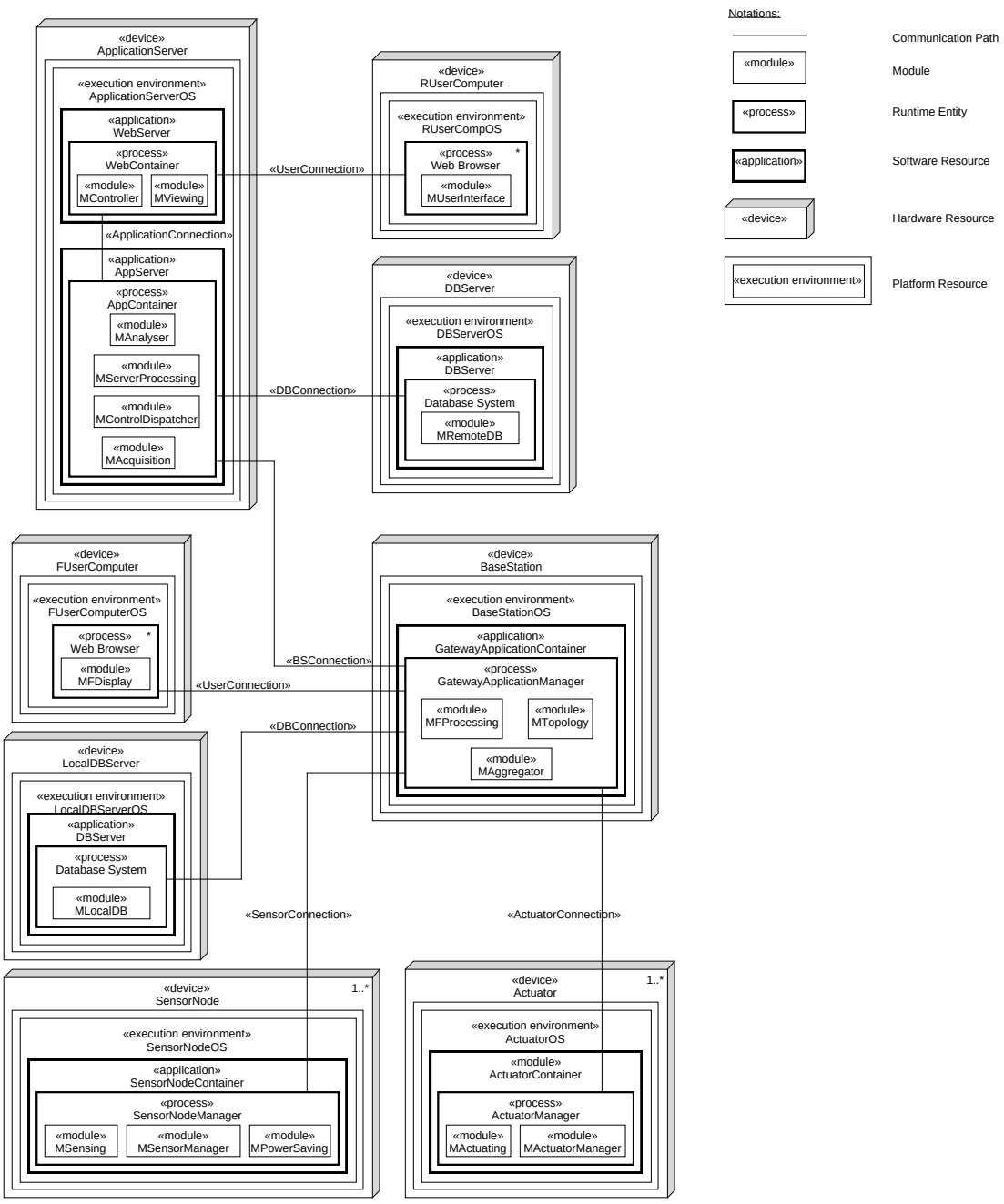


Figure 7.3: Execution View of RefSEISs

7.2.4 Code View

The code view describes how the system modules and interfaces presented in Section 7.2.2 are partitioned into source files. Then these source files are organized into directories. This view originated from the application of code viewpoint described in Section 6.4. The code

view describes how the software of SEIS is implemented to perform the specified functional architectural requirements: FN1: Data Acquisition, FN2: Storage, FN3: Phenomena Analysis, FN4: Visualization, FN5: Phenomena Control, FN6: Sensor Management, and FN7: Actuator Management.

The code view presents the actual modules that should be implemented in a particular programming language. Such a view indicates how the system is built out of implementation artefacts, such as modules, tables, classes, etc. This view is useful in guiding software experts mainly developers and maintainers during the software implementation and maintenance respectively. Although, the code view is usually product-specific, for the case of RefSEISs, this view present high level of abstractions. The source files and directories are represented using components and package notations and are also associated with their corresponding stereotypes to clarify their meanings. According to global analysis, the source files have been organized while maintaining the primitive architecture styles that have been used initially in the module view of RefSEISs. This view unifies the configurations of the existing SEISs [Zhang et al., 2008, Nithya and Vanamala, 2018, Greenwood et al., 2006, Seal et al., 2012, Mirchandani and Head, 2001, Curiac and Volosencu, 2010, Hartung et al., 2006, Guthi, 2007, Jadhav and Deshmukh, 2012, Alasia, 2013, Raju, 2014, Kotta et al., 2011, Huang et al., 2015, Zhou et al., 2010, Sunkpho and Ootamakorn, 2011, Udo and Isong, 2014, Lozano et al., 2012, Nguyen et al., 2015, Ujang et al., 2013, Ramesh, 2014] while reflecting the specified modules described in Section 7.2.2.

The directory structure for the code view of the RefSEISs is shown in Figure 7.4 follows the layering and module decomposition to the subdirectories for subsystems: sensor-actuator and Information Control Centre. Each major directory contains all the modules for the process as specified in the execution view in Section 7.2.3 as well as enforcing the design constraints or usage dependencies established in module view as described in Section 7.2.2. The sensor-actuator subsystem consists of *Gateway*, *LocalDBServer*, *SensorNode* and *Actuator* which are made up of source codes, library and configuration files for pulling up the appropriate files, packages and system services as required. The *Gateway* contains the *GSource* directory which is made up of *GatewayApp* source files for implementing the gateway application, *GConfig* configuration files for configuring the gateway and *Communication* library for facilitating communication between the gateway and other nodes. The *LocalDBServer* directory contains source code for the local database to be used in the field. The *SensorNode* contains the *SensorSketch* source file, and *Loggings* configuration files for implementing the sensor node manager and configuring the sensor node respectively, while *SensorLib* and multiple *Communication* library files are used for managing the sensors and communication with other nodes. Similarly, the *Actuator* contains the *ActuatorSketch* source, *Communication*

library and *Loggings* configuration files for implementing the actuator manager, managing the communication with other nodes and configuring the actuator.

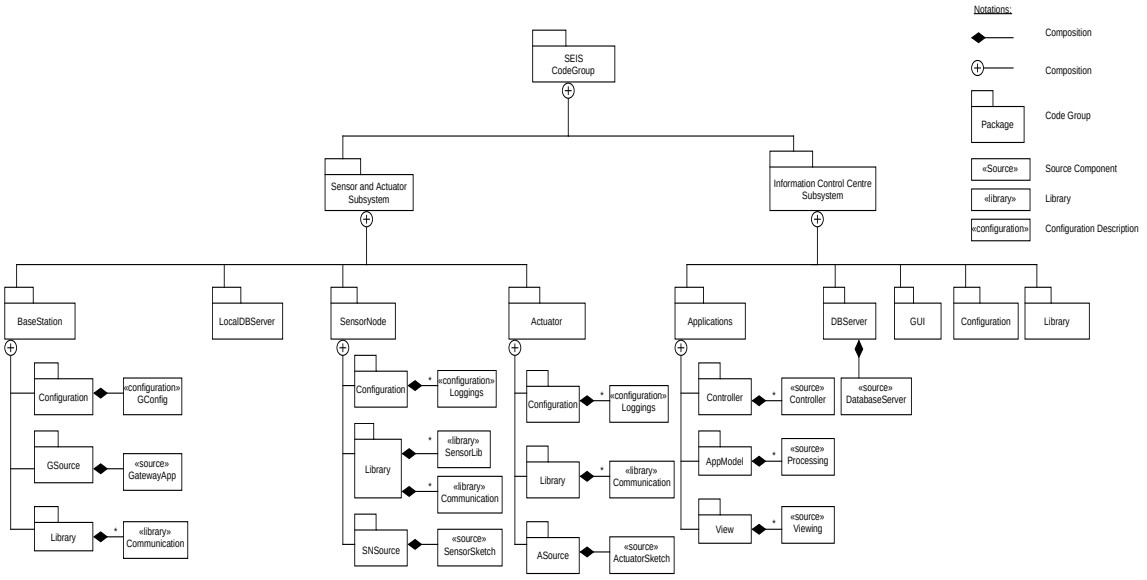


Figure 7.4: Code View of RefSEISs

The information control centre subsystem directory is organized based on the layering structure of the module view as described in Section 7.2.2. Such directory contains a set of source components for the intended applications, data and user access services as well as configurations through *Applications*, *DbServer*, *GUI* and *Configuration* directories respectively. In which the *Applications* directory reflects MVC by having *AppModel*, *View* and *Controller* directories for implementing the processing tasks or business logic of the system and processes handling user inputs. The *DBServer* directory contains *DBSource*, for implementing the database. The *Configuration* for configuration files of information control centre subsystem. The communication subsystem is not explicitly expressed but is included in both sensor-actuator and information control centre subsystems directories.

7.2.5 Topology View

The topology view provides an overview of the network topology of the RefSEISs by describing the relationship and possible connections between nodes of the system. The topology view demonstrates the configuration of physical resources as described by the execution view in Section 7.2.3 while addressing the specified functional architectural requirements such as FN1: Data Acquisition, FN5: Phenomena Control, FN6: Sensor

Management, and FN7: Actuator Management. The topology view resulted from the application of topology viewpoint described in Section 6.5.

The topology view describes the configurations, possible connections and distribution of SEISs nodes mainly *ApplicationServer*, *RUserComputer*, *DBServer*, *LocalDBServer*, *FUSerComputer*, *BaseStation*, *SensorNodes*, *Sensors*, and *Actuators*. The *LocalDBServer*, *FUSerComputer*, *SensorNodes*, *Actuators*, and *BaseStation* are deployed in the field of interests. Given that these *SensorNodes* possess limited energy and computation resources. The energy costs of data transmission of these sensor nodes are usually higher than that of computation [Ganesan et al., 2004]. This requires maximization of local computation while minimizing the transmission of raw data [Sabit et al., 2009]. There are various mechanisms for reducing the power consumption of sensor networks based on the taxonomy of energy consumption sources of WSN as shown in Figure 7.5 [Abdelaal, 2015]. To mention the few, such mechanisms have been used to optimize the energy consumption of sensor networks in various SEISs, i.e. tree-topology clustering control management [Zhang et al., 2008, Jadhav and Deshmukh, 2012, Liyang et al., 2005], routing protocols [Jiménez and García, 2015, Castillo-Effen et al., 2004], clustering algorithm for routing in real-time forest fire detection system [Liyang et al., 2005], state switching through adaptive duty cycling mechanisms [Jeličić et al., 2011, Kumar and Kishore, 2017, Marin Perez et al., 2012], and phenomena detection that uses threshold in aggregating the observed data [Ramesh, 2014].

In the context of RefSEISs, the state switching using an adaptive duty cycling and topology control process using network clustering tree-topology mechanisms are considered to be the most effective power consumption optimization techniques. Such that the network cluster tree-topology architecture style is used in the configuration of the sensor-actuator subsystem as shown in Figure 7.6 with three-tier nodes; end-nodes, routers and coordinators as adopted from [Zhang et al., 2008, Jadhav and Deshmukh, 2012]. The sensor nodes are end-nodes deployed in the field of interest collect data and then transmit the collected data to other *SensorNodes*. The *SensorNodes* which collect data from other *SensorNodes* are considered to be routers that transmit the data packets to the *BaseStation*. The *BaseStations* are coordinators which act as gateways between wireless sensor networks and wired network. The *BaseStation* concerns with data filtering, integration, storage of data locally in *LocalDBServer* and transmission of data packets to the *ApplicationServer* in the information control centre subsystem for further processing. These coordinators (*BaseStations*) configure and manage the network to the lower nodes, i.e. routers and end nodes. The control commands are flowing from the *ApplicationServer* in the information control centre subsystem to the *BaseStation* in the sensor-actuator subsystem and then executed by *Actuators* as end nodes. The *BaseStation* is responsible for managing both *SensorNodes* and *Actuators*. The *BaseStation* is also

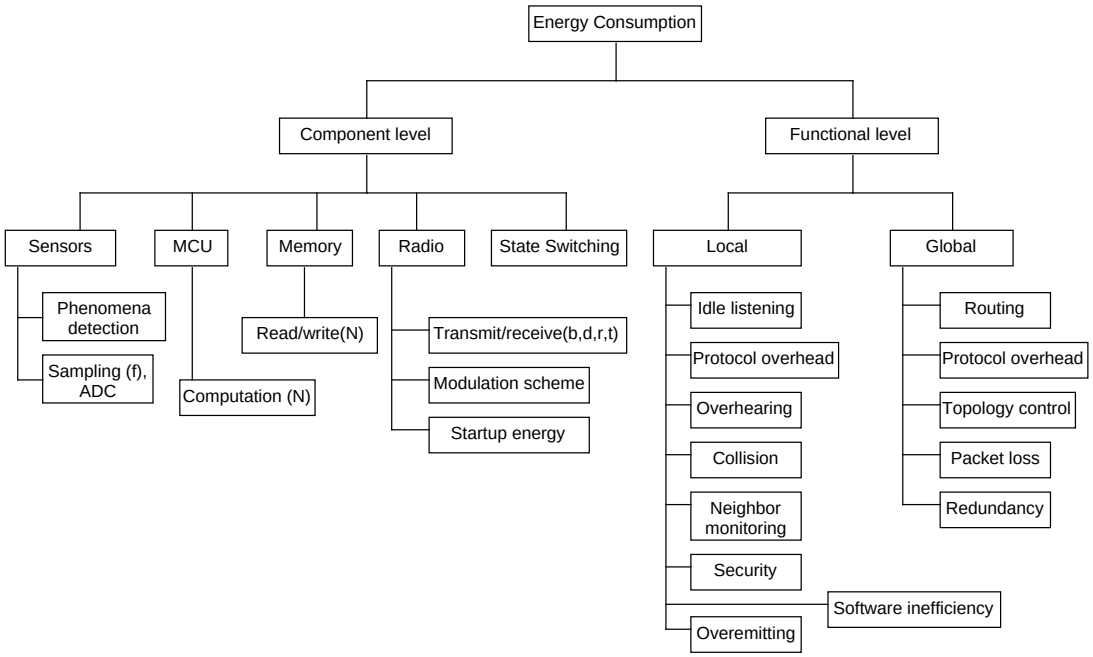


Figure 7.5: Taxonomy of energy consumption sources in WSNs from [Abdelaal, 2015].

connected to multiple *FUserComputer* for allowing the user to access the system in the field. In the information control centre subsystem the *ApplicationServer* processes and stores data in the *DBServer*, and supports queries from users through multiple *RUserComputers*.

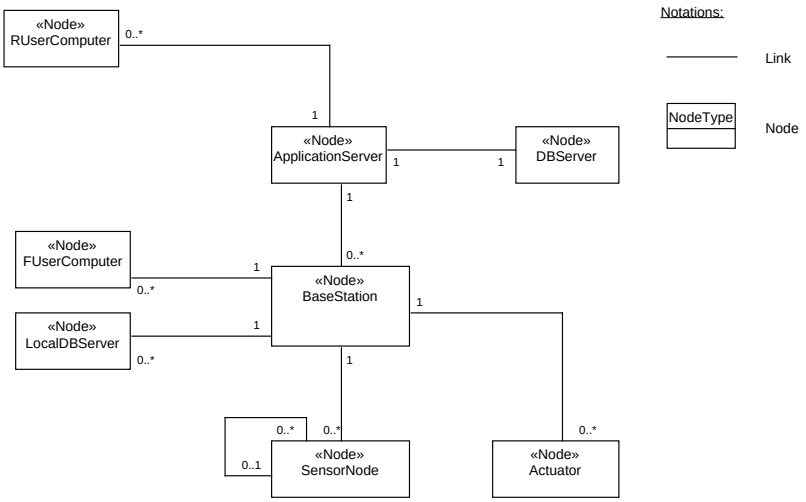


Figure 7.6: Topology View of RefSEISs

7.2.6 Data View

The data view describes the data entities and their relationship involved in SEISs from how data is created, accessed, updated, transmitted and stored. The data view demonstrates the static data structure of each software implementation module that is originated from the module view presented in Section 7.2.2 while addressing the specified functional architectural requirements such as FN1: Storage, FN6: Sensor Management, and FN7: Actuator Management. The data view resulted from the application of data viewpoint described in Section 6.6.

Additionally, this data view intends to show how the data collected by various heterogeneous sensor nodes. This is achieved through the adoption of Key-Value type table architectural style from [Kim et al., 2013] in association with sensing and actuating profiles, hence enable the seamless integration of various data types. This data view has substantial impacts on both maintainability and interoperability. Such a view is described based on the module view provided in Section 7.2.2, in which the communication between modules or subsystems is associated with the exchange of information. Such information is expressed in terms of classes which are organised in packages of their corresponding module as shown in Figure 7.7.

In sensor-actuator subsystem, the *MSensing* contains a *Sensor* class for storing information about the end nodes (sensors) which sense the environmental parameters and a *Measurement* for storing those measured parameters. The *MSensorManager* uses a *SensorNode* as a data source for storing information about the sensor node and the *MPowerSaving* uses *SensingControl* for imposing duty cycling on the operations of sensor nodes. The *MAggregator* is associated with *Aggregate* for measurements (aggregates) that have derived from various sensor nodes through multi-hoping such that the sensor nodes near the base station send the information collected by other sensor nodes. Both *Measurement* and *Aggregate* store the observed data using Key-Value type table architecture style in which the data declared of type any to accommodate any type of data. The base station is responsible with the field processing, contains a gateway which sends the data collected from various sensor nodes to the information control centre and receives the control instructions from the information control centre and then executes them through actuators. The *MFProcessing* is associated with *Gateway* as a data source for storing information about the gateway and *FusedData* for storing integrated information derived from various sensor nodes. Both *Aggregate* and *FusedData* support the storage of any data type of sensor nodes (observed environmental parameter) to be stored in the database without further modifications. A base station requires a *DHandler* to connect to the *IFUserAccess* for accessing the system locally in the field

through *MFDisplay* which is associated with *FDisplay* as data entity for storing information about the display. Similarly, a base station requires a *DBHandler* which has the database connection information in the *IFDataAccess* to access the database *MLocalDB* using *LocalDB* as a data entity for storing information about the local database. The *MActuatorManager* uses *Actuator* as a data entity for storing information about the actuator and execute control actions implemented by *MActuating* which uses *Task* for storing the information about the control actions.

In the information control centre subsystem, the *MServerProcessing* responsible with the business logic of an application through a models which encapsulate methods to access and manipulate data i.e., databases, files, etc. The information about models is provided by a *Model*. A *Model* utilizes an *Acquisition* to store information about the observations received from various gateways in the fields. The *Model* requires the *DBHandler* which stores the information about the connection in the *IRDataAccess* to access the database *MRemoteDB*. The *MRemoteDB* is associated with the *RemoteDB* as a data source for storing information about the database. The *MAnalyser* uses *Analysis* for storing the analysed data. The *MControlDispatcher* contains *ControlDispatcher* and *ControlInstruction*. The *ControlDispatcher* data source is used for storing information about the dispatcher of the control instruction from the information control centre subsystem to the sensor-actuator subsystem. The *ControlInstruction* is used for storing information of the control instructions. The *MViewing* controls the way data is displayed and how the user interacts with the system. The *MViewing* is associated with *View* data entity for storing information about the views. The *MController* is responsible for handling the events which are triggered by either a user or a system. The *MController* contains *Controller* data entity for storing information about the controller. The controller accepts requests using a *Request* that stores information about the requests in the *IRUserAccess* from the user via a *MUserInterface*. The *MUserInterface* has *User* which provide the information of the user. Similarly, the controller sends response which its information is stored in the *Response* in the *IUserAccess* to the user. The *MController* interacts with both *MServerProcessing* and *MViewing* for retrieving the required data and generating views respectively.

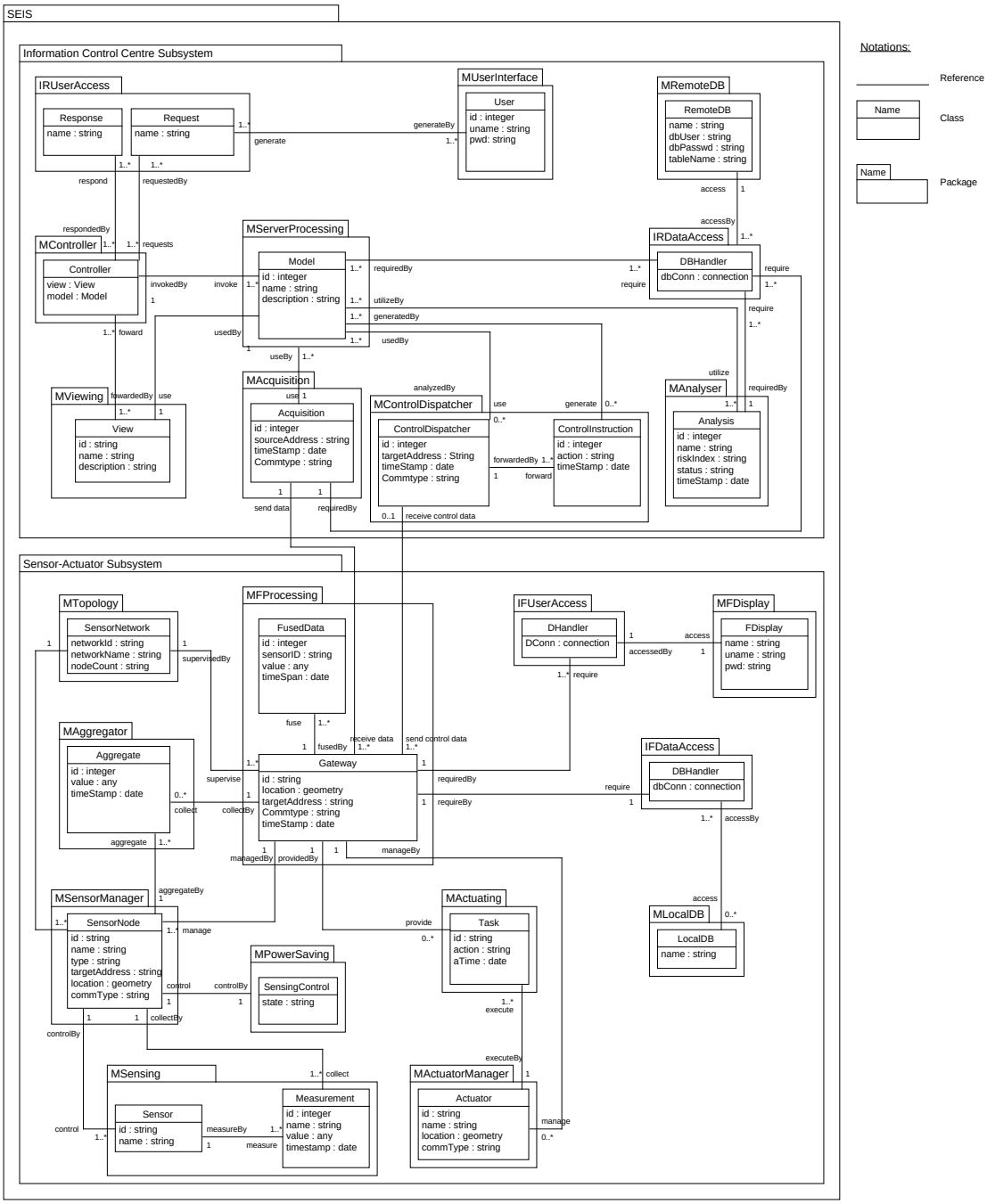


Figure 7.7: Data View of RefSEISs

7.3 Use of RefSEISs

The RefSEISs encompasses the architectural knowledge and artefacts mainly a set of stakeholders, concerns, requirements and designs that can be reused in the construction of new

SEISs, comparing and improving the existing SEISs. The elements of architecture knowledge and artefacts of RefSEISs are either variable or common elements; hence they can be changed, deleted or unchanged to satisfy the needs of a particular scenario. However, this section demonstrates the essential steps towards the construction of new concrete SEIS using RefSEISs (an instance of the reference architecture for SEISs) as a baseline:

1. Description of the business case that adopts the domain model of SEISs.
2. Description of the stakeholders and their concerns for an instance refer to Section 5.1.
3. Description of the functional and non-functional requirements that should be addressed by the resulted architecture instance refer to Section 5.2.
4. Description of the global analysis to identify the potential architectural factors and strategies that have a strong influence over the quality of architectural design 7.1.
5. Selection of the viewpoints and then design the views for this new architecture instance refer to Chapter 6 and Section 7.2 respectively.
6. Evaluation of the resulted architecture instance as described in Chapter 10.

7.4 Summary

In this chapter, the architecture views of RefSEISs have been derived from the architecture viewpoints described in Chapter 6 while addressing the stakeholder concerns and requirements specified in Chapter 5. These views are abstract such that they do not delve into specific implementation details. This will be handled while describing the software architectures instances of the RefSEISs, in which the concepts defined in the reference architecture will be refined, extended or specialized according to the problem at hand. The views of RefSEISs adopts some of the essential architectural styles from the existing SEISs to achieve some critical quality attributes. The proposed RefSEISs can be used as a baseline for improving and comparing the existing SEISs and designing a new concrete SEIS by concretising the software elements. In Chapter 8, a proof of concept will be demonstrated to prove the applicability of the proposed RefSEISs in construction of concrete SEIS. And Chapter 9 demonstrates the mapping of the RefSEISs to the existing SEISs.

Part IV

Evaluation

Part IV demonstrates the application of the proposed RefSEISs on real-world systems. The proposed RefSEISs is first applied in the construction of the concrete architecture of SEISs. A forest fire detection system is used as a case of study as described in Chapter 8. In which the requirements of such a system are identified while referring to the defined requirements of the RefSEISs. Then to fulfil such requirements, an architecture of forest fire detection system is constructed as referred to the proposed RefSEISs. Similarly, to demonstrate the applicability of the proposed RefSEISs on the existing SEISs. Some of the existing systems such as the Forest Fire Monitoring System (IPNAS) [Stipanicev et al., 2018], Urban Air Quality Monitoring System [Ujang et al., 2013], Flood Risk Assessment System [Amirebrahimi et al., 2016] and Indoor Air Quality Monitoring System [Abraham and Li, 2016] are mapped onto the RefSEISs as described in Chapter 9. Finally, the RefSEISs is verified by checking the fulfilment of the specified requirements by the proposed designs in Chapter 10.

Chapter 8

Application: Forest Fire Detection System

In this chapter, a case study is presented to demonstrate the application of the proposed RefSEISs on real-world systems. As a proof-of-concept, the proposed RefSEISs from the Part III is used to build an architecture of a Forest Fire Detection System (FFDS) as an example of how the proposed architecture RefSEISs can be applied in the construction of new concrete SEIS. This chapter consists of three sections; Section 8.1 provides an introduction about the forest fire detection system. Section 8.2 presents an architecture for forest fire detections system, while the architecture implementation is presented in Section 8.3. Finally, Section 8.4 provides the overall summary of the chapter.

8.1 Introduction

As explained in Section 3.2, forest fires are one of the greatest evils happening in the world today which cause significant economic damage, loss of lives of living organisms, climate change and destruction of properties. The combination of both early detection of forest fires and application of rapid and appropriate intervention is crucial for the forest fire damage minimization. This is where SEISs are required to monitor remotely potential forest fire parameters such as humidity, temperature, smoke, etc. to control and prevent the occurrence of forest fires. The development of forest fire detection system begins with the architectural description of the system.

8.2 Software Architecture

The architecture design of forest fire detection system follows the description of the RefSEISs proposed in Part III. In this section, the description of the architecture of forest fire detection system is provided starting with the stakeholders and concerns in Section 8.2.1, followed by the establishment of the requirements in Section 8.2.2, and then the design of architecture views in Section 8.2.3.

8.2.1 Stakeholders and Concerns

The stakeholders and concerns of forest fire detection system are deduced from the RefSEISs as detailed in Section 5.1 which include development experts (developers, maintainers, testers, architects), acquirers, operators, and users mainly firefighters, and residents. These stakeholders and their concerns are as follows:

A. Acquirers

Acquirers initiate, develop, manage and maintain the whole project of development and maintenance of forest fire detection systems. These stakeholders include fire agencies, non-governmental organizations, forest management groups, landowners, forest fire researchers, government and national authorities. These stakeholders concern with improving public safety, early detection of forest fires, reduce costs of controlling forest fires and improve the information to predict wildfire behavior and allocation of adequate firefighters to fight forest fires.

B. Development experts

The development experts include developers, testers, sensor-actuator experts and architects, are responsible for the actual development of forest fire detection system. Hence they are concerns with data modeling and access, development, code management and organization, effective topology configurations of field nodes and installation of the system on site, etc.

C. Operators

Operators include forests operators are responsible for providing and supporting the daily operations of the system. Hence they are interested with managing and ensuring the fulfillment of their services. The concerns of these stakeholders include; data acquisition, storage, access and transmission, code management and organization, topology configurations of field nodes and installation of the system on site, etc..

D. Users

Users include firefighters, and residents. The users are interested with the use of the system for getting information on forest fires for both early detection of fires, well-allocation of firefighters, and effective evacuation of residents in case of fire.

8.2.2 Requirements Establishment

After the identification of stakeholders and their concerns, the functional and non-functional requirements of the system are identified. These requirements are defined based on this work motivation and preliminaries of the existing systems which serve as the critical inputs for the software architecture development.

Functional Requirements

The functional requirements of forest fire detection system are deduced from the functional requirements of RefSEISs described in Section 5.2.1. In which five primary functions of the system are identified;

FFN1 Data Acquisition: The system must be able to acquire data from sensors spread over the forest.

FFN2 Data Storage: The system must be able to store collected data.

FFN3 Forest Fire Analysis: The system must be able to use data processing methods based on models of prediction and approximation algorithms to predict the occurrence of fire and generating fire alarms.

FFN4 Visualization: The system must be able to display or visualize data.

FFN5 Sensor Management: The system must be able to configure, manage, and display the nodes deployed in the forest.

Non-Functional Requirements

The development of a fully functional forest fire detection system requires the following non-functional requirements or challenges to be considered explicitly as derived from the non-functional requirements of RefSEISs identified in Section 5.2.2;

FNFN1 Energy Efficiency: The system should be energy efficient. Since the battery-powered nodes are involved in continuous monitoring of forest parameters. The

deployment of these nodes in the forest make it hard and infeasible to be replaced or recharged often.

FNFN2 Maintainability: The system should be easily maintainable to avoid additional burden of human involvement in the forest.

FNFN3 Interoperability The system should be interoperable to support the integration of data originated from various sources or nodes.

8.2.3 Architecture Views

The design of forest fire detection system is a multi-view perspective as referred to the architecture views of RefSEISs proposed in Section 7.2. In which, the abstract views of RefSEISs have been concretised to satisfy the needs of forest fire detection system. As described in Section 1.1, some elements are common while others are variable.

Conceptual View

The conceptual view of forest fire detection system refers to the conceptual view of the RefSEISs presented in Section 7.2.1. In which the functional entities of the system and their relationships with each other are described in terms of components and connectors. This view addresses the concerns of acquirers, users and development experts by describing how the functionalities of the systems are fulfilled, and the system is decomposed.

The conceptual view addresses the following requirements of the system; FFN1: Data Acquisition, FFN2: Storage, FFN3: Forest Fire Analysis, FFN4: Visualization, FFN5: Sensor Management. This is achieved through the configurations of *SensorNode*, *BaseStation*, *Forest Fire Detection System Server*, *Client* and *RemoteDB* components. In which the Client-Server architecture style, central controller component, and decouple the user interaction model strategy through MVC pattern are employed as adapted from the RefSEISs. As depicted in Figure 8.1, in which the *data*, and *control* connectors in both sensor-actuator and information control centre subsystems are concretized into various types of connectors such as *IPC*, *USB*, *HTTP*, *MySQL*, *UART* and *1-Wire* to reflect the concrete communication mechanisms mainly the protocols to be obeyed by each of the ports or roles. Some of the components specified in the conceptual view of the RefSEISs are common components which have been adapted as they are in both sensor-actuator and information control centre subsystems, i.e. in *Acquisition*, *SProcessing*, *Aggregator*, *BSDisplay*, *Analyser*, *Viewing* etc. Other components have been removed mainly the functionalities of actuators, since this system does not involve actuators.

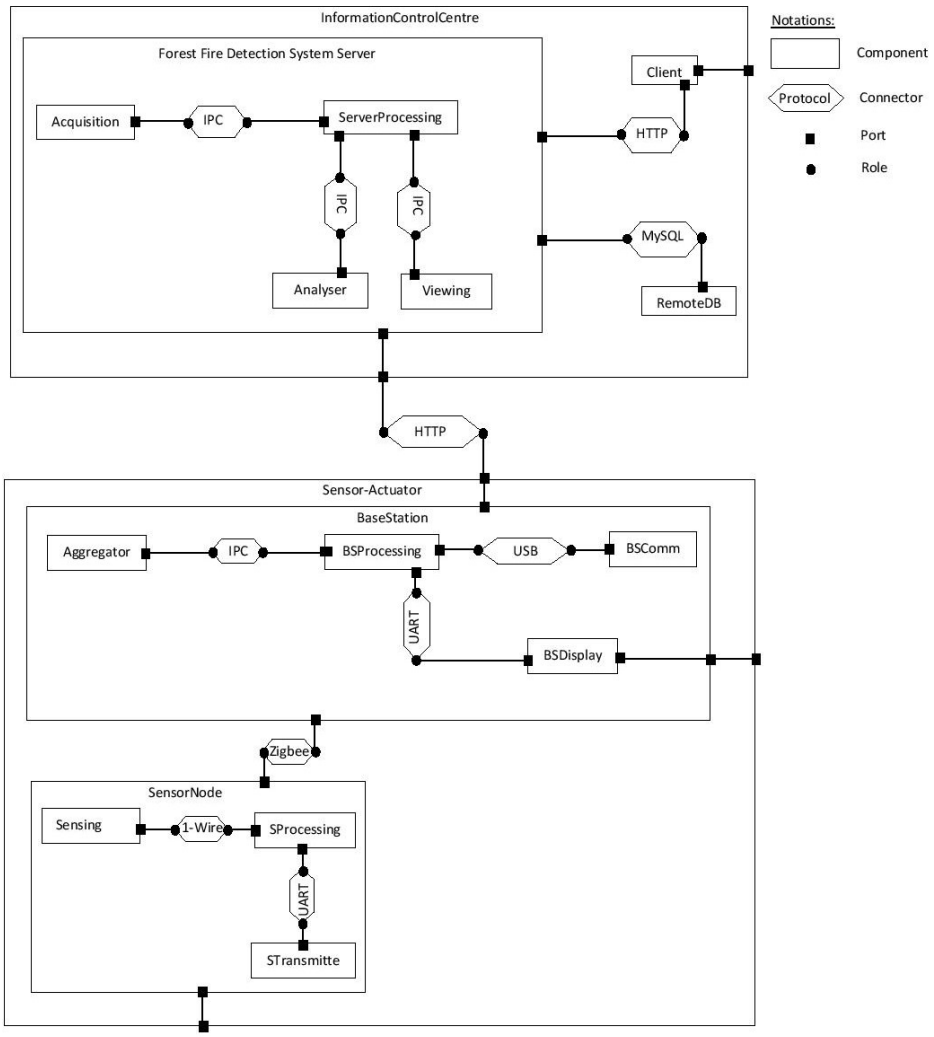


Figure 8.1: Conceptual View

The sensor-actuator subsystem is decomposed into two main components; *BaseStation*, and *SensorNode*. The *BaseStation* communicates with *SensorNodes* through *Zigbee* connector. The *BaseStation* manages both sensor nodes in the forest. The *BaseStation* integrates data from various *SensorNodes* and then transmits the collected data to the information control centre subsystem for further processing. The *SensorNodes* measures the environmental parameters of the forest i.e. temperature, humidity, smoke, etc.. With the application of decouple the user interaction model strategy and separation of concerns; the *BaseStation* is further decomposed into *Aggregator* for acquiring data from sensors, *BSProcessing* for local processing through *IPC* connectors, *BSDisplay* for allowing the user (mainly an operator) to view data, as well as control data acquisition, and *BSComm* is used for transmission of

the collected data or receive information to and from sensor nodes and information control centre subsystem. The *BaseStation* defines and sets the policies for collecting and processing the data (environmental parameters). The *SensorNode* is further decomposed into *Sensing*, *SProcessing* and *STransmitter* using *1-Wire* and *UART* connectors for collecting, processing and transmitting the observed environmental parameters to the *BaseStation*.

The information control centre subsystem employs a client-server architecture style, central controller component, and decouple the user interaction model strategy through Module-View Control (MVC) pattern. A user interface is expressed as a *Client* component as an operator uses a port to send requests to the *Forest Fire Detection System Server* component as the controller. The interaction between the *Client* and *Server* components is facilitated by *HTTP* connector. The *Server* component uses another port to access the model in the *ForestDB* component via *MySQL* connector while processing the requests of the clients and send them back as views. A *Server* is further decomposed into *Acquisition*, *ServerProcessing*, *Analyser* and *Viewing*. The *Acquisition* for receiving the observed information from the forest. The *ServerProcessing* processes and analyses the observed information through *Analyser*. The *Viewing* generates potential views for displaying information as required by clients.

The communication subsystem is demonstrated explicitly within both subsystems (sensor-actuator subsystem and information control centre subsystem) through connectors which encompass protocols to be obeyed by each of the ports or roles. This conceptual configuration is described using UML Class diagram using special graphical symbols for the component, connector, role and port to make the description more clear, easily and natural to read.

Module View

The module view of forest fire detection system refers to the module view of the RefSEISs presented in Section 7.2.2. This view addresses the concerns of operators and development experts by describing the software implementation modules of the system. This view maps the elements of the conceptual view into the implementation modules as shown in Table 8.1.

The module view addresses the following requirements of the system; FFN1: Data Acquisition, FFN2: Storage, FFN3: Forest Fire Analysis, FFN4: Visualization, FFN5: Sensor Management. This is achieved through the configurations of software implementation modules as depicted in Figure 8.2. In which the module view of forest fire detection system is designed as a three-layered style; GUI, Application, and Data Access. The strategies; MVC with a central controller component and modules with low coupled and high cohesion are adopted as well. The first layer, *GUI Layer* incorporates visualization modules which provide and receive information to and from users using *MFDisplay* and *MUserInterface* at

Table 8.1: Mapping between Conceptual and Module Architecture Views

Conceptual Element	**Module or Subsystem**
Sensing	MSensing
SProcessing, STransmitter, 1-Wire and UART connectors	MSensorManager, MPowerSaving,
Aggregator	MAggregator
BSProcessing, BSComm, IPC, UART and USB connectors	MTopology, MFProcessing
Analyser	MAnalysis
Viewing	MViewing
DataAcq, ServerProcessing, and IPC connectors	MServerProcessing, MAlerting
RemoteDB	ForestDB
BSDisplay	MFDisplay
Client	MUserInterface
UAccess	IUserAccess
DBAccess	IDataAccess

the forest and in the information control centre respectively. The business logic modules of the forest fire detection system are situated in the *Application Layer* which consists of the application server with *MServerProcessing*, *MController*, *MViewing*, *MAlerting* and *MAnalysis* modules in the information control centre subsystem for integrating, analysing, alerting and predicting the potential occurrence of forest fires, while *MSensorManager*, *MPowerSaving*, *MAggregator*, *MAcquisition* and *MFProcessing* modules are situated in the sensor-actuator subsystem for collecting data in the field. All these are common modules adopted as they are, while the *MRemoteDB* in the *DataAccess Layer* of RefSEISs is concretised into the *ForestDB* for storing data collected from the forest. The *MPowerSaving* module is associated with a duty cycle mechanism to reduce the power consumption of sensor nodes and hence extends the life cycle of sensors. The communication between layers are facilitated by *IDataAccess* and *IUserAccess* interfaces. Both sensor-actuator and information control centre subsystems belong to the forest fire detection system.

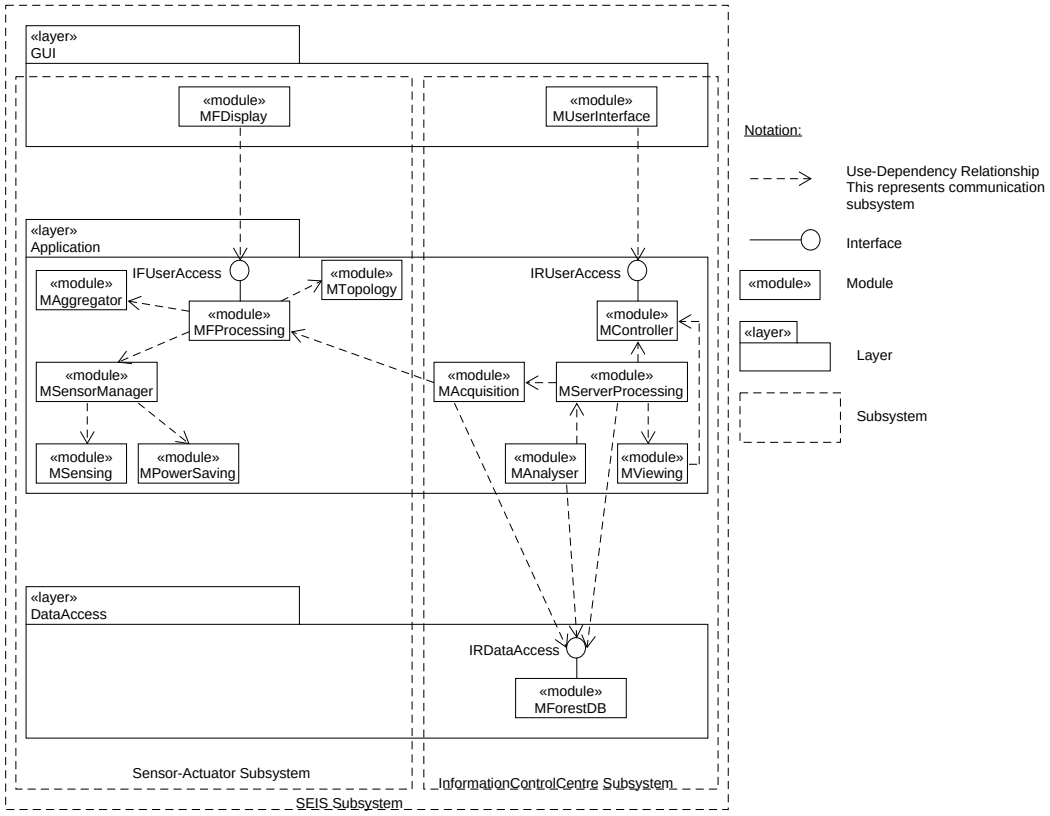


Figure 8.2: Module View

Execution View

The execution view of forest fire detection system refers to the execution view of the RefSEISs presented in Section 7.2.3. In which the corresponding modules from the module view are assigned to the runtime entities and their allocation to the hardware. This view addresses the concerns of operators and development experts and the specified architectural requirements of the system by describing how the system is integrated, components of the system interact and deployed.

As depicted in Figure 8.3, the execution view supports maintainability by utilizing the strategies inherited from the module view such as client-server architecture style in which a single-server and base station processes are used to serve many clients. This minimizes the number of executable processes and associated context switches. Most components of the execution view of the RefSEISs are variable components which have been refined to reflect the concrete components of the system. For instance, the *ApplicationServer*, *ApplicationServerOS*, *AppContainer*, *DBServer*, *SensorNodeOS*, and *GatewayApplication-*

Container of RefSEISs are replaced by *FFDSServer* (Forest Fire Detection System), *Linux OS*, *FFDSContainer*, *MySQLServer*, *Arduino* and *Raspbian OS* respectively. Also the *UserConnection*, *DBConnection BSConnection*, *ApplicationConnection*, and *SensorConnection* are concretized to *HTTP*, *MySQL*, *HTTP*, *IPC*, and *ZigBee* respectively.

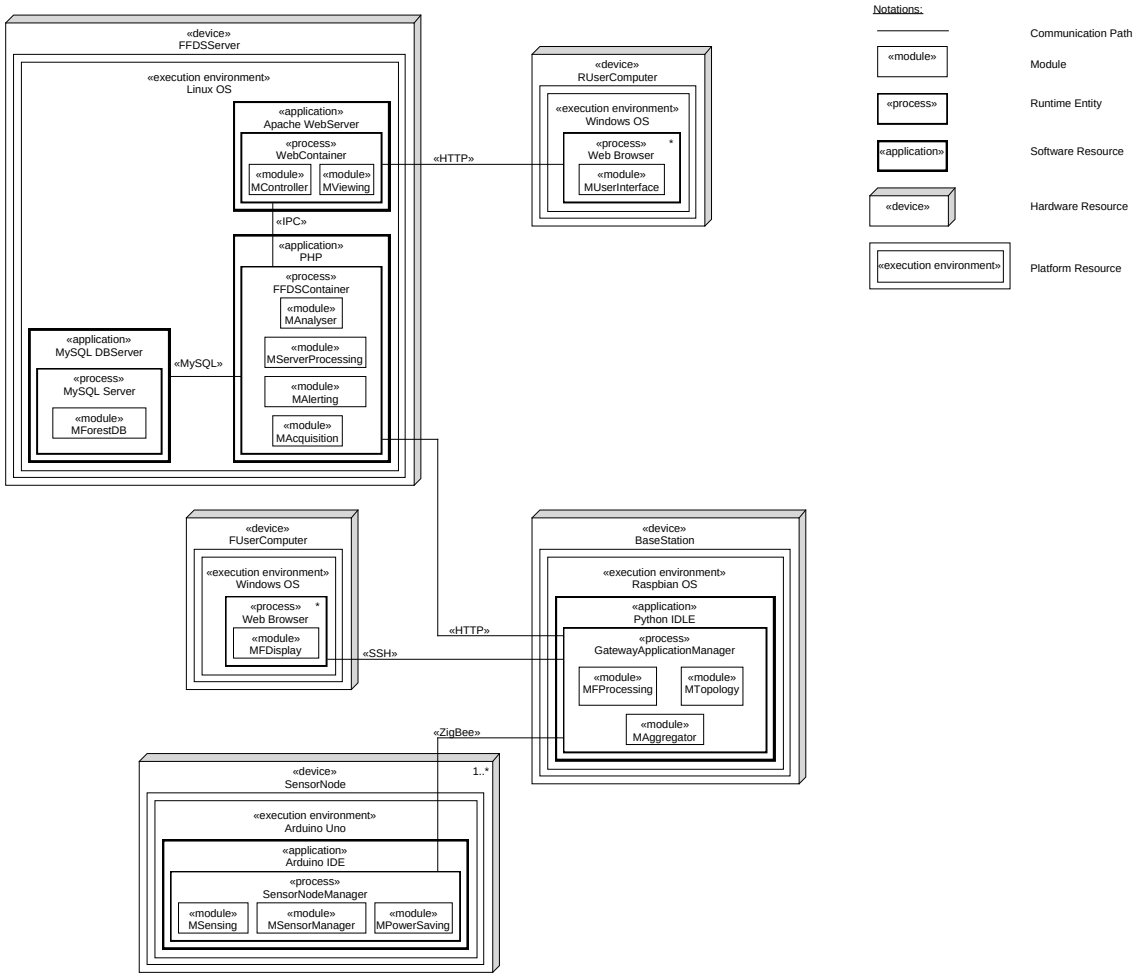


Figure 8.3: Execution View

In sensor-actuator subsystem, the devices deployed in the forest include *SensorNode*, *BaseStation*, and *FUserComputer*. Such devices contain *SensorNodeManager*, *GatewayApplicationContainer* and *WebBrowser* processes respectively which are executed in their corresponding operating systems. Each process is associated with one or more modules as described from the module view in Section 7.2.2. The *SensorNodeManager* process deploys all the modules of sensor nodes, i.e. *MSensing*, *MSensorManager*, and *MPowerSaving* while that of *BaseStation* include *MAggregator*, *MFProcessing*, and *MTopology* are deployed in the *GatewayApplicationManager* process and *WebBrowser* process is used for accessing

the system locally in the forest. The *SensorNodeManager* process communicates to the higher level nodes mainly a *GatewayApplicationManager* process through wired or wireless communication mechanisms.

The information control centre subsystem is deployed into *FFDSServer*, and *RUserComputer* devices which consist of one or more *WebBrowser*, one *WebContainer*, *FFDSContainer* and *DBServer* processes. These processes are also associated with their corresponding modules. The *WebContainer* process handles the requests of users on the server's side by deploying *MController* and *MViewing* modules. The *WebBrowser* process deploys *MUserInterface* module to allow users to access the system. The *FFDSContainer* process is associated with *MAcquisition*, *MAnalyser*, *MServerProcessing*, and *MAlerting* modules for integrating data gathered by *BaseStations*, processing, and analysing possible occurrence of fire as well as alerting the users in case of fire. The *DBServer* process deploys *MForestDB* module. The Application Server is connected to field nodes through *BaseStation*. Finally, the communication subsystem is reflected explicitly through the communication paths: *Zigbee*, *IPC*, *HTTP*, *HTTP*, and *MysQL*.

Code View

The code view of forest fire detection system refers to the code view of the RefSEISs presented in Section 7.2.4. Such a view shows how the modules of forest fire detection system are mapped into system source files. This view addresses the concerns of operators and development experts related to how the system is integrated and identified system modules or components are implemented. The code view shows the directory structure of forest fire detection system as depicted in Figure 8.4 which contain generated codes of the system and runtime loadable files that correspond to the runtime processes in the execution view. Each subsystem (sensor-actuator and information control centre subsystems) has sets of source components.

The code view is usually platform dependent, this system requires multiple programming languages to depict their corresponding classes, interfaces and packages. This is reflected through the variable components of the code view of RefSEISs which were concretized to their corresponding files. For instance, in the sensor-actuator subsystem, the *Gateway* directory consists of *Aggregator.py* the source file for implementing the gateway application. The *xbee_serial_array.h* and *cgi-bin* libraries are used for facilitating communication to the sensor nodes and information control centre subsystem. The *Config.txt* used to store configuration files of the base station. The *SensorNode* directory contains *MyConfig.h* and *arduino.ini* for the configurations of the sensor node. The sensor nodes source files

is *SensorSketch.ino* for implementing the sensor node manager. The *MQ2.h*, *DHT.h* and *XBee.h* are libraries of the sensor node for facilitating communication between smoke, temperature-humidity and base station.

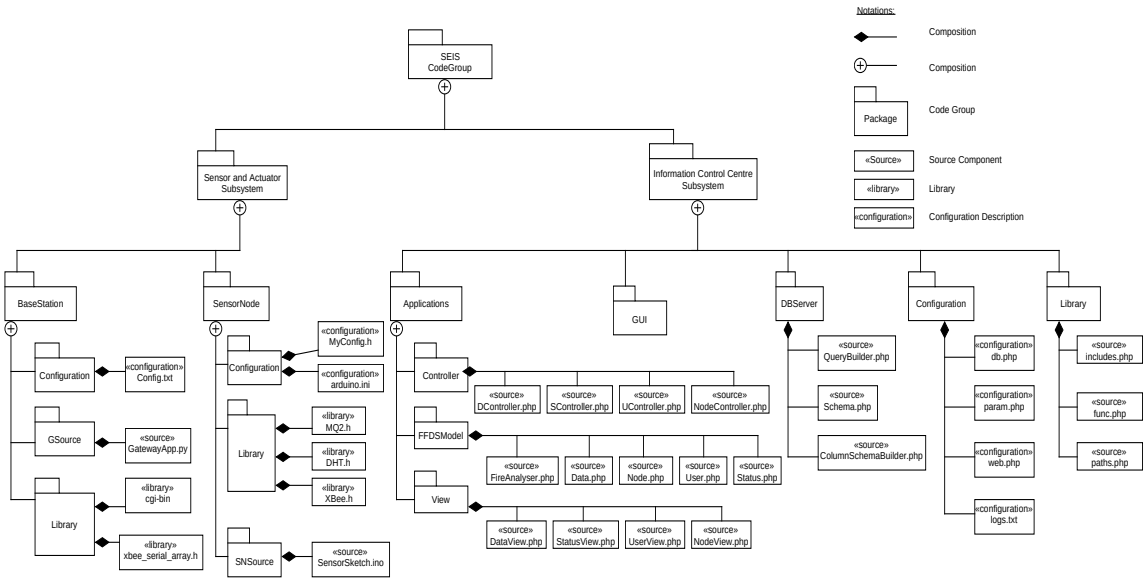


Figure 8.4: Code View

The information control centre directory contains a set of source components for the intended applications, database services and configurations through *ApplicationServer*, *DB-Server*, and *GUI* directories respectively. In which the *ApplicationServer* directory contains source codes of *Controllers*, *FFDSModels* and *Views* for implementing the software modules of forest fire detection system through various components, i.e. *DController.php*, *FireAnal-yser.php*, *StatusView.php*, *Data.php*, *Node.php* etc. The *Configuration* directory consists of *web.php*, *db.php*, *param.php* and *logs.txt* for configuring the webserver, database . The *Library* contains *include.php*, *func.php* and *paths.php* for declaring public interfaces and shared files. The *DatabaseServer* contains *DBSource* and *Configuration* directories for storing the files of implementing and configuring database server respectively. The *DBSource* includes *Schema.php*, *QueryBuilder.php* and *ColumnSchemaBuilder.php* while *Configuration* directory include *db.php*. All these files reflect modules and runtime processes that implement the actual application for analysing the forest fire parameters, predict the occurrence of fire and issue alerts in case of fire.

Topology View

The topology view refers to the topology view of the RefSEISs presented in Section 7.2.5 which describe the connections between the nodes. The topology view addresses the concerns of operators and development experts while addressing issues related to the how nodes are distributed, and communicate in the forests while focusing on the construction of energy efficient system since the system is made up the resource (mainly power, memory and processing) constrained sensor nodes. The topology view of forest fire detection system is depicted in Figure 8.5.

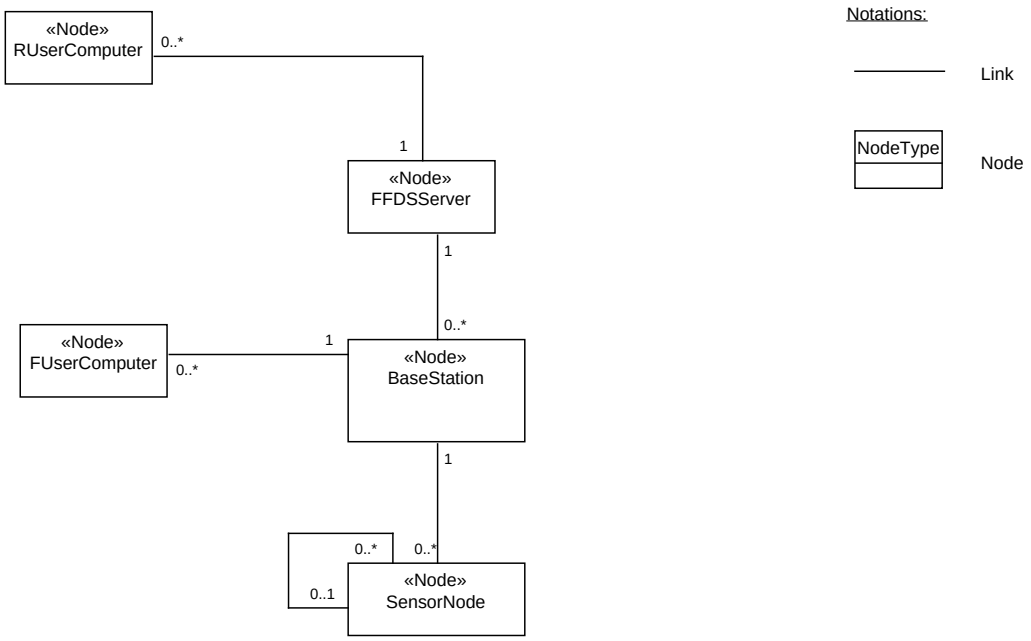


Figure 8.5: Topology View

A tree topology is adopted from the RefSEISs using a ZigBee protocol in the sensor-actuator subsystem as specified in execution view. Then the *SensorNodes* are considered to be end nodes collecting the raw data (smoke, humidity and temperature parameters) from the forest and send the data to the *SensorNode*. The *SensorNodes* are considered to be the routers responsible for transmitting the collected data and aggregates of other *SensorNodes* via multi-hop routings to the *BaseStation*. The *SensorNodes* are connected forming a tree topology. The *BaseStation* is a coordinator used for managing the *SensorNodes*, and integrating data from various *SensorNode*s and transmit them to the information control centre subsystem through a *FFDSServer* for further processing. The *BaseStation* is also connected to multiple *FUserComputer* for allowing the user to access the system in the field. In the information

control centre subsystem, the *FFDSServer* deploys the application and web containers as well as database server processes and stores data in the *DBServer* and supports queries from users through multiple *RUserComputers*.

Data View

This data view refers to the data view of the RefSEISs presented in Section 7.2.4 as shown in Figure 8.4. The data view addresses concerns of operators and development experts related to how data is created, accessed, updated and stored in the system. The data view of the RefSEISs with Key-Value type table architectural style has been referred, except the classes related to actuators, i.e. *Actuator* and *Tasking* have been omitted. This data view addresses the FFN2: Storage, and FN5: Sensor Management architectural requirements.

The data view of forest fire detection system is depicted in Figure 8.6. In sensor-actuator subsystem, the *MSensing* contains a *Sensor* class for storing information about the end nodes (sensors) which sense the environmental parameters and a *Measurement* for storing those measured parameters i.e. temperature, humidity and smoke. The *MSensorManager* uses a *SensorNode* as data source for storing information about the sensor node and the *MPowerSaving* uses *SensingControl* for imposing duty cycling on the operations of sensor nodes. The *MAggregator* is associated with *Aggregate* for measurements (aggregates) that have derived from various sensor nodes through multi-hoping such that the sensor nodes near the base station send the information collected by other sensor nodes. Both *Measurement* and *Aggregate* store the observed data using Key-Value type table architecture style in which the data declared of type any to accommodate any type of data. The base station is responsible with the field processing, contains a gateway which sends the data collected from various sensor nodes to the information control centre for further processing. The *MFProcessing* is associated with *Gateway* as a data source for storing information about the gateway and *FusedData* for storing integrated information derived from various sensor nodes. Both *Aggregate* and *FusedData* support the storage of any data type of sensor nodes (observed environmental parameter) to be stored in the database without further modifications. A base station requires a *DHandler* to connect to the *IFUserAccess* for accessing the system locally in the field through *MFDisplay* which is associated with *FDisplay* as data entity for storing information about the display.

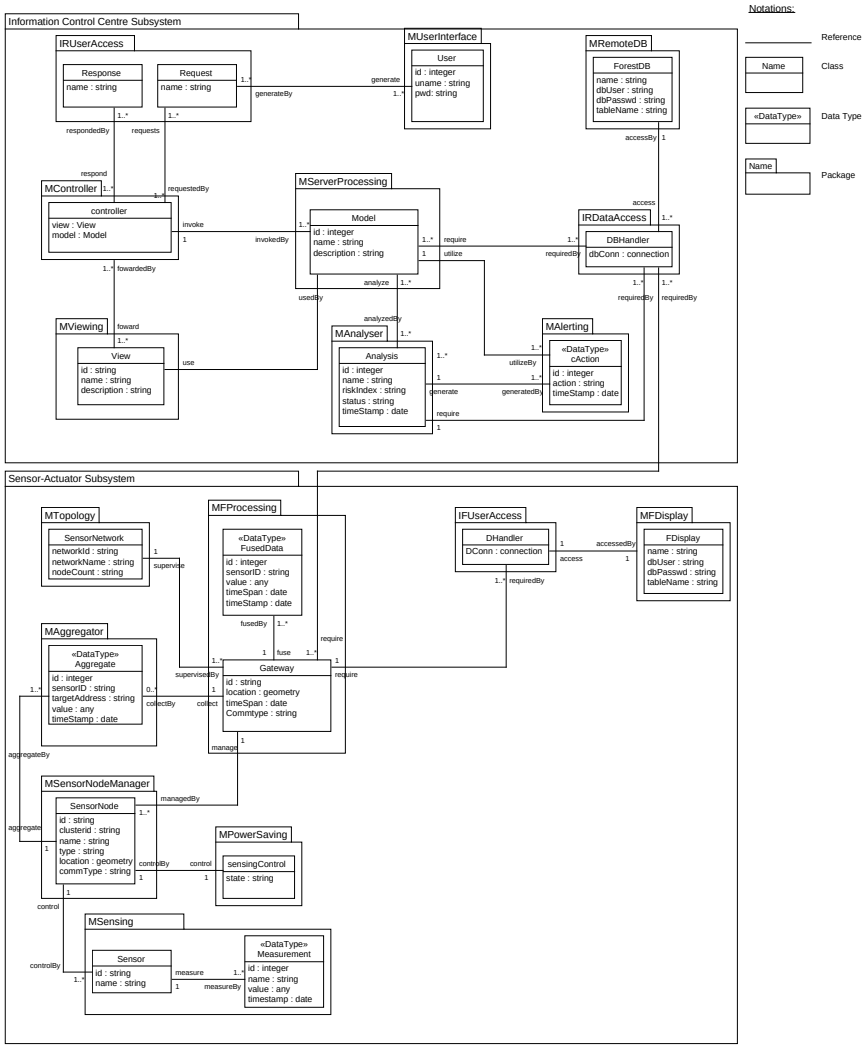


Figure 8.6: Data View

In the information control centre subsystem, the *MServerProcessing* responsible with the business logic of forest fire detection system through a models which encapsulate methods to access and manipulate data. The information about models is provided by a *Model*. A *Model* utilizes an *Acquisition* to store information about the observations received from various gateways in the forest. The *Model* requires the *DBHandler* which stores the information about the connection in the *IRDataAccess* to access the database *MForestDB*. The *MForestDB* is associated with the *ForestDB* as a data source for storing information about the database. The *MAnalyser* uses *Analysis* for storing the analysed data. The *MAlerting* contains *Alert* is used for storing information about alerts generated by *MServerProcessing*. The *MViewing* controls the way data is displayed and how the user interacts with the system. The *MViewing*

is associated with *View* data entity for storing information about the views. The *MController* is responsible for handling the events which are triggered by either a user or a system. The *MController* contains *Controller* data entity for storing information about the controller. The controller accepts requests using a *HTTPRequest* that stores information about the requests in the *IRUserAccess* from the user via a *MUserInterface*. The *MUserInterface* has *User* which provide the information of the user. Similarly, the controller sends response which its information is stored in the *HTTPResponse* in the *IRUserAccess* to the user. The *MController* interacts with both *MServerProcessing* and *MViewing* for retrieving the required data and generating views respectively. This data view supports the FNFN2: Maintainability and FNFN3: Interoperability.

8.3 Architectural Prototype Implementation

The software architectural prototype of forest fire detection system has been developed based on the software architecture described in Section 8.2. This section describes such a prototype of that system starting with the used hardware, assumptions and prototype.

8.3.1 Hardware

The hardware that used for the implementation of forest fire detection system includes; (i) Two Arduino Uno R3 with XBee transceivers as sensor nodes consisting temperature and humidity sensor DHT11 and gas smoke sensor MQ for collecting environmental parameters such as humidity, temperature and smoke from sensors on the forest. (ii) A single Raspberry Pi Model B with XBee transceiver as the base station. Such a base station aggregates the observed data from the forest remotely and transmits the collected data to the information control centre for further processing. (iii) A personal computer as the information control centre. In which, data processing methods are deployed to analyse and predict the occurrence of forest fires. The following diagrams show the Raspberry Pi Model B and Arduino Uno R3 which have been used in this prototype.

8.3.2 Assumption

In this thesis, the sensor nodes are assumed to be static. Thus, their addresses and locations are known. The sensor reports the power level of their battery level as they send data. Considering that this system collects temperature, humidity and smoke values, the chosen forest fire danger index was a Swedish index called the Angstrom Fire Index [Willis et al., 2001]. This index calculates the likelihood of a forest fire occurrence using the observed

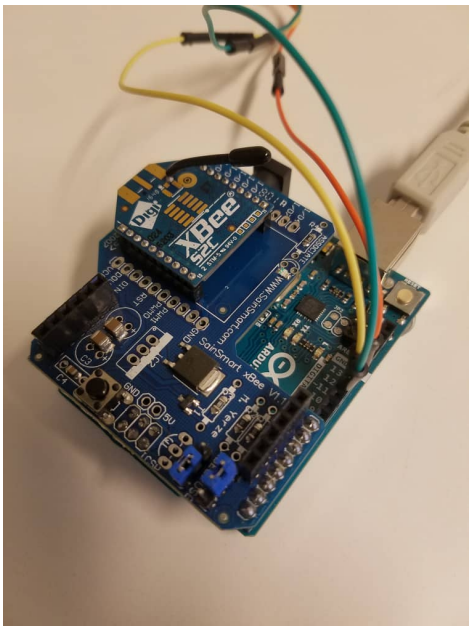


Figure 8.7: Arduino Uno R3

Figure 8.8: Raspberry Pi Model

temperature and humidity values. The Angstrom fire index calculation (I) is expressed as [Willis et al., 2001]:

$$I = \frac{R}{20} + [\frac{(27-T)}{10}] \tag{8.1}$$

where

- I= Angstrom Index
- R= Relative Humidity (%)
- T= Air Temperature (oC)

Interpretation:

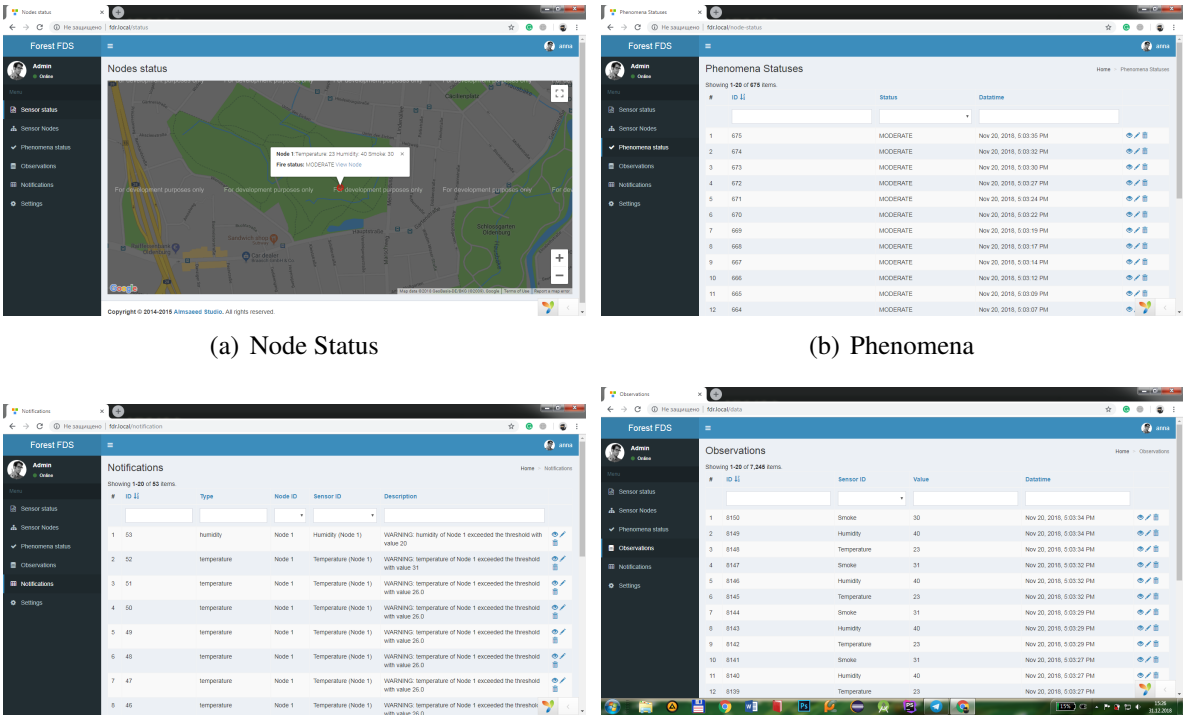
- >4 : Fire occurrence unlikely (unlikely)
- 4.0-2.5 : Fire conditions unfavourable (low probability)
- 2.5 - 2.0 : Fire conditions favourable (moderate)
- <2.0 : Fire occurrence very likely (high probability)

This calculation, for the designed system, is done in the PHP script responsible for calculating the likelihood of fire occurrence while processing the received data at the information control centre and then send alarms or notify an operator. Similarly, the MQ2 sensor

activates the alarm, i.e. buzzer when the observed value is above the threshold value 400 gas concentration.

8.3.3 Prototype

The prototype of the forest fire detection system intends to predict the probability of fire using temperature, humidity, and gas smoke data. Temperature and humidity (DHT11) and gas (MQ2) sensors are connected to a sensor node (Arduino Uno R3). The sensor node collects environmental information from sensors and sends it to the Gateway (Raspberry Pi 3 Model B) via Zigbee. The gateway receives the data in XBeeDataFrames format and after preprocessing sends the appropriate temperature, humidity, and smoke values to API of Information Control System through a web server in HTTP GET Request format. The web server uses PHP as the scripting language to process and store the incoming data into the MySQL database. Then a fire analyser model consumes the data by analyzing the probability fire probability based on an Angstroms Fire Index formula [Willis et al., 2001]. A user then views the data and fire status via a web browser that uses a combination of JavaScript and Google Maps. The information control centre was implemented using LAMP stack with Yii MVC framework [Safronov and Winesett, 2014]. Figure 8.9 depicts some of the features of the forest fire detection system.



(a) Node Status

(b) Phenomena

(c) Notifications

(d) Observations

Figure 8.9: Screenshots of the architectural prototype

8.4 Summary

This chapter demonstrated the applicability of the proposed RefSEISs in the construction of new concrete SEISs mainly forest fire detection system. The RefSEISs encompasses architecture knowledge such as requirements, viewpoints, views, etc. that can be reused in the development of particular SEISs. Most of the architecture knowledge of RefSEISs have been reused, the software architecture description of forest fire detection system employed short time and efforts. This chapter answered the third research question **(RQ3)**: How to apply the proposed reference architecture for SEISs?

Chapter 9

Validation of RefSEISs

This Chapter describes the conceptual validation of the proposed RefSEISs by mapping it to some of the existing architectures of SEISs presented in Chapter 3. Differences are depicted, and a conclusion summarizes the applicability of such reference architecture. The RefSEISs is mapped to four SEISs as described in their respective sections; (i) Forest Fire Monitoring System (IPNAS) in Section 9.1, (ii) Urban Air Quality Monitoring System in Section 9.2, (iii) Indoor Air Quality Monitoring System in Section 9.3, and (iv) Flood Risk Assessment System in Section 9.4. Finally, Section 9.5 provides the overall summary of the chapter.

9.1 Forest Fire Monitoring System (IPNAS)

The forest fire monitoring system called IPNAS is automated surveillance associated with automatic detection of forest fires. IPNAS is supported by Ministry of science, education and sport of Republic Croatia [Stipanicev et al., 2018].

The architecture and the mapping of the Croatian Integral Forest fire Monitoring System IPNAS onto RefSEISs is shown in Figure 9.1. The software architecture of IPNAS represents the module view of RefSEISs as described in Section 7.2.2. The IPNAS is based on the fields units and a central processing unit. The field units consist of video cameras and mini-meteorological stations which are connected through wireless LAN to a central processing unit for further processing. The data collected by the field units is stored in the databases (GIS database, SQL database and Data warehouse) which is represented by field and remote databases in the *Data Access Layer* of RefSEISs. With the use of data interface, IPNAS offers various services including user-friendly camera control, automatic fire detection and archival retrieval of both data and video which is reflected through user sensor node, processing, control dispatcher modules in the *Application Layer* of RefSEISs. The IPNAS is web-based with

a user interface displayed in a standard web browser. A user can access the system through tunnelled SSL (Secure Socket Layer) VPN (Virtual Private Network). This user interface reflects the user interface modules of RefSEISs. The data access, application and GUI layer represents software modules for *Sensor-Actuator*, *Communication*, and *Information Control Centre* subsystems. In regards to IPNAS, the module view of the RefSEISs covers each element of the architecture though the architecture was too abstract and modules were not explicitly defined. Therefore, this reflects an appropriate mapping to the definition of the RefSEISs.

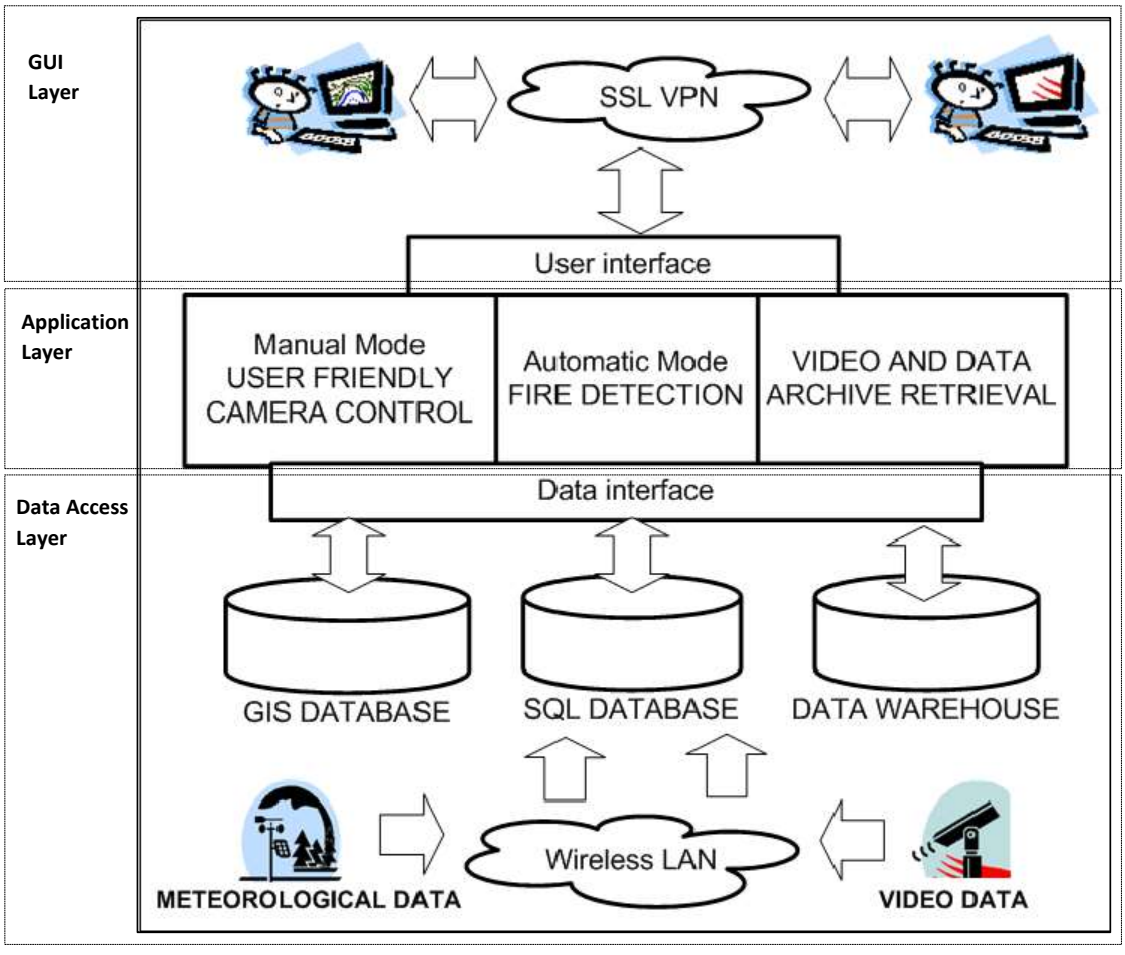


Figure 9.1: Mapping of IPNAS architecture from [Stipanicev et al., 2018] and RefSEISs

9.2 Urban Air Quality Monitoring System

The urban air quality monitoring system concerns with monitoring and managing of air pollution in urban areas using 3D spatial city model in the dispersion model in the aspect of model scaling, data acquiring, 3D visualization, and visual analysis [Ujang et al., 2013]. This urban air quality monitoring system has been designed in such a way that air quality parameters are sent directly from the sensor nodes in the sensor-actuator subsystem to the information control centre subsystem for analysis. Figure 9.2 depicts an integrated data view of urban air quality modeling with a 3D city model mainly CityGML which is an open-standard data model and its comparison against the data view of RefSEISs.

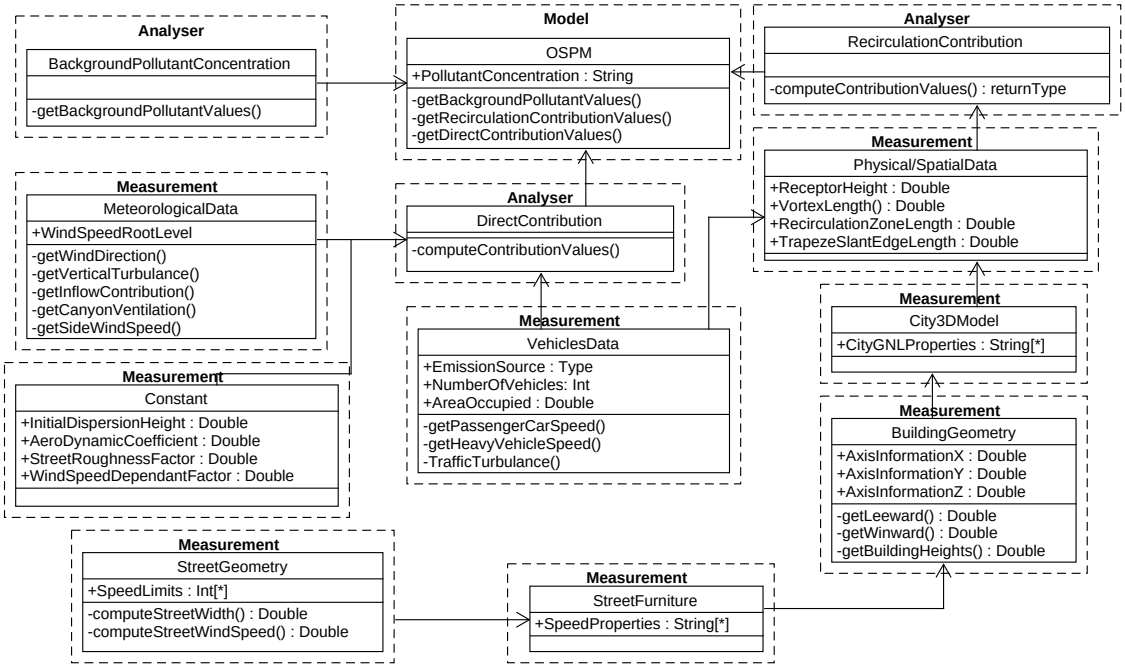


Figure 9.2: Mapping Air Pollution Dispersion Architecture from [Ujang et al., 2013] and RefSEISs

The CityGML consists of geometrical attributes which are essential in air pollution dispersion model [Ujang et al., 2013]. The urban air pollution dispersion model involves the Operational Street Pollution Model (OSPM) which is represented by *Model* in the RefSEISs. The OSPM determines the concentration of pollution through background pollutant concentration, direct contribution and recirculation contribution. The background pollutant concentration, direct contribution and recirculation contribution present the *Analysers* of RefSEISs. And finally, the meteorological data, constants, vehicle data, street geometries, street furniture, 3D city models, building geometries and physical or spatial data are actual

data values represented by the *Measurements* of RefSEISs as some of them originated from various sensor nodes, i.e. meteorological data, vehicle data, street geometry, etc.. Since all the elements of the data model of urban air quality system are mapped onto some data entities of the RefSEISs, then this is appropriate to the definition of the RefSEISs.

9.3 Indoor Air Quality Monitoring System

An indoor air quality monitoring is a SEIS which facilitate the detection and improvement of the indoor air quality [Abraham and Li, 2016]. The mapping of the architecture of an indoor air quality monitoring system onto the RefSEISs is shown in Figure 9.3. This architecture represents a topology view of RefSEISs as described in Section 7.2.5. The main components of the system include the sensor nodes as both end device and router, the base station as coordinator, database and web server. The base station aggregates the data from the sensor nodes periodically using wireless connections and either mesh or tree topology algorithm. Then the aggregated data is sent to the server and stored in the database server for further processing. To access the data and manage the system remotely, a web server provides a convenient web interface for users. The sensor nodes as end devices, sensor nodes as router and base station represent the *SensorNode* and *BaseStation* field nodes in the RefSEISs, while the database and web server present the *Database Server* and *Application Server* of RefSEISs. The web clients can access the system through client machines i.e. *RUserComputer* as reflected in RefSEISs. Considering the architecture of an indoor air quality monitoring system, each element of such a system is mapped to the corresponding node of the topology view of the RefSEISs. Therefore this makes the definition of the RefSEISs to hold.

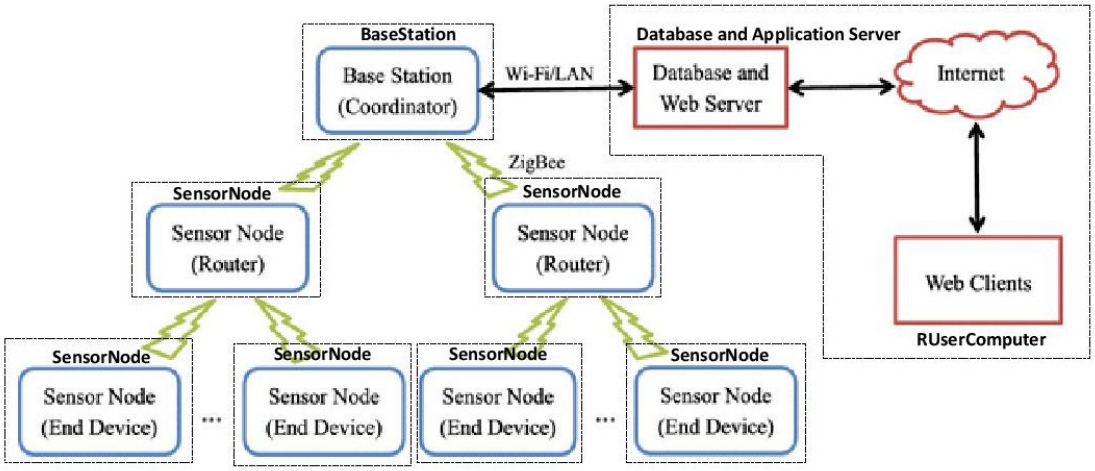


Figure 9.3: Mapping of Indoor Air Quality Monitoring System Architecture from [Abraham and Li, 2016] and RefSEISs

9.4 Flood Risk Assessment System

The flood risk assessment system intends to provide a detailed assessment of flood risks to proposed buildings [Amirebrahimi et al., 2016]. The system supports current planning needs, allows for the identification of the type and the monetary costs of potential flood damages at building individual component level, and also generates a visualization of damages through an interactive 3D environment.

Figure 9.4 shows the architecture and mapping of the flood risk decision support system onto RefSEISs. Such architecture consists of four layers; i.e. Data Layer (DL), Data Access Layer (DAL), Business Layer (BL), and the Presentation Layer (PL) which reflect the module view of RefSEISs as described in Section 7.2.2. The Data Layer of the system contains both required spatial and non-spatial data to undertake the damage/risk assessment and visualization processes. While the Data Access Layer (DAL) is an intermediate layer between the Data and Business layers as well as providing simplified access and retrieval mechanisms for data stored in the data storage, e.g. files or databases reflecting the database modules of the RefSEISs. Both Data Layer and Data Access Layer belong to *Data Access Layer* of RefSEISs. The Business Layer concerns with the execution and maintenance of the logical processes and rules that are required for calculations. The modules of this layer include water infiltration modeling, damage and costs assessment, etc., such modules reflect the server processing, analyser and control dispatcher modules of the RefSEISs. The outputs

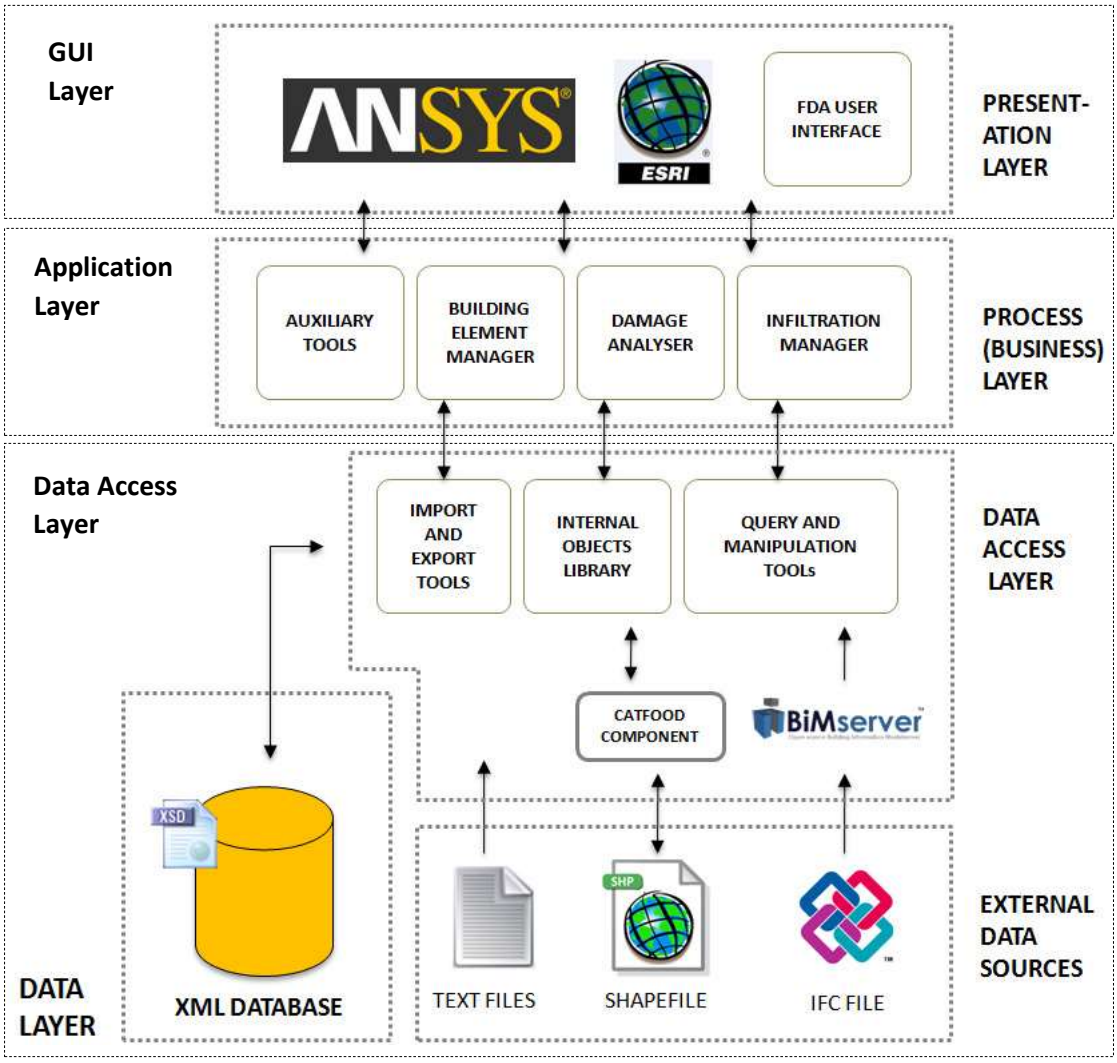


Figure 9.4: Mapping of Flood Risk Assessment System Architecture from [Amirebrahimi et al., 2016] and RefSEISs

of these modules are then transmitted to Presentation Layer for the presentation of the system to the user through a user interface, which in turn the data is sent back to Data Access Layer for the export and storage. The user interface of this architecture reflects the user interface module of RefSEISs. The Business Layer represents the *Application Layer* of RefSEISs. The Presentation Layer consists of the User Interface (UI) which allows the interactions of the users to the system, i.e. capturing users requests and presenting responses. Such user interface reflects the user interface module of RefSEISs. The Presentation Layer reflects the *GUI Layer* of RefSEISs.

Considering the architecture of flood risk assessment system, the RefSEISs covers each module of the architecture though modules were not explicitly defined. This demonstrates an appropriate mapping to the definition of RefSEISs.

9.5 Summary

This Chapter presented one of the RefSEISs evaluation approaches with the intention of answering the fourth research question: *How to evaluate the proposed reference architecture for SEISs (RefSEISs)?*. In which the applicability of the proposed RefSEISs on the existing SEISs has been demonstrated since the proposed RefSEISs has been successfully used in mapping the existing SEISs. In such mappings, the key components or elements of the RefSEISs have been utilized. Then Chapter 10 proceed with the requirements verifications of the RefSEISs.

Chapter 10

RefSEISs Requirements Verifications

In this Chapter, the RefSEISs is verified by checking the fulfilment of the specified requirements by the proposed designs. The primary goal of this evaluation is to determine the practical feasibility of the proposed RefSEISs and demonstrate its adherence to the identified quality attributes. As part of the validation, the application scenario is recalled and use its implementation to evaluate the concrete architecture derived from the RefSEISs.

The proposed RefSEISs is required to fulfil the requirements presented in Section 5.2. Therefore this chapter describes how those specified requirements have been satisfied. Such that each requirement is associated with the description of the part of the reference architecture that handles such a requirement. This requirements verification is done based on both functional and non-functional requirements as described in Section 10.1 and Section 10.2 respectively.

10.1 Functional Requirements Verifications

Table 10.1 describes the functional requirements of the RefSEISs and its relation to the design.

Table 10.1: Functional Requirements Verifications

Requirement	**Design Verification**
(FN1) Data Acquisition: The reference architecture must allow the development of SEIS that can acquire data (environmental parameters) in both non-real-time and real-time from sensors spread over the field of interest.	The RefSEISs incorporates the *SensorNode* component, *MAcquistion* module, *SensorNodeManager*, and *SensorNode* directory in conceptual, module, execution, and code views as described in Section 7.2.1, Section 7.2.3 and Section 7.2.4 respectively to enable the data acquisition in both real-time and non-real time from various sensors in the field of interest. Furthermore, the adoption of *Client-Server architectural style* in the RefSEISs allows a user to retrieve the data by sending requests to the server in the information control centre subsystem. The server is responsible for handling requests of the clients. This can be achieved by the server to command the sensor node to collect data.
(FN2) Storage: The reference architecture must support the development of SEISs that able to store the collected data for further processing.	The RefSEISs consists of the *LocalDB*, and *RemoteDB* in the conceptual view as described in Section 7.2.1, *MLocalDB* and *MRemoteDB* in the module view as described in Section 7.2.2, and *DBServer* in the execution view as described in Section 7.2.3 software elements in conceptual, module and execution views. All these software elements support the storage of data in both local and remote databases in the field of interest as well as information control centre which may be located far from the field. In addition, the data view of the RefSEISs provides the description of data entities involved from the collection of raw data from the sensor to the execution of control actions by the actuators in the field of interests. Such description facilitates the identification of suitable columns needed in the creation of those databases.

Continued on next page

Table 10.1 – *Continued from previous page*

Requirement	Design Verification
(FN3) Phenomena Analysis: The reference architecture must allow the development of SEIS that can establish analysis models on the processed information to gain an understanding of the environmental phenomena of interest, predict the possible occurrence of such environmental phenomena.	The RefSEISs utilizes the *ServerProcessing* and *Analyser* in the conceptual view as described in Section 7.2.1 and *MServerProcessing*, *MAnalyser* and *MControlDispatcher* modules in module view as described in Section 7.2.2, The *MServerProcessing*, *MAnalyser* and *MControlDispatcher* modules are deployed in *AppContainer* modules in the execution view as described in Section 7.2.3. All these software elements of the RefSEISs incorporate analysis algorithm to analyse and predict the environmental phenomena of interest.
(FN4) Visualisation: The reference architecture must enable the development of SEIS that is capable of visualizing collected environmental parameters, predictions and propagation of environmental phenomena.	The RefSEISs uses *BSDisplay* and *Client* components in the conceptual view in Section 7.2.1, *MFDisplay* and *MUserInterface* modules in the module view in Section 7.2.2, and the *Web Browser* processes in the execution in Section 7.2.3 to allow users of the system to access the system and hence manage to visualize collected environmental parameters, predictions and propagation of environmental phenomena.
(FN5) Phenomena Control: The reference architecture should support the development of SEIS that can manipulate environmental parameters to control environmental phenomena in the field of interest.	The RefSEISs incorporates the *Actuator* component, *MActuating* module, *ActuatorManager*, and *Actuator* directory in the conceptual, module, execution, and code views as described in Section 7.2.1, Section 7.2.2, Section 7.2.3 and Section 7.2.4 respectively to enable the execution of control actions in real-time by various actuators in the field of interest. These software elements are associated with control instructions for controlling a specified environmental phenomenon.

Continued on next page

Table 10.1 – *Continued from previous page*

Requirement	Design Verification
(FN6) Sensor Management: The reference architecture should enable the development of SEIS that can coordinate, monitor and control sensors spread over the field of interest.	The RefSEISs utilizes the *BaseStation* component of the conceptual view in Section 7.2.1, *MFProcessing* module of module view in Section 7.2.2, and *BaseStation* device of the execution view in Section 7.2.3 are responsible for coordinating, monitoring and controlling sensors which are distributed over the field of interest. Additionally, the field nodes networks of SEISs mainly sensor-actuator networks are modeled as tree-topology clustering networks through topology views as described in Section 7.2.5. In which the *BaseStation* coordinate, monitor, and control various *SensorNodes* to sample, store and transmit data in real-time as described in the topology view.
(FN7) Actuator Management: The reference architecture should enable the development of SEIS that can coordinate, monitor and control actuators spread over the field of interest.	The RefSEISs utilizes the *BaseStation* component of the conceptual view in Section 7.2.1, *MFProcessing* module of module view in Section 7.2.2, and *BaseStation* device of the execution view in Section 7.2.3 are responsible for coordinating, monitoring and controlling actuators which are dispersed over the field of interest. Additionally, the field nodes networks of SEISs mainly sensor-actuator networks are modeled as tree-topology clustering networks through topology views as described in Section 7.2.5. In which the *BaseStation* coordinate, monitor, and control various *Actuators* to sample, store and transmit data in real-time as described in the topology view.

10.2 Non-Functional Requirements Verifications

The RefSEISs incorporated the non-functional requirements in two levels; domain and reference architecture levels.

10.2.1 Domain Level

As explained in Section 2.2.4, the RefSEISs has been assessed using a scenario based evaluation method. Various scenarios have been created by considering possible ways that SEISs fulfil the specified domain level requirements described in Section 5.2.2. Since it is clearly infeasible to identify and include all possible scenarios, thus three scenarios were defined to demonstrate how the specified requirements are fulfilled; (i) Optimization of power consumption. (ii) The change of environmental phenomena analysis algorithms, database or sensor measuring types. (iii) Use of heterogeneous sensor nodes. This is demonstrated through the evaluation of the proposed forest fire detection system presented in Chapter 8. Therefore this section described how the aforementioned scenarios using forest fire detection system cover the evaluation of the recurring non-functional requirements of SEISs domain mainly energy efficiency, interoperability and maintainability.

Scenario 1: Optimization of Power Consumption

As described in Section 7.2.5, there are various mechanisms for optimizing the power consumption of sensor networks in various SEISs, i.e. tree-topology clustering control management [Zhang et al., 2008, Jadhav and Deshmukh, 2012, Liyang et al., 2005], routing protocols [Jiménez and García, 2015, Castillo-Effen et al., 2004], clustering algorithm for routing in real-time forest fire detection system [Liyang et al., 2005], state switching through adaptive duty cycling mechanisms [Jeličić et al., 2011, Kumar and Kishore, 2017, Marin Perez et al., 2012], and phenomena detection that uses threshold in aggregating the observed data [Ramesh, 2014]. The RefSEISs adopts a cluster tree-topology architecture style which is considered to be the most effective, suitable and low-power consumption topology for sensor networks. This type of topology extends the life cycle of the entire sensor network, by minimizing the data transmission distance and allowing only the nodes near the base station to send the data directly while the rest send data through other sensor nodes (multi-hoping).

Besides the computation burden over battery-powered sensing nodes has been reduced through an edge computing device mainly a base station that has its local processing unit, and databases. This is achieved through the use of the *MPowerSaving* module in module view as described in Section 7.2.2. In the proposed design, the duty cycle of the sensor node is directly regulated by time interval, such that the sensor node sample (or collect) data every two minutes. This implies the sensor node was changing states from sleeping to awake, listen and then transmit every after two minutes. Similarly, another experiment was run to measure the voltage used by the sensor node without *MPowerSaving* module that

contains the duty cycle algorithm, i.e. the sensor nodes samples or collect and transmit the data simultaneously. The microwatt-meter device has been used for measuring the power consumption of the sensor node. The results obtained showed that the sensor node without the duty cycle algorithm consumed high power than the sensor node with the duty cycle algorithm as shown in Figure 10.1. This implies the use of *MPowerSaving* module optimizes the energy consumption of sensor nodes.

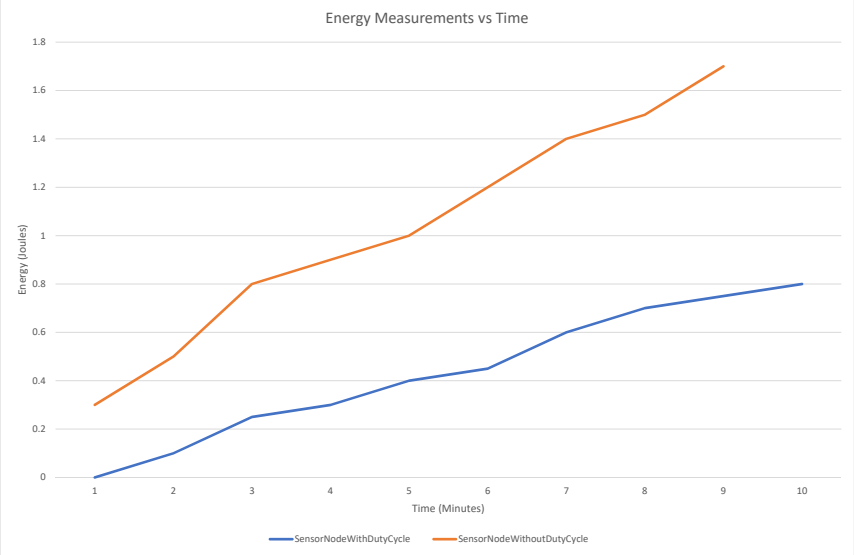


Figure 10.1: Power Analysis at the Sensor Node Level

Scenario 2: The change of environmental phenomena analysis algorithms, database or sensor measuring types

The incorporation of maintainability demands in an early phase of the system design is crucial for achieving a well-functional and reasonable cost system since the resulted system will be able to accommodate the changes or new requirements throughout its lifetime. Maintainability is one of the software quality attributes that can be expressed naturally through change scenarios [Kazman et al., 1994]. Some of the potential change scenarios of forest fire detection system as concrete SEIS that could arise as new requirements in the future include; the change of forest fire detection analysis algorithms, database or sensor measuring types.

The advancements of technologies and change of demands of the acquirers promote the need of novel forest fire detection analysis algorithms to suit the demands. The database schema of the forest fire detection system can be changed to another schema type, i.e. from SQL to Oracle. Also, the sensor measuring types could be changed to meet new demands and increasing the accuracy of the system, for instance, the forest fire detection system can include CCTV cameras in addition to temperature, humidity, and smoke measuring types. These changes could affect many components or software elements of the system and possibly led to the reconfiguration of the whole system. However, the forest fire detection system was designed using a module view as described in Section 8.2.3, which employed a layered architectural style with loosely coupled software modules. The modules were associated with the separation of the user interface and database model from the application logic through MVC strategy. Such design facilitates maintainability by enabling these changes (e.g. analysis algorithms, database or sensor measuring types) on a per module basis, i.e. *MAnalyser*, *MForestDB* or *MSensorManager* respectively rather than on the whole system and hence reduce ripple-effect. This makes the SEISs easier to replace components or modules, add, remove or to upgrade a component or module without affecting the other components or modules.

Furthermore, the code view of forest fire detection system described in Section 8.2.3 shows how the specified software module of the system are mapped into system source files. Such that each module was placed in a certain source file that can be specified using a different name but contains the functionality of such module, for example, *MAnalyser*, and *MSensorManager* of module view are placed in *FireAnalyser.php* and *SensorSketch.ino* source code files of the code view as presented in Section 8.2.3. Such description enables multiple developers to focus in different parts of the system concurrently and hence integrate changes independently of each other.

Scenario 3: Use of heterogeneous sensor nodes

Interoperability enables multiple software systems to incorporate or work with each other. The interoperability of SEISs is ensured by handling the heterogeneity of sensor nodes at two levels; data and network as described in Section 5.2.2. The interoperability at the network level is ensured when multiple and heterogeneous sensor nodes with different protocols form a network, such that manages to communicate or exchange information with each other. The communication between heterogeneous sensor nodes of SEISs is handled using an interoperable topology view as described in Section 7.2.5. The topology view supports different kinds of standards and communication protocols (wired or wireless). Such a view manages the heterogeneous sensor nodes effectively through the base stations which contains

gateways for transmitting data to the server for further processing. The base stations possess the management functionality which tends to reduce memory and computation requirements on sensor nodes as well as bridge the gap between heterogeneous sensor nodes and to provide way to manage, query and interact with them, for example a general message exchange mechanism that uses XML as message style and SOAP as transmission protocol [Vodel et al., 2012] can be imposed in sensor nodes to enable uniform data to be transmitted across the network. Furthermore, a user can perform management tasks using a base station, and this can be remotely located on the Internet.

At the data perspective, the interoperability is ensured by the easy integration of data originated from heterogeneous sensor nodes. This is achieved using the data view as described in Section 7.2.5 which focus on how to acquire, process and combine such heterogeneous data. The data view provides an integration data structure which includes all the SEISs data entities and their relationship involved in all stages like data measurement, processing, storage and finally data analysis tasks. In this data view, the sensor node as an object has been modeled using both structural and behavioural features, i.e. *SensorNode* provides sensor global information (name, id, type, etc.), *Measurement* for describing the variables that a sensor can measure (temperature, light, humidity, etc.), *Aggregate* describes data that collected from other sensor nodes and *SensingControl* is used for describing the sensor operating state/mode (on, off, sleep, etc.). Similarly, the base station is associated with *Gateway*, *SensorNetwork* and *FusedData* for providing the information about the gateway, network topology and data that is integrated from various sensor nodes. Both *Measurement* and *FusedData* allow any data to be stored. The sensor nodes, base station and server adopt a Key-Attribute value table architecture style to support the seamless integration of various data types hence enable the seamless integration of data originated from heterogeneous sensor nodes. For instance, the forest fire detection system managed to integrate data from three types of sensors, i.e. smoke, temperature and humidity as shown in Figure 10.2

10.2.2 Reference Architecture Level

At the reference architecture level, the proposed RefSEISs fulfils the following requirements:

NFN4 Generality

The RefSEISs models abstract architectural elements of SEISs independent of existing technologies and protocols. These elements are considered to be common and variants abstracted elements for the implementation of SEISs as described in Chapter 7. The RefSEISs makes good use of best practices while avoiding reliance on specific, and concrete technologies.

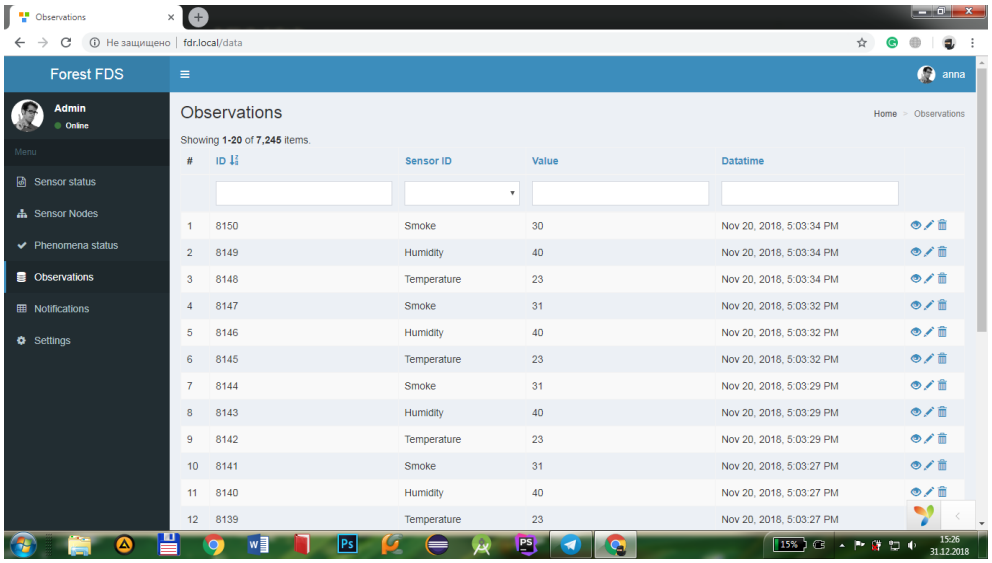


Figure 10.2: Data from Heterogeneous Sensor Nodes

NFN5 Completeness

The RefSEISs covers all the critical dimensions for reporting a reference architecture as described in [Angelov et al., 2012] as shown in Table 10.2. This demonstrates that the proposed RefSEISs has achieved the completeness requirement as specified in Section 5.2.2.

Table 10.2: The RefSEISs against the Dimensions of Reference Architecture

Dimension	**Sub-Dimension**	**RefSEISs**
Context	Who defines it?	It is defined as a part of a research.
	Where will it be used?	It aims to facilitate the development and maintenance of a SEIS.
	What is the maturity stage of the domain?	The RefSEISs is considered as preliminary because to the best of our knowledge; a comprehensive and robust reference architecture is missing.

Continued on next page

Table 10.2 – *Continued from previous page*

Dimension	**Sub-Dimension**	**RefSEISs**
Goal	Why is it defined?	The RefSEISs aims to facilitate the development and maintenance of concrete SEISs by providing the conceptual, module, code, execution, topology, and data views as described in Section 7.2.1, Section 7.2.2, Section 7.2.3, Section 7.2.4, Section 7.2.5 and Section 7.2.6 respectively.
Design	What is described?	The RefSEISs is described in terms of high-level component-connector, modules, runtime entities, files and directories, nodes-links, and data entities using conceptual, module, execution, code, topology, and data views respectively and design principles of the reference architecture.
	How is it described?	It is described based on ISO/IEC/IEEE 42010:2011 using UML diagrams.
	How is it represented?	It is described as high-level representations using semi-formal approaches.
Instantiation	How is it instantiated?	The RefSEISs has been instantiated into the concrete architecture of forest fire detection system, and its prototype has been implemented.
Evaluation	How is it evaluated?	The RefSEISs have been evaluated using scenarios for functional requirements, quality attributes and its prototype has been assessed.

NFN6 Applicability

The applicability of the RefSEISs has been demonstrated in the construction of the new concrete architecture of SEISs mainly forest fire detection system and mapping of the existing SEISs on the reference architecture as described in Chapter 8 and Section 9 respectively.

All the specified requirements of RefSEISs in Chapter 5 have been addressed and verified. Architecture documentation is never complete because it is impossible to cover all concerns and details. However, the documentation of the proposed RefSEISs manages to capture enough knowledge which is relevant and useful for the development and maintenance of SEISs.

10.3 Summary

This chapter finalized Part IV of the thesis while addressing the fourth research question: *How to evaluate the proposed reference architecture for SEISs (RefSEISs)?*. The RefSEISs is verified by checking the fulfilment of both functional and non-functional requirements with the designs. Results indicated that the RefSEISs has a positive influence in achieving the requirements of SEISs. This RefSEIS presents, therefore, good perspectives to be adopted and contribute to the development and maintenance of SEISs. Then Chapter 11 concludes this thesis with the summary of the main contributions and future work.

Part V

Conclusion

This is the final part of the thesis, in which the conclusion is written, lessons learned and future research areas identified as presented in Chapter 11.

Chapter 11

Conclusion and Recommendations

This is the final chapter which provides the overall findings of this thesis and future research options. Section 11.1 summarises the work reported in this thesis. After that, the answers to research questions presented in the Section 1.2 are outlined in Section 11.2. Then the recommendations for further research follows in Section 11.3. Finally Section 11.4 proceeds with summarising the contributions of this thesis.

11.1 Thesis Summary

The thesis is divided into five parts. Part I commenced with the challenges which include the motivation, research objectives and research methodology. Then Part II presented the foundations which describe the current state of work in software architectures and SEISs. Both contents are essential for the development of RefSEISs with a strong emphasize on quality attributes. Part III described the RefSEISs based on ISO/IEC/IEEE 42010:2011 standard and Siemens View Model. Part IV has demonstrated the applicability of the proposed RefSEISs in the development of forest fire detection system as a concrete SEIS, mapping to the existing SEISs and evaluation of the RefSEISs. Then the conclusion and future recommendations are provided in Part V. A central learned lesson of this thesis is that a systematic approach to deal with software architecture is essential in the context of SEISs. Since many decisions are repeatedly executed over multiple designs situation in different SEISs with similar subsystems. Then the reuse of the established best practices through the use of reference architecture is chosen as a basis for the systematic approach for rapid development and maintenance of SEISs.

11.2 Answers to Research Questions

The primary objective of this thesis is to propose a reference architecture for smart environmental information systems (RefSEISs) to facilitate the development and maintenance of SEISs. In Section 1.2, four research questions have been identified to achieve the specified objective. This section summarizes the answers to these questions based on the findings of this research.

RQ1: Which are essential aspects or perspectives that are required to describe SEISs?

The analysis on the existing SEISs, i.e. forest fire detection systems, flood detection systems, air pollution detection systems, landslides detection systems and road traffic control systems as SEISs representatives have been conducted and described in Section 3.2, Section 3.3, Section 3.4, Section 3.5 and Section 3.6 respectively. Such analysis revealed that the existing SEISs are stand-alone solutions which were associated with different architectures covering various aspects, i.e. conceptual, module, execution, topology and data. Thus this thesis proposes a set of six viewpoints for SEISs which reflect the essential aspects or perspectives required in the description of SEISs. Such viewpoints include the conceptual, module, execution, code, topology and data viewpoints as defined in Chapter 6. The first four viewpoints namely conceptual, module, execution, and code viewpoints were adopted from Siemens view model which were modified to fit the context of SEISs.

RQ2: How to describe a reference architecture for SEISs (RefSEISs)?

The second research question is discussed in Part III where a reference architecture for SEISs is developed. The RefSEIS encompasses requirements, architecture designs, knowledge and software elements derived from the predefined existing architectures to facilitate the development and quality of SEISs. The description of the RefSEISs is based on ISO/IEC/IEEE 42010:2011 standard and Siemens View Model.

The initial fundamental concepts of the RefSEISs documentation is described in Chapter 5, in which the stakeholders, their concerns and the overall architecture requirements were identified based on the analysis of the existing SEISs in Chapter 2. The acquirers, development experts, operators, and users are the primary stakeholders of SEISs. The concerns of these stakeholders are related to the general idea and feasibility of SEIS, fulfilment and decomposition of the SEISs functionalities, system integration, creation, access, manipulation and storage of data, communication mechanisms and protocols involved in SEISs, system components or modules implementation, maintainability, energy efficiency and interoperability concerns of SEISs. These concerns are intertwining and covering the full life-cycle of SEISs. The analysis of these concerns led to the identification of the recurring functional

and non-functional requirements of RefSEISs as presented in Section 5.2. The functional requirements include FN1: Data Acquisition, FN2: Storage, FN3: Phenomena Analysis, FN4: Visualization, FN5: Phenomena Control, FN6: Sensor Management, and FN7: Actuator Management. The non-functional requirements include both at the domain level; NFN1: Energy Efficiency, NFN2: Maintainability, and NFN3: Interoperability while at the reference architecture level; NFN4: Generality, NFN5: Completeness and NFN6: Applicability.

Afterwards, Chapter 7 described the software design of RefSEISs fulfilling the concerns and requirements in terms of six architectural views referring to their corresponding viewpoints defined in Chapter 6 and associated with the established architecture knowledge (best practices) of existing SEISs presented in Chapter 3. Such views include conceptual, module, execution, code, topology and data views. The conceptual view divides the functionalities of the system in terms of components and connectors as described in Section 7.2.1. The module view maps the identified components and connectors into the implementation modules and their dependency relationships as presented in Section 7.2.2. The execution view assigns the implementation modules from the module views into runtime entities and their allocation to the hardware as demonstrated in Section 7.2.3. The code view shows the implementation modules into files and directories as described in Section 7.2.4. The topology view describes the relationships and possible connections between the nodes in the field of interest as presented in Section 7.2.5. And finally, the data view represents the data entities involved in SEISs as described Section 7.2.6. Each view can be demonstrated using various diagrams.

This multi-views perspective of RefSEISs partitioned SEISs into different architectural views, based on the principle of separation of concerns, is found to be more effective and useful in meeting both functional and nonfunctional requirements. The use of UML in the development of various system software artefacts facilitate the smooth discussions among stakeholders and the creation of constructive feedback loops. In the design of reference architecture views, the layered, client-server, and Key-Value type table architecture styles, tree-topology clustering, a central controller component, and decoupling of the user interaction module strategy through the MVC pattern have been used to achieve maintainability, energy-efficiency, and interoperability.

RQ3: How to apply the proposed reference architecture for SEISs?

As a case study, the forest fire detection system as concrete SEIS has been developed to prove the applicability of the proposed RefSEISs in Chapter 8. Such that the RefSEISs was applied in the construction of forest fire detection system as a new concrete SEIS using the instantiation steps described in Section 7.3. When applied, less time and efforts have been utilized in the construction of the forest fire detection system since most of the

architecture knowledge encompassed by the RefSEISs have been reused. Additionally, the proposed RefSEISs has been used in mapping the some of the existing SEISs, i.e. Forest Fire Monitoring System (IPNAS) in Section 9.1, Urban Air Quality Monitoring System in Section 9.2, Indoor Air Quality Monitoring System in Section 9.3, and Flood Risk Assessment System in Section 9.4. These mappings demonstrated that the proposed RefSEISs can provide insights into how to improve the existing SEISs since the similarities and differences have been drawn.

RQ4: How to evaluate the proposed reference architecture for SEISs?

The fourth and last objective is discussed in Chapter 10, which involve the evaluation of the proposed RefSEISs by discussing some scenarios based on the implementation of forest fire detection system architectural prototype and demonstrating the fulfilment of the specified requirements of the RefSEISs by the proposed designs.

In the context of an architectural prototype implementation, the RefSEISs was verified to be relevant since the specified requirements of both forest fire detection system and RefSEISs were satisfied. While the mapping of the existing SEISs on the RefSEISs validates the applicability of the proposed reference architecture in the existing SEISs. The essential conclusion that can be drawn from this evaluation is that the proposed RefSEISs managed to meet both functional and non-functional requirements; hence the RefSEISs is considered to be useful and relevant for the SEISs.

11.3 Future Works

Although in this research, essential software elements or components of the proposed reference architecture for smart environmental information systems (RefSEISs) have applied and the specified quality attributes of the reference architecture have been achieved, there is a room for further improvements. An obvious further step is to create more concrete architectures with the guidance of RefSEISs since only one case scenario have been used to demonstrate the applicability of RefSEISs in the creation of concrete SEISs. By conducting more case studies with the proposed reference architecture, its practical use can be shown while its quality improved.

In this work, tools were not the primary concern. Then a future research project could investigate practical, and useful tools or infrastructure for the constructions, and expansions of concrete SEISs using the RefSEISs (instantiations). The instantiation of this reference architecture is still performed in ad hoc and based entirely on the expertise of software architect. This research opens up further investigations regarding the standardization, and

protocols since during development of SEISs, it is essential to have the list of the best suitable options provided by the tool.

All these proposed future works can be integrated with the high-level views adopted within this thesis and thus generate a more explicit and detailed reference architecture that can be used as a guideline for the constructions of SEISs.

11.4 Main Contributions

The SEISs is coupled with the existence of various hardware devices, extensive use of off-the-shelf components, growth of the size, various stakeholders and multiple programming languages. The inevitable complexity of such systems makes the *development and maintenance of the SEISs to be difficult, time-consuming and require extensive knowledge of the domain*. Dealing with such complexity can be challenging for even most experienced software engineers such as architects, developers and maintainers if start from scratch. Since, in spite of the considerable relevance and established architectural knowledge of the existing SEISs, there is a lack of a *reference architecture for SEISs* that supports the development of SEISs. This thesis takes the step towards improving the development and maintenance of SEISs by developing a *reference architecture for SEISs*. The main contributions of this thesis are;

Set of Viewpoints for SEISs: An effective approach to describe an understandable and easier maintainable software architectural description is by partitioning the software architecture into a number of separate views. Each view addresses one aspect of the architecture using a specific set of models from a specific viewpoint. Therefore this thesis proposes a required *set of Viewpoints for SEISs* which include *conceptual*, *module*, *execution*, *code*, *topology*, and *data* viewpoints. Some of these viewpoints, i.e. conceptual, module, code and execution viewpoints are adopted from Siemens view model. Furthermore, these viewpoints have been integrated to ensure conceptual integrity and consistency in the construction of a well-integrated system. Such viewpoints are applied in the development of the *reference architecture for SEISs (RefSEISs)*.

Reference Architecture for SEISs: The description of the RefSEISs follows the ISO/IEC/IEEE 42010:2011 standard for the architectural description and encompasses the best practices established in the existing SEISs. Thus, the proposed RefSEISs includes various software artefacts and knowledge such as the sets of stakeholders, concerns,

requirements and architectural designs that could be reused in the construction of concrete SEISs architectures.

Applications: The proposed RefSEISs has been applied to several applications. First, the RefSEIS has been applied in the *construction of forest fire detection system* as new concrete SEIS to demonstrate the applicability of the proposed RefSEISs in real scenarios and verification of the design with the requirements. Secondly, a conceptual validation is presented in which the proposed RefSEISs is used for *mapping some of the existing SEISs*; *Forest Fire Monitoring System (IPNAS)*, *Urban Air Quality Monitoring System*, *Flood Risk Assessment System* and *Indoor Air Quality Monitoring System*. These mappings demonstrated the proposed RefSEISs provides insights into how to improve the existing SEISs since the similarities and differences of existing SEISs with respect to RefSEISs have been described.

The proposed RefSEISs supports effectively the constructions of new concrete SEISs since the time and efforts required for developing and maintaining forest fire detection system have been substantially reduced. And also the mappings have demonstrated that the proposed RefSEISs can be used to compare and improve the existing SEISs by describing the similarities, differences, and how such SEISs can be improved. Results indicated that the RefSEISs has a positive influence in achieving the requirements of SEISs. This RefSEIS presents, therefore, good perspectives to be adopted and contribute to the development and maintenance of SEISs. To conclude, this thesis has proposed a reference architecture for smart environmental information systems (RefSEISs) based on literature reviews and architecture analysis, underlining its functionality and other factors that influence the development and maintenance of SEISs. Also, the areas that require further improvements have been identified to establish perfect RefSEISs.

Bibliography

Mohamed Abdelaal. Distributed techniques for energy conservation in wireless sensor networks. In *Doctoral Consortium - DCSENSORNETS, (SENSORNETS 2015)*, pages 9–20. SciTePress, 2015.

W. Abdelmoez, M. Shereshevsky, R. Gunnalan, H. H. Ammar, Bo. S. Bogazzi, M. Korkmaz, and A. Mili. Quantifying software architectures: an analysis of change propagation probabilities. *Computer Systems and Applications in The 3rd ACSIEEE International Conference*, 3:124, 2005.

Sherin Abraham and Xinrong Li. Design of a low-cost wireless indoor air quality sensor network system. *International Journal of Wireless Information Networks*, 23:57–65, March 2016.

Frank J. Affonso and Elisa Y. Nakagawa. A reference architecture based on reflection for self-adaptive software. In *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse*, pages 129–138, September 2013.

Tanuj Ahuja, Vanita Jain, and Shriya Gupta. Smart pollution monitoring for instituting aware travelling. *International Journal of Computer Applications*, 145(9):4–11, July 2016.

A. R. Al-Ali, Imran Zualkernan, and Fadi Aloul. A mobile gprs-sensors array for air pollution monitoring. *IEEE Sensors Journal*, 10(10):1666–1671, October 2010.

Marketing Alasia. An early warning system for forest fires, 2013. URL http://www.fire-watch.de/system-overview. Accessed: 2017-12-30.

Jaspa Albers. Comparative analysis of the forest fire situation in central-eastern europe l, 2012.

A. Alesheikh, A., K. Oskouei, A., F. Atabi, and H. Helali. Providing interoperability for air quality in-situ sensors observations using gml technology. *International Journal of Environmental Science & Technology*, 2:133–140, June 2005.

Ahmad A. A. Alkhatib. A review on forest fire detection techniques. *International Journal of Distributed Sensor Networks*, 10(3), March 2014.

Robert Allen and David Garlan. A case study in architectural modelling: The aegis system. In *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD '96, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7361-3. URL http://dl.acm.org/citation.cfm?id=857204.858260.

Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, July 1997. ISSN 1049-331X. doi: 10.1145/258077.258078. URL http://doi.acm.org/10.1145/258077.258078.

Abdelkrim Amirat. Generic model for software architecture evolution. In *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 139–143, November 2012.

Sam Amirebrahimi, Abbas Rajabifard, Priyan Mendis, Tuan Ngo, Soheil Sabri, and Australia . A planning decision support tool for evaluation and 3d visualisation of building risks in flood prone areas. In *Floodplain Management Association National Conference*, May 2016.

Samuil Angelov, Jos J. M. Trienekens, and Paul Grefen. Towards a method for the evaluation of reference architectures: Experiences from a case. In *Software Architecture*, pages 225–240, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Samuil Angelov, Paul Grefen, and Danny Greefhorst. A framework for analysis and design of software reference architectures. *Information Software Technologies*, 54(4):417–431, April 2012.

Malik Arslan, Zainab Riaz, Adnan Kiani, and Salman Azhar. Real-time environmental monitoring, visualization and notification system for construction h&s management. *Journal of Information Technology in Construction (ITcon)*, 19:72–91, June 2014.

Ángel Asensio, Carlos Trasvina-Moreno, Ruth Blasco, Álvaro Marco, and Robert Casas. Wireless sensor networks in traffic management systems. *AICT*, pages 60–68, April 2015.

Yunus Emre Aslan, Ibrahim Korpeoglu, and Özgür Ulusoy. A framework for use of wireless sensor networks in forest fire detection and monitoring. *Computers, Environment and Urban Systems*, 36(6):614–625, 2012.

World Road Association. A guide for practitioners: Road network operations and intelligent transport systems), 2018. URL https://rno-its.piarc.org/en/its-basics-what-its-basic-its-concepts/stakeholders. Accessed: 2018-08-19.

Ioannis N. Athanasiadis and Pericles A. Mitkas. An agent-based intelligent environmental monitoring system. *In Management of Environmental Quality, An International Journal*, pages 238–249, 2004.

Muhammad Babar and Ian Gorton, editors. *Software Architecture*. Springer-Verlag Berlin Heidelberg, 2010.

Muhammad A. Babar and Ian Gorton. *Software Architecture*. Springer, 2011.

M. Bahrepour, N. Meratniat, and P. Havinga. *Automatic Fire Detection: A Survey from Wireless Sensor Network Perspective*. CTIT Technical Report Series. Centre for Telematics and Information Technology (CTIT), Netherlands, 2008.

Ziv Baida. Stakeholders and their concerns in software architectures, November 2001.

Bartosz Balis, Marian Bubak, Danie Harezlak, Piotr Nowakowski, Maciej Pawlik, and Bartosz Wilk. Towards an operational database for real-time environmental monitoring and early warning systems. *International Conference on Computational Science (ICCS)*, pages 2250–2259, 2017.

Elena Baralis, Tania Cerquitelli, Silvia Chiusano, Paolo Garza, and Mohammad R. Kavoosifar. Analyzing air pollution on the urban environment. In *39th IEEE International Convention In Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1464–1469, May 2016.

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2012.

Martin Bauer, Mathieu Boussard, Nicola Bui, Francois Carrez, Christine Jardak (SIEMENS, Jourik De Loof (ALUBE, Carsten Magerkurth (SAP, Stefan Meissner, Andreas Nettsträter (FhG IML, Alexis Olivereau, Matthias Thoma (SAP, Walewski Joachim, Julinda Stefa, and Alexander Salinas. Internet of things – architecture iot-a deliverable d1.5 – final architectural reference model for the iot v3.0, July 2013.

Douglas W. Bennett. *Designing Hard Software : The essential tasks*. Manning Publications Co., Greenwich Connecticut, 1997.

Luis Bernardo, Rodolfo Oliveira, Ricardo Tiago, and Paulo Pinto. A fire monitoring application for scattered wireless sensor networks - a peer-to-peer cross-layering approach. In *Proceedings of the International Conference on Wireless Information Networks and Systems (WINSYS 2007), Barcelona, Spain*, pages 189–196, July 2007.

Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. ISBN 0-201-67494-7.

Jan Bosch and Peter Molin. Software architecture design: evaluation and transformation. In *IEEE Engineering of Computer Based Systems Symposium (ECBS)*, 1999.

Mike Botts, George Percivall, Carl Reed, and John Davidson. *OGC® Sensor Web Enablement: Overview and High Level Architecture*, pages 175–190. Springer Berlin Heidelberg, Berlin, Heidelberg, October 2008.

Kechar Bouabdellah, Houache Noureddine, and Larbi Sekhri. Using wireless sensor networks for reliable forest fires detection. In *Proceedings of the 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013), Halifax, Nova Scotia, Canada*, pages 794–801, June 2013.

Alan W. Brown and John A. McDermid. The art and science of software architecture. *First European Conference (ECSA) Lecture Notes in Computer Science*, 4758:439–466, 2007.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996. ISBN 0471958697, 9780471958697.

Natural Resources Canada. Canadian wildland fire information system (cwfis), 2018. URL http://cwfis.cfs.nrcan.gc.ca. Accessed: 2018-01-30.

Valentina Casola, Andrea Gaglione, and Antonino Mazzeo. A reference architecture for sensor networks integration and management. In *GeoSensor Networks*, pages 158–168. Springer Berlin Heidelberg, 2009.

Mauricio Castillo-Effen, D.H. Quintela, Wilfrido Moreno, Ramiro Jordan, and Wayne Westhoff. Wireless sensor networks for flash-flood alerting. In *Fifth IEEE International Caracas Conference on Devices, Circuits and Systems*, pages 142–146, December 2004. ISBN 0-7803-8777-5. doi: 10.1109/ICCDCS.2004.1393370.

Rashed Chowdhury. Consensus seasonal flood forecasts and warning response system (ffwrs): An alternate for nonstructural flood management in bangladesh. *Environmental management*, 35:716–25, July 2005.

Andrea L. Clements, William G. Griswold, Abhijit RS, Jill E. Johnston, Megan M. Herting, Jacob Thorson, Ashley Collier-Oxandale, and Michael Hannigan. Low-cost air quality monitoring tools: From research to practice (a workshop summary). *Sensors*, 17(11):2478, 10 2017.

Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. SEI Series in Software Engineering. Addison-Wesley, Boston, MA, 2001.

Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architecture - Views and beyond; SEI Series in Software Engineering*. Addison-Wesley, 2005.

Mario Collotta, Giovanni Pau, Gianfranco Scatà, and Tiziana Campisi. A dynamic traffic light management system based on wireless sensor networks for the reduction of the red-light running phenomenon. *Transport and Telecommunication*, 15(01):1–11, January 2014.

European Commission. European forest fire information system, 1998. URL http://effis.jrc.ec.europa.eu/. Accessed: 2017-11-30.

Daniel-Ioan Curiac and Constantin Volosencu. Urban traffic control system architecture based on wireless sensor-actuator networks. In *2nd International Conference on Manufacturing Engineering, Quality and Production Systems*, pages 1–5, March 2010.

C. C. Dakave and M. S. Gaikwad. Landslide detection and alert system using psoc. *International Journal of Innovative Research in Computer Science and Technology (IJIRCST)*, 3 (3), May 2015. ISSN 2347–5552.

Lívia Castro Degrossi, Guilherme G. Do Amaral, Eduardo S. M. De Vasconcelos, João Porto de Albuquerque, and Jo Ueyama. Using wireless sensor networks in the sensor web for flood monitoring in brazil. In *10th International ISCRAM Conference*, pages 458–462. ISCRAM Association, May 2013.

Christian Del Rosso. Continuous evolution through software architecture evaluation: a case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(5): 351–383, September 2006.

Remco M. Dijkman, Dick A. C. Quartel, and Marten J. van Sinderen. Consistency in multi-viewpoint design of enterprise information systems. *Information and Software Technology*, 50(7-8):737–752, June 2008.

K. Dimitropoulos, O. Gunay, K. Kose, F. Erden, F. Chaabene, F. Tsalakanidou, N. Grammalidis, and E. Cetin. Flame detection for video-based early fire warning for the protection of cultural heritage. In *Progress in Cultural Heritage Preservation*, pages 378–387, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Liliana Dobrica and Eila Niemelä. A survey on software architecture analysis methods. *IEEE Trans. Software Engineering*, 28:638–653, 2002.

Bettina Enders, Torsten Heverhagen, Michael Goedicke, Peter Tröpfner, and Rudolf Tracht. Towards an integration of different specification methods by using the viewpoint framework. *Transactions of the SDPS*, 6(2):1–23, 2002.

Miguel Ángel Esbrí, Jose Fernando Esteban, Martin Hammitzsch, Matthias Lendholt, and Edward Mutafungwa. Dews: Distant early warning system: Innovative system for the early warning of tsunamis and other hazards, 2011. URL http://www.dews-online.org. Accessed: 2018-12-28.

Richard F. Paige, Nikolaos Drivalos, Dimitrios S. Kolovos, Kiran Fernandes, Christopher Power, Gøran K. Olsen, and Steffen Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software and System Modeling*, 10:469–487, October 2011. doi: 10.1007/s10270-010-0158-8.

EU FP7. Network statistics of environmental change, resources and ecosystems (secure), 2016. URL https://www.gla.ac.uk/research/az/secure/. Accessed: 2018-04-30.

E. Gamma, R. Helm, R.and Johnson, and J. Vlissides. Design patterns. elements of reusable object-oriented software, 1994.

C. Gane and T. Sarson. *Structured Systems Analysis: tools and techniques*. Prentice-Hall, Englewood-Cliffs, NJ, 1977.

Deepak Ganesan, Alberto Cerpa, Wei Ye, Jerry Zhao, and Deborah Estrin. Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing*, 64(7): 799–814, July 2004.

Zhifu Gao and Linsheng Huang. A forest fire monitoring and early warning system based on the technology of multi-sensor and multilevel data fusion. In *2015 2nd International Conference on Electrical, Computer Engineering and Electronics*, pages 195–199. Atlantis Press, January 2015.

David Garlan, Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. *SIGSOFT Softw. Eng. Notes*, 19(5):175–188, December 1994. ISSN 0163-5948. doi: 10.1145/195274.195404. URL http://doi.acm.org/10.1145/195274.195404.

Priya George, Disha D. Malavika, and Soumya Deekshitha. Traffic monitoring system using ir sensors. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 5(IV):568–578, June 2017.

Jaap Gordijn, Hans de Bruin, and Hans Akkermans. Scenario methods for viewpoint integration in e-business requirements integration. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 7 - Volume 7*, HICSS '01, page 7032, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0981-9.

Phil Greenwood, Danny Hughes, Barry Porter, Paul Grace, Geoff Coulson, Gordon Blair, Francois Taiani, Florian Pappenberger, Paul Smith, and Keith Beven. Using a grid-enabled wireless sensor network for flood management. In *4th ACM Conference on Embedded Networked Sensor Systems*, pages 10–13, January 2006.

Group. IEEE recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, pages 1–23, 2000.

Oliver Günther. Environmental information systems. *SIGMOD Rec.*, 26(1):3–4, March 1997.

Anjaiah Guthi. Implementation of an efficient noise and air pollution monitoring system using internet of things (iot). *International Journal of Advanced Research in Computer and Communication Engineering*, 7(5):237–242, October 2007.

David Harel. Statecharts: A visual formalism for complex systems. *Science Computer Programming*, 8(3):231–274, June 1987.

Cian Harrington. *Definition and Verification of a Set of Reusable Reference Architectures for Hybrid Vehicle Development*. PhD thesis, Cranfield University, 2012.

Alexander Hars. *Rahmenbedingungen für die Nutzung von Referenzdatenmodellen*, pages 6–40. Gabler Verlag, Wiesbaden, 1994.

Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, MobiSys '06, pages 28–41, New York, NY, USA, 2006. ACM.

Stefanie Hass, Kai Walter, Frank Niemeyer, Christian Arnhardt, Kristine Asch, Raffig Azzam, Ralf Bill, Tomas Fernandez-Steeger, Stefan D. Homfeld, and Hartmut Ritter. Sensor-based landslide early warning system (slews), development of a spatial data infrastructure with integrated real-time sensor data as a basis for early warning systems exemplifying landslides. *Geotechnologien science report 10*, pages 69–70, 2008.

Stefanie Hass, Kai Walter, Frank Niemeyer, Christian Arnhardt, Kristine Asch, Raffig Azzam, Ralf Bill, Tomas Fernandez-Steeger, Stefan D. Homfeld, and Hartmut Ritter. Slews – a prototype system for flexiblereal time monitoring of landslides using an open spatial data

infrastructure and wireless sensor networks. *Geotechnologien science report*, 13:3–15, January 2009.

Mohamed Hefeeda and Majid Bagheri. Forest fire modeling and early detection using wireless sensor networks. *Ad Hoc and Sensor Wireless Networks*, 7:169–224, 2009.

Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. ISBN 0-201-32571-3.

A. Howard, Ashok Kochhar, and J. Dilworth. Application of a generic manufacturing planning and control system reference architecture to different manufacturing environments. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 213(4):381–396, April 1996.

Jian Huang, Runqiu Huang, Nengpan Ju, Qiang Xu, and Chaoyang He. 3d webgis-based platform for debris flow early warning: A case study. *Engineering Geology*, 197(C):57–66, 2015.

ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, January 2001.

ISO/IEC/IEEE. Systems and software engineering - architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, January 2011.

P. S. Jadhav and V. U. Deshmukh. Forest fire monitoring system based on zig-bee wireless sensor network, 2012.

Vana Jeličić, Michele Magno, Giacomo Paci, Davide Brunelli, and Luca Benin. Design, characterization and management of a wireless sensor network for smart gas monitoring. In *4th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 115–120, June 2011.

Víctor P. G. Jiménez and Julia M. F. García. Simple design of wireless sensor networks for traffic jams avoidance. *Journal of Sensors*, 2015:1–7, April 2015.

Simon Jirka, Arne Bröring, Peter Kjeld, Jon Maidens, and Andreas Wytzisk. A lightweight approach for the sensor observation service to share environmental data across europe. *Transactions in GIS*, 16:293–312, 2012.

Youngtae Jo, fJinsup Choi, and Inbum Jung. Traffic information acquisition system with ultrasonic sensors in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 10(5):1–12, May 2014.

Mohamed A. Kafi, Yacine Challal, Djamel Djenouri, Abdelmadjid Bouabdallah, Lyes Khelladi, and Nadjib Badache. A study of wireless sensor network architectures and projects for traffic light monitoring. *Procedia Computer Science*, 10:543–552, December 2012.

Igorce Karafilovski, Vladimir Zdraveski, and Dimitar Trajanov. Case studies of forest fire detection systems. *11th International Conference on Informatics and Information Technologies (CIIT)*, pages 243–248, April 2014.

Ruthbetha Kateule and Andreas Winter. Viewpoints for sensor based environmental information systems. In *Environmental Informatics-Stability, Continuity, Innovation*, pages 211–217, September 2016.

Harpreet Kaur and Pardeep Singh. Uml (unified modeling language): Standard language for software architecture development. *International Symposium on Computing, Communication, and Control (ISCCC 2009)*, pages 118–125, 2009.

Rick Kazman, Len Bass, Mike Webb, and Gregory Abowd. Saam: A method for analyzing the properties of software architectures. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 81–90, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

K. K. Khedo, R. Perseedoss, and A. Mungur. A wireless sensor network air pollution monitoring system. *International Journal of Wireless & Mobile Networks (IJWMN)*, 2(7): 31–45, May 2010.

Kavi K. Khedo and Vishwakarma Chikhooreeah. *Low-Cost Energy-Efficient Air Quality Monitoring System Using Wireless Sensor Network*, chapter 7. IntechOpen, Rijeka, October 2017.

Ahmad Kheir, Hala Naja, Mourad Oussalah, and Kifah Tout. Overview of an approach describing multi-views/ multi-abstraction levels software architecture. In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE*, pages 140–148. SciTePress, 2013.

J. Kim, C. Lee, T. Kwon, G. Park, and J. Rhee. Development of an agricultural data middleware to integrate multiple sensor networks for an farm environment monitoring system. *Journal of Biosystems Engineering*, 38:25–32, 2013.

Slobodan Kletnikov, Jugoslav Achkoski, Nikola Kletnikov, Igorche Karafilovski, Rumen Stainov, and Rossitza Goleva. Design of advanced system for monitoring of forest area and early detection of forest fires using drones, camera and wireless sensor network. *The 13th Annual International Conference on Computer Science and Education in Computer Science*, pages 281–296, 2017.

H. Z. Kotta, K. Rantelobo, S. Tena, and G. Klau. Wireless sensor network for landslide monitoring in nusa tenggara timur. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 9(1):9–18, April 2011.

P. Kruchten. Architectural blueprints - the "4+1" view model of software architecture. *IEEE Software*, 12(6):42–50, November 1995.

Philippe Kruchten, Henk Obbink, and Judith Stafford. The past, present, and future for software architecture. *IEEE Software*, 23(2):22–30, March 2006. ISSN 0740-7459.

Kiran D. Kumar and Suresh T. V. Kishore, G. snd Kumar. Fire monitoring system for fire detection using zigbee and gprs system. *IOSR Journal of Electronics and Communication Engineering*, 12:23–27, February 2017.

Somansh Kumar and Ashish Jasuja. Air quality monitoring system based on iot using raspberry pi. *International Conference on Computing, Communication and Automation (ICCCA2017)*, pages 1341–1346, 2017.

Hsu-Yang Kung, Jing-Shiuan Hua, and Chaur-Tzuhn Chen. Drought forecast model and framework using wireless sensor networks. *Journal of Information Science and Engineering*, 22(4):751–769, July 2006.

Michael Larsen, Steve De La Salle, and Dave Reuter. A reusable control system architecture for hybrud powertrains. *SAE International*, 111(3):2579–2583, 2002.

D. S. Lee, M. K. Holland, and N. Falla. The potential impact of ozone on materials in the uk. *Atmospheric Environment*, 30:1053–1065, 1999.

Bennett P. Lientz and E. Burton Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980. ISBN 0201042053.

Freya H. Lin, Timothy K. Shih, and Won Kim. An implementation of the cordra architecture enhanced for systematic reuse of learning objects. *IEEE Transactions on Knowledge and Data Engineering*, 21(6):925–938, June 2009.

SW Lin, B Miller, J Durand, G Bleakley, A Chigani, R Martin, B Murphy, and M Crawford. The industrial internet of things volume g1: reference architecture. *Industrial Internet Consortium*, pages 10–46, 2017.

Yu Liyang, Wang Neng, and Xiaoqiao Meng. Real-time forest fire detection with wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing*, volume 2, pages 1214–1217, 2005.

Jesús Lozano, José Ignacio Suárez, Patricia Arroyo, José M. Ordiales, and Fernando Álvareza. Wireless sensor network for indoor air quality monitoring. In *Chemical Engineering Transactions*, volume 30, pages 319–324, January 2012.

David C. Luckham and James Vera. An event-based architecture definition language. *IEEE Trans. Softw. Eng.*, 21(9):717–734, September 1995. ISSN 0098-5589.

Yajie Ma, Mark Richards, Moustafa Ghanem, Yike Guo, and John Hassard. Air pollution monitoring and mining based on sensor grid in london. *Sensors*, 8(6):3601–3623, 2008.

Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. *SIGSOFT Softw. Eng. Notes*, 21(6):3–14, October 1996. ISSN 0163-5948. doi: 10.1145/250707.239104. URL http://doi.acm.org/10.1145/250707.239104.

Rafael Marin Perez, Javier García-Pintado, and Antonio Skarmeta Gómez. A real-time measurement system for long-life flood monitoring and warning applications. *Sensors*, 12 (4):4213–4236, 2012.

Silverio Martínez-Fernández, Claudia P. Ayala, Xavier Franch, Helena Martins Marques, and David Ameller. Towards guidelines for building a business case and gathering evidence of software reference architectures in industry. *Journal of Software Engineering Research and Development*, 2(1):7, August 2014.

Saoudi Massinissa, Bounceur Ahcene, Euler Reinhardt, Kechadi Tahar, and Alfredo Cuzzocrea. Intelligent data mining techniques for emergency detection in wireless sensor networks, July 2016.

Nenad Medvidovic. Formal modeling of software architectures at multiple levels of abstraction. In *In Proceedings of the California Software Symposium*, pages 28–40, 1996.

Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. Modeling software architectures in the unified modeling language. *ACM Trans. Softw. Eng. Methodol.*, 11(1):2–57, January 2002. ISSN 1049-331X. doi: 10.1145/504087.504088. URL http://doi.acm.org/10.1145/504087.504088.

Johannes Meier and Andreas Winter. Model consistency ensured by metamodel integration. In *6th International Workshop on The Globalization of Modeling Languages (GEMOC), co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018)*, pages 408–415, Copenhagen, 10 2018. CEUR Proceedings of MODELS 2018 Workshops.

Johannes Meier, Heiko Klare, Christian Tunjic, Colin Atkinson, Erik Burger, Ralf Reussner, and Andreas Winter. Single underlying models for projectional, multi-view environments. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*, page (to appear), Prag, February 2019. SCITEPRESS.

Ulrich Meissena and Frank Fuchs-Kittowskib. Crowdsourcing in early warning systems. *7th International Congress on Environmental Modelling and Software Society (iEMSs)*, 2014.

Kheireddine Mekkaoui and Abdellatif Rahmoun. Short-hops vs. long-hops -energy efficiency analysis in wireless sensor networks. In *Third International Conference on Computer Science and its Applications (CIIA11)*, volume 825, pages 13–15, 12 2011.

Paulo F. Merson. Data model as an architectural view. *Research, Technology, and System Solutions*, October 2009.

James A. Milke and Thomas J. McAvoy. Analysis of signature patterns for discriminating fire detection with multiple sensors. *Fire Technology*, 31(2):120–136, 1995.

Pitu Mirchandani and Larry Head. A real-time traffic signal control system: Architecture, algorithms, and analysis. *Transportation Research Part C: Emerging Technologies*, 9(6): 415–432, 12 2001. doi: 10.1016/S0968-090X(00)00047-4.

Gerrit Müller, Eiri Hole, Robert Cloutier, Dinesh Verma, Roshanak Nilchiani, and Mary Bone. The concept of reference architectures. *Syst. Eng.*, 13(1):14–27, February 2010. ISSN 1098-1241.

Mark Moriconi, Xiaolei Qian, and R. A. Riemenschneider. Correct architecture refinement. *IEEE Transactions on Software Engineering*, 21:356–372, April 1995.

Johan Muskens, Michel R. V. Chaudron, and Rob Westgeest. Software architecture analysis tool - software architecture metrics collection. *Proceedings of the 3rd Progress workshop on embedded systems, Utrecht, Neatherlands*, 3:128–139, October 2002.

Johan Muskens, V Chaudron, Michel R, and Christian Lange. Investigations in applying metrics to multi--view architecture models. *Systems Engineering*, pages 372–379, September 2004.

Elisa Y. Nakagawa, Flavio Oquendo, José C. Maldonado, and Milena Guessi. Consolidating a process for the design, representation, and evaluation of reference architectures. In *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014*, pages 143–152, April 2014. ISBN 978-1-4799-3412-6.

Elisa Yumi Nakagawa, Pablo Oliveira Antonino, and Martin Becker. Reference architecture and product line architecture: A subtle but critical difference. In *Software Architecture*, pages 207–211. Springer Berlin Heidelberg, 2011.

Chinh D. Nguyen, Tan D. Tran, Nghia D. Tran, Tue H. Huynh, and Duc T. Nguyen. Flexible and efficient wireless sensor networks for detecting rainfall-induced landslides. *International Journal of Distributed Sensor Networks*, 11(11):235954, November 2015.

Priya A.P Nithya and C.K. Vanamala. Wireless sensor network (wsn) based weather monitoring in flood disaster management by using iot. *International Journal of Emerging Technology in Computer Science and Electronics (IJETCSE)*, 25:27–32, May 2018.

Bello Nuhu, Arulogun O.T., Ibrahim Adeyanju, and Abdullahi I.M. Wireless sensor network for real-time flood monitoring based on 6lowpan communication standard. *APTIKOM Journal on Computer Science and Information Technologies*, 1:12–22, September 2016.

University of Arizona, University of California PATH Program, Savari Networks, Inc, SCSC, and Econolite. Multi-modal intelligent traffic signal system, 2013.

OMG. OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1, August 2011.

Valiallah Omrani and Seyyed Ali Razavi Ebrahimi. Software architecture viewpoint models: A short survey. *ACSIJ Advances in Computer Science: an International Journal*, 2(5): 55–62, November 2013.

World Meteorological Organization(WMO). Manual on flood forecasting and warning, 2011. URL https://library.wmo.int/. Accessed: 2017-12-30.

Dejan Pajk, Mojca Stemberger, and Andrej Kovačič. Reference model design: An approach and its application. In *Proceedings of the International Conference on Information Technology Interfaces, ITI*, pages 455–460, 2012.

Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, pages 45–77, 2008.

G. Peinel, T. Rose, and R. S. Jose. Customised information services for environmental awareness in urban areas. *7th World Congress on Intelligent Transport Systems*, November 2000.

Luciano Picarelli. Improved knowledge of landslide hazard. *Landslides*, 11:745–745, October 2014.

Peter Popic, D. Desovski, Walid Abdelmoez, and Bojan Cukic. Error propagation in the reliability analysis of component based systems. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 53–62, December 2005.

Philipp Preuner, Anna Scolobig, Joanne Linnerooth-Bayer, David Ottowitz, Stefan Hoyer, and Birgit Jochum. A participatory process to develop a landslide warning system: Paradoxes of responsibility sharing in a case study in upper austria. *Resources*, 6(4), October 2017. ISSN 2079-9276.

Prima D. Purnamasari, Evan G. Sumbayak, Vicky D. Kurniawan, and Wulan R. Apriliyanti. Co pollution warning system for indoor parking area using fpga. *International Journal of Reconfigurable and Embedded Systems (IJRES)*, 2(2):64–75, July 2013.

Janus Putman. *Architecting with RM-ODP*. Prentice-Hall, Upper Saddle River, NJ, 2001.

Hemadri Prasad Raju. *Real time mobile monitoring of air quality in urban areas using solid state GAS sensors GPS and wireless network*. PhD thesis, St Peters University, 2014.

Maneesha Vinodini Ramesh. Design, development, and deployment of a wireless sensor network for detection of landslides. *Ad Hoc Networks*, 13:2–18, February 2014.

Kudva Ranjini, A. Kanthimathi, and Y. Yasmine. Design of adaptive road traffic control system through unified modeling language, February 2011.

Andreas Reidt, Matthias Pfaff, and Helmut Krcmar. Der referenzarchitekturbegriff im wandel der zeit. *HMD - Praxis Wirtschaftsinform.*, 55:893–906, 2018.

M. Rosemann and W. M. P. van der Aalst. A configurable reference modeling language information systems, 2007.

Nick Rozanski and Ein Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2 edition, 2011. ISBN 032171833X, 9780321718334.

Nick Rozanski and Eoin Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, Boston, 2005.

H. Sabit, A. Al-Anbuky, and H. Gholam-Hossein. Distributed wsn data stream mining based on fuzzy clustering. *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, pages 395–400, 2009.

Mark Safronov and Jeffrey Winesett. *Web Application Development with Yii 2 and PHP*. Packt Publishing, 2014.

F. Samadzadegan, H. Zahmatkesh, M. Saber, and H. J. G. khanlou. An interoperable architecture for air pollution early warning system based on sensor web. *ISPRS - Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1: 459–462, September 2013.

Ricardo Sanz and Janusz Zalewski. Pattern--based control systems engineering. *IEEE Control Systems Magazine*, 23(3):43–60, July 2003.

Annisa N. Sari, Adi Susilo, and Edi Susilo. The role of stakeholders in flood management: Study at ponorogo, indonesia. *The International Journal Of Engineering And Science (IJES)*, 2:01–10, October 2013.

Richard Schmidt. *Software Engineering: Architecture-driven software development*. Morgan Kaufmann, 2013.

Robert W. Schwanke. Architectural requirements engineering: Theory vs. practice, 2003. URL https://cs.uwaterloo.ca/~straw03/finals/Schwanke.pdf. Accessed: 2018-08-28.

Victor Seal, Arnab Raha, Shovan Maity, Shouvik K. Mitra, Amitava Mukherjee, and Mrinal K. Naskar. A simple flood forecastig scheme using wirelesss sensor networks. *International Journal of Adhoc Sensor and Ubiquitious Computing*, 3:45–60, March 2012.

Kewei Sha, Weisong Shi, and Oliver Watkins. Using wireless sensor networks for fire rescue applications: Requirements and challenges. *IEEE International Conference on Electro/Information Technology*, pages 239–244, May 2006.

Dhirendra Sharma. Real time online early forest fire detection using wsn in the western himalayan region of india. *International Journal of Engineering Science and Computing (IJESC)*, 6:3926–3928, 2016.

Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.

Ramiz Shikhaliyev. About methods for visualizing network monitoring. In *Problems of Cybernetics and Informatics (PCI)*, pages 1–2, September 2012. ISBN 978-1-4673-4500-2. doi: 10.1109/ICPCI.2012.6486280.

Byungrak Son and Jung-Gyu Kim. A design and implementation of forest-fires surveillance system based on wireless sensor networks for south korea mountains. In *IJCSNS International Journal of Computer Science and Network Security*, volume 6, 2006.

Darko Stipanicev, Tomislav Vuko, Maja Štula, and Ljiljana Bodro. Forest fire protection by advanced video detection system - croatian experiences, October 2018.

Vanessa Stricker, Kim Lauenroth, Piero Corte, Frédéric Gittler, Stefano De Panfilis, and Klaus Pohl. Creating a reference architecture for service-based systems: A pattern-based approach. In *Future Internet Assembly*, pages 149–160. IOS Press, 2010.

Riny Sulistyowati, Hari A. Sujono, and Ahmad K. Musthofa. A river water level monitoring system using android-based wireless sensor networks for a flood early warning system. *Proceedings of Second International Conference on Electrical Systems, Technology and Information (ICESTI)*, pages 401–408, 2015.

Jirapon Sunkpho and Chaiwat Ootamakorn. Real time flood monitoring and warning system. In *Songklanakarin Journal of Science and Technology*, volume 33, pages 227–235, April 2011.

K. Tavladakis and N. C. Voulgaris. Development of an autonomous adaptive traffic control system. In *In ESIT '99 - The European Symposium on Intelligent Techniques*, March 1999.

Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 51–63, New York, NY, USA, 2005. ACM.

Edward Udo and Etebong B. Isong. Flood monitoring and detection system using wireless sensor network. *The Asian Journal of Computer and Information Systems (ISSN: 2321 – 5658)*, 1:2321–5658, 01 2014.

Uznir Ujang, François Anton, and Alias A. Rahman. Unified data model of urban air pollution dispersion and 3d spatial city model: Groundwork assessment towards sustainable urban development for malaysia. *Journal of Environmental Protection*, 4(7):701–712, July 2013.

V. Ustad, A.S. Mali, and S. S. Kibile. Zigbee based wireless air pollution monitoring system using low cost and energy efficient sensors. *International Journal of Engineering Trends and Technology (IJETT)*, 10(9):456–460, April 2014.

Antonio Vallecillo and Etsi Informática. Rm-odp: The iso reference model for open distributed processing. *DINTEL Edition on Software Engineering*, pages 69–99, March 2001.

Hans van der Veer and Anthony Wiles. Achieving technical interoperability: The etsi approach. In *ETSI White Paper*, volume 3, April 2008. URL https://www.etsi.org/images/files/ETSIWhitePapers.

Francis Vanek, Peter Jackson, and Richard Grzybowski. Systems engineering metrics and applications in product development: A critical literature review and agenda for further research. *Systems Engineering*, 11:107–124, February 2008.

Bharathi Varadharajulu and D. Sridharan. Uml as an architecture description language. *International Journal of Recent Trends in Engineering*, pages 230–232, May 2009.

Matthias Vodel, Wolfram Hardt, and René Bergelt. A generic data processing framework for heterogeneous sensor-actor-networks. *International Journal On Advances in Intelligent Systems*, pages 483–492, December 2012.

Brett Warneke and Kristofer S. J. Pister. Mems for distributed wireless sensor networks. In *9th International Conference on Electronics, Circuits and Systems*, volume 1, pages 291–294, September 2002.

Gunarathna W.A.S.R., Dissanayake S.A., Darshana D.G.T., Bandara H.M.P.M., and D. Dhammearatchi. Suraki bhoomi: Landslide early warning system. *International Journal of Scientific and Research Publications (IJSRP)*, 7(7):544–548, July 2017.

WHO. Air quality guidelines for europe, european series 91. *Copenhagen, Denmark: World Health authority Regional Publications*, pages 1–198, 1999.

WHO. *Ozone and other photochemical oxidants*, volume 2, pages 175–186. WHO Regional Publications, Copenhagen: WHO Regional Office for Europe, 2000.

Carla Willis, Brian Van Wilgen, Kevin Tolhurst, Colin Everson, Peter D'Abreton, Lionel Pero, and Gavin Fleming. The development of a national fire danger rating system for south africa, 2001. URL http://www.daff.gov.za/doaDev/sideMenu/ForestryWeb/dwaf/cmsdocs/Elsa/Docs/Fire/DevofNatFireDangerRatingSystem2001.pdf. Accessed: 2018-08-30.

Andreas Winter. *Referenz-Metaschema für visuelle Modellierungssprachen*. PhD thesis, Koblenz University, 2000.

Alexander L. Wolf. Succeedings of the second international software architecture workshop (isaw-2). *SIGSOFT Softw. Eng. Notes*, 22(1):42–56, January 1997. ISSN 0163-5948.

Elias Yaacoub, Abdullah Kadri, Mohammed Mushtaha, and Adnan Abu-Dayya. Air quality monitoring and analysis in qatar using a wireless sensor network deployment. In *9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 596–601, July 2013.

Mohd A. M. Yunus, Sallehuddin Ibrahim, Mohd Taufiq M. Khairi, and Mahdi Faramarzi. The application of wifi-based wireless sensor network (wsn) in hill slope condition monitoring. *Jurnal Teknologi*, 73(3):75–84, February 2015.

Junguo Zhang, Wenbin Li, Ning Han, and Jiangming Kan. Forest fire detection system based on a zigbee wireless sensor network. *Frontiers of Forestry in China*, 3:369–374, September 2008.

Kan Zheng, Shaohang Zhao, Zhe Yang, Xiong Xiong, and Wei Xiang. Design and implementation of lpwa-based air quality monitoring system. *IEEE Access*, 4:3238–3245, January 2016.

Binbin Zhou, Jiannong Cao, X. Zeng, and Hejun Wu. Adaptive traffic light control in wireless sensor network-based intelligent transportation system. In *IEEE 72nd Vehicular Technology Conference - Fall*, volume 38, pages 1–5, September 2010.

Binbin Zhou, Jiannong Cao, X. Zeng, and Hejun Wu. Adaptive traffic light control of multiple intersections in wsn-based its. *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pages 1–5, May 2011.

Declaration of Authorship

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichungen, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Ruthbetha Kateule, 17.09.2019