Architecting Integrated Internet of Things Systems

Ömer Köksal

Architecting Integrated Internet of Things Systems

Ömer Köksal

Thesis committee

Promotor
Prof. Dr B. Tekinerdogan
Professor of Information Technology
Wageningen University & Research

Other members
Prof. Dr M. Akşit, University of Twente, The Netherlands
Prof. Dr U. Aßmann, TU Dresden, Germany
Prof. Dr M. G. J. van den Brand, Eindhoven University, The Netherlands
Prof. Dr J. H. Trienekens, Wageningen University & Research

This research was conducted under the auspices of the Wageningen School of Social Sciences (WASS)

Architecting Integrated Internet of Things Systems

Ömer Köksal

Thesis
submitted in fulfillment of the requirements for the degree of doctor
at Wageningen University
by the authority of the Rector Magnificus,
Prof. Dr A.P.J. Mol,
in the presence of the
Thesis Committee appointed by the Academic Board
to be defended in public
on Friday 6th of July 2018
at 11:00 a.m. in the Aula.

Ö. Köksal
Architecting Integrated Internet of Things Systems
218 pages

PhD thesis, Wageningen University, Wageningen, NL (2018)
with references and summary in English

ISBN: 978-94-6343-854-4
DOI: https://doi.org/10.18174/446106

to my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 Background

The internet is the interconnected networks to connect billions of computers and other devices. The origin of the Internet dates back to Advanced Research Projects Agency Network (ARPANET) that led to the development of protocols for interconnection of networks into a single network. Initially computers of the University of California Los Angeles and the Stanford Research Institute were interconnected after which many other universities and institutes became part of the ARPANET. The development of TCP/IP in 1970s and its implementation in ARPANET further paved the way and adoption for the ARPANET which was later renamed as the "Internet" in 1984.

The internet usage was dominated by e-mail and file transfer between 1970s and 1980s. With the invention of World Wide Web (WWW) in 1989 by Tim Berners Lee who wrote the first web browser, internet gained further momentum. The number of internet users increased rapidly from thousands to billions in 2000s. In 2018, more than 50% of the world population is now using the internet and this number will increase in the near future. The internet was a disruptive technology and had a dramatic impact on the global culture and social life.

Until recent time, the internet was primarily used for interconnecting computers any time and any place but this required human interaction and monitoring. The internet of things (IoT) is a new paradigm that adds a new dimension to the current information and communications technologies (ICTs), whereby the dimension "Anything communication" is added to the communication capabilities as shown in Figure 1-1. The IoT enables anytime, anyplace connectivity for anything, by linking the objects of the real world with the virtual world. In the IoT world physical things and virtual things, all interact with each other in the same space and time.

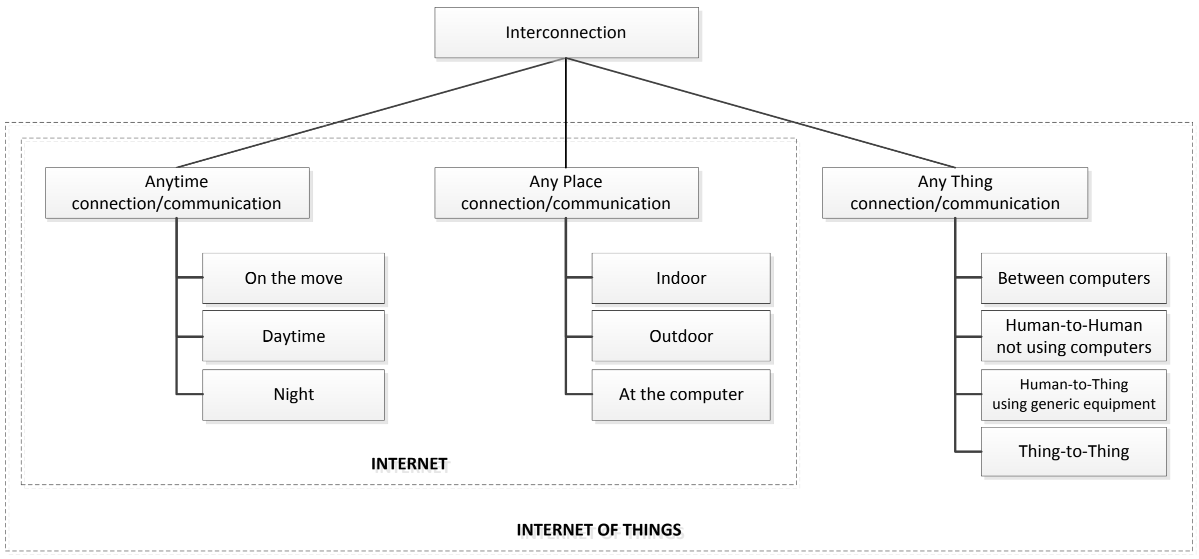

Interconnection
Anytime connection/communication
Any Place connection/communication
Any Thing connection/communication
On the move
Daytime
Night
Indoor
Outdoor
At the computer
Between computers
Human-to-Human not using computers
Human-to-Thing using generic equipment
Thing-to-Thing
INTERNET
INTERNET OF THINGS

Figure 1-1: Dimensions of Communication – IoT adds the "Any Thing" dimension

It is predicted that the number of devices connected to internet will be more than 50 billion (Evans, 2011). This means, that most of the internet traffic will not be among human beings but be between devices and all things connected to the internet resulting in more complicated and much wider Internet of Things.

The rise of the IoT is mainly dependent on three factors (Daniel Kellmereit, 2013): (1) Miniaturization, electronic devices are getting more powerful and energy efficient (2) Affordability, costs of electronic components and networks are consistently going down (3) De-wireization, more and more things are becoming wireless.

Since the IoT is the result of technological progress in many fields such as wireless sensor networks, machine-to-machine communication, mobile computing, ubiquitous computing, and embedded systems, the IoT might express different meanings to different people. The term Internet of Things was first used by Kevin Ashton in 1999 who was one of the co-founders of Auto-ID that aimed to investigate and understand what came next after the barcode. The Auto-ID has investigated RFID and realized that all physical objects can be traced via the internet by tagged them with RFID transponders. The IoT is defined by the International Telecommunication Union (ITU) as “The IoT is the network of physical objects or "things" embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data. Here “thing” is defined as: an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks (ITU, 2005). McEwen and Cassimally (McEwen & Cassimally, 2014) formulate the IoT with a simple equation as: “Physical Object + Controller, Sensor, and Actuators + Internet = IoT”.

Although the IoT system integrates many different entities it still deals with the design of a single system. Very often it is required though to integrate multiple IoT-based systems with other systems (Figure 1-2).

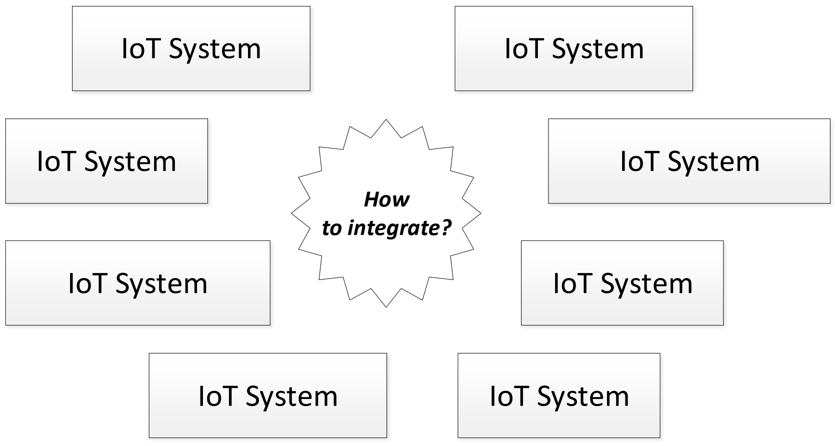


Figure 1-2: Integration of different IoT systems

An IoT system can typically realize a distributed system in which heterogeneous devices are connected over the internet. A distributed system consists of multiple software components that are located on networked computers, but act and run as a single system. The computers that are in a distributed system can be connected by a local network and be physically close to each other, or they can be connected in a wide area network and geographically distant.

Distributed systems offer many benefits over centralized systems, including scalability, concurrency and redundancy.

To reduce the effort for developing distributed systems, common middleware architectures have been introduced. The middleware provides common services such as name and directory services, discovery, data exchange, synchronization, and transaction services (Myerson, 2002). The publish/subscribe middleware adopts an event-driven approach based on publish/subscribe communication pattern. The publish/subscribe pattern has gained broad attention in the development of loosely coupled, scalable large-scale applications. One of the important and popular publish-subscribe middleware is the Data Distribution Service for Real-Time Systems (DDS) that has been defined by the Object Management Group (OMG) to provide a standard data-centric publish-subscribe specification for distributed systems. It appears that DDS has been applied to different domains including development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains.

1.2 Objectives and Research Questions

Despite of the overall vision of the IoT, the integration of multiple heterogeneous devices over the internet remains an important challenge. Hence, the main objective of this thesis is to analyze and design integrated IoT systems. In this context we have defined the following research questions:

RQ1. *What are the characteristic features of IoT systems?*

Before tackling the integration of IoT devices it is important to identify the current state of the IoT and likewise identify and describe the common and variant IoT features.

RQ2. *How to design the architecture for an IoT-based system?*

Our focus in this thesis will be at the design level. The architectural design of an IoT system is one of the early key artefacts that has a huge impact on the subsequent artefacts in the overall lifecycle. However, designing the IoT architecture is not trivial. For this we will investigate the current architecture approaches for IoT and propose an approach to guide the architect in designing feasible IoT architectures.

RQ3. *What are the identified obstacles in the DDS domain?*

For connecting the devices in an IoT system we will explicitly consider the adoption of middleware and hereby in particular DDS middleware. In the literature both the concepts of IoT and DDS have developed in parallel ways. For investigating the adoption of DDS for IoT it is important to identify and describe the features of IoT and herewith the current obstacles.

RQ4. *What are the solution directions for the identified obstacles of DDS?*

After addressing the obstacles of DDS, we will identify and describe the proposed solution directions. In particular, we will focus on obstacles and solution directions related to the adoption of DDS for IoT.

RQ5. *What are the approaches for integrating multiple IoT-based systems?*

IoT-based systems are often not standalone systems and require the integration with other systems. For investigating the integration of IoT-based systems several integration approaches can be used. We focus on most common integration patterns to investigate the integration of IoT-based systems.

RQ6. *How to design a DDS-based IoT system?*

DDS middleware targets the high-performance computing hence it is an important communication protocol for the IoT-based systems. We will provide an approach for designing the architecture of DDS-based IoT systems.

RQ7. *How to derive feasible deployment alternatives for DDS-based systems?*

A DDS-based system usually consists of multiple participant applications each of which has different responsibilities in the system. These participants can be allocated in different ways to the available resources, which leads to different configuration alternatives. We will provide a systematic approach for deriving feasible deployment alternatives based on the application design and the available physical resources.

1.3 CASE STUDIES

In order to illustrate our approaches for the research questions we have defined and used two different case studies, namely Farm Management Information Systems (FMIS) and Smart Traffic System (STS) in the context of Smart City Engineering.

1.3.1 Farm Management Information Systems

Precision farming adopts advanced technology to increase the amount of production and economic returns, often also with the goal to reduce the impact on the environment. One of the key elements of precision farming is the Farm Management Information Systems (FMIS) that supports the automation of data acquisition and processing, monitoring, planning, decision making, documenting, and managing the farm operations. An increased number of FMISs now adopts Internet of Things (IoT) technology to further optimize the targeted business goals.

Nowadays, FMIS adopt IoT technologies to further optimize the precision farming goals. IoT-based FMIS have different functional and quality requirements than traditional FMIS such as communication protocols, the amount of the data size to be processed, the security level, safety level, and time performance. In order to develop an IoT-based FMIS, one should design the proper architecture that meets the corresponding requirements.

The FMIS case study will be explored using two different industrial case studies on precision farming including smart wheat production FMIS, and greenhouse FMIS.

1.3.2 Smart Traffic System

For the near future, it is expected that a large part of the world population will live in urban areas. This will have a huge impact on future personal lives and mobility. A smart city uses information and communication technology (ICT) to enhance the quality and performance of urban services, to reduce costs and resource consumption, and to engage more effectively and actively with its citizens. Sectors that have been developing smart city technology include government services, transport and traffic management, water and waste, health care, and energy. Smart city applications are developed with the goal to improve the management of urban flows and allowing for real time responses to challenges. One of the important applications in smart city engineering includes the development of smart traffic system (STS). Traffic is already a large problem in many cities and this problem will be even bigger in the future. Many people spend a considerable amount of time in traffic, which leads to unnecessary waste of human resource, time and increase of CO2 emissions. STS provides different capabilities such as traffic light management, congestion detection, traffic regulation, shared parking platform, etc. For example, shared parking platform optimizes the search for finding a suitable parking slot by guiding the drivers to the available nearest parking spots in real-time.

STS consists primarily of sensors and vehicles. Sensors are the devices that monitor the environment and provide the corresponding data. Vehicles use the sensor data and publish their position and other relevant information to the STS. In order to manage vehicle and sensor data several IoT technologies might be used. In essence, STS is a data-intensive system and hence OMG's DDS Middleware is very suitable to realize STS.

1.4 Research Methodology

In order to provide answers to the defined research questions we have applied a set of research methodologies including:

- Systematic literature review (SLR)
- Design science research, and
- Case study research

The details of the applied research methodologies will be given in the next subsections. Table 1-1 shows the research methodologies for the identified research questions as adopted in the different chapters.

Table 1-1: Applied Research Methodologies for the identified research questions

Research Methodology	**Ch-1**	**Ch-2**	**Ch-3**	**Ch-4**	**Ch-5**	**Ch-6**	**Ch-7**
Systematic Literature Review		RQ1 RQ2			RQ3 RQ4		
Design Science Research			RQ2	RQ5		RQ6	RQ7
Case Study Research			RQ2				RQ7

Figure 1-3 shows the workflow depicting the adopted research methodologies used in the thesis together with the contributions of each step. Firstly, we have applied two literature reviews in parallel. In chapter 2 we have applied a domain analysis for the IoT and our contribution in this chapter is the domain model for the IoT. In chapter 5 we have applied systematic literature review methodology for deriving the obstacles and solution directions for the Data Distribution Service middleware.

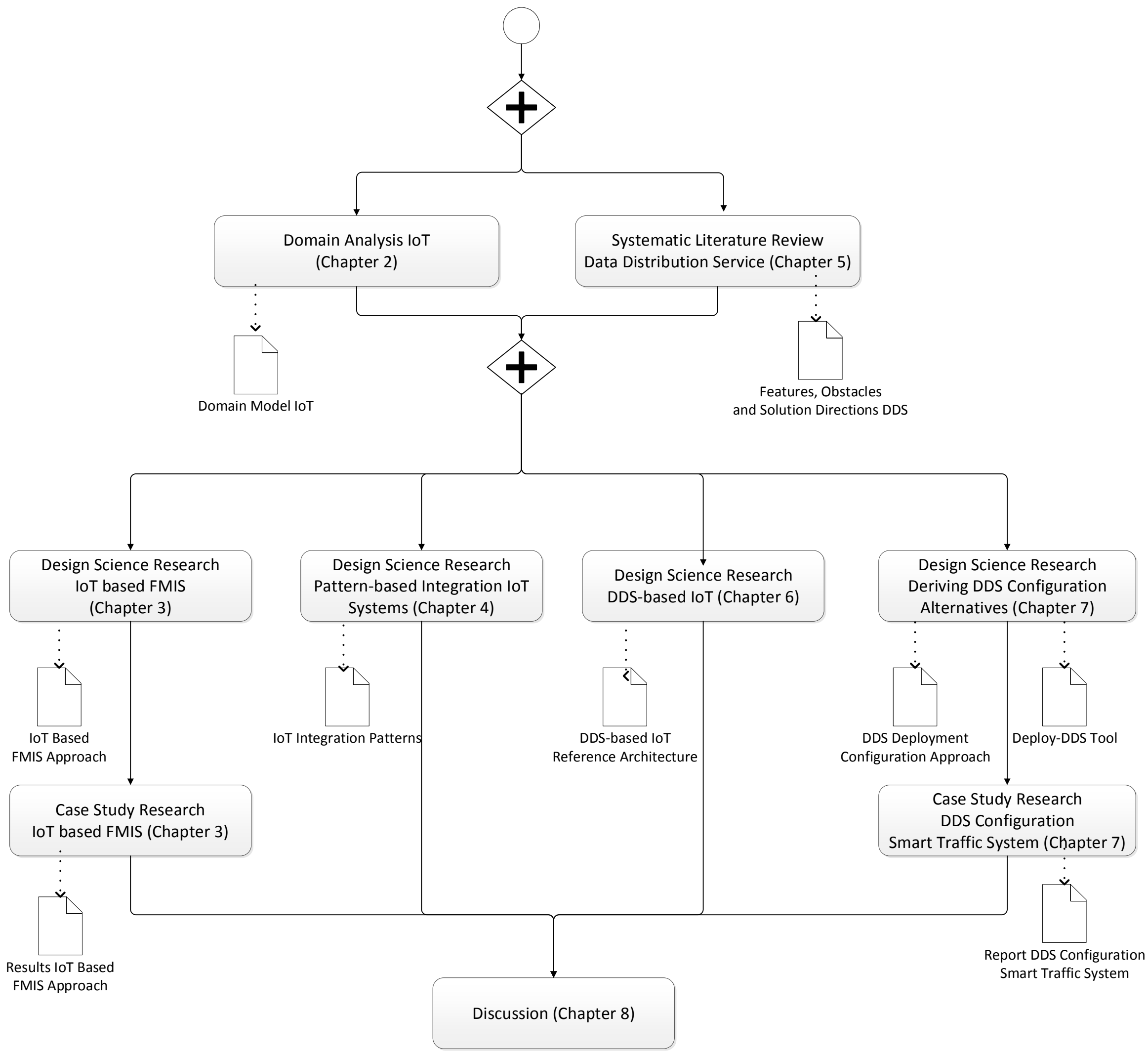

Domain Analysis IoT (Chapter 2)
Systematic Literature Review Data Distribution Service (Chapter 5)
Domain Model IoT
Features, Obstacles and Solution Directions DDS
Design Science Research IoT based FMIS (Chapter 3)
Design Science Research Pattern-based Integration IoT Systems (Chapter 4)
Design Science Research DDS-based IoT (Chapter 6)
Design Science Research Deriving DDS Configuration Alternatives (Chapter 7)
IoT Based FMIS Approach
IoT Integration Patterns
DDS-based IoT Reference Architecture
DDS Deployment Configuration Approach
Deploy-DDS Tool
Case Study Research IoT based FMIS (Chapter 3)
Case Study Research DDS Configuration Smart Traffic System (Chapter 7)
Results IoT Based FMIS Approach
Report DDS Configuration Smart Traffic System
Discussion (Chapter 8)

Figure 1-3: Research methodologies used in the thesis and contributions

Subsequently we have applied four parallel design science research activities. In chapter 3, we performed design science research for architecting an IoT-based FMIS. In chapter 4 we defined the pattern-based integration approach. In chapter 6 we designed the DDS based IoT architecture. Finally, we used design science research in chapter 7 in which we defined an approach for generating feasible DDS deployment alternatives.

The case study research has been applied in chapter 3 and chapter 7. In chapter 3 we applied case study research for evaluating the proposed architecture design approach. In chapter 7 we applied case study research for evaluating the DDS deployment configuration approach using also the developed Deploy-DDS tool.

1.4.1 Systematic Literature Review

For answering research questions RQ1, RQ2, RQ3, and RQ4 we have applied an SLR. A systematic literature review or systematic review for short is a well-defined and rigorous method to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest. The goal of an SLR is to give a fair, credible and unbiased evaluation of a research topic using a trustworthy, rigorous and auditable method. The inception of systematic reviews is based on the evidence-based concept which is devised in the field of medicine. The success of evidence-based medicine has triggered many other disciplines to adopt a similar SLR approach, including for example psychiatry, nursing, social policy, and education. Similarly, the concept of evidence-based software engineering (EBSE) (Dybå, Kitchenham, & Jorgensen, 2005) has been introduced together with guidelines for performing systematic literature reviews in software engineering (Kitchenham & Charters, 2007). There are several reasons for undertaking a systematic literature review including summarizing the existing evidence concerning a treatment or technology, identifying any gaps in current research in order to suggest areas for further investigation, providing a framework/background in order to appropriately position new research activities, examining the extent to which empirical evidence supports/contradicts theoretical hypotheses, or assisting in the generation of new hypotheses. The goal of EBSE is to improve the quality of software-intensive systems and provide insight to stakeholder groups whether practitioners are using best practice or not. In our study, we aimed at identifying the obstacles regarding the DDS concepts. Different approaches have been presented in the literature for conducting SLRs in different domains. We followed the guidelines for performing SLRs as proposed by Kitchenham and Charters (Kitchenham & Charters, 2007). The SLR has been applied in chapter 2 and chapter 5. In chapter 2 we have applied a literature review to identify the features of IoT. In chapter 5 we have applied the complete protocol to identify the features and obstacles of DDS middleware.

1.4.2 Design Science Research

For answering the research questions RQ2, RQ5, RQ6, and RQ7 we have applied design science research.

In this thesis we apply the design science methodology according to Hevner (Hevner, 2007). Design science research follows three iterative cycles: relevance cycle, design cycle and rigor cycle. The relevance cycle motivates the desired improvement that should be brought about to an environment. It also leads to a list of requirements and associated criteria for evaluating the research results. The design cycle turns the requirements into new design artefacts using an existing body of design knowledge. The rigor cycle contributes to the body of design knowledge. We applied case study methodology for the relevance cycle. We applied feature modeling and architecture modeling techniques in the design cycle. We applied case study methodology, demonstration and review of related work for the rigor cycle (Hevner, 2007).

1.4.3 Case Study Research

For answering the research questions RQ2 and RQ7 we have case study research in chapter 3 and chapter 7, respectively.

Our primary objective is to evaluate the impact of the developed architecture design method for IoT-based FMIS. To evaluate the above research questions, we have applied the case study research protocol as defined by Runeson and Höst (Runeson & Höst, 2008). Based on this we have followed the five steps: (1) case study design (2) preparation for data collection (3) execution with data collection on the studied case (4) analysis of collected data (5) reporting. We explain the execution and details of each step in the corresponding chapters.

1.5 CONTRIBUTIONS

This thesis provides the following contributions:

1. Identification of the current features of the IoT systems

To identify the features of IoT systems we have applied a feature-based domain analysis approach. Hereby feature diagrams have been used to model the common and variant features of IoT-based systems. The feature diagram has been organized based on reference architecture for IoT that includes multiple different layers. We have in particular focused on session layer that which is responsible for setting up and taking down of the association between the IoT connection points. For supporting the communication among the different IoT entities many different communication protocols are now available in practice. Based on the resulting feature diagram we can explicitly characterize the existing session layer communication protocols for a given IoT system. Further we have defined the criteria for selecting the identified communication protocols for the different conditions.

2. Software architecture design of IoT-based FMIS

We provide a systematic approach for guiding the architect in designing IoT-based FMIS. To this end, we adopt a feature-driven domain analysis approach to model the various different precision farming requirements. Further, based on FMIS and IoT reference architectures we describe the steps and the modeling approaches for designing the IoT-based FMIS architecture.

3. Architecting and designing integrated IoT-based systems

One of the key challenges in IoT is coping with the heterogeneous set of systems and the integration of these systems in the same communication network. Several studies have focused on this integration aspect and addressed this at different levels of abstraction. Unfortunately, the different approaches are scattered and fragmented over the different studies and it is not clear how to cope with the integration concern within a single IoT system but also across multiple IoT systems that need to be integrated. To this end this we provide a comprehensive and systematic approach for identifying the key integration concerns in the IoT system architecture and describe the currently provided solutions. For this we adopt a pattern-based approach in which generic architecture solution structures are provided to these recurring integration concerns. We illustrate our approach for addressing the integration of IoT-based systems within the context of smart city engineering.

4. Identification of obstacles of DDS middleware and corresponding solution directions

Data Distribution Service (DDS) is a standard data-centric publish-subscribe programming model and specification for distributed systems that can be used for integrating IoT systems. DDS has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. Various papers have been written on the application of DDS, however, there has been no attempt to systematically review and categorize the identified obstacles. We present the results of a systematic literature review (SLR) that has been conducted by a multiphase study selection process using the published literature since the introduction of DDS in 2003. We reviewed 468 papers that are discovered using a well-planned review protocol, and 34 of them were assessed as primary studies related to our research questions. Based on the SLR we have identified 11 basic categories of obstacles and the corresponding research challenges in DDS.

5. Architecture design of a DDS-based IoT system

Focusing on the architecture design of DDS-based IoT systems firstly, we describe the requirements for IoT systems and present the IoT reference architecture. Then we provide a DDS-based architecture for IoT systems based on the Views and Beyond Approach. We have performed a systematic approach in which we adopted architecture viewpoints for modeling DDS, IoT and finally DDS-based IoT systems. Since both the DDS and IoT are often represented as layered structures we have applied the layered viewpoint to represent the DDS-based IoT. Further we have also defined the deployment view for DDS-IoT. We can state that we succeeded to integrate and represent the architecture models that can be used to model DDS-based IoT systems for various application domains.

6. Systematic approach for deriving feasible deployment alternatives for DDS-based systems

We have provided a systematic approach by extending the DDS UML Profile, and an extensible tool framework to figure out the important design concern in DDS-based applications: the selection of the feasible deployment alternative given the application model, the physical resources, and the execution configurations. So far, this problem has not been explicitly addressed in the DDS literature. In general, the deployment configuration is selected manually based on expert knowledge. We provide a systematic approach for deriving feasible deployment alternatives based on the system design and the available physical resources. The approach showed to be useful in the modeling, the design and the evaluation of the DDS deployment alternatives. Furthermore, we have evaluated the approach for a relevant IoT case study on smart city engineering.

7. Tool support for designing DDS deployment alternatives

To support the method for finding feasible deployment alternatives in DDS-based systems we developed the Deploy-DDS tool. The tool supports the selection and generation of deployment architectures of DDS based systems with given system design and physical resources. The tool can be used to perform an evaluation during the design phase and generate the selected feasible configurations. The adoption of different algorithms and the ability to add new algorithms support the system architect also in the experimentation of different algorithms.

1.6 THESIS OUTLINE

Figure 1-4 shows the organization of the thesis. After this introduction section, chapter 2 presents the feature-driven domain analysis of Internet of Things. Hereby the common and variant features of IoT systems are explicitly defined and characterized. Chapter 3 describes an approach for architecting IoT systems for the farm management information system domain. Chapter 4 elaborates on the design process and considers the design of the integration of multiple IoT systems. Chapter 5 provides the results of a systematic literature review for identifying the current obstacles and solution directions when adopting DDS. This is in particular important for integrating multiple IoT systems. Based on the results of the earlier chapters, chapter 6 presents the DDS based architecture design for IoT systems. Chapter 7 presents an approach for deriving the feasible DDS configuration alternatives. Finally, chapter 8 provides the general discussion and reflects on the contributions of the thesis.

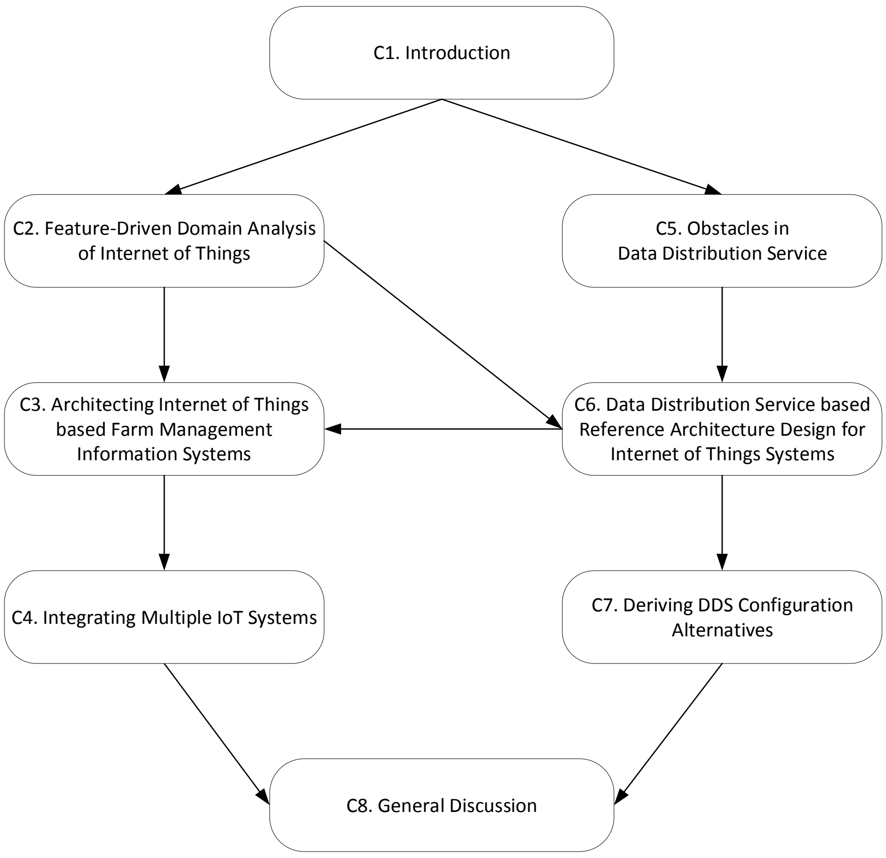

C1. Introduction
C2. Feature-Driven Domain Analysis of Internet of Things
C5. Obstacles in Data Distribution Service
C3. Architecting Internet of Things based Farm Management Information Systems
C6. Data Distribution Service based Reference Architecture Design for Internet of Things Systems
C4. Integrating Multiple IoT Systems
C7. Deriving DDS Configuration Alternatives
C8. General Discussion

Figure 1-4: Thesis outline

2

FEATURE DRIVEN DOMAIN ANALYSIS OF THE INTERNET OF THINGS[1]

[1] This chapter is based on the following published paper:

- Ö. Köksal and B. Tekinerdogan, "Feature-driven domain analysis of session layer protocols of internet of things," in Proceedings - 2017 IEEE 2nd International Congress on Internet of Things, ICIOT 2017, 2017, pp. 105–112.

Abstract

The Internet of Things (IoT) architecture is defined as a layered structure in which each layer represents a coherent set of services. For supporting the communication among the different IoT entities many different communication protocols are now available in practice. For practitioners, it is often not clear which communication protocol is suitable for the various conditions in which the IoT systems need to be operated. In this chapter, we focus on the session layer which is responsible for setting up and taking down of the association between the IoT connection points. We adopt a feature-driven domain analysis whereby we define the common and variant features that are related to communication protocols in the session layer. Based on the resulting feature diagram we explicitly characterize the existing session layer communication protocols. Further we define the criteria for selecting the identified communication protocols for the different conditions.

Keywords: Internet of Things, Session Layer Protocols, Message Queuing Telemetry Protocol, Extensible Messaging and Presence Protocol, Data Distribution Service, Advanced Message Queuing Protocol and Constrained Application Platform.

2.1 INTRODUCTION

Internet of Things (IoT) can be described as connecting all devices to the internet. The word “thing” can be defined as any physical device except computers since we have already connected computers to the internet. Mobile phones, tablets, medical devices, sensors, actuators are typical devices used in internet of things concept.

IoT connects billions of different devices with different data and connection characteristics. So, several network technologies and communication protocols are required to connect these devices to the internet.

In order to accomplish connection requirements, internet of things phenomenon consists of several layers including device/data link layer, network layer, session layer, and application layer as well as security and management layers. Each of these layers might use different protocols for the same purpose. From this perspective IoT can be defined as concourse of devices connected by communication software using different communication protocols. Every layer of the IoT architecture includes its own set of possible communication protocols.

Currently there are dozens of communication protocols that are defined by various different organizations and vendors. For practitioners, it is often not clear which communication protocol is suitable for the various conditions in which the IoT systems need to be operated.

In this chapter, we focus on the session layer which is responsible for setting up and taking down of the association between the IoT connection points. The session layer provides services related issues of the session such as initiation, maintenance, and disconnection. As such, frequency and duration of various types of sessions are related with the session layer. Also, session information might enforce encryption and other security measures.

Selection of the session layer protocol depends on many factors such as data size, number of devices to be connected, latency, etc. Depending on the application requirements different session layer protocols might be used in session layer of the IoT application.

We adopt a feature-driven domain analysis whereby we have identified the important knowledge sources and extracted and modeled the important features of the session layer communication protocols. The result of the domain analysis process, as such, is a feature model that defines the common and variant properties of the session layer communication protocols. Based on the resulting feature diagram we explicitly characterize the existing session layer communication protocols. Further we define the criteria for selecting the proper session layer communication protocol for different conditions.

The remainder of the chapter is organized as follows: Section 2.2 provides a short background on IoT architecture. Section 2.3 describes the feature model of IoT session layer protocols. Section 2.4 presents a survey for the current session layer protocols and provides selection criteria. Section 2.5 presents the related work and finally section 2.6 concludes this chapter.

2.2 THE IOT ARCHITECTURE

Various reference architectures have been provided for the IoT. In general, IoT architecture is represented as a layered architecture with various set of layers. Hereby, a layer simply represents a grouping of modules that offers a cohesive set of services. Based on the literature (Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015; Gazis et al., 2015; Gilchrist, 2016; Karagiannis, Chatzimisios, Vazquez-Gallego, & Alonso-Zarate, 2015; McEwen & Cassimally, 2014; Palattella et al., 2013; Pandya & Champaneria, 2015; Schneider, 2016; Sheng et al., 2013; Vermesan & Friess, 2014) we provide the reference architecture as shown in Figure 2-1.

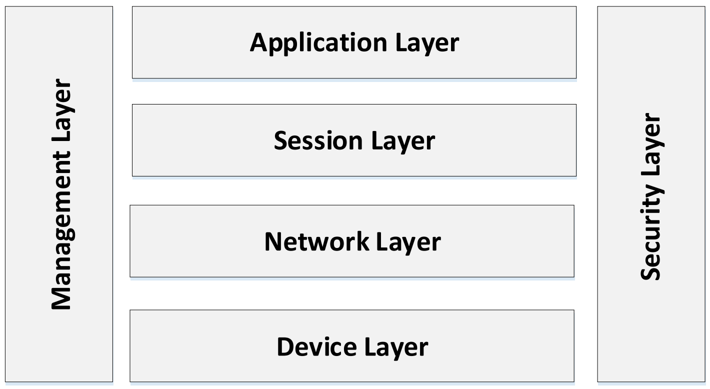


Figure 2-1: The IoT Reference Architecture

The reference architecture consists of four layers including device/datalink layer, network layer, session layer, and application layer. The device layer includes the capabilities for the things in the network. The network layer provides functionality for networking connectivity and transport capabilities. The IoT layered architecture consists of functionality for generic support capabilities (such as data processing or data storage), and specific support capabilities for the particular applications. The application layer contains the IoT application.

The Security layer is a side-car layer relating to the other four layers and provides the security functionality. Finally, the management layer supports capabilities such as device management, local network topology management, and traffic and congestion management.

2.3 Feature Driven Domain Analysis

In this section, we provide a feature-driven overview of IoT "Session Layer" protocols. For this purpose, we have carried out a thorough domain analysis process in which we selected and studied relevant set of studies that explicitly deal with IoT Communication Protocols. The domain analysis steps that we have adopted are shown in Figure 2-2.

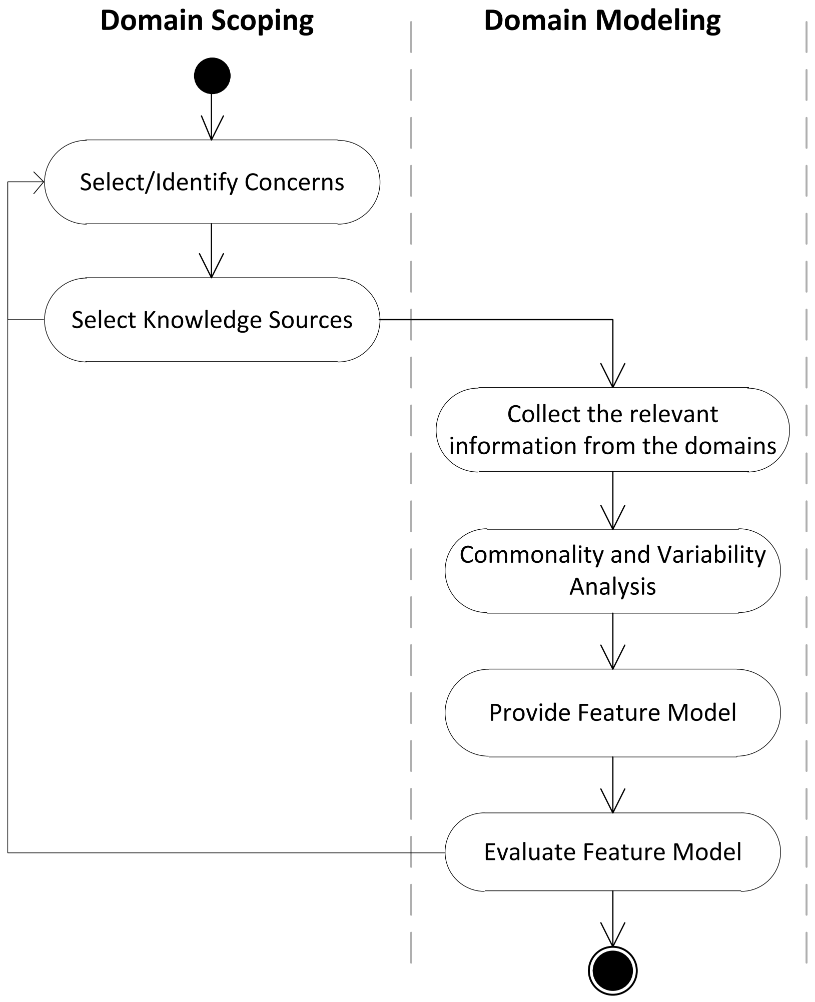


Figure 2-2: Domain Analysis Process

The process consists of two basic activities including domain scoping and domain modeling. In the scoping process, we define the scope of the domain analysis process and select the set of knowledge sources. In the domain modeling process the feature diagram is provided. A feature diagram is a tree with the root representing a concept (e.g., a software system), and its descendent nodes are features. Feature diagrams show both the mandatory and the variant features. Variant features are usually represented as optional or alternative features. A feature configuration is a set of features which describes a member of a communication protocol. A feature constraint further restricts the possible selections of features to define configurations. In our overview the root node represents the problem category, while the features represent the sub-problems, and optionally the sub-sub-features define the possible solutions, if these were described. The overall legend (abstract syntax) for the problem feature diagrams is given in Figure 2-3.

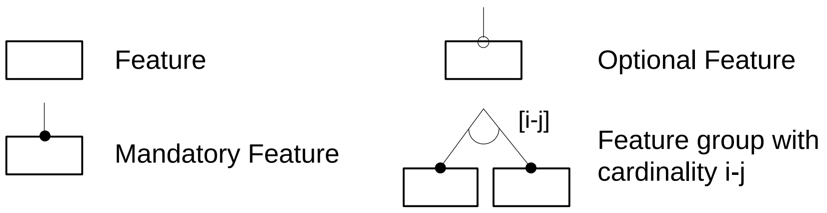


Figure 2-3: Legend for the feature diagrams

During the domain scoping process, we have looked at not only scientific papers but also considered websites and white papers of the important vendors and stakeholders in the IoT domain. It appears that there is a plethora of communication protocols that can be used in the different layers. In this paper, we focus on the communication protocols of the session layer. The selected list of important sources that we have considered is shown in Table 2-1.We do not claim that the set of sources is comprehensive but an analysis of these selected studies shows a convergence and agreement on the current set of protocols.

Table 2-1: Selected set of primary studies discussing the IoT protocols

ID	Primary Study
1	IoT – From Research and Innovation to Market Deployment (Vermesan & Friess, 2014)
2	Standardized Protocol Stack for the IoT (Palattella et al., 2013)
3	A Survey on the IETF Protocol Suite for the IoT (Sheng et al., 2013)
4	A Survey of Technologies for the IoT (Gazis et al., 2015)
5	IoT: Survey and Case Studies (Pandya & Champaneria, 2015)
6	IoT: A Survey on Enabling Technologies, Protocols and Applications (Al-Fuqaha, Guizani, et al., 2015)
7	Industry 4.0 - The Industrial IoT (Gilchrist, 2016)
8	RTI Whitepaper (Schneider, 2016)
9	Designing the Internet of Things (McEwen & Cassimally, 2014)
10	A survey on the Application Layer Protocols for the IoT (Karagiannis et al., 2015)

During the domain modeling process, we extracted the relevant data from the knowledge sources, compared the identified protocols as discussed in the different studies, derived the common and variant properties and mapped these to the feature diagram. The final step was the evaluation of the feature diagram which resulted in several iterations until we could consolidate the feature diagram.

2.4 FEATURE MODEL OF THE IOT SESSION LAYER PROTOCOLS

Figure 2-4 shows the top-level feature diagram that we could derive from the primary studies. In essence, communication protocols are distinguished for the four layers of the IoT architecture.

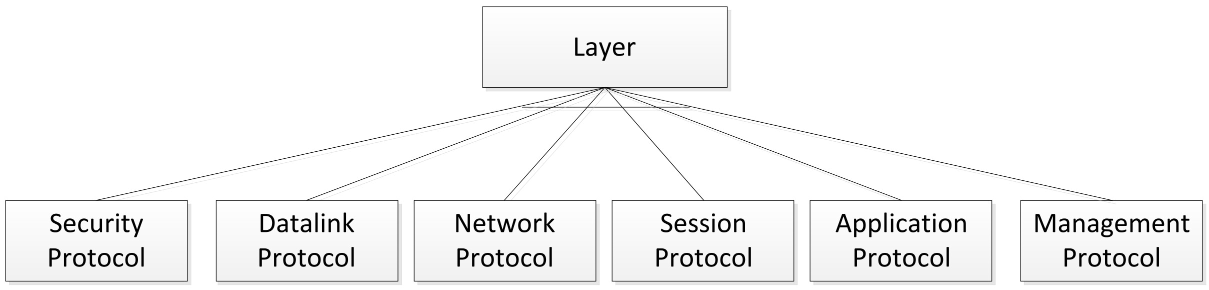


Figure 2-4: Top level feature diagram of the IoT

Based on the feature diagram as defined in the previous section we will characterize the existing session layer communication protocols provided in the literature. Figure 2-5 shows the feature diagram that we derived from the domain analysis to the IoT communication protocols.

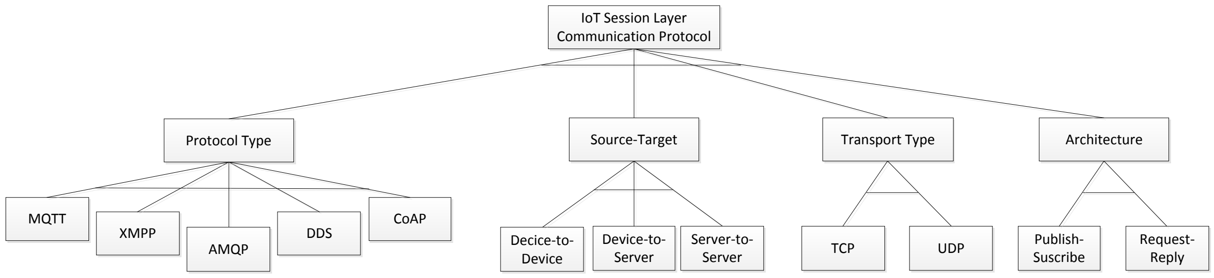


Figure 2-5: Feature diagram of session layer communication protocols of the IoT

The top-level mandatory features in the feature diagram are protocol type, source-target, transport type and architecture. The protocol type feature defines the protocols that we identify from the selected primary studies. These identified protocols are the following:

- Message Queuing Telemetry Transport (MQTT): One of the most popular protocols to collect device data and communicate with servers (Schneider, 2016).
- Extensible Messaging and Presence Protocol (XMPP): is based on exchanges of XML messages in real time that is defined to connect devices to servers (IETF, 2011).
- Advanced Message Queuing Protocol (AMQP): A queuing system designed to connect servers to each other (OASIS, 2011).
- Data Distribution Service (DDS): A fast data bus for integrating devices and systems (OMG, n.d.-a).
- The Constrained Application Protocol (CoAP): A specialized web-based protocol to be used in constrained nodes and constrained networks (Shelby, Hartke, & Bormann, 2004).

Please note that, although in some references Representational State Transfer (REST) is given an example of the communication protocol, it is not a real standard but a framework like SOAP (Gilchrist, 2016). So, it is not given here as a session layer protocol.

As given in Figure 2-5, there are three types of source-target relations available in session layer protocols: Device-to-Device (D2D), Device-to-Server (D2S), and Server-to-Server (S2S). In some references these features are also named Machine-to-Machine (M2M), Machine-to-Cloud (M2C), and Cloud-to-Cloud (C2C) respectively.

Session layer protocols are closely related with the transport layer. For all communication protocols, transport layer could be either UDP or TCP. Some protocols like DDS support both UDP and TCP. Addressing scheme (unicast, broadcast, or multicast) might be important depending on the application requirements.

The selection of transport layer protocol is important since using TCP and/or UDP changes the characteristics of the communication from performance and security perspectives. If low power devices and networks will be used, TCP is generally not available. So, the protocol that will be used must support UDP. On the other hand, not using TCP might introduce some security drawbacks. Because, security tools of TCP (SSL/TLS) are not available in UDP.

2.5 PUBLISH-SUBSCRIBE COMMUNICATION PATTERN

All of the session layer protocols given in this chapter support publish-subscribe communication pattern as given in Figure 2-6. A typical Publish-Subscribe system defines a Publish-Subscribe Domain which consists of a group of Participants which are deployed on a number of Application Nodes. Each Participant defines a number of Publisher and Subscribers that reads/writes Data Objects/Events. Data Objects/Events are elements of data exchange model of the publish-subscribe system.

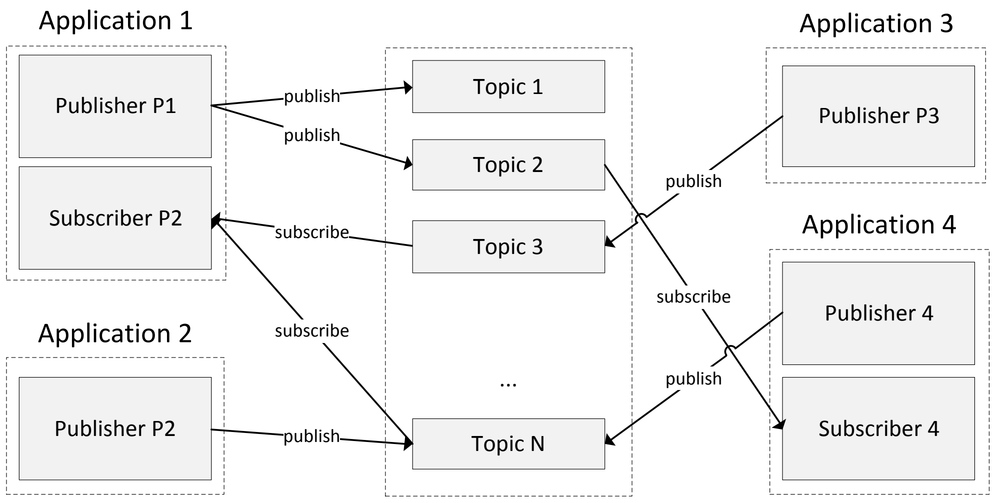


Figure 2-6: Publish-Subscribe Communication Pattern

Three different types of decoupling can be identified between the subscribers and publisher (Eugster, Felber, Guerraoui, & Kermarrec, 2003). Time decoupling refers to the fact that interacting components do not need to be actively participating in the interaction at the same time. Publishers might publish events independent of the subscribers, and subscribers might get notified about the occurrence of events even if the original publisher of the event is disconnected. Space decoupling refers to the fact that publishers and subscribers might not know each other and do not hold any reference to each other. Finally, synchronization decoupling refers to the fact that publishers and subscribers are not blocked during their actions. The Architecture Model of a middleware can be either centralized or decentralized denoting whether the data flows through a central unit or not. Further, the architecture model can include a broker that manages the data flow. The architecture can be unbrokered, i.e. there is no broker defined, or multi-brokered, whereby multiple brokers manage the data flow.

2.6 Selecting Communication Protocol

As we can observe from the domain analysis we can distinguish different protocols and different criteria for selecting these protocols. In this section, we will provide an evaluation framework for characterizing communication protocols and describe the identified communication protocols using the evaluation framework.

2.6.1 Evaluation Framework

In our domain analysis process, we have also extracted the criteria that have been mentioned when describing the communication protocols. We have summarized these criteria in the evaluation framework as shown in Table 2-2.

Table 2-2: Criteria for selecting communication protocols

Criteria	Description
Standard organization	What is the standardization organization?
Source-Target	What are the possible source-target relations?
Real-Time	Does the architecture allow real-time communication?
Brokered/Bus based	Is the architecture brokered or bus-based (unbrokered)?
Communication pattern	Is the adopted communication pattern including Pub/Sub or Request/Reply?
Message/Data centric	Is the protocol message or data centric?
Transport	What is the transport protocol (TCP/UDP)?
Interoperable	What is the level of interoperability?
QoS	Are the quality of service parameters defined?
Mobile support	Does the system provide mobile support?
Web/Application based	Is the protocol web-based or application-based?
Security	What is the adopted security protocol?
License	What is the license level of the communication protocol?

2.6.2 Session Layer Communication Protocol Types

Using the evaluation framework, we will characterize the five identified communication protocols in more detail.

1) MQTT

Message Queuing Telemetry Transport MQTT (OASIS, 2014) is a light-weight messaging protocol introduced by IBM in 1999. It is an open source protocol which is standardized by OASIS (OASIS, 2010) in 2013. It is designed for limited bandwidth networks and small code footprints. The main purpose of this protocol is remote monitoring. Data from devices is to be controlled and monitored within servers/cloud. It is especially suitable for large networks of small devices. Thousands of sensors in a single location can be connected for analysis.

MQTT mainly collects data from devices and transport to servers. So mainly it uses Device-to-Server (D2S) communication although Device-to-Device (D2D) communication is possible.

MQTT does not need to be fast, its real time is measured in seconds. MQTT uses broker based publish-subscribe architecture as given in Figure 2-7.

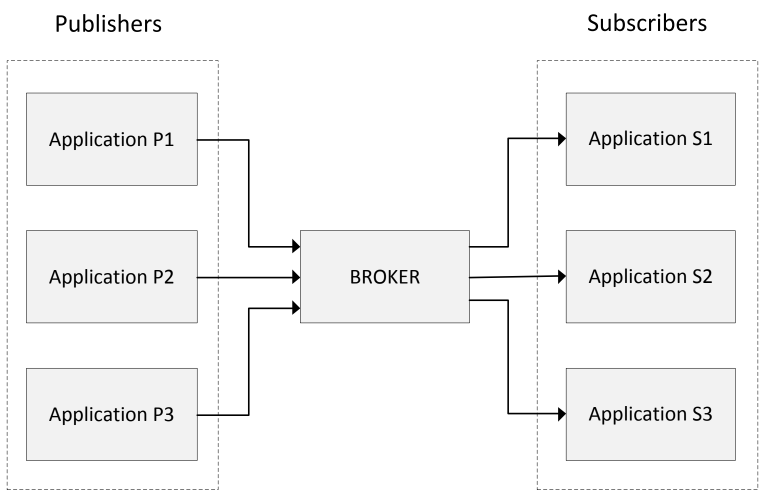


Figure 2-7: MQTT communication architecture

The architecture of MQTT consists of three main components namely publishers, subscribers and the broker. Publishers are generally the light weight sensors. They send their data to the broker and immediately sleep (if possible). Brokers classify sensor data coming from publishers into specified topics. Subscribers connect to the broker to get new/updated data. Subscribers are the applications that are interested in certain data. This architecture enables easier one to many messaging and low coupling between publishers and subscribers on contrary to tight coupling between client and servers in hypertext transfer protocol (HTTP).

MQTT supports hierarchical topics structures (like subject, sub-subject, sub-sub-subject) (Vermesan & Friess, 2014). Although MQTT supports asynchronous communication, it supports interoperability partially since the data is not negotiated between publishers and subscribers. Message formats must be known by clients. Also, it does not support labeling messages with types or metadata. MQTT use TCP for transport since device data shall not be lost. On the other hand, TCP might cause decreasing network efficiency as the number of nodes increase. Also, MQTT does not support dynamic discovery of nodes.

MQTT-S protocol is an extension of MQTT protocol that is suitable for constrained networks on which TCP is not available. This protocol allows MQTT to be used for sensor networks such as ZigBee (McEwen & Cassimally, 2014). Since MQTT protocol is simple, it might suffer from hacking (Vermesan & Friess, 2014). So, there are some secured versions of MQTT to introduce data encryption. For example, SMQTT is the secured version of MQTT protocol that is purposed to enhance security. This protocol uses encryption broadcasting in which one message is encrypted and delivered to multiple nodes. Although this type of encryption is widely used in IoT applications key generation and encryption algorithms are not standardized (Singh, Rajan, Shivraj, & Balamuralidhar, 2015).

2) XMPP

XMPP is a widely used session layer protocol based on XML. XMPP was standardized by Internet Engineering Task Force (IETF) (IETF, 2017). The use of XML makes this protocol extensible. It is a widely-used protocol for consumer IoT applications as well as for Software Defined Networks (SDN). On the other hand, the use of XML messages causes extra overhead and increase power consumption. Because of the high-power consumption and the complexity of the standard makes the protocol not suitable for embedded sensors with limited resources as well as sensor networks.

XMPP was initially called Jabber and designed for message exchange / instant messaging (IM) applications, i.e. people to people communication by text messaging. As the above definition implies XMPP uses Device-to-Server (D2S) communication. In order to address a device, it uses a simple and powerful addressing scheme in name@domain.com format. But it is not suitable for Device-to-Device communications (Schneider, 2016). It is near real-time and scalable to thousands of nodes. Its real time is measured in seconds. Since it is designed for near-real time applications it supports low-latency small messages. XMPP uses broker-less architecture. It supports publish/subscribe pattern as well as request response architecture. XMPP does not provide Quality of Services (QoS). XMPP uses TCP for the transport.

In XMPP, before transmitting data, its' data must be encoded. Hence, it is useful for devices with large communication traffic where extra security is required. As it was stated in the above paragraphs, XMPP was designed for instant message exchange and adopted to IoT later. New extensions for XMPP are still being offered to enhance the protocol application to the IoT. For example, in order to add service discovery XEP-0030, to add concentrators for connecting legacy sensors and devices XEP-0035 and to transport over HTTM, XP-0124 standards were added.

3) AMQP

AMQP is another session layer protocol especially designed for finance industry. AMQP is standardized by OASIS (OASIS, 2011). The main purpose of AMQP protocol is to handle thousands of queued transactions. It is mostly suitable for server-based analysis functions. It tracks messages and guarantee the delivery of the messages. Messages are delivered in three ways (1) At most once (once or never) (2) At least once (multiple delivery might exist) (3) Exactly once. AMQP sends transactional messages between servers (Schneider, 2016). So, it is a Server-to-server (S2S) protocol. The main features of AMQP architecture is message orientation, queuing and routing, reliability and security (Vermesan & Friess, 2014). AMQP is a message centric brokered protocol that supports publish-subscribe communication pattern similar to MQTT Figure 2-8. Unlike MQTT, the broker in AMQP consists of two main components: exchange and queues (Salman & Jain, 2017). Exchange receives messages from publishers and distributes them to the queues with respect to predefined rules. When data is available subscribers get the data in the subscribed to queues that are basically topics (OASIS, 2011). In addition, AMQP supports point-to-point communication and discovery of nodes which is managed by the broker.

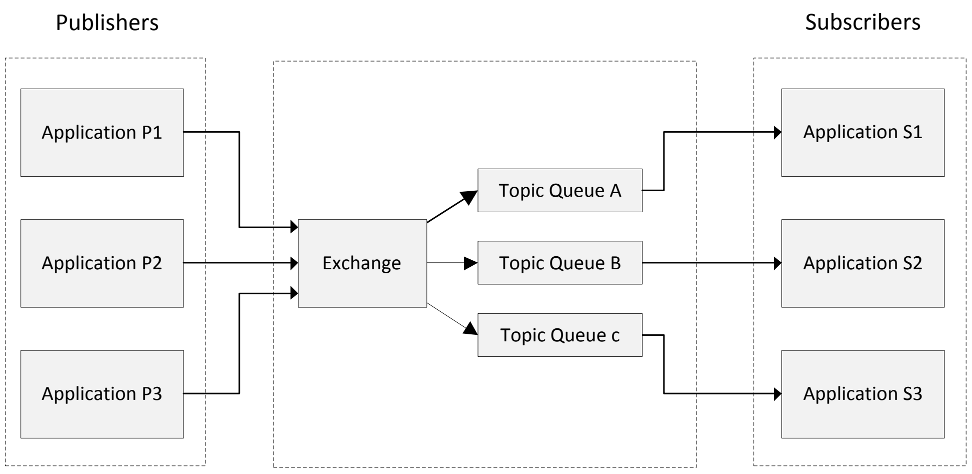


Figure 2-8: AMQP communication architecture

AMQP also provides strictly reliable point-point connection. Endpoints acknowledge the acceptance of messages. AMQP requires implementations from different vendors must be interoperable. AMQP is a wire-protocol that means any application that conform data format of AMQP can work with any other compatible application from a different vendor being language independent. In order, not to lose transactions, AMQP is designed to run over TCP. Using TCP as the transport might cause poor performance with increasing number of nodes. One of the key features of the AMQP is the security. It uses authentication and/or encryption based on SASL and/or TLS (Vermesan & Friess, 2014).

4) DDS

DDS is standardized by Object Management Group (OMG) (OMG, n.d.-b) in 2004 and the latest release is submitted in 2015 (OMG, 2015b). DDS is a data centric, scalable, real-time middleware for high performance machine-to-machine communications. DDS might get/send enormous data from many simultaneous publishers/subscribers. On the other hand, DDS, from the IoT perspective, has some obstacles in wide area network communication, wireless communication and mobile computing domains (Köksal & Tekinerdogan, 2017b).

Unlike MQTT and XMPP, DDS provides M2M communication. DDS is a real-time standard whereby real-time is measured in milliseconds or microseconds (Schneider, 2016). DDS uses bus-based communication architecture as given in Figure 2-9.

DDS support publish-subscribe pattern as many other session layer protocols providing operating system and programming language independency. DLRL layer might be used to convert data in application objects to the data-centric format of DCPS layer. DDS provides automatic discovery of nodes as well as supporting Quality of Service (QoS). DDS provides more than 20 QoS (depending on the vendor) in all open source and commercial implementations which simplifies complex network programming. QoS are also useful for several quality factors such as reliability, durability and scalability. DDS provides decoupled communication between participants. DDS has an interoperability standard (DDSI) to guarantee interoperability (OMG, 2014) between different vendors.

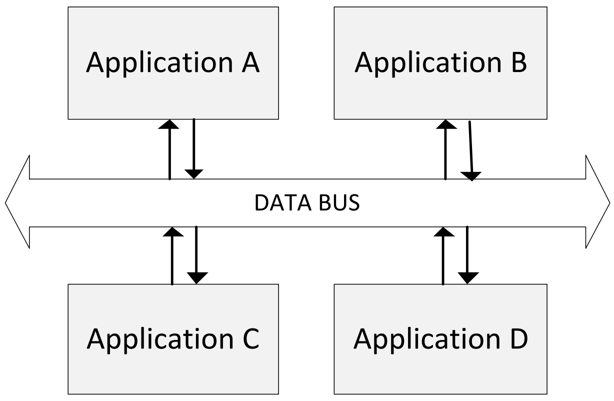


Figure 2-9: Typical bus-based communication architecture

As explained in above sections, most of the session layer protocols just support TCP in the transport layer. But DDS also supports UDP and multicast UDP. In fact, one of the powerful features of the DDS is supporting multicast UDP that enables high performance M2M communication. On the other hand, since multicast and UDP transports are not supported by many Wide Area Networks (WAN), some additional concepts like Interconnection Services or Routers shall be used in DDS systems to assure end-to-end QoS in WANs (Köksal & Tekinerdogan, 2017b).

DDS has both open source and commercial implementations. Lightweight versions are also available to run with limited or high-performance computing resources in embedded devices.

DDS was initially designed for LAN. But, as the interest in working with DDS in WAN increase, security issues gained more importance in DDS based systems. Security specification of DDS is just released (OMG, 2016) after a long-term beta version. DDS uses AES for data encryption and HMAC-SHA for message authentication.

5) CoAP

CoAP is a specialized document transfer protocol similar to HTTP which is standardized by IETF (IETF, 2017). It includes key Web concepts like URIs and internet media types. Representational State Transfer (REST) is the standard interface between client and servers of HTTP. But, REST requires high power consumption with respect to low power devices. CoAP is designed to provide RESTful interface for low power devices. Similar to MQTT-S, the main use of CoAP is in constrained nodes and constrained networks which do not support TCP. It is designed to interface with HTTP. It is a simple protocol with low overhead supporting multicast communication (Xi Chen, 2014).

Like DDS, CoAP is used in machine-to-machine (M2M) communications. CoAP uses a request-reply architecture providing automatic discovery of services and resources. It supports asynchronous communication and one-to-one communication between client and server.

CoAP use four different types of messaging depending on the application requirements. Confirmable and non-confirmable messages are for reliable and unreliable messaging respectively. For direct communication between client and server piggyback messages (acknowledgement) are used. Finally, separate messaging is used for server messages other than acknowledgement (Xi Chen, 2014). CoAP supports content negotiation since it is designed to be interoperable. Although, TCP is commonly used in HTTP, in low power devices and microcontrollers UDP is easier to implement. Since CoAP is designed to work on constrained networks, it uses UDP in the transport layer supporting both broadcast and multicast addressing. CoAP does not support TCP.

Although supporting UDP is important in constrained networks, not using TCP brings security issues since security tools of TCP (SSL/TLS) are not available in UDP. CoAP uses Datagram Transport Layer Security (DLTS) over UDP for secure communication. DLTS is similar to the TLS over TCP that provides authentication, automatic key generation and cryptography. Please note that DLTS has some drawbacks. First of all, it does not support multicast which is one of the major advantages of CoAP. Also, using DLTS causes packet increase in network traffic causing increased power consumption (Karagiannis et al., 2015).

2.6.3 Overall summary

In the previous sub-sections, we have described the session layer communication protocols using the evaluation framework in Table 2-2. In Table 2-3 we summarize the overall results. The table can be used to select the communication protocol for different situations.

Table 2-3: Adopted criteria for selecting communication protocols

Characteristics	**AMQP**	**CoAP**	**DDS**	**MQTT**	**XMPP**
Standard	OASIS	IETF	OMG	OASIS	IETF
Source-Target	S2S	D2D	D2D	D2S	D2S
Real-Time	No	No	Yes	No	Near RT
Broker/Bus based	Broker-based	Broker-based	Bus-based	Broker-based	Bus-based
Com. pattern	Pub/Sub	Request-Reply	Pub/Sub	Pub/Sub	Pub/Sub
Message/Data centric	Message	Data	Data	Message	Data
Transport	TCP	UDP	TCP/UDP	TCP	TCP
Interoperable	Yes	Yes	Yes	Partial	Yes
QoS	Yes	Yes	Yes	Yes	Yes
Mobile support	Yes	Yes	Yes	Yes	Yes
Web/App. based	App. based	Web based	App. based	App. based	App. based
Security	TLS/SSL	DTLS	AES/HMAC-SHA	TLS/SSL	TLS/SSL
License	Open source & Commercial	Open source & Commercial	Open source & Commercial	Open source & Commercial	Open source & Commercial

Given the requirements for different source-target communication it appears that different protocols are needed. For example: if the application will provide device-to-server communication MQTT and XMPP might be used. Regarding real-time constraints, only XMPP and DDS seem to be feasible. These protocols also require bus-based architecture. The other protocols do not provide real-time performance and mainly broker-based.

All the session layer protocols, except CoAP use publish-subscribe communication pattern. CoAP uses request-reply. Interoperability is an important concern for all the communication patterns. The TCP protocol is the most frequent used transport layer protocol, while the UDP protocol is only used in DDS and CoAP.

All the protocols provide QoS parameters. Further, both open source and commercial versions are available for all the protocols, and mobile support is provided.

Security is an important issue for all the protocols and this handled in different ways. The TLS/SSL protocol is an important protocol for data encryption.

The communication protocols are either message centric or data centric. All the protocols are application based, except CoAP which is web-based.

All the above criteria can be used to select the proper communication protocol given the contextual requirements. Based on the analysis practitioners might select the feasible protocol. In case more than one protocol is feasible additional functional and non-functional requirements might be considered.

2.7 CONCLUSION

For supporting the communication among the different IoT entities many different communication protocols are now available in practice. In this chapter, we have provided the results of the analysis to the IoT session layer protocols. For this we have adopted a systematic domain analysis process in which we first selected the important key knowledge sources that describe IoT session layer protocols. We have adopted feature modeling to model the common and variant features of the server layer communication protocols. It appeared that all the protocols adopt publish-subscribe architecture except CoAP. CoAP uses Request-Reply. To characterize the communication protocols in more detail we have provided an evaluation framework that includes the important criteria which we derived from the literature as well. The evaluation framework in the end appeared to be very useful not only for characterizing a single communication protocol but also for comparing the different communication protocols. The feature model of the communication protocols can be used by practitioners to select feasible communication protocols for their situational requirements. Researchers can use the results of this study to elaborate on further research on session layer protocols.

3

ARCHITECTING INTERNET OF THINGS BASED FARM MANAGEMENT INFORMATION SYSTEMS[1]

[1] This chapter is based on the following submitted paper:

- Ö. Köksal and B. Tekinerdogan, “Architecture Design Approach for Internet of Things Based Farm Management Information Systems”, Precision Agriculture Journal, 2018.

Abstract

Precision farming adopts advanced technology and the corresponding principles to increase the amount of production and economic returns, often also with the goal to reduce the impact on the environment. One of the key elements of precision farming is the Farm Management Information Systems (FMIS) that supports the automation of data acquisition and processing, monitoring, planning, decision making, documenting, and managing the farm operations. An increased number of FMISs now adopts Internet of Things (IoT) technology to further optimize the targeted business goals. Obviously IoT systems in agriculture typically have different functional and quality requirements such as choice of communication protocols, the amount of the data size to be processed, the security level, safety level, and time performance. For developing an IoT-based FMIS, it is important to design the proper architecture that meets the corresponding requirements. For guiding the architect in designing the IoT-based farm management information system that meets the business objectives a systematic approach is provided. To this end a design-driven research approach is adopted in which feature-driven domain analysis is used to model the various different precision farming requirements. Further, based on a FMIS and IoT reference architectures the steps and the modeling approaches for designing IoT-based FMIS architectures are described. The approach is illustrated using two different relevant industrial case studies on precision farming in Turkey, one for smart wheat production in Konya, and the other for smart green houses in Antalya.

Keywords: Precision Farming, Farm Management Information System, Internet of Things, Architecture Design

3.1 INTRODUCTION

Precision farming adopts advanced technology and the corresponding principles to increase the amount of production and economic returns, often also with the goal to reduce the impact on the environment. Similar related terms are used for the same purpose such as precision agriculture, site-specific farming, site-specific crop management, prescription farming, and satellite farming (Adamchuk, Hummel, Morgan, & Upadhyaya, 2004; Rains & Thomas, 2009; N. Zhang, Wang, & Wang, 2002). Precision farming builds on advanced technology such as cloud computing, remote sensing, data-driven farming, big data analytics, and IoT. Several important benefits of precision farming have been provided in the literature including optimizing production efficiency, optimizing quality of the crop, minimizing environmental impact, minimizing risk, Conservation of resources, reducing cost, increasing profit, and better management decisions (Rains & Thomas, 2009; Sørensen et al., 2010; Sørensen, Pesonen, Bochtis, Vougioukas, & Suomi, 2011; N. Zhang et al., 2002).

One of the key elements of the precision farming is the FMIS. Although FMIS started as a simple record keeping system, modern FMISs are sophisticated systems with advanced modules to supporting comprehensive set of farming operations (Fountas et al., 2015). With the introduction of IoT FMIS and precision farming in general have gained a new momentum. IoT helps in smart and automated information gathering and fusing as well as monitoring sensor data coming from different machines, animals, plants, other farms and greenhouses and other systems such as unmanned air and land vehicles. In this way, the decision making and planning in the agricultural domain can be further supported which can lead to even more effective and efficient farming. With the help of IoT, farming practices such as yield monitoring, cultivar selection, pest management, irrigation, etc. can be applied more precisely. Crop yield can be monitored and precise crop maps which show high and low production areas can be obtained readily (Rains & Thomas, 2009).

For developing an IoT-based FMIS it is important to design the proper IoT architecture which represents the overall gross level structure of the system. IoT-based farm management information systems typically have different functional requirements such as the type of crop, the feasible type of sensors and communication protocols, and the amount of the data size to be processed. Besides of functional requirements also quality requirements such as security level, safety level, time performance, and overall cost of development and operation are also different for different applications.

The different requirements typically require a different IoT architecture. In the literature, several reference architectures for FMIS and IoT have been proposed that can be reused to derive the IoT application architecture. Deriving the proper architecture however is far from trivial and this can impede the success of the IoT system.

The objective of this study is to contribute to the current state-of-art of FMIS by enhancing the current architecture design approaches for IoT-based FMIS. In particular, the study provides an answer to the following research question: *What is a suitable architecture design approach for designing IoT-based FMIS?* The presented approach adopts a feature-driven domain analysis approach to model the various different precision farming requirements. Further, based on FMIS and IoT reference architectures the steps and the modeling approaches for designing the IoT-based FMIS architecture are described. The approach is illustrated using two different relevant industrial case studies on precision farming in Turkey, one for developing IoT-based FMIS for smart wheat production in Konya, and the other for smart tomato production in greenhouses in Antalya.

The remainder of the chapter is organized as follows. In section 3.2 we provide the background on IoT, precision farming and architecture design. In section 3.3, we describe two case studies and the problem statement. In section 3.4 we present the approach for deriving concrete application architectures. Section 3.5 presents the feature model for FMIS and IoT, which will be used to support the design of the IoT-based FMIS. Section 3.6 presents the reference architecture for FMIS. Section 3.7 illustrates how our approach is applied to the case studies of Section 3.3. Section 3.8 presents the discussion. Section 3.9 presents the related work and finally section 3.10 concludes the chapter.

3.2 Background

3.2.1 Internet of Things

Until recent time, the internet was primarily used for interconnecting computers any time and any place, but this required human interaction and monitoring. The IoT is a new paradigm that adds a new dimension to the current information and communications technologies (ICTs), whereby the dimension "Anything communication" is added to the communication capabilities. The IoT enables anytime, anyplace connectivity for anything, by linking the objects of the real world with the virtual world. In the IoT world physical things and virtual things, all interact with each other in the same space and time.

The IoT is the result of technological progress in many parallel and often overlapping fields, including those of embedded systems, ubiquitous and pervasive computing, mobile telephony, telemetry and machine-to-machine communication, wireless sensor networks, mobile computing, and computer networking. In essence, IoT combines the concepts “Internet” and “Thing” and the provided definitions can be interpreted how these have addressed these two concepts. What is important is that IoT adds a new dimension to the current ICTs, which already provide "any time" and "any place" communication. Many definitions of IoT can be found in the literature. A feasible definition of IoT for the context of this chapter is the following (ITU, 2005): *The Internet of Things (IoT) is the network of physical objects or "things" embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data*.

Various reference architectures have been provided for the IoT. In general, IoT architecture is represented as a layered architecture with various set of layers. Hereby, a layer simply represents a grouping of modules that offers a cohesive set of services. Based on the literature (Al-Fuqaha, Guizani, et al., 2015; Gazis et al., 2015; Palattella et al., 2013; Pandya & Champaneria, 2015; Sheng et al., 2013) we provide the reference architecture as shown in Figure 3-1.

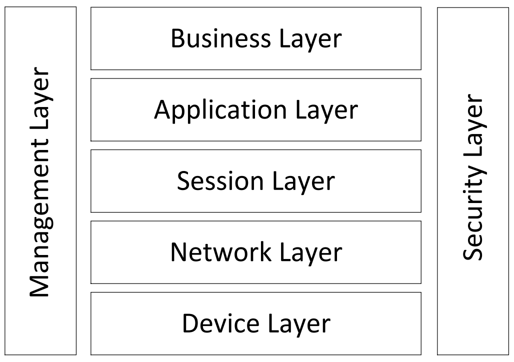


Figure 3-1: IoT reference architecture

The reference architecture includes the following layers: device layer, network layer, session layer, application layer, and business layer, management layer, and security layer. The device layer consists of sensors and physical devices. This layer identifies and collects data and specific information generated by sensors and physical devices. The data gathered is passed to the network layer. In essence the device layer bridges thus, the gap between the physical world and the digital world. The network layer provides functionality for networking connectivity and transport capabilities. This layer is also called transport layer. This layer securely transmits data gathered from sensors to the session layer. Transmission medium can be wired or wireless. The session layer is responsible for service management and consists of functionality for setting up and taking down of the association between the IoT connection points. Several session layer standards and protocols are offered by different organizations. Although most of these standards and protocols use TCP or UDP for transport, they have different architectures and characteristics that are suitable for different purposes. The application layer contains the IoT application and manages the application using the data from the session layer. The implemented IoT application can be, for example, smart farming, smart city, and smart home. The business layer defines business logic and workflows. This layer is responsible from the management of all IoT systems, services and applications within the domain. The business models are defined in this layer based on the data gathered from the application layer. The data is analyzed to build the required business models and define the strategies. The security layer is a side-car layer relating to the other five layers and provides the security functionality. Similarly, the management layer is a side-car layer supporting capabilities such as device management, local network topology management, and traffic and congestion management (Khan, Khan, Zaheer, & Khan, 2012; Köksal & Tekinerdogan, 2017a).

3.2.2 Architecture Design

Software architecture for a program or computing system consists of the structure or structures of that system, which comprise elements, the externally visible properties of those elements, and the relationships among them (Bass, Clements, & Kazman, 2003; Clements et al., 2011; Tekinerdogan, 2014). Software architecture forms one of the key artifacts in the entire software development life cycle since it embodies the earliest design decisions and includes the gross-level components that directly impact the subsequent analysis, design and implementation (Apel, Batory, Kästner, & Saake, 2013). It is generally accepted that software architecture design plays a fundamental role in coping with the inherent difficulties of the development of large-scale and complex software. Research on architecture design in the last two decades has resulted in different useful techniques and approaches.

Architectural drivers define the concerns of the stakeholders which shape the architecture. A stakeholder is defined as an individual, team, or organization with interests in, or concerns relative to, a system (Bass et al., 2003). Each of the stakeholders' concerns impacts the early design decisions that the architect makes. A common practice is to model different architectural views for describing the architecture according to the stakeholders' concerns (Clements et al., 2011; Demirli & Tekinerdogan, 2011; Tekinerdogan, 2014). An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern (Clements et al., 2011). Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An *architectural framework* organizes and structures the proposed architectural viewpoints.

A recent software architecture framework approach is the so-called Views and Beyond approach (V&B) (Clements et al., 2011). The approach distinguishes three different categories of viewpoints or styles including Module, component-and-connector, and allocation styles. Module view category is used for documenting a system's principal units of implementation and Component and Connector category is used for documenting the system's units of execution. Deployment View category that is used to document the relationships between a system's software and its development and execution environments.

A software architecture that addresses the concerns of specific stakeholders is here referred to as concrete architecture. A concrete architecture defines the boundaries and constraints for the implementation and is used to analyze risks, balance trade-offs, plan the implementation project and allocate tasks (Tekinerdogan, 2014). Concrete architectures can be viewed as instances of reference architectures, which are generic designs. In turn, a reference architecture is derived from the knowledge and experiences accumulated in designing concrete architectures in the past (Angelov, Grefen, & Greefhorst, 2012; Cloutier et al., 2010). The concrete architectures differ from one case to the next depending on the requirements of the stakeholders involved. Reference architectures can be used descriptively to "capture the essence of existing architectures" or prescriptively to guide the development of new ones (Cloutier et al., 2010).

3.3 Case Studies and Problem Statement

In this section, we describe the problem statement that we illustrate using two important industrial case studies of precision farming in Turkey. The case studies have been selected based on their relevance and their difference with respect to the functional and quality requirements. The case studies include the development of IoT-based FMIS for wheat production and tomato production in Turkey (Figure 3-2). We first describe the details of each case study and then follow up with the problem statement.

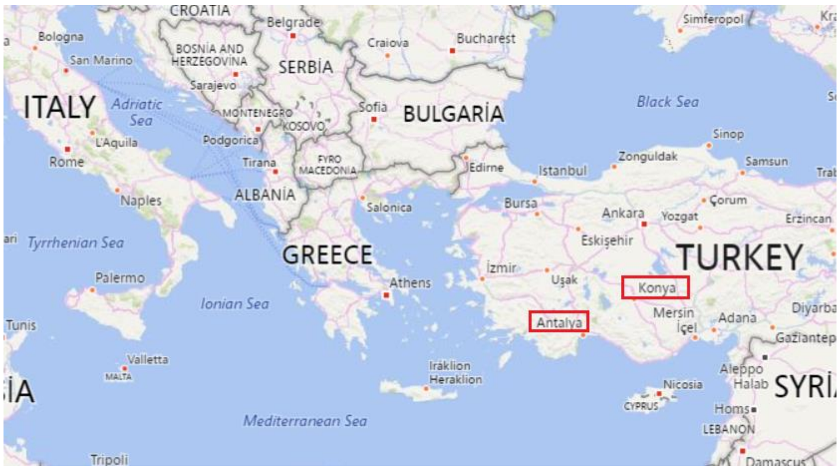


Figure 3-2: Location of Konya and Antalya cities [Bing Imagery]

3.3.1 Case Study: Wheat Production

Turkey has 23.9 million hectares of cultivated farms. Hereby, grain production takes 49% of this area. Wheat production constitutes 67% of the total grain production (Turkish Land Crop Office, 2017). Turkey's wheat production is about 20 million tons yearly (Turkish Ministry of Agriculture, 2018). As such, wheat production is one of the most relevant agriculture businesses in Turkey.

One of the key regions of wheat production in Turkey is the region of Konya which is far from costal area and located on a big plane near to middle of part of Turkey (Figure 3-2). Konya has a primarily terrestrial climate, and large plateaus and big arable farms of the city make Konya a suitable place for wheat production. Konya is the greatest wheat producer city of Turkey and has 3 million tons of wheat production yearly.

3.3.2 Case Study: Tomato Production in Greenhouses

The second case study includes the tomato production in greenhouses in Antalya. Tomato production in the world is 170 million tons yearly and almost 12 million tons of this production is produced in Turkey. Turkey exports tomato to many countries, and the total export is about 600000 tons. Tomato is produced both in farms and greenhouses. 51% of greenhouse production is tomato in Turkey.

Antalya is located in the south of Turkey just above the Mediterranean Cost as shown in Figure 3-2. The typical Mediterranean climate of the city is suitable for vegetable and fruit production. Currently, especially greenhouse farming is very common in Antalya. 80% of glass greenhouses and 50% of plastic greenhouses of Turkey exists in Antalya

3.3.3 Problem Description

Generally, wheat and tomato are produced with traditional farming in these regions. A general observation from governmental reports is that a small part of the farmers uses traditional FMIS (Turkish Land Crop Office, 2017; Turkish Ministry of Agriculture, 2018). Despite the use of FMIS, several problems in the agricultural sector could still be identified.

- Inefficient crop production

To meet the growing population in Turkey, it is important to increase the crop production. Over the last years, the percentage of the crop that is needed to feed the population tends to get lower. Hence, it is required that the production is increased, which requires a more effective and efficient crop production. According to the Turkish Statistical Institute (TSI), Turkey is one of the top 10 wheat producers in the world (Figure 3-3). But this production is not sufficient for Turkey’s growing internal demand.

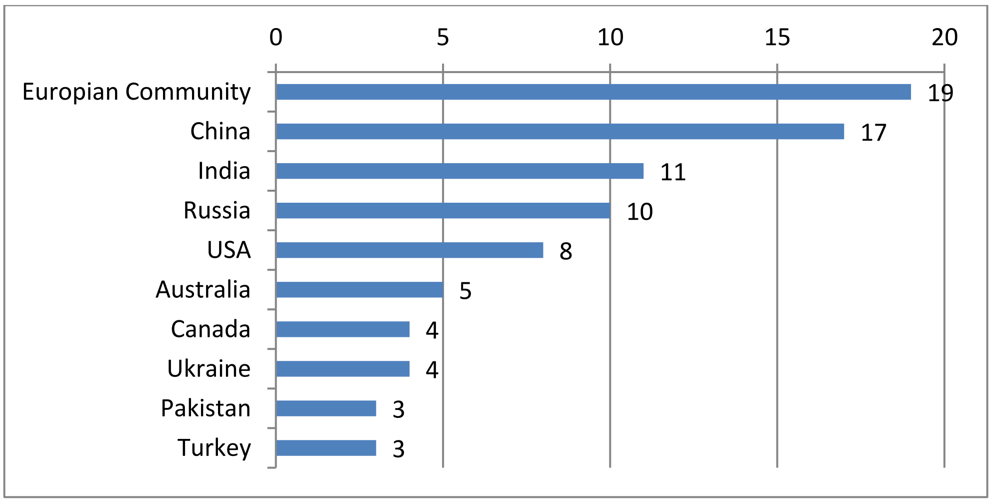


Figure 3-3: Percentages of top 10 wheat producer in the world in 2016-17

In order to compensate the need, Turkey imports more than 4 million tons of wheat each year from other countries. From the efficiency point (tons/hectare) of view also improvements are required and possible. Turkey's wheat production efficiency is about 2.68 tons/hectare (Figure 3-4).

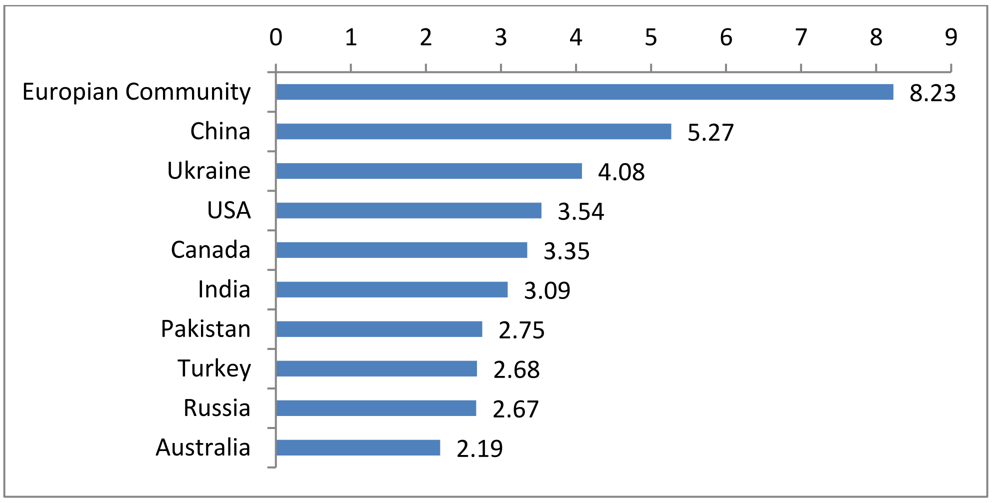


Figure 3-4: Efficiency in wheat production [tons/hectare] in 2016-17

- Inefficient usage of soil

Turkey has 2.2 million farmers and 148 million decares arable farms but, 17% of arable farms are fallow lands (Turkish Ministry of Agriculture, 2018). One of the reasons for this is due to lack of insight and support regarding decision making on crop production, soil fertilization, and pesticide management.

- Increase in cost of farming inputs

In the last years, the cost of fertilizers, fuel-oil and chemicals of pest prescriptions have dramatically increased. On the other hand, the usage of these resources has not been effectively used and monitored. This has inversely affected farming and greenhouse production profit. As a result of this, the number of farmers in these domains have decreased and herewith the overall production has lowered. To solve these problems, it is required to better monitor and manage the input resources, and likewise decrease the overall costs.

These problems can be to some extent tackled by focusing on an improved business and logistics process, by applying total quality and precision farming principles, and appropriate sensing and effector technologies. Yet, these solutions remain limited compared to the adoption of IoT that provides a further substantial optimization by enabling the integration of various technologies such as (wireless) sensor networks, mobile computing, cloud network, data analytics, and decision support systems.

To cope with these problems a strategic decision is to focus on IoT (Dlodlo & Kalezhi, 2015; Ma, Zhou, Li, & Li, 2011). IoT enables the usage of sensors to measure the required parameters (e.g. soil quality), support the decision-making process using services such as data analytics, and use actuators to execute the proper action at the right time and right place. This is for example the case for the wheat and tomato production that has been described in the previous sections. With the introduction of IoT for both cases several benefits are envisioned. Firstly, determining the variability in yield potential might allow optimizing production at each site. With the help of precision farming practices such as nutrient management and soil management quality of the soil can be improved. Also, pest management allows mapping pest populations and obtaining better prescriptions reducing pesticide usage and causing minimal environmental impact. Managing farming practices and obtaining profit maps help reducing the risk in agriculture. Better irrigation, fertilization practices, and pest management strategies save resources to be used. Crop production problems can be solved precisely and in less time with precision farming. Further, long term data can be collected and analyzed over time, providing better strategic management decisions. Saving input materials and resources enables reducing the work needed and cost. Finally, reducing cost and improving quality will increase profits obtained. So far with the existing FMIS these goals could not be fully realized or are only achieved to a limited extent. Although it is envisioned that IoT will be worthwhile to realize the above goals it is not easy to develop an IoT-based FMIS system.

In the literature, various different reference architectures have been proposed for both IoT and FMIS. Recently, the two concepts are further integrated leading to an IoT-based FMIS architecture. Unfortunately, deriving a concrete application architecture for the specific farming situation is far from trivial. This is because the existing architectures are usually represented as reference architectures that are too abstract and do not consider further details that are required to derive the application architecture. For deriving the concrete architectures for a particular context, the different features of FMIS and IoT should be selected. This includes for example the different management functionality, the security protocols, the device communication protocols, and the cloud services. For each of these many different selections can be made and the combination of these leads to a broad design space.

Obviously, given a description for the precision farming system we can identify many different architecture design alternatives. Since the architecture has a direct systemic impact on the overall IoT-based precision farming it is important to derive the proper architecture to meet the overall precision farming requirements of the various stakeholders. For guiding the architect in deriving the customized concrete architecture a systematic approach is necessary

3.4 FMIS Development Method

In Figure 3-5, we show the proposed development approach for deriving an IoT-based FMIS application architecture. The approach consists of two basic activities including *Domain Engineering* and *FMIS Development*. In essence, the approach is based on the product line engineering process as described in the literature (Apel et al., 2013; Capilla, Bosch, Trinidad, Ruiz-Cortés, & Hinchey, 2014; Tüzün, Tekinerdogan, Kalender, & Bilgen, 2015).

The Domain Engineering activity focuses on developing and preparing the artefacts for developing application FMIS. Hereby, the first step includes the development of an IoT FMIS family feature model that defines the common and variant features of the different FMISs. The subsequent step focuses on developing the reference architecture for IoT-based FMIS.

The final step in the domain engineering activity aims to develop the reusable components that will be necessary to develop the FMIS based on the reference architecture. The following sections elaborate on the development of the family feature model and present the corresponding reference architecture.

The FMIS development activity focuses on developing a particular IoT-based FMIS. Hereby, the FMIS will be developed based on reuse of the artefacts in the domain engineering activity. The first step in the application engineering includes the selection of the features of the application. Further, the features will include both features for IoT and FMIS. These will be usually different for different FMISs in different contexts. Based on the selected features the specific FMIS application architecture will be developed using the reference architecture of the domain engineering activity. The final step includes the implementation of the FMIS. Hereby, the earlier developed components in the domain engineering activity will be reused. Very often an FMIS simulation system can be developed to validate the system before deciding on the large-scale investment. In the following sections we will elaborate on the activities of Figure 3-5.

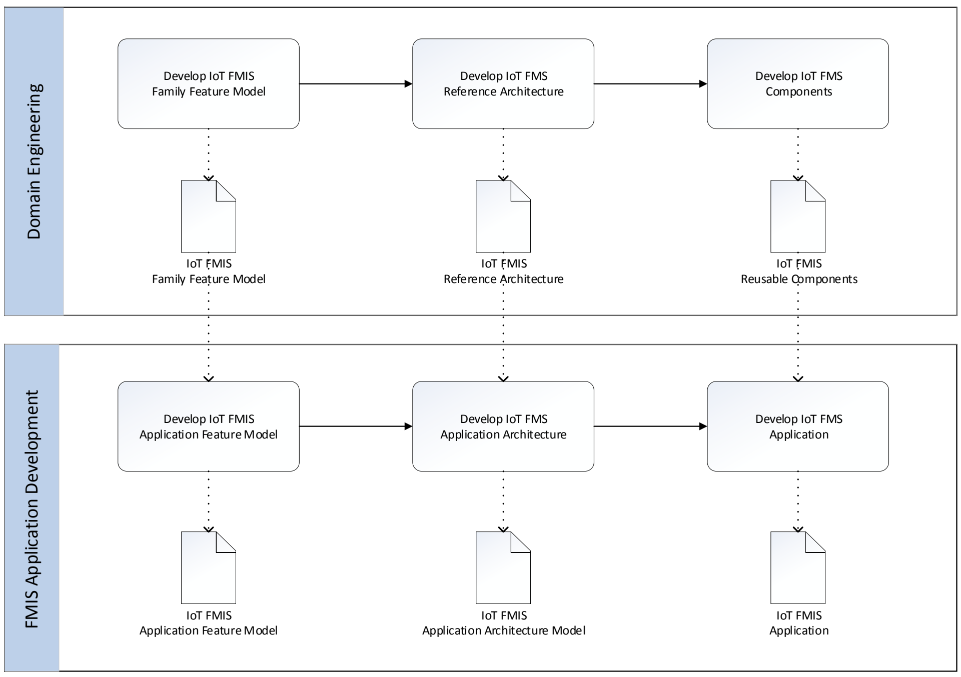

Domain Engineering
Develop IoT FMIS Family Feature Model
Develop IoT FMS Reference Architecture
Develop IoT FMS Components
IoT FMIS Family Feature Model
IoT FMIS Reference Architecture
IoT FMIS Reusable Components
FMIS Application Development
Develop IoT FMIS Application Feature Model
Develop IoT FMS Application Architecture
Develop IoT FMS Application
IoT FMIS Application Feature Model
IoT FMIS Application Architecture Model
IoT FMIS Application

Figure 3-5: FMIS development approach

3.5 FAMILY FEATURE MODEL

The first step of the domain engineering activity of the proposed approach in Figure 3-5 is the development of a family feature diagram for IoT-based FMIS. A feature diagram is a tree that is in particular used to model the commonality and variability of a specific domain or system. The feature diagram includes a root node representing the domain or system that includes features representing the essential characteristics or externally visible properties of the system (Tekinerdogan, Sozer, & Aksit, 2012). Features may have sub-features which can lead to a hierarchical tree. Features can be mandatory or variant. Variant features are usually represented as optional or alternative features. Optional features can be selected or not, whereby alternative features require the selection of one of the defined features.

A feature configuration is a set of features which describes a member of a communication protocol. A feature constraint further restricts the possible selections of features to define configurations. The overall legend (abstract syntax) for the problem feature diagrams is given in Figure 2-3.

For deriving the feature models, we have carried out a domain analysis process in which we selected and studied relevant set of studies that explicitly deal with IoT, Precision Farming and Farm Management Information Systems, respectively.

The domain analysis consists of two basic activities including domain scoping and domain modeling. In the scoping process, we define the scope of the domain analysis process and select the set of knowledge sources. In the domain modeling process the feature diagram is provided.

During the domain scoping process, we have looked at not only scientific papers but also considered websites and white papers of the important vendors and stakeholders in the IoT and precision farming domains. The selected list of important sources that we have considered for IoT are shown in Appendix C, the list of sources for precision farming are shown in Appendix D. We do not claim that the set of sources is comprehensive but an analysis of these selected studies shows a convergence and agreement on the concepts. In the following we first describe the feature diagram for IoT in section 3.5.1 and then the feature diagram for precision farming in section 3.5.2.

3.5.1 Feature Model for IoT

Based on the primary studies given in Appendix C – Primary Studies for Deriving Characteristics of IoT, we have obtained the top-level feature diagram of IoT as given in Figure 3-6.

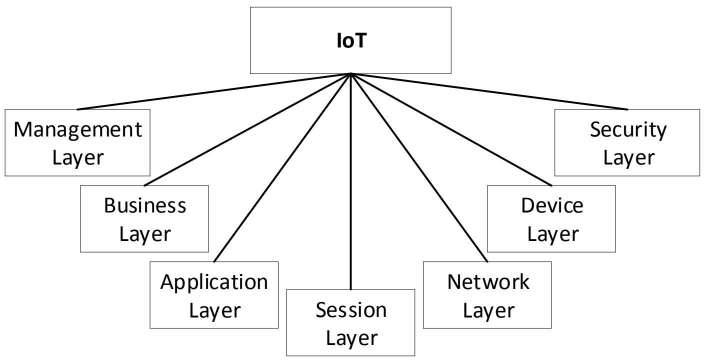


Figure 3-6: Top-level feature diagram of the IoT

In essence, the top-level figure diagram presents the design features and as such includes the mandatory features for the layers of the earlier defined IoT reference architecture in Figure 3-1. The feature diagram states that all the layers are mandatory for setting up an IoT system.

For each of the layers we can derive a detailed feature diagram that represents the commonality and variability for the corresponding layer. Among these IoT layers it appears that the decisive layer is the session layer that includes the protocols for initiating the connection and the further communication session. Figure 2-5 shows the feature diagram that we derived from the domain analysis to the IoT session layer communication protocols.

The top-level mandatory features in the feature diagram are protocol type, source-target, transport type and architecture. The protocol type feature defines the protocols that we could identify from the selected primary studies. These identified protocols are the following:

- Message Queuing Telemetry Transport (MQTT): One of the most popular protocols to collect device data and communicate with servers (OASIS, 2014).
- Extensible Messaging and Presence Protocol (XMPP): is based on exchanges of XML messages in real time that is defined to connect devices to servers (IETF, 2011).
- Advanced Message Queuing Protocol (AMQP): A queuing system designed to connect servers to each other (OASIS, 2011).
- Data Distribution Service (DDS): A fast data bus for integrating devices and systems (OMG, 2015b).
- The Constrained Application Protocol (CoAP): A specialized web-based protocol to be used in constrained nodes and constrained networks (IETF, 2013).

As given in Figure 2-5, there are three types of source-target relations available in session layer protocols: Device-to-Device (D2D), Device-to-Server (D2S), and Server-to-Server (S2S). In some studies, these features are also called Machine-to-Machine (M2M), Machine-to-Cloud (M2C), and Cloud-to-Cloud (C2C) respectively.

Session layer protocols are closely related to the transport layer. For all communication protocols, transport layer could be either UDP or TCP. Some protocols like DDS support both UDP and TCP. Addressing scheme (unicast, broadcast, or multicast) might be important depending on the application requirements. The selection of transport layer protocol is important since using TCP and/or UDP changes the characteristics of the communication from performance and security perspectives. If low power devices and networks will be used, TCP is generally less feasible, and likewise the UDP protocol is used instead. On the other hand, TCP is required for supporting security and the common security protocols of (SSL/TLS) are not available using UDP.

The architecture of the session layer protocols can be either publish-subscribe or request-reply. In publish-subscribe architecture, participants send data to a topic on which several subscribers that are registered to this topic might read data. In this architecture publishers and subscribers do not need to know each other, and do not need to be alive at the same time, i.e. this communication type provides time and space uncoupling. This type of communication is well suited for the same data that must flow from one producer to many consumers. On the other hand, request-reply architecture, senders and receivers do need to know each other. Hereby, the requester sends a request message and waits for the response. When the replier receives the request, it responds with a reply message. The session layer protocols of IoT generally use publish-subscribe architecture except in the case of CoAP. CoAP uses a request-reply architecture. There are many criteria to select the right IoT session layer protocol depending on the application requirement. Table 2-3 summarizes the selection of proper IoT session layer protocol. Further information on this issue is provided in (Köksal & Tekinerdogan, 2017a).

Given the requirements for different source-target communication it appears that different protocols are needed. For example, if the application will provide device-to-server communication, MQTT and XMPP might be used. Regarding real-time constraints, only XMPP and DDS seem to be feasible. These protocols also require bus-based architecture. The other protocols do not provide real-time performance and are mainly broker-based.

All the session layer protocols, except CoAP use publish-subscribe communication pattern. CoAP uses request-reply. Interoperability is an important concern for all the communication patterns. The TCP protocol is the most frequent used transport layer protocol, while the UDP protocol is only used in DDS and CoAP.

All the protocols provide QoS parameters. Further, both open source and commercial versions are available for all the protocols, and mobile support is provided.

Security is an important issue for all the protocols and this is handled in different ways. The TLS/SSL protocol is an important protocol for data encryption.

The communication protocols are either message centric or data centric. All the protocols are application based, except CoAP which is web-based.

All the above criteria can be used to select the proper communication protocol given the contextual requirements. Based on the analysis practitioners can select the feasible protocol. In case more than one protocol is feasible, additional functional and non-functional requirements might be considered.

3.5.2 Feature Model for Precision Farming

The previous sub-section has presented the feature diagram for IoT systems. For IoT-based precision farming the other important domain is of course the domain of precision farming itself. Similar to the IoT domain again we have applied a domain analysis process in which we searched for relevant primary studies on precision farming and based on these selected studies we derived a feature diagram to represent the common and variant features. The selected primary studies are listed in Appendix D – Primary Studies for Deriving Characteristics of FMIS. Based on the literature we could identify the following sub-domains for precision farming:

- Global Positioning Systems (GPS),
- Geographical Information Systems (GIS),
- Sensors,
- Variable Rate Technology (VRT),
- Yield Monitoring (YM),
- Yield Mapping (YMAP), and
- Farm Management Information Systems (FMIS).

Based on the above sub-domains, Figure 3-7 shows the top-level feature diagram for the precision farming. In the following subsections, we describe each feature in more detail.

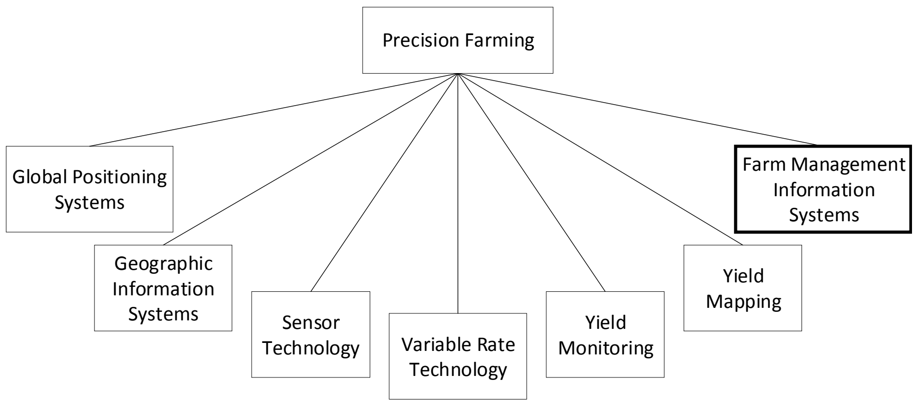

Precision Farming
Global Positioning Systems
Geographic Information Systems
Sensor Technology
Variable Rate Technology
Yield Monitoring
Yield Mapping
Farm Management Information Systems

Figure 3-7: Top-level feature diagram of precision farming

3.5.2.1 Global Positioning System

Global Position System (GPS) is a satellite system that provides location and time information to GPS receivers in real time. Location information consists of latitude, longitude, and, elevation values. In essence, these values are the input of GIS software.

As a tool of precision agriculture, GPS enables receivers to calculate their geo-location. This precise location information supports mapping and analyzing important farming data such as amount of crop and water usage. Further, GPS enables farmers to work during low visibility conditions such as rain, fog, and darkness.

3.5.2.2 Geographic Information System

Geographic Information System (GIS) software is used to input, store, analyze, and display geographical information of the field. GIS enables detailed analysis of farming data in several map like forms: yield maps, soil maps, light maps, etc. Further, GIS is used in many farming applications such as farm planning, tractor guidance, and variable rate applications (Rains & Thomas, 2009).

3.5.2.3 Sensor Technology

In precision farming, different types of sensors are used mainly to measure crop, soil, and weather properties. Sensors can be used by fixing them into the field or they can be used to make measurements while in motion which are called on-the-go measurements. Typical on-the-go measurements are performed to measure yield and soil properties.

In order to communicate with sensors, several standards were defined. ISOBUS is the common specification of the manufacturers on the uniform application of the International standard ISO 11783 Serial control and communications data network. This standard defines an open communication protocol at physical and application layer level and is based on Controller Area Network (CAN) protocol ISO 11898-1. AgroXML (Schmitz, Martini, Kunisch, & Mösinger, 2009) is a popular standardized language for exchanging data in precision farming. AgroXML is based on the eXtensible Markup Language (XML) and is used in communication between FMIS and external systems.

3.5.2.4 Variable Rate Technology

Variable Rate Technology (VRT) consists of variable rate control systems. VRT applications, typically use GPS and GIS software. Differential GPS (DGPS) might also be used in VRT applications as mounted on tractors or other vehicles to provide precise location information of the vehicle. VRT is used to obtain site-specific information in seeding, fertilizer, lime, and pest management applications (Rains & Thomas, 2009).

3.5.2.5 Yield Monitoring

Yield Monitoring is the most direct method to assess the field production and how it should be better managed. A yield monitor measures the crop as it is harvested. Yield monitoring usually measures crop weight, impact forces, and the time an array of light beams are broken. If used, images of the plants can be acquired using unmanned land and air vehicles as well as satellites and direct cameras. This information is important in predicting planting places and times and harvesting times (Rains & Thomas, 2009).

3.5.2.6 Yield Mapping

As the yield is measured, data are stored on a computer along with the GPS coordinates at the point where the yield was measured. Mapping software can then create a yield map. The yield map can show yield variability as well as yield production. Yield variability is illustrated on a map by a change in color, where each color represents a range in. To maintain some level of convention, red is suggested to represent low yields and green high yields. The map legend will tell you how to read what each color indicates. Lack of yield variability would mean that the map shows mostly one color. Yield production can be found by calculating the yield for the entire field (Rains & Thomas, 2009).

3.5.3 Farm Management Information System

Farm Management Information System (FMIS) software is a core part of the precision farming. FMIS is used to collect and process data to manage all farming operations. The top-level feature diagram for FMIS is given in Figure 3-8. The right part of the feature diagram has been derived from the FMIS functions as defined by the survey results in (Fountas et al., 2015). The left part focus on IoT related functionality of FMIS including Collection of Data, Processing Data, Visualization of Data, Communication with External Systems, and System Management. In this chapter we will further focus on the IoT FMIS aspects.

The data acquisition feature defines the gathering of data from sensors and other systems used in precision farming such as tractors, agribots and unmanned vehicles. The data processing feature represent functions for processing the gathered data whereby useful information is extracted using data mining, machine learning, and image processing. The data visualization feature includes displaying processed data in different forms. Apart from classical tables, reports, and monitoring tools, dedicated visualization maps are essential for precision farming applications. Hereby, several maps such as yield maps, soil maps, lighting maps, and profit maps. are displayed for different purposes. The system management feature defines the management of data acquisition, processing, visualization, and external system communication features of FMIS. The quality related functions such as reliability, scalability, extensibility, and security are considered in this feature. Finally, communication with external systems feature defines the communication with external systems such as, a weather forecast system. Each of these features are shown in a different color in the figure to refer these in later sections.

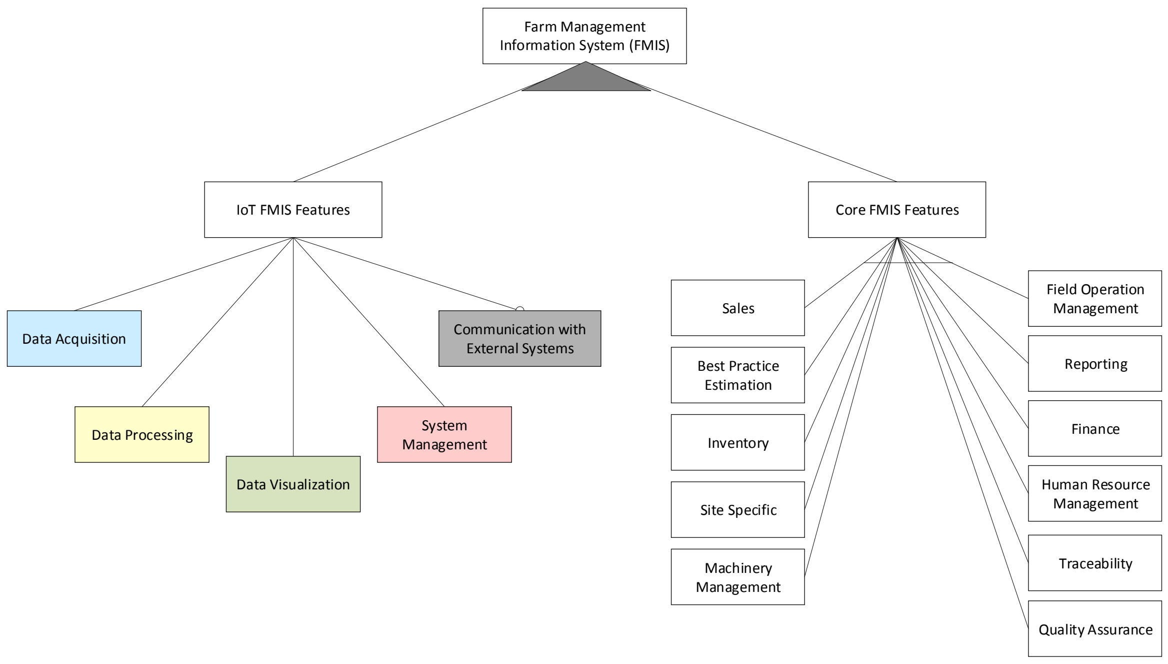


Figure 3-8: Top-level feature diagram of FMIS

3.5.4 Feature Model for IoT-based FMIS

In principle, IoT and FMIS are independent concepts and as such these have been modeled separately in the previous sub-sections. For the selection of the application features we could in principle select the features from the IoT feature diagram as defined in section 3.5.1 and the features from the FMIS feature diagram as explained in section 3.5.1. Alternatively, the notion of IoT-based FMIS can be considered as the integration of both concepts that needs to be separately considered. To this end, Figure 3-9 shows the integrated family feature diagram that we have derived from the feature models for IoT and FMIS. Hereby we focused on FMIS as the dominant decomposition and the integrated IoT features in the separate leaves of the FMIS feature tree. The colors define the features of the feature model in Figure 3-8. The detailed feature diagram given in Figure 3-9 can in principle be further extended with respect to particular project requirements. For the context of this chapter though, the provided feature models are sufficient to illustrate our approach.

In IoT-based FMIS, data acquisition consists of IoT data acquisition feature and conventional data acquisition to support legacy systems. IoT data acquisition contains 5 alternative IoT session layer protocols as discussed in section 3.5.1 namely MQTT, XMPP, AMQP, DDS, and CoAP. Depending on the application one or more protocols for IoT communication can be selected for the application FMIS. In section 3.5.1 we have described the criteria for this. Traditional data acquisition feature consists of ISOBUS and Controller Area Network (CAN) protocols. As we have stated before, other (legacy or non-IoT) protocols can be added to the feature diagram.

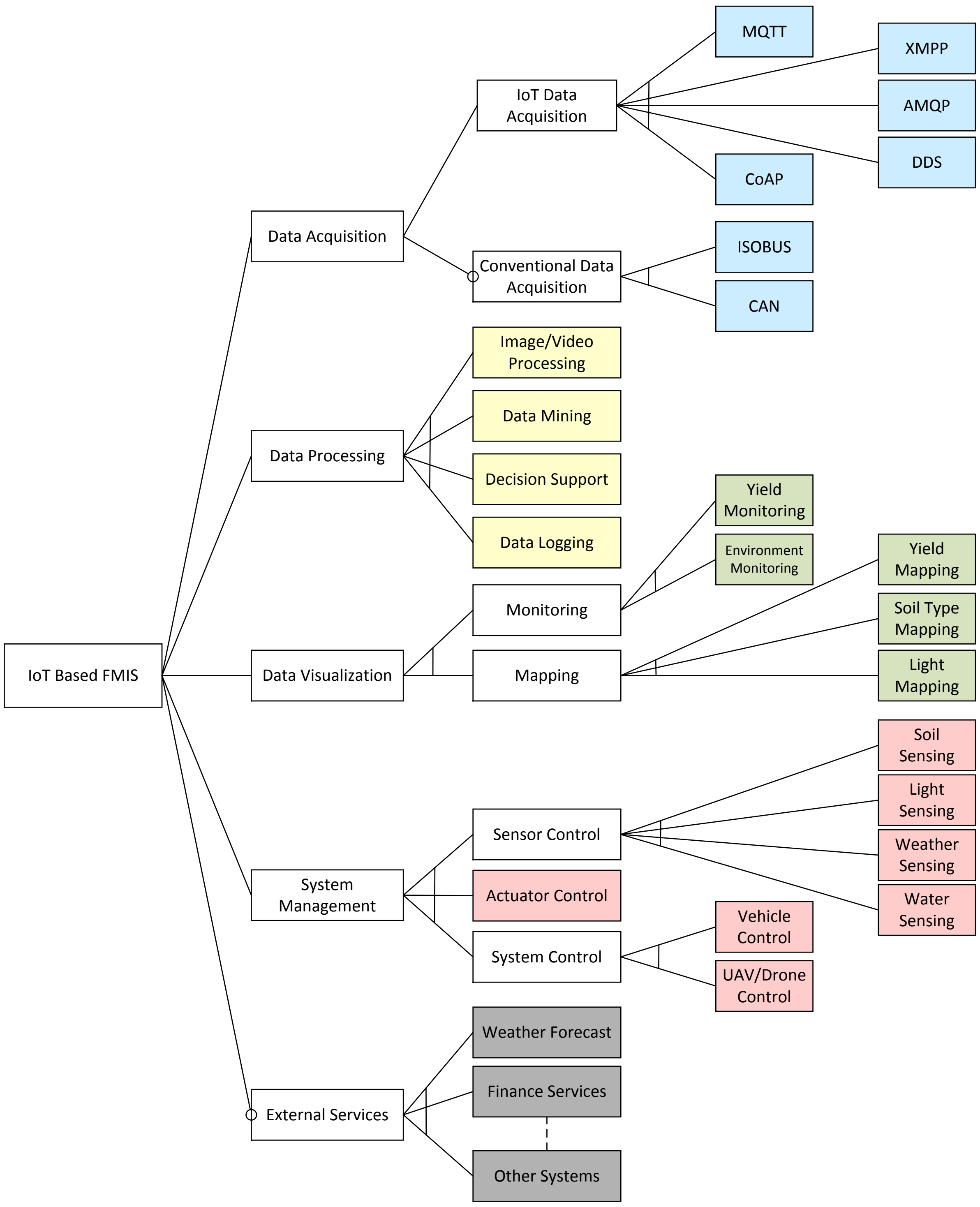


Figure 3-9: Family feature diagram of FMIS

Data processing features mainly depend on the application type and include Image/Video processing, data mining, data logging, and decision-support features. One or more features might be used at the same time. Depending on the application requirements these features can be extended to use different processing features.

Data visualization consists of monitoring and mapping functions. Monitoring consists of environment monitoring and yield monitoring functions. Mapping includes yield, soil-type, and light mapping features.

System management includes sensor control, actuator control, and system control features. Sensor control consists of several sub-features such as soil sensing, light sensing, weather sensing, and water sensing. Also, system control includes vehicle control and UAV/Drone control features.

Finally, external services feature contains externally communicated systems such as weather forecast, finance services, and other external systems.

3.6 Reference Architecture for FMIS

Once we have developed the family feature models for IoT-based FMIS systems, the next step is the development of the reference architecture for the potential systems. In fact, in the literature several reference architectures have already been proposed for FMIS systems (Beck, 2001; Fountas, Wulfsohn, Blackmore, Jacobsen, & Pedersen, 2006; Nikkilä, Seilonen, & Koskinen, 2010; Sørensen et al., 2010, 2011). However, in general these reference architectures are either at a conceptual level and/or do not consider IoT aspects explicitly. Hence, in this section, we will introduce the reference architecture for IoT-based FMIS. For this we will use selected viewpoints of the "Views and Beyond" architecture framework (Clements et al., 2011) including the decomposition viewpoint, Layered Viewpoint and Deployment Viewpoint.

3.6.1 Decomposition View

The Decomposition view is used to show how system responsibilities are partitioned across modules and how these modules are decomposed into sub-modules. Usually, the features in the feature diagram are realized by one or more modules in the decomposition view. The decomposition view of the architecture depicts the overall structure of the architecture which is reasonably decomposed into modular implementation units. It is regarded as a fundamental view of the architecture since it serves as an input for other views (e.g. work allocation view) and helps to communicate and define the structure of the software. The proposed reference decomposition view for the IoT-based FMIS system is given in Figure 3-10.

The modules in the decomposition view are colored to make the link with the earlier defined features in the feature diagrams of FMIS. In essence, the decomposition view includes the modules for data acquisition, data processing, data visualization, system management, and modules for communication with external services. The decomposition view includes all the possible modules for the various IoT-based FMIS applications. Note that, in this case, for each feature in the earlier diagram one module has been defined in the decomposition view. Further we have not depicted the lower level functionalities such as node discovery and directory and name services. In section 3.3.7, we will explain the configuration of a specific decomposition view from this reference decomposition view.

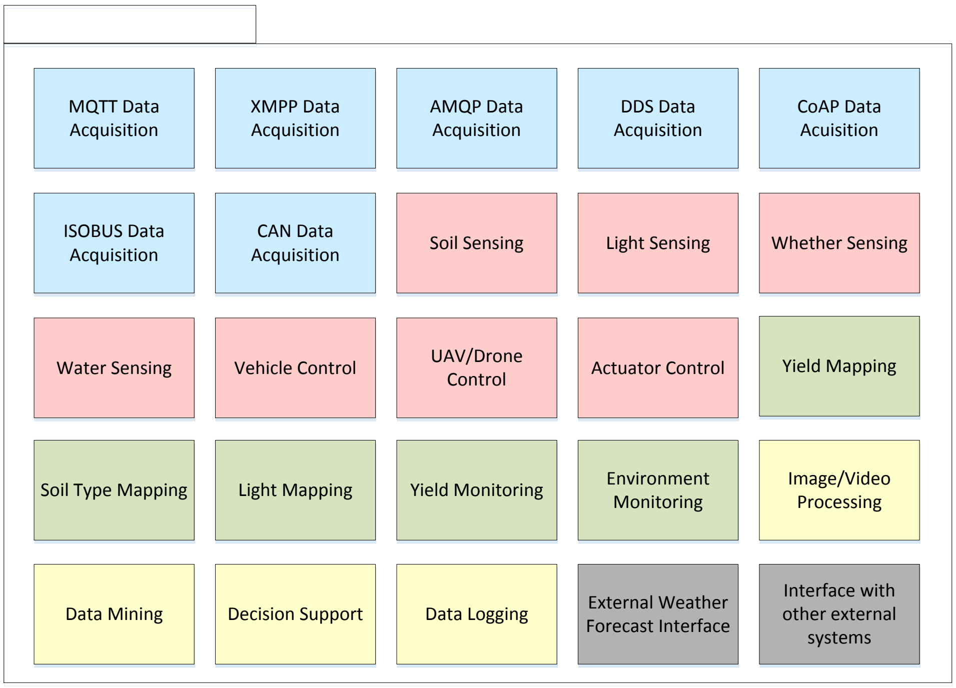


Figure 3-10: IoT-based FMIS – decomposition view

3.6.2 Layered View

The Layered view is similar to decomposition view since it reflects the division of software into units. The difference is that in a layered view, modules are structured into layers, which interact based on a strict ordering relation. This means that if layer A is allowed to use layer B, layer A's implementation can use any public facilities of Layer B. However, layer B cannot use any facilities of layer A.

Figure 3-11 shows the layered view for the IoT-based FMIS system. Here the dominant decomposition is taken from the IoT layered view as it was given in Figure 3-1, and likewise it includes the same layers of the IoT reference architecture. The specific details are primarily in the higher-level layers including the business layer, the application layer and the data acquisition layer. The FMIS business layer includes all required farm management operations logic and workflows such as Fertility Management, Nutrient Management, Pest Management, Weed Management, and Irrigation Management. The FMIS application layer realizes the functionalities for FMIS Data Processing, Data Visualization, System Management, and Communication with External Systems. The FMIS Data Acquisition Layer is for data adaptation between IoT session layer and FMIS, i.e. this layer provides the connection with the session layer protocols of the IoT. This layer includes IoT session layer interfaces such as MQTT interface, XMPP interface, and CoAP interface. To support non-IoT systems, the module conventional interface also takes place in this layer.

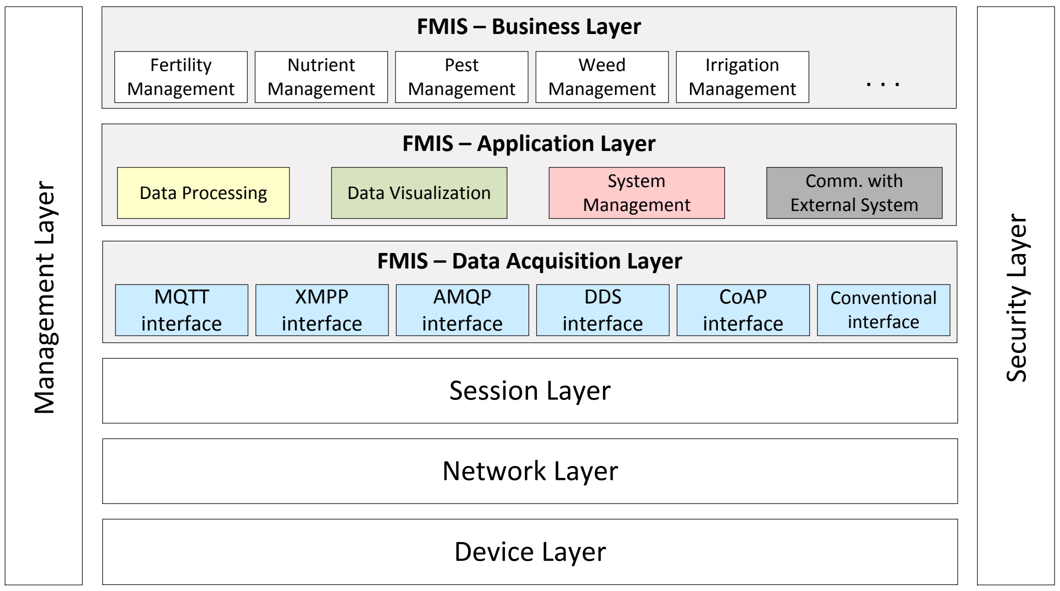


Figure 3-11: IoT-based FMIS – layered view

3.6.3 Deployment View

The earlier defined views (decomposition and layered) focus on modeling the software modules of the IoT-based FMIS. The deployment view elaborates on these views and is used to show the allocation of the identified software modules to the hardware of a computing platform. The deployment view of IoT-based FMIS is given in Figure 3-12. Hereby, data processing module is deployed on the Central Cloud Server and Client (Farmer) nodes. The other nodes are dedicated to sensors, actuators, and cameras. The main sensors in the UAV/Drone and Satellite are cameras. Vehicles can have their own sensors, actuators, and cameras. Hence, these items can be assigned to different nodes.

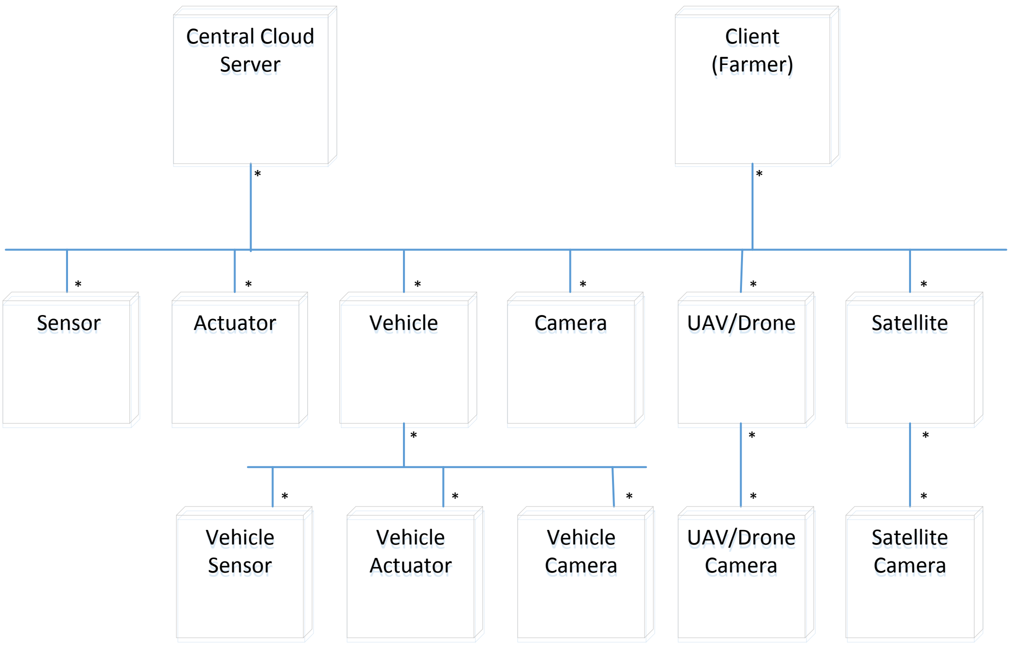


Figure 3-12: IoT-based FMIS – deployment view

3.7 CASE STUDY EVALUATION

3.7.1 Case Study Protocol

Our primary objective is to evaluate the impact of the developed architecture design method for IoT-based FMIS. To evaluate the defined research questions, we have applied the case study research protocol as defined by Runeson and Höst (Runeson & Höst, 2008). Based on this we have followed the five steps: (1) case study design (2) preparation for data collection (3) execution with data collection on the studied case (4) analysis of collected data (5) reporting.

Table 3-1 shows the case study design elements. We have applied the case study research both for a retrospective case and prospective case. The retrospective case included a system which was developed before and for which there was already an existing architecture and the required design documents. The prospective case includes the system that is planned to be developed.

Table 3-1: Case study design

Case Study Design Activity	**Retrospective Case Study (Wheat Production)**	**Prospective Case Study (Tomato Production)**
Goal	Comparing and assessing the method feasibility and recommended application architecture	Assessing the effectiveness of the method Assessing the practicality of the method
Research Questions	RQ1. To which extent is the derived application architecture in alignment with the decision of the case study?	RQ1. To which extent does the method support the architecture design of the IoT-based FMIS? RQ2. How practical is the method for deriving the IoT-based FMIS application architecture?
Background and Source	- Official requirements documents - Official architecture design documents - Project Manager and System architects	- Official requirements documents - Project Managers and System architects
Data Collection	Indirect data collection based on document analysis (the design documents and technical reports)	Indirect data collection and direct data collection through semi-structured interviews (mix of open and closed questions)
Data Analysis	Qualitative Data Analysis using Radar Charts	Qualitative Data Analysis using Radar Charts

The goal for the retrospective study was to compare the earlier results with the results that are produced by the proposed method. In this way, it was aimed to evaluate the effectiveness of the method. For the prospective case study, we aimed to evaluate both the effectiveness and the practicality of the approach. The research questions were defined accordingly as it is shown in the Table 3-2.

Table 3-2: Questionnaire for the interview

	Questions
Q1	With information at hand, are you planning to increase the adoption of IoT in the future?
Q2	Do you think that this reuse-based architecture design method is more effective than the architecture design method that you adopted so far?
Q3	Do you think that the provided recommended application architecture is of high quality?
Q4	Do you think that the reference architecture is of high quality?
Q5	Is the method and the reference architecture sufficient to derive the application architecture?
Q6	Do you think that the method is practical?
Q7	Will you use the method again?
Q8	Do you think that the application of the method can provide a competitive advantage to the organization?
Q9	Has the usage of the method enhanced your knowledge on IoT-based FMIS?
Q10	Do you have any suggestions for improving the method?
Q11	Do you have any suggestions for improving the family feature models?
Q12	Do you have any suggestions for improving the reference architecture?

For the adopted background and sources in the case study research we have used official design documents and interviewed project managers and system architects. For the retrospective case study, the requirements and design documents were available, whereas for the prospective case study only the requirements document was available. For both case studies we had contact with and interviewed the project managers and system architects. The project manager had more than 20 years of experience in farm management system. The two system architects had more than 15 years experienced in designing farm management systems.

For both case studies we use a qualitative data analysis approach using radar charts. For the retrospective case study, we used indirect data analysis by analyzing the requirements documents, applying the method and comparing the results of the method with the existing architecture. For the prospective case study, we used a direct and indirect data analysis approach. For the direct data analysis, we conducted semi-structured interviews, in which a list of predefined set of questions was asked to the project manager and software architects. The predefined questions are listed in Table 3-2. The questions included a 5-point Likert scale (strongly disagree to strongly agree) for the possible answers. Besides of this a further explanation was asked for each question.

The interview was organized as follows:

1. First a meeting was scheduled with the project manager and system architects for the initial interview. The goal of this interview was to capture the initial thoughts and experience on IoT adoption.
2. In the second step we gave a short presentation about the goal of the developed method. Also, we shortly explained the operation of the method as well as the final outcome.
3. In the third step we applied the method both for the retrospective case (section 3.7.2.1) and prospective case (section 3.7.2.2).
4. In the fourth step, the researchers analyzed the architecture design that resulted from the application of the method to the retrospective case and the prospective case.

5. In the fifth step, the researchers held a post interview with the subjects with the purpose of identifying the impact of the method and its practicality.
6. In the sixth step, the researchers collectively assessed data from the initial interview, report delivered by the method, and the post interview. The assessment was carried out separately and later was discussed together to derive the lessons learned.

In the following subsection 3.7.2 we first discuss the results of the above process after the first two steps. Subsequently in section 3.8 we discuss the evaluation in step 4, 5 and 6.

3.7.2 FMIS Architecture Design

In this section we describe the application of the approach to the retrospective cases study (section 3.7.2.1) and prospective case study (section 3.7.2.2). As stated before, the application architecture is derived from the family feature model and the reference architecture. As described in section 3.4 and the approach in Figure 3-5 the FMIS development method includes the development of the application feature model and the application architecture.

3.7.2.1 Retrospective Case Study: IoT-based Wheat Production

Figure 3-13 shows the feature model for smart wheat production that is derived after the analysis of the existing case study.

This application feature model is obtained by reusing the feature model for FMIS given in Figure 3-9 and selecting the features that are needed for this case study. As shown in the figure, for this case study, MQTT session layer protocol of IoT is chosen. The main reason for this was because open source implementations of MQTT could be used and MQTT supports TCP and D2S communications which were considered necessary in the given context. Likewise, MQTT feature of the IoT Data Acquisition will be used. Also, in order to support conventional data acquisition with tractors used in wheat production ISOBUS and CAN communications shall be supported. Almost all data processing and data visualization features are required for smart wheat production. For this retrospective case study, we will integrate our FMIS with the external weather forecast system only.

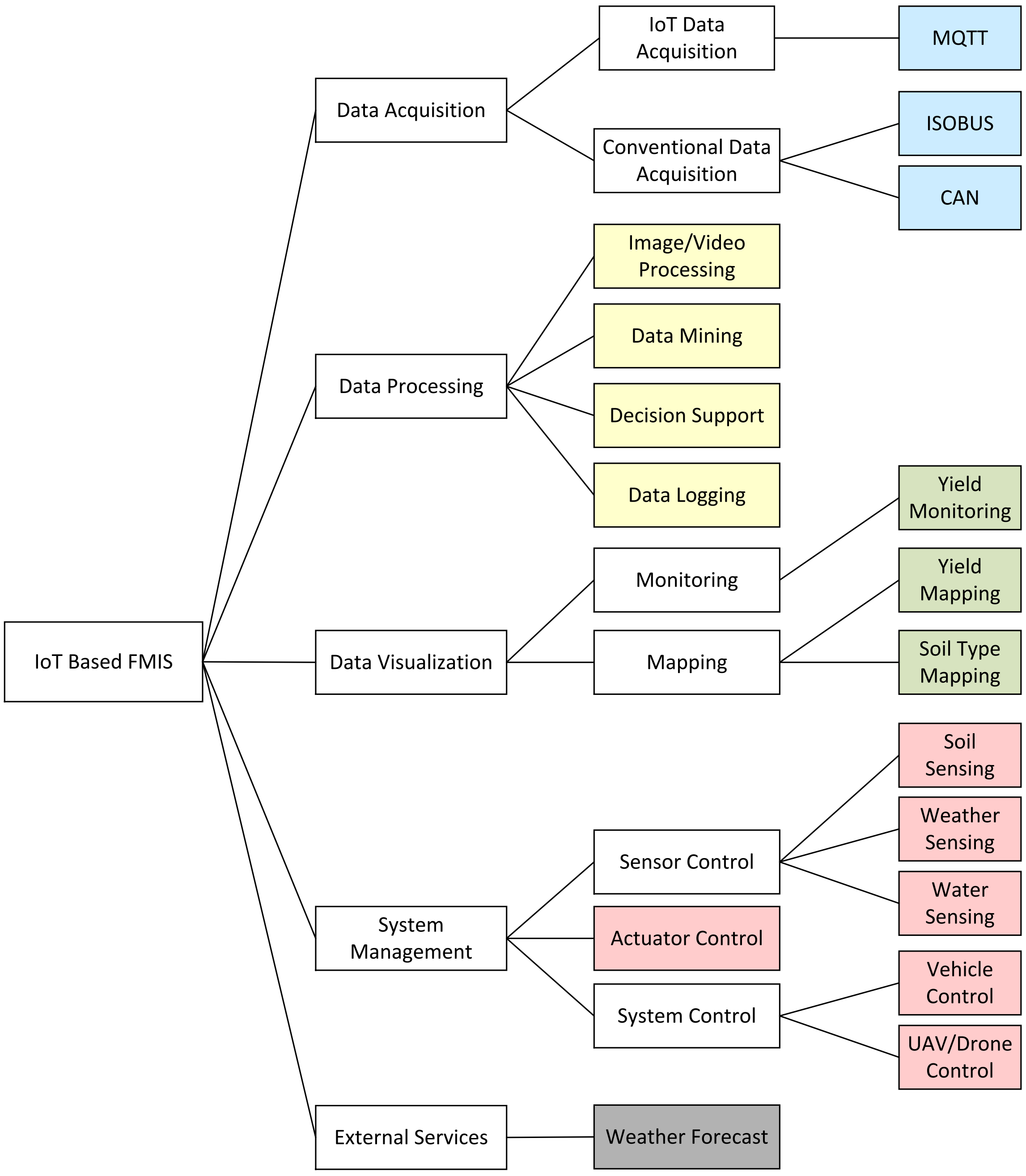


Figure 3-13: Application feature diagram of FMIS for IoT-based smart wheat production – Retrospective case study

Decomposition View

Based on the selected features as defined in Figure 3-13, the application architecture can now be derived. As we have discussed before, the architecture of a system is usually described using multiple different architecture views. For each of the required architectural view it is indeed necessary to develop the application architecture view. Figure 3-14 shows the decomposition view of the Smart Wheat Production that is obtained using the reference decomposition view given in Figure 3-10. As explained above, MQTT, ISOBUS, and CAN data acquisition modules will be used to support IoT communications. All the sub-features of the system management feature of the family feature model will be used except light sensing.

Light sensing feature is used to obtain light maps in the greenhouses. Yield monitoring, yield mapping, and soil type mapping modules will be used to implement data visualization features. All data processing modules namely image/video processing, data mining, decision support, and data logging modules will be implemented. Finally, a single external communication interface: external weather forecast interface module will be implemented.

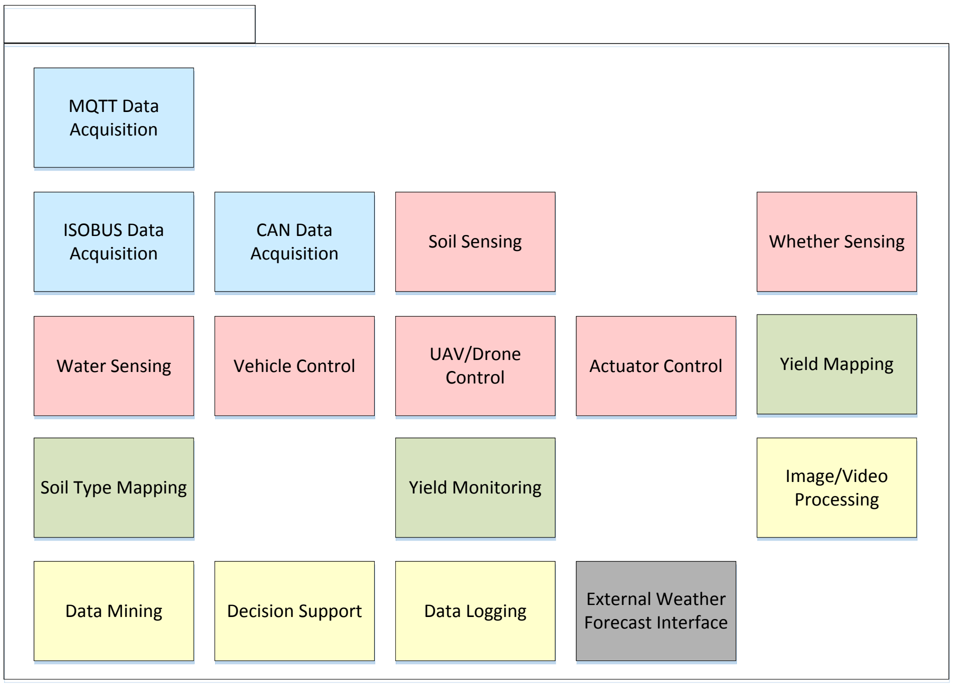


Figure 3-14: IoT-based FMIS – Decomposition view for IoT-based smart wheat production – Retrospective case study

Layered View

Figure 3-15 shows the layered view of smart wheat production. Similar to the other views, this view is also customized from the reference layered view diagram given in Figure 3-11. Here the modules of the decomposition view are distributed over the layers in the layered view. The modules MQTT Interface and Conventional Interface are allocated to the FMIS-Data Acquisition Layer. The FMIS-Application layer includes the modules Data Processing, Data Visualization, System Management and Communication with External System. The FMIS-Business Layer includes Fertility Management, Nutrient Management, Pest Management, Weed Management, and Irrigation Management. We assume that the other layers and the modules in these layers are similar as defined in the reference architecture.

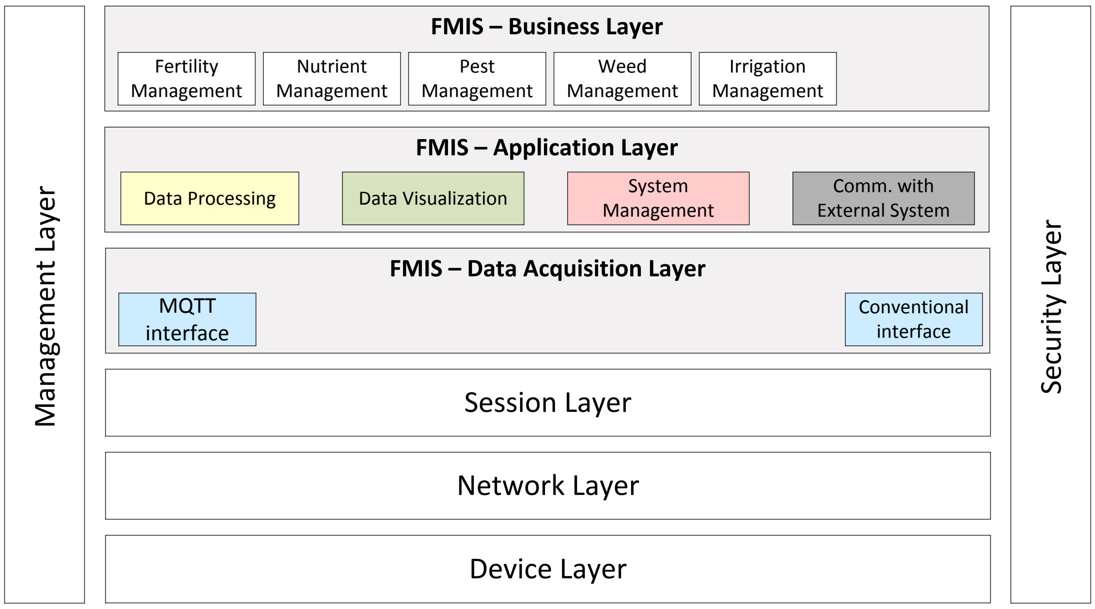


Figure 3-15: IoT-based FMIS – Layered view for IoT-based smart wheat production – Retrospective case study

Deployment View

The deployment view of the smart wheat production case study is given in Figure 3-16.

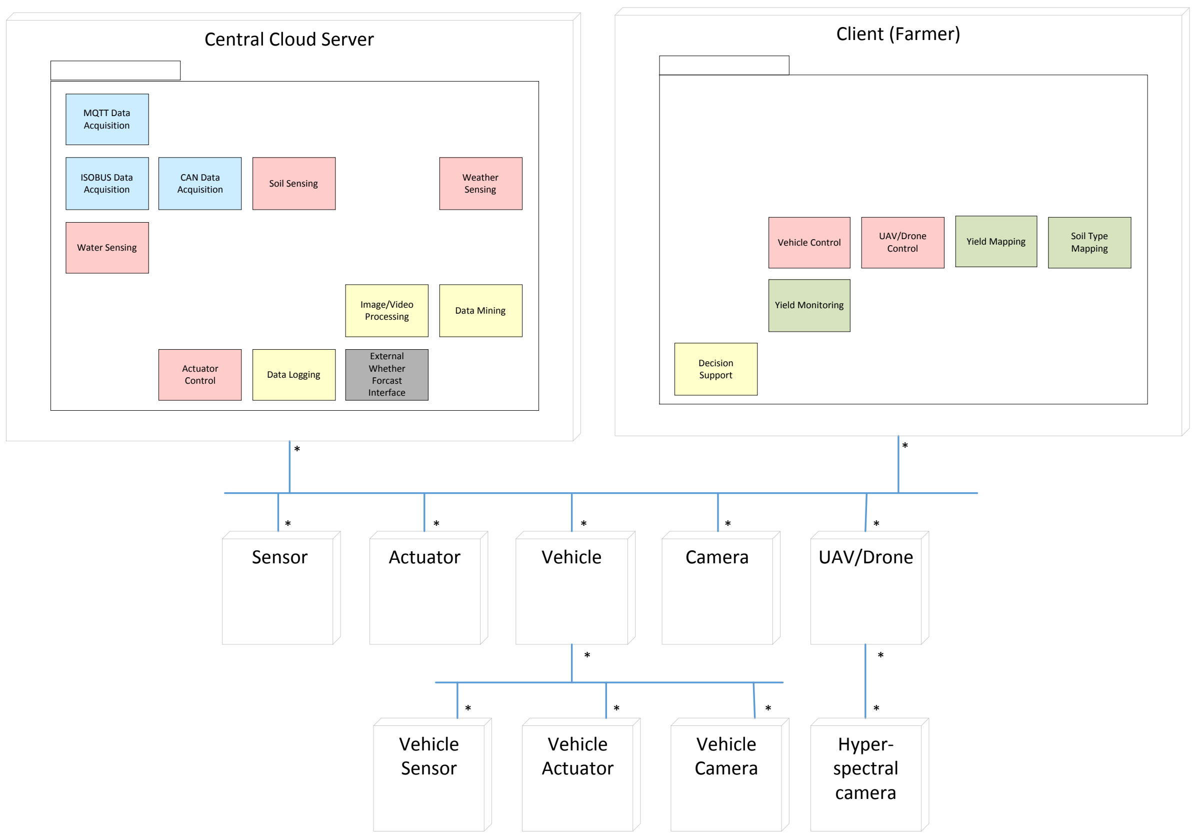


Figure 3-16: IoT-based FMIS – Deployment view of IoT-based smart wheat production – Retrospective case study

The required software modules given in the decomposition view are deployed to a central cloud server and client (farmer). Since there is no satellite to be used it is omitted for this case. A subset of the features from the ground sensors actuators, and cameras, on-the-go sensors, actuators will be used. These will be deployed on vehicle, tractor or UAV/Drone nodes.

3.7.2.2 Prospective Case Study: IoT-based Tomato Production in Greenhouses

In this section, we will show the application of our approach to the prospective case study: IoT-based smart tomato production in greenhouses.

Application Feature Model

Figure 3-17 shows the feature model for IoT-based smart tomato production in greenhouses. This feature model is obtained again by reusing the feature model for FMIS given in Figure 3-9 and selecting the features for smart tomato production in greenhouses.

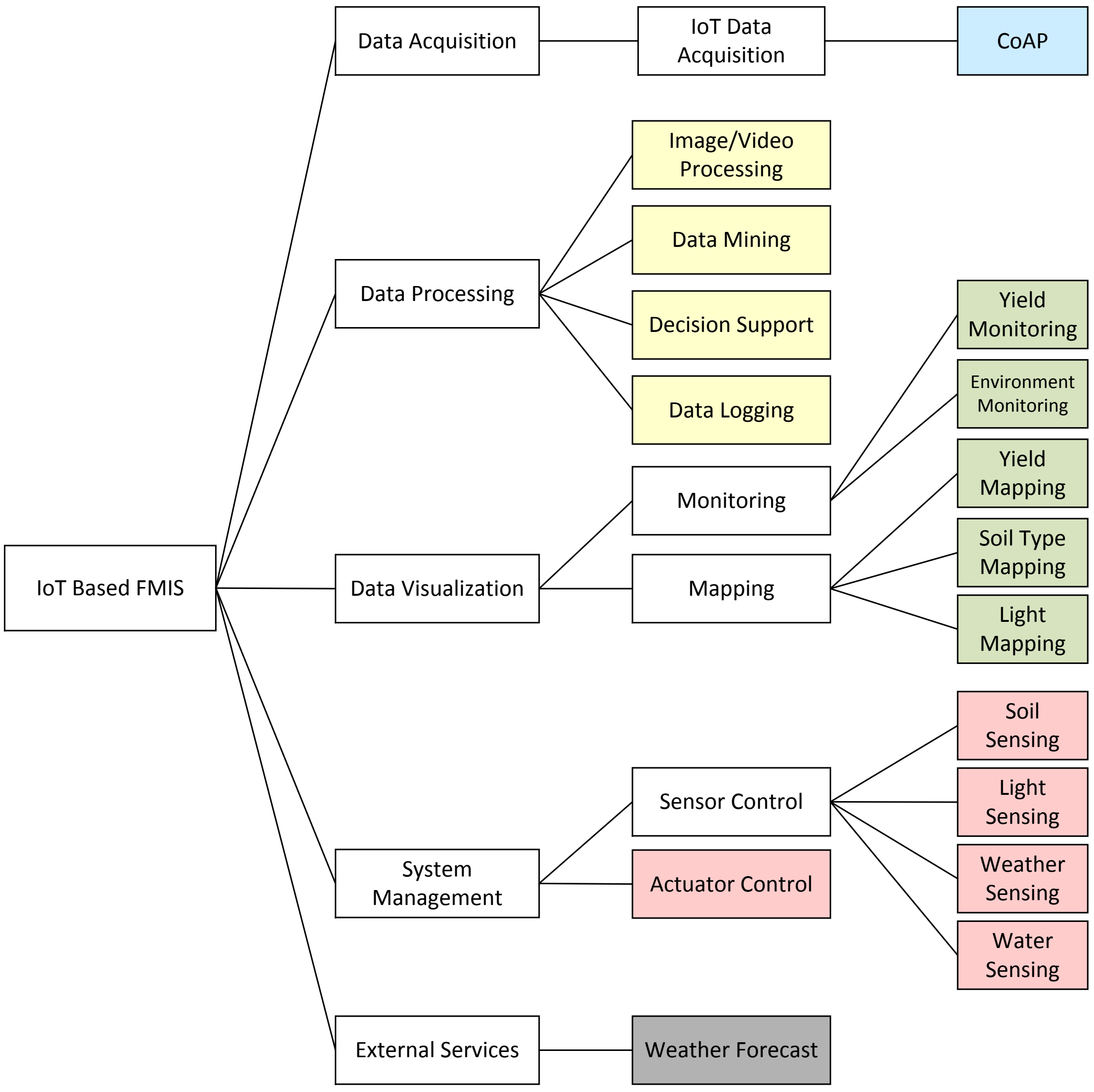


Figure 3-17: Application feature diagram of FMIS for IoT-based smart tomato production in greenhouse – Retrospective case study

For this case study, CoAP session layer protocol of IoT is chosen and CoAP feature of the IoT Data Acquisition will be used. Since there will be no need to support conventional data acquisition, ISOBUS and CAN communications will not be supported. All data processing and data visualization features will be implemented for IoT-based smart tomato production in greenhouses. As in the previous case, we will integrate our FMIS with just the external weather forecast system only.

Decomposition View

Figure 3-18 shows the decomposition view of our second case study. This view is also obtained using the reference decomposition view given in Figure 3-10. Here, for data acquisition only CoAP is selected. System management functionalities except vehicle and UAV/drone control will be supported. All data processing and data visualization modules will be implemented. Similar to the first case study, a single external communication interface: external weather forecast interface module will be implemented as well.

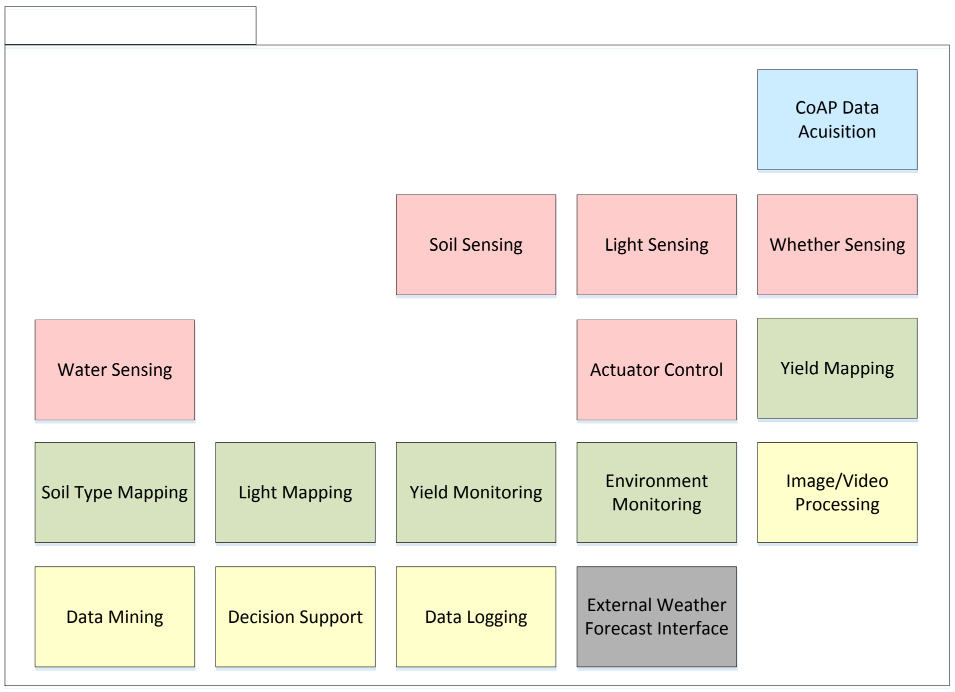


Figure 3-18: IoT-based FMIS – Decomposition view for IoT-based smart tomato production in greenhouse – Prospective case study

Layered View

Figure 3-19 shows the layered view of IoT-based smart tomato production in greenhouses that is customized from the reference layered view diagram which is given in Figure 3-11. In this case, only CoAP interface exists in the Data Acquisition Layer of FMIS Application Layer. As in the first case study, all required precision farming applications are required in the Precision Farming Application Layer.

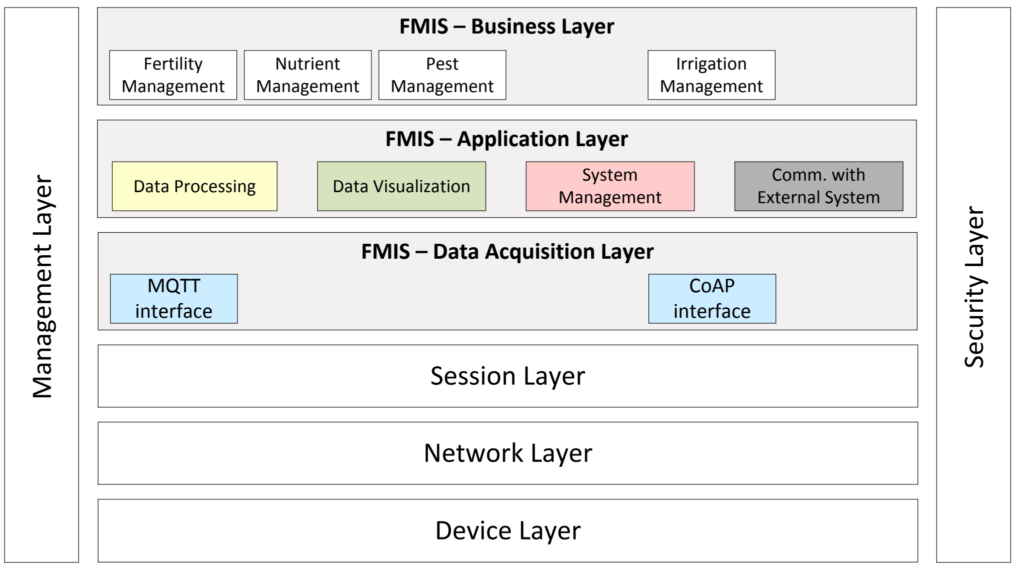


Figure 3-19: IoT-based FMIS – Layered view for IoT-based smart tomato production in greenhouse – Retrospective case study

Deployment View

Deployment view of the IoT-based smart tomato production in greenhouses is given in Figure 3-20. The required software modules that are given in the decomposition view in Figure 3-18 are deployed to central cloud server and client (farmer). Since there is no satellite, vehicle, and UAV/drone to be used in this case they are all excluded in the deployment view. Only fixed sensors, actuators, and cameras will be deployed in this case.

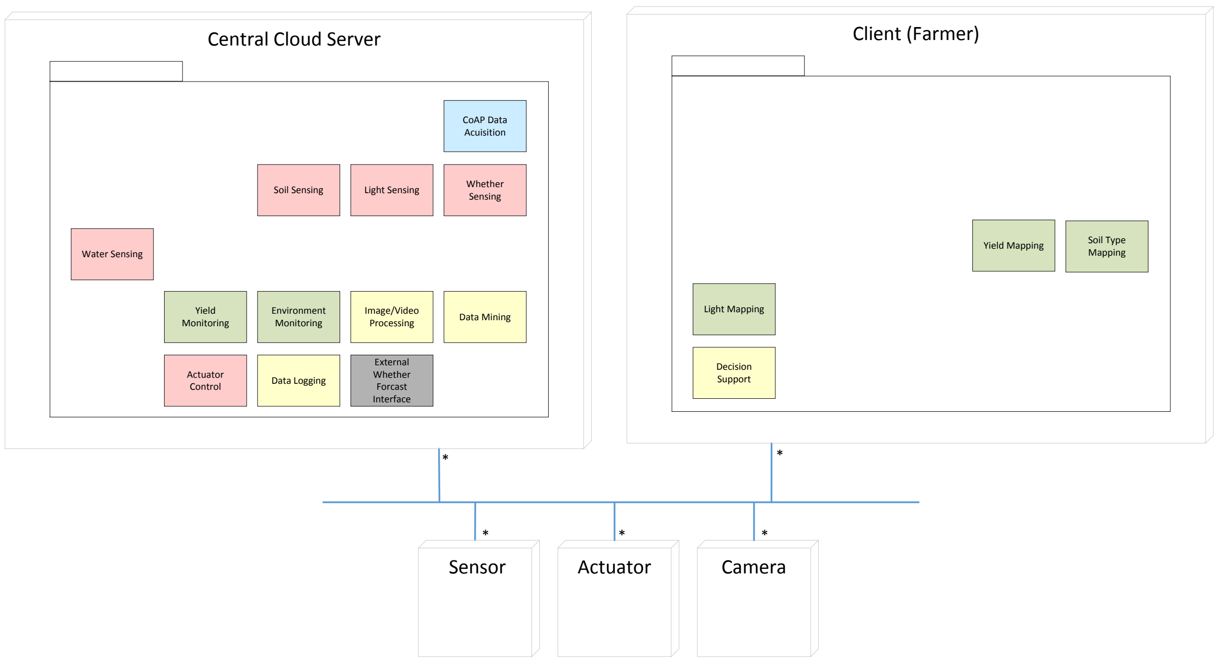


Figure 3-20: IoT-based FMIS – Deployment view of IoT-based smart tomato production in greenhouse – Retrospective case study

3.7.3 Result of the Evaluation for the Retrospective Case Study

In the previous subsection, we have shown the application of the approach for both the retrospective and prospective case studies. As defined in the case study protocol we analyzed the effectiveness and practicality of the approach. The results of the interview are shown as a radar chart in Figure 3-21. In this section we discuss this and the overall evaluation of the retrospective case study.

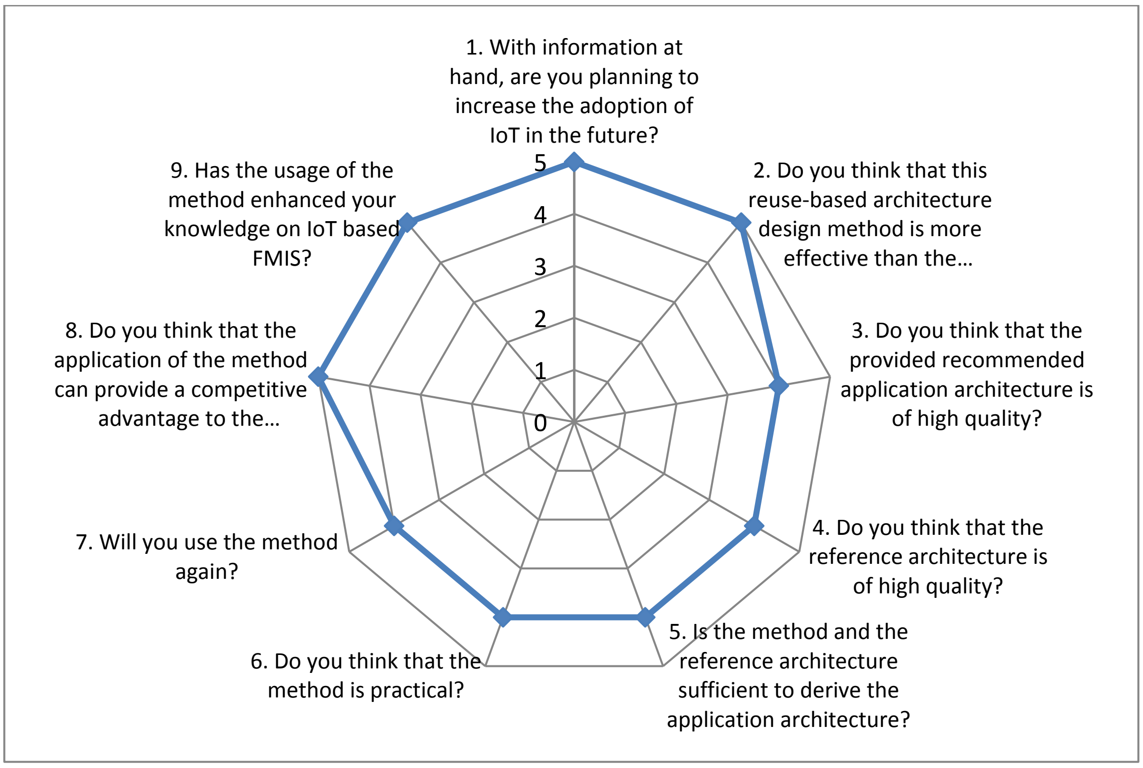


Figure 3-21: Results of the interview presented in radar chart – Retrospective study

Effectiveness of the approach

For assessing the effectiveness of the approach, we analyzed the resulting application architecture and carried out the post-interviews. For this retrospective study the application architecture was before described in different document formats including MS PowerPoint and MS Visio. Further we could access some design documents. It should be stated that the design as such was not as we had properly defined according to a well-known viewpoint approach and not properly designed. Nevertheless, we were able to analyze the existing application architecture and compare it with the application architecture that we derived using our own approach. For the comparison we identified three different relations (1) convergence (2) deviation (3) absence. The convergence relation implies that the similar architecture elements could be identified in both the architecture designs. Deviation implies that the resulting application architecture had additional elements that were not defined in the existing architecture. Finally, absence defines that the resulting application architecture had missing elements that were defined in the existing architecture.

Overall, the result of our analysis showed that the resulting application architecture was quite similar to the existing architecture. In general, the convergence was very high (>90%). We could identify though several deviation and absence relations. With respect to deviation in our resulting application architecture the modules *UAV/Drone Control* and *External Weather Forecast Interface* were not defined in the existing architecture. On the other hand, we could identify also some absence relations. For example, the resulting application architecture did not have the module *Finance Interface Module, Farmer Data Module, and Simulation Module* that were explicitly defined in the existing application architecture. This became also apparent in the post-interviews. In the interview, the questions 2, 3, 4 and 5 relate directly to the effectiveness of the approach. The architects provided a score of at least 4 for all these questions indicating that the approach for this case was considered is largely effective. The interview architects indicated *"Although we could identify some deviations and absent architectural elements, the resulting architecture matched our existing architecture very closely. And this was done in a pretty short period of time. For us, this was quite convincing."*

Practicality of the approach

The practicality of the method was assessed though questions 6 to 9 of the questionnaire. The architects gave at least a score of 4 for all these questions indicating that they were quite satisfied with the practicality of the method. They added: "*Usually designing the architecture requires lots of effort. In particular for IoT-based FMIS it is needed that all the concepts are well understood and the architectural decisions are made properly. With the reference architecture and the family feature models we could direct see the possible elements and could make the design decision quite quickly*".

Another interesting statement that was made was "The method helped us to think explicitly about our design decision and to communicate this early on. For the retrospective case we observed that we could have adopted other design decision which would perhaps be better. For the prospective case we were already guided to make the proper decisions. The method turned out really to be practical and useful".

For the question "*Will you use the method again?*" the answers were positive again and both architects indicated that they would use this method for the subsequent project. The architects also had some suggestions for improvement. "*The method adopts a family feature model and reference architecture. It would be helpful to indicate that specific delta modules are allowed when deriving the application architecture*".

3.7.4 Result of the Evaluation for the Prospective Case Study

The results of the evaluation for the prospective case study is shown in the radar chart of Figure 3-22. The answers to the provided questions were positive and got a score of 4 or higher. In the following we discuss again the effectiveness and practicality of the proposed approach for the prospective case study.

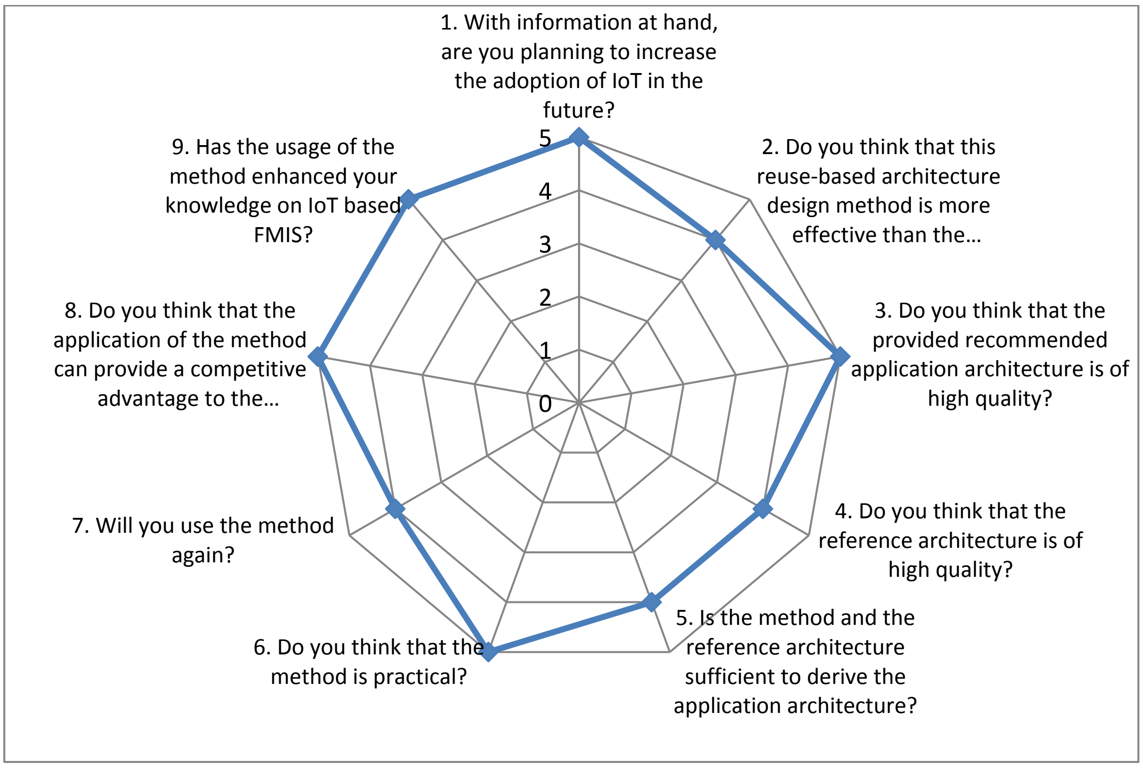


Figure 3-22: Results of the interview presented in radar chart – Prospective study

Effectiveness of the approach

For the retrospective case study, we compared the resulting architecture with the existing architecture to assess the effectiveness of the approach. Further we used the results of the interview as represented in the corresponding radar chart. For this prospective case study, we could not compare the results with an existing application architecture since only requirements document was provided and the application architecture had still to be designed. We used the requirements document to identify the required application feature diagram. Based on this, as discussed in section 3.2.2, we derived the application architecture. The effectiveness of the application architecture and the overall approach was based on the results of the interview.

As shown in the radar chart in Figure 3-22 again the scores for the questions 2 to 5 related to effectiveness were at least 4. This indicated that the approach was effective for the given case. The architects noted "*We could easily follow how the application architecture was derived based on our defined requirements. The resulting application architecture directly meets all our defined requirements. After this process we will develop the system and evaluate it also at the code level*".

Practicality of the approach

For assessing the practicality of the approach, we considered again the results of questions 6 to 10. Again, it appeared that these were all score with at least 4. Similar to the retrospective case study the approach was found practical and easy to use. Similar statements as in the retrospective case study were made. Some of the interesting different statements were the following.

"For us designing the architecture usually takes lots of effort. The approach helped us to derive application architecture in a very short period"

"The approach helped us not to derive the application architecture but also discuss the design decisions which we found very useful"

"In the beginning we were quite skeptical about the approach and expected a more academic exercise. However, the learning curve for the approach after the two hours introduction was quite low. We will use the approach as well. We might add new modules to the reference architecture which are dedicated to our company's domain".

3.8 DISCUSSION

The introduction of IoT has led to the notion of IoT-based FMIS to support the precision farming goals. In this chapter, we aimed to integrate the IoT systems with the FMIS to align both systems and create additional value that cannot be achieved if these are considered separately. This integration effort leads to the enhancement of the current FMIS systems with new modules that support the precision farming operations based on IoT. In our approach these required new modules have been explicitly defined in addition to the traditional FMIS modules. The overall approach as such provides an integrated view of the overall system to better support the architecture design of IoT-based FMIS.

The method that we have discussed can be adopted for deriving IoT-based FMIS architecture for multiple different systems. Hence, we focus on the whole product family of IoT-based FMIS systems rather than on a single system. The notion of product families or product line engineering and the corresponding systematic reuse is discussed in detail in the product line engineering community (Clements, 2006). Our method is inspired and customizes the product line engineering approach in which reference models are developed and applications are developed by reusing these reference models. The reference feature diagram that we have shown aims to target and integrate the domains of IoT and FMIS. The focus in this chapter was primarily to illustrate the overall method. The feature diagrams as well as the reference architecture design could be easily extended. We have discussed the architectures for IoT and FMIS separately and illustrated the integration of both for supporting IoT-based FMIS systems. The architecture can be extended in two ways. First of all, we could of course detail the different views to provide an even more comprehensive result. This would require for example to further detail the modules that are needed in the decomposition view. Secondly, we could extend the architecture representations with other architecture views. We have chosen three architecture views including decomposition view, layered view and deployment view. If needed other architecture views in the architecture documentation process could be used as well. Here again due to space limitations and the focus on the method rather than on the detailed output of the case studies we have chosen for the given scope. The complete versions of the feature diagrams as well the detailed implemented architecture designs have not been shown due to confidentiality issues.

The reference architecture is designed in such a way that it is generic enough to derive different concrete architectures. Nevertheless, like it is the case for all reference models, the reference architecture does not provide all the details. Likewise, a system which requires very dedicated features that were not anticipated before would not be covered by the reference architecture. Furthermore, our focus has been on illustrating the reference architecture and the approach for deriving a concrete architecture. This appeared to be useful and practical. However, we do not claim that the reference architecture is complete and further research can be used to refine and enhance the reference architecture. For example, the device layer and the related functionality have not been discussed in detail in this article. This could though be easily added without loss of generality and applicability of the proposed approach.

Although, we have showed our approach for two important case studies in the smart agri-food sector the method can be actually applied for the development of other FMISs. We have not focused on the implementation of these systems. The reason for this is because of confidentiality and the goal to prescribe the system-to-be in the prospective case study. For the prospective case study, it is decided to develop first a simulation system to evaluate the outcome of the method. We consider this as part of our future work.

In this chapter we have provided both the reference architecture for IoT-based FMIS and the overall approach to derive a concrete architecture. The idea of systematic guidelines for deriving a concrete architecture could also be used for enhancing the use of existing IoT-based reference architectures.

Although our method has illustrated the development of IoT-based FMIS systems we could even use the method for developing traditional FMIS systems. In that case we would omit the IoT architecture part and just focus on the development of reference models for FMIS.

This chapter describes a domain-driven design approach to design IoT-based FMIS and support the architect in deriving a concrete IoT-based FMIS architecture. Several other important issues need of course to be taken into account to realize effective precision farming. Important aspects include the acceptability of the provided IoT technology by the relevant stakeholders including the end-users, development of cost-effective transition strategies, and farm management and agricultural economics. Detailed discussion on economics and profitability of IoT solutions in the agriculture domain have been addressed by multiple studies including (T. W. Griffin & Lowenberg-DeBoer, 2005; Terry W. Griffin et al., 2018; Kutter, Tiemann, Siebert, & Fountas, 2011; Lowenberg-DeBoer, Erickson, & Vogel, 2000; Schimmelpfennig, 2016; S. Wolfert, Ge, Verdouw, & Bogaardt, 2017).

Adopting IoT-based FMIS is not trivial and usually requires large economic investments. To justify these up-front investments the return-on-investment both with respect to cost and quality should be defined. Further, IoT-based FMIS requires changes to farm equipment or totally new farm equipment, that the farmers are not used to. Hence, it is important to analyze the acceptability and adoption scenarios, and provide clear transition strategies for the efficient introduction, usage and maintenance of precision farming. Due to the concrete scope of the thesis we have not further elaborated on these in this chapter.

We have applied a systematic case study research to validate our approach. Each empirical study usually has to deal with a few potential threats to validity. In the following we discuss these for our case study research shortly and describe the mitigation strategy for each threat.

Construct validity refers to what extent the operational measures that are studied really represent what the researchers have in mind and what is investigated according to the research questions (Yin, 2009). Table 3-3 shows various identified threats to construction validity together with the counter measures.

Table 3-3: Threats to validity and applied counter measures in case studies

Threat	Countermeasure
Inappropriate analysis of existing requirements and architecture (for retrospective case study)	To ensure that we have understood all the requirements we have organized a meeting. The missing artefacts were reverse engineering and discussed with the architects.
Incorrect interpretation of the descriptions of the questions by the interviewed persons	We have applied the principles described in Kitchenham and Pfleeger (Kitchenham & Pfleeger, 2002) for constructing the questions and answers. To ensure uniqueness of interpretations of the questions, we have provided detailed explanations.
Incorrect interpretation of the description of the answers by the interviewed persons, and likewise the wrong selection of answers	Same thing is also true for the answers of the questions. Especially for the Likert-scale questions. In most of the cases, it is difficult to differentiate for example between a “Strongly Agree” and “Agree”. This was also one of the comments we have gathered in the trial runs. To mitigate this per each Likert-scale question, we have tried to define each scale as much as possible to avoid confusion.
Incorrect interpretation of the open questions by the interviewed persons	To mitigate this threat, we have verified the interpretation of the questions with interviewed persons.
Incorrect interpretation of the researchers to the provided answers of the interviewed persons	To mitigate this threat both researchers were present in the interview to achieve observer triangulation.

Internal Validity relates to a causal relationship between treatment and the outcome. In the case of retrospective case study, it has been relied on existing design documentation and related literature. There could be missing information in the cases that would affect the outcome. To mitigate this threat several iterations were applied to derive both the application feature model and the application architecture. In the prospective case, the lack of proper requirements documentation could have an impact on the derived decisions. To mitigate this threat, this has been discussed with the interviewed persons in detail and several iterations were adopted.

External Validity concerns the ability to generalize the results of the study. In the case study evaluation both a retrospective and prospective case study were adopted which were also in different domains. This was done to support triangulation and likewise extend the external validity. The approach was considered effective for both case studies but due to the small number of participants a stronger statement could not be provided. In the future work a repetition of this study with multiple other case studies with an increased number of participants would further justify the claims of this chapter and also support the quantitative evaluation.

3.9 Related Work

There are several studies that discuss the adoption of internet technologies to support FMIS and cope with complexity (Fountas et al., 2015; Kaloxylos et al., 2012; Kruize et al., 2016; Murakami et al., 2007; Nikkilä et al., 2010; Sørensen et al., 2010; Steinberger, Rothmund, & Auernhammer, 2009; J. Wolfert, Verdouw, Verloop, & Beulens, 2010). These studies have focused on different issues including the adoption of service-oriented architectures for FMIS (Murakami et al., 2007; Steinberger et al., 2009; J. Wolfert et al., 2010), the development of data exchange standards for supporting interoperability over the internet (Schmitz et al., 2009), the adoption and implementation of geographic information systems (GIS) (Seelan, Laguette, Casady, & Seielstad, 2003). Although the main focus of these studies is integration and operation of an FMIS over the internet the adoption of IoT is not explicitly considered.

In (Murakami et al., 2007), a distributed service-oriented reference architecture is proposed for the development of information systems for precision agriculture. This web-based approach is focused on communication between software services on a service bus. In (Schmitz et al., 2009), the so-called AgroXML is proposed as a standardized language based on XML, to be used for data exchange in FMIS. In (Nikkilä et al., 2010), a web-based approach is defined to implement connectivity requirements arising from the internet and the management of GIS data. In (Sørensen et al., 2010), a new model for FMIS is proposed to have better information handling focusing on internal data connection, external information collection, plan generation, and report generation in FMIS. In (Kaloxylos et al., 2012), an architecture is proposed to provide support and integration of different stakeholders and services, and interworking with the external services.

There are some studies in the literature related to web-based architectures (Chaudhary, Sorathia, & Laliwala, 2004; Steinberger et al., 2009). These studies present architectures to enhance the effectiveness of web-based decision support system on which data can be requested for further use via a web portal and a web service interface.

Instead of full FMIS most architecture academic research on FMIS is restricted to individual component of an FMIS such as predicting crop yield, implementing a special sensor, and the usability of an FMIS (Nikkilä et al., 2010). There are actually few studies that explicitly discuss FMIS architectures in a comprehensive manner, and only specific focus of the architecture is considered instead. For example, in (Linseisen, 2001) , FMIS architecture is discussed by focusing on an information system gathering and storing high accuracy GPS data. In (Beck, 2001), an architecture, based on implementing object databases, CORBA middleware, and Java languages is proposed to provide easier development, maintenance, and easier integration of information systems.

Advancements in the functionality of academic and commercial FMIS are presented in (Fountas et al., 2015). This study investigates commercial and academic FMIS packages and performs cluster analysis between them. The authors indicate that commercial packages tend to target daily farm office tasks such as budgeting, finance, recordkeeping, machinery management, and documentation. On the other hand, academic FMISs deal with compliance to standards, automated data capture, and interoperability issues.

There are also studies that discuss traditional on-site FMIS software. However, these studies mainly focus on the improvement of information integration of traditional FMIS and do not take IoT technologies into account. For example, in (Verdouw, Wolfert, & Tekinerdogan, 2016), the authors propose an architecture to improve the standardization and integration of data, application, and process. A service-oriented architecture (SOA) based solution is proposed to improve the information integration implementing business process management (BMP) concepts.

Related to IoT is the research on wireless sensor networks which is reviewed in (Aqeel-Ur-Rehman, Abbasi, Islam, & Shaikh, 2014; Jawad, Nordin, Gharghan, Jawad, & Ismail, 2017). These studies primarily focused on comparing sensors and communication technologies such as ZigBee, Bluetooth, Wifi, Sigfox, Wibree, long range radio and GPRS. Although these protocols might increase the number of possibilities to communicate data in IoT, these researches do not directly offer architectural solution for FMIS.

This chapter has focused on applying IoT for FMIS in particular. However, IoT has also been applied in different application domains. The application of IoT in agriculture has been reviewed in (Verdouw et al., 2016). This review showed that IoT concept got attention of scientific community in 2010 and since then number of researches are increased. Total 168 papers and books were reviewed in this paper. Top topics of these studies are food supply chains, arable farming, general agriculture, greenhouse horticulture, and livestock farming, and open-air horticulture including orchards respectively with respect to number of papers published. On the other hand, it is stated that IoT applications mostly focus on basic functionalities, including tracking, tracing, monitoring, and event management. It is concluded that although IoT receives an increasing level of attention, it is still in its infancy in the agriculture and food domain suffering from lack seamless integration and advanced solutions.

3.10 Conclusion

FMISs are being more and more applied in many different farming systems. Several architectures for FMIS have been proposed in the literature but these are usually abstract, and it is not trivial to derive the application FMIS architecture for the corresponding context of the farm system. In this chapter we have provided an architecture design method for deriving application architectures for various different FMISs. For this we have adopted the reference architectures of internet of things (IoT) and FMIS and defined a novel IoT-based FMIS. We have provided the architecture design method for deriving the customized application FMIS architecture. To support the design of the application architecture we have adopted a domain driven approach whereby we defined a family feature diagram representing the common and variant features of IoT-based farm management information systems. We have illustrated our approach using a systematic case study approach. Hereby we have adopted both a retrospective and prospective case study including the IoT-based wheat production and IoT-based tomato FMIS, respectively. The case study research showed that the approach was both effective and practical. It appeared that both the reference architecture that we have provided as well as the corresponding method appeared to be very useful to derive the customized application FMIS architecture. Since in general developing IoT systems is not trivial adopting a systematic approach appears to be useful in not only the final results but also the intermediate steps that support the communication between the stakeholders and the overall guidance of the design decisions. The contribution of this chapter can be useful for both researchers who do research on IoT-based FMIS as well as practitioners who aim to architect different FMIS systems. The future work will apply our approach for other farm management systems. Further focus will be on the architecture design and integration of multiple different FMISs.

4

PATTERN-BASED INTEGRATION OF INTERNET OF THINGS SYSTEMS[1]

[1] This chapter is based on the following accepted paper to be published in Proceedings of the 2018 International Conference on Internet of Things:

- B. Tekinerdogan and Ö. Köksal, "Pattern-Based Integration of Internet of Things Systems", International Conference on Internet of Things (ICIOT), Seattle, USA, 2018.

Abstract

The Internet of Things (IoT) is the network of physical devices embedded with sensors, actuators, and connectivity which enables these objects to connect and exchange data. Cleary the IoT has a pervasive impact on the society and an increasing number of systems are now based on IoT. One of the key challenges in IoT is coping with the heterogeneous set of systems and the integration of these systems in the same communication network. Several studies have focused on this integration aspect and addressed this at different levels of abstraction. Unfortunately, the different approaches are scattered and fragmented over the different studies and it is not clear how to cope with the integration concern within a single IoT system but also across multiple IoT systems that need to be integrated. To this end this chapter provides a comprehensive and systematic approach for identifying the key integration concerns in the IoT system architecture and describing the currently provided solutions. For this we adopt a pattern-based approach in which generic architecture solution structures are provided to these recurring integration concerns. We illustrate our approach for addressing the integration of IoT-based systems within the context of smart city engineering.

Keywords: Internet of Things, Architecture Design Patterns, Smart City Engineering

4.1 INTRODUCTION

The Internet of Things (IoT) is the result of technological progress in many parallel and often overlapping fields, including those of embedded systems, ubiquitous and pervasive computing, mobile telephony, telemetry and machine-to-machine communication, wireless sensor networks, mobile computing, and computer net-working. In essence, IoT combines the concepts “Internet” and “Thing” and the provided definitions in the literature can be interpreted how these have addressed these two concepts. What is important is that IoT adds a new dimension “any-thing” to the current communication technologies (ICTs), which already provide "any time" and "any place" communication.

To support the design of IoT systems, various reference architectures have been provided in the literature. In general, IoT architecture is represented as a layered architecture with various set of layers representing a grouping of modules that offer a cohesive set of services. Based on the literature (Al-Fuqaha, Guizani, et al., 2015; Gazis et al., 2015; Palattella et al., 2013; Pandya & Champaneria, 2015) we provide the reference architecture as shown in Figure 2-1.

The reference architecture consists of seven layers including Device Layer, Network Layer, Session Layer, Cloud Layer, Application Layer, Management Layer, and Security Layer (Köksal & Tekinerdogan, 2017a). Usually these layers can be distributed in different ways over the different nodes in the IoT system. Using the IoT reference architecture various different IoT systems can be designed. Each such IoT system integrates the various devices within the same network. Yet, the scope of an IoT system is often within a particular scope and the integration with other IoT systems or non-IoT systems is not a trivial task.

Cleary the IoT has a pervasive impact on the society and an increasing number of systems are now based on IoT. One of the key challenges in IoT is coping with the heterogeneous set of systems and the integration of these systems in the same communication network. Several studies have focused on this integration aspect and addressed this at different levels of abstraction. Unfortunately, the different approaches are scattered and fragmented over the different studies and it is not clear how to cope with the integration concern within a single IoT system but also across multiple IoT systems that need to be integrated. To this end this chapter provides a comprehensive and systematic approach for identifying the key integration concerns in the IoT system architecture and describing the currently provided solutions. For this we adopt a pattern-based approach in which generic architecture solution structures are provided to these recurring integration concerns. We illustrate our approach for addressing the integration of IoT-based systems within the context of smart city engineering.

4.2 Case Study: Smart City Engineering

In this section, we define a case study that will be used to illustrate the problem statement and the approach in further sections. The case study that we consider is within the context of smart city engineering (Yoshikawa, Sato, Hirasawa, Takahashi, & Yamamoto, 2012). One of the important applications in smart city engineering includes the development of smart traffic system (STS). STS provides different capabilities such as traffic light management, congestion detection, traffic regulation, shared parking platform, etc. The high-level reference architecture of STS is depicted in Figure 4-1 (Tekinerdogan, Celik, & Köksal, 2018).

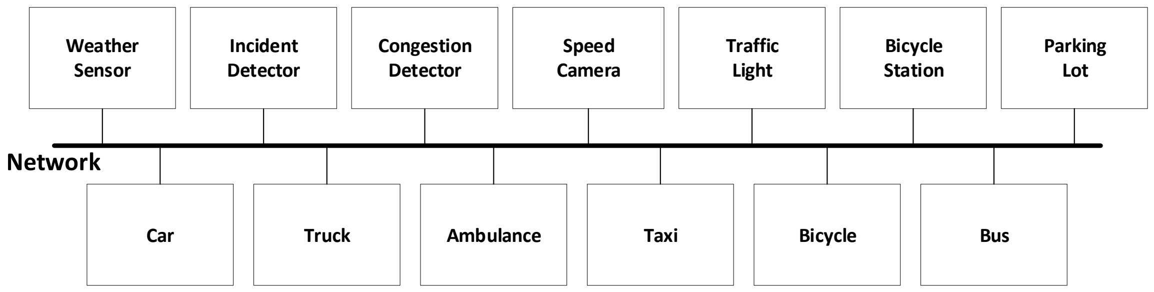


Figure 4-1: Conceptual Architecture for Smart Traffic System

Although the above case integrates many different entities it still deals with the design of a single system, in this case an STS. Very often it is required though to integrate the STS with other systems in the smart city engineering context, such as city energy consumption system, the weather information system, the security system, the air quality control system, the smart lighting system etc. (Figure 4-2).

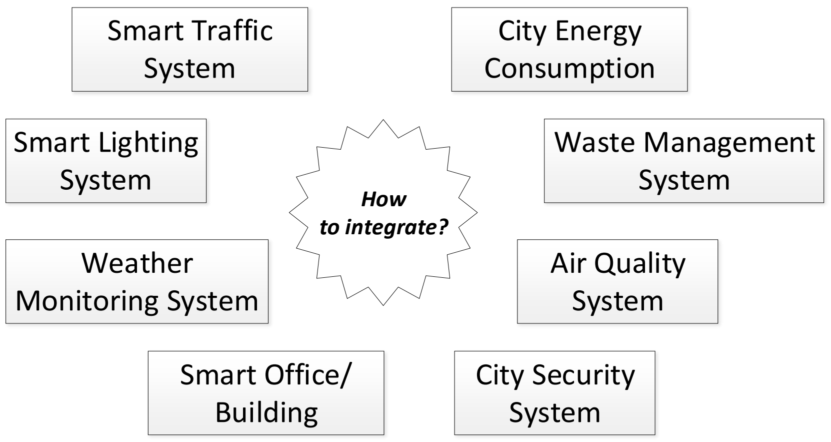


Figure 4-2: Integration of different IoT systems in the context of Smart City Engineering

Integrating all these systems in a coherent manner is not trivial and requires careful consideration. We will elaborate on this in the next sections.

4.3 INTEGRATION FRAMEWORK

The integration of IoT systems can be considered at different abstraction levels. We will discuss the integration based on the four layers of the architecture as defined in Figure 2-1. To illustrate this need for integration at different levels Figure 4-3 shows the integration of different IoT systems.

As shown in Figure 4-3 we distinguish the following types of integration in IoT systems (1) *Session Layer Integration*: (1a) *Protocol Integration via IoT Gateway* (1b) *Protocol Integration via Middleware*, (2) *Cloud Layer Integration*, and (3) *Application Layer Integration*. For describing the integration solutions, we will adopt a design pattern-based approach. A design pattern represents a generic solution to recurring problems. Design patterns play an important role in the engineering design process and can be applied at the different levels in the lifecycle including the architecture design, detailed design, and the code. In this chapter, we will mainly focus on architectural patterns which focus on the gross-level structure of the system and its interactions (Bushmann, Meunier, & Rohnert, 1996). In the following, for each level we will describe the possible architectural patterns that can be used in the integration of IoT systems. Hereby we will also shortly indicate the advantages and disadvantages of the adopted architecture design pattern.

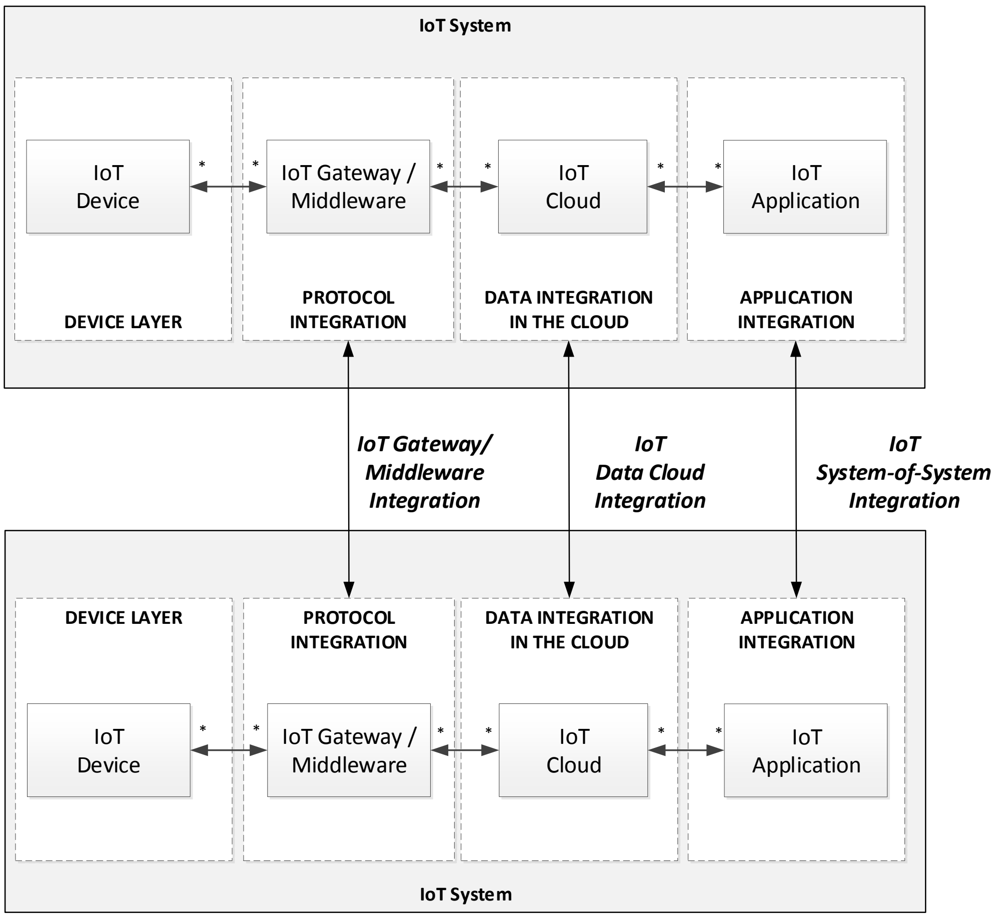


Figure 4-3: System Integration of IoT-based systems in different layers

4.3.1 Protocol Integration via IoT Gateway

Multiple session layer protocols exist in the IoT domain to integrate the different things in the IoT as shown in Table 4-1 (Köksal & Tekinerdogan, 2017a). The issue of heterogeneous devices adopting different communication protocols impedes the integration of these de-vices in the same IoT systems. An IoT gateway acts as a portal between two elements of one or multiple IoT systems, allowing them to share information by communicating between the adopted IoT protocols. An IoT gateway, as such, bridges the gap between devices, cloud, and the computer or mobile device providing a communication link between the devices and cloud.

Table 4-1: Selected characteristics of the session layer protocols

Characteristics	**AMQP**	**CoAP**	**DDS**	**MQTT**	**XMPP**
Broker/Bus based	Broker-based	Broker-based	Bus-based	Broker-based	Bus-based
Com. pattern	Pub/Sub	Request-Reply	Pub/Sub	Pub/Sub	Pub/Sub
Message/Data centric	Message	Data	Data	Message	Data
Real-Time	No	No	Yes	No	Near RT
Source-Target	S2S	D2D	D2D	D2S	D2S
Transport	TCP	UDP	TCP/UDP	TCP	TCP

Figure 4-4 shows the different gateway patterns used for integration of IoT systems. In the classical protocol integration hardware/software gateways are used to format and translate data coming from one protocol type to a different protocol type as given in Figure 4-5a. This type of protocol integration is successful as long as the number of devices to be integrated is not excessive. However, for a large-scale set of devices it is not easy to handle all the heterogeneous protocols and technologies of the IoT and design a suitable gateway without causing anomalies such as timing and collusion problems. With the possible addition of even more protocols and technologies developed in IoT domain, this problem will become even less manageable (Olivieri & Rizzo, 2015). In order to solve the scalability problems and to provide more efficient gateways the following solutions are proposed in the literature.

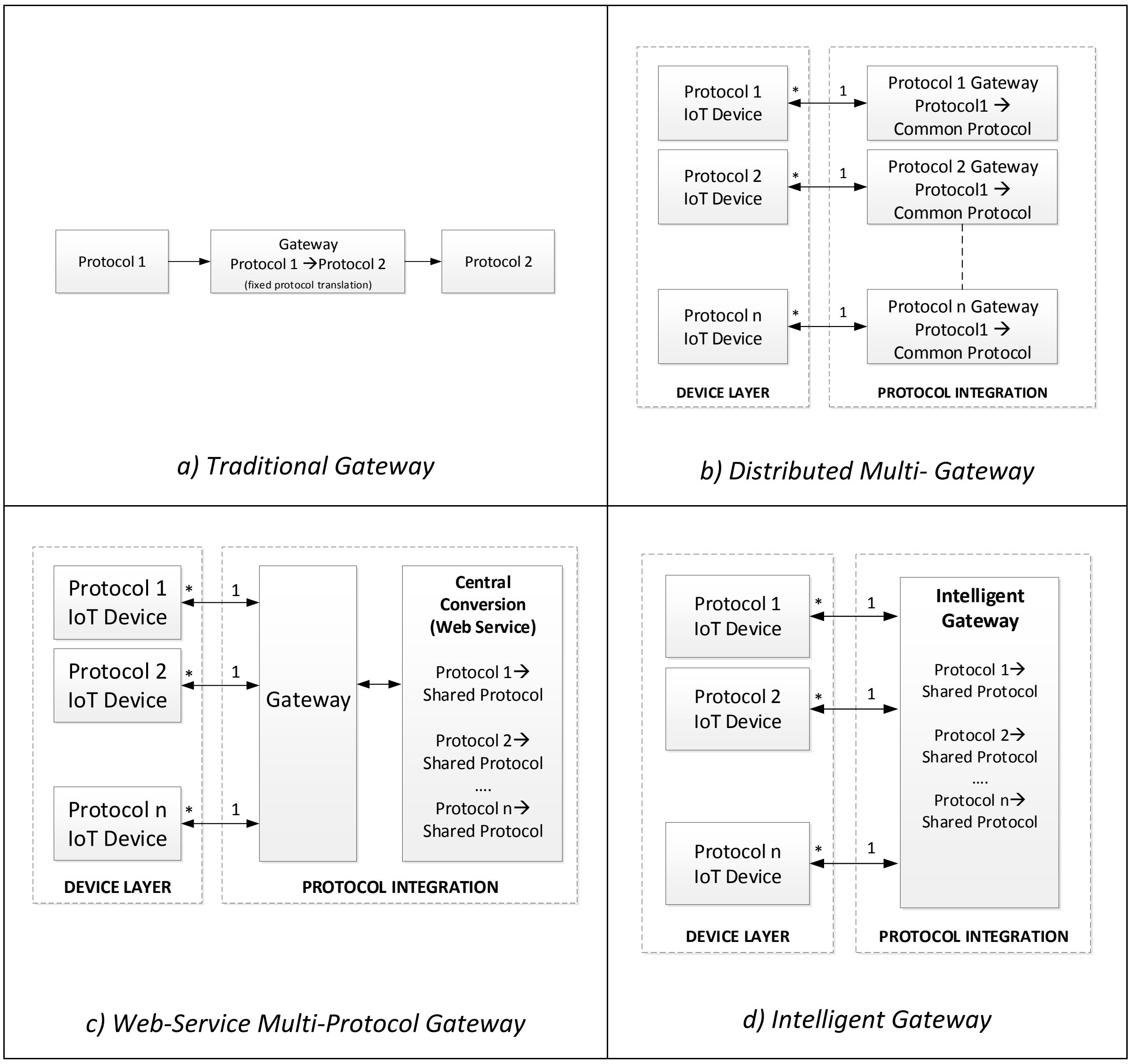

Protocol 1
Gateway
Protocol 1 →Protocol 2
(fixed protocol translation)
Protocol 2
a) Traditional Gateway
Protocol 1 IoT Device
Protocol 2 IoT Device
Protocol n IoT Device
Protocol 1 Gateway Protocol1 → Common Protocol
Protocol 2 Gateway Protocol1 → Common Protocol
Protocol n Gateway Protocol1 → Common Protocol
DEVICE LAYER
PROTOCOL INTEGRATION
b) Distributed Multi- Gateway
Gateway
Central Conversion (Web Service)
Protocol 1→ Shared Protocol
Protocol 2→ Shared Protocol
Protocol n→ Shared Protocol
c) Web-Service Multi-Protocol Gateway
Intelligent Gateway
d) Intelligent Gateway

Figure 4-4: Different gateway patterns used for integration of IoT systems

4.3.1.1 Distributed Multi-Gateway Approach

In this approach multiple gateways are used to cope with the different set of protocols in the IoT system in Figure 4-5b (Olivieri & Rizzo, 2015). Hereby, the protocols are treated singularly or as a subset of the selected protocols in each gateway. Each gateway translates its protocol to a common shared protocol. The gateways themselves can communicate using the common protocol. By combining gateways dedicated to different technologies multi-protocol scenarios can be generated.

4.3.1.2 Web-Service Multi-Protocol

Instead of having a gateway for each protocol it is also possible to provide a central gateway that is connected to a central conversion server. This so-called web-service multi-protocol pattern is shown in Figure 4-5c (Olivieri & Rizzo, 2015). In this approach, gateways receive raw data from sensors which are translated to a shared format by connecting with a web-service. In contrast to the distributed multi-gateway there is only one gateway which does the translation among the protocols.

4.3.1.3 Intelligent Gateways

For translating to different protocols, the gateway can be provided the required translation functionality as shown in Figure 4-5d. In this case the gateway will not de-pend on a separate central server such as in the case of web-service multiprotocol gateway but include the functionality for translating the protocols. Hence, we can indicate this as an intelligent gateway solution. This solution is, for example adopted by M. Diaz-Cacho et al. (Diaz-Cacho, Delgado, Falcon, & Barreiro, 2015) and (Al-Fuqaha, Khreishah, Guizani, Rayes, & Mohammadi, 2015). In these solutions intelligent gateways convert the incoming protocol data to a common shared protocol data which is in this case is extended MQTT. However, the intelligent gateway can in principle also provide different kind of functionality and mapping of the protocols.

4.3.2 Integration via Middleware

The alternative way of overcoming the protocol heterogeneity other than using a gateway is the use of a middleware to be used as an abstraction layer. This pattern is shown in Figure 4-5.

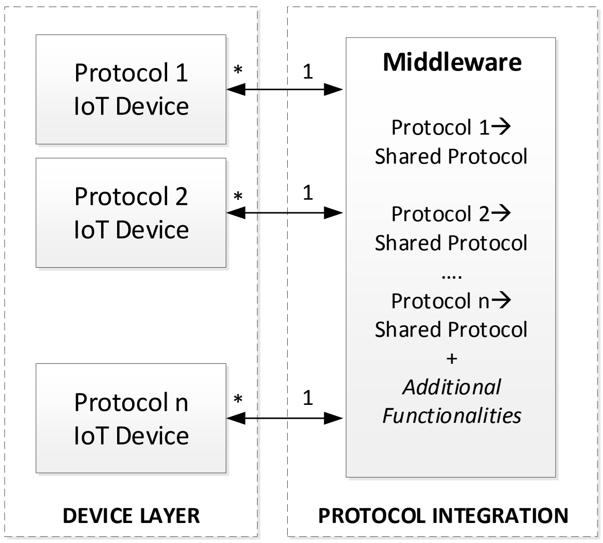


Figure 4-5: Integration of protocols using middleware

This goes beyond the intelligent gateway solution that includes only functionality for translation among the protocols. In case of a middleware solution also additional functionality such as naming and directory services, security aspects, reliability and other functional and quality services can be also provided. The primary aim of using middleware is to provide seamless integration of systems by hiding the communication and various low-level acquisition aspects (Calbimonte, Sarni, Eberle, & Aberer, 2014).

There are studies offering the use of an IoT middleware to integrate IoT-based systems in the literature. A. Ngu et al. (Ngu, Gutierrez, Metsis, Nepal, & Sheng, 2017) provides a survey about IoT middleware integration. Lomotely et al. (Lomotey, Pry, Sriramoju, Kaku, & Deters, 2017) proposes a middleware to be used as an abstraction layer to address variation in device semantic and protocols that limit the interoperability of the systems. The proposed middleware uses enhanced environment features to match the appropriate communication protocol to aid pushing data from sensors to cloud infrastructure.

4.3.3 Integration in the Cloud

Another integral component of the IoT is cloud computing. In general, three types of cloud computing models are defined including Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Öztürk & Tekinerdogan, 2011; Tekinerdogan & Öztürk, 2013; Tekinerdogan, Öztürk, & Doğru, 2011). The IaaS model shares hardware resources among the users. Cloud providers typically bill IaaS services according to the utilization of hardware resources by the users. The PaaS model is the basis for the computing platform based upon hard-ware resources. It is typically an application engine similar to an operating system or a database engine, which binds the hardware resources (IaaS layer) to the soft-ware (SaaS layer). The SaaS model is the software layer, which contains the business model. In the SaaS layer, clients are not allowed to modify the lower levels such as hardware resources and application platform. Clients of SaaS systems are typically the end-users that use the SaaS services on-demand basis. For adopting cloud-based integration the different clients are considered the individual systems in the overall System of Systems (SoS). The integration pattern based on the IaaS, PaaS, or SaaS in the cloud layer is shown in Figure 4-6a.

An important benefit of IoT is the generation of data that can be further used to derive information to support the decision-making process. The data is typically stored in the cloud which can be used to support analytical and computational tasks on these data allowing centralized access to the generated IoT services (Botta, De Donato, Persico, & Pescape, 2014).

Figure 4-6b shows the pattern for the data integration in the cloud. Hereby, the integration of the systems is primarily based on the integration of the data from the different IoT systems. Since each IoT system can use its own type of data platform and the corresponding data structures and formatting, the integration will need to support data interoperability. For this it is needed to adopt a common data format and platform that is adopted at a central cloud node. Incoming data from different nodes will be typically mapped to a shared data format. Subsequently a data fusion and/or data conversion process will be carried out to synthesize the data. The cloud node will typically include analytics modules for processing the data for descriptive, diagnostic, predictive or prescriptive analytics.

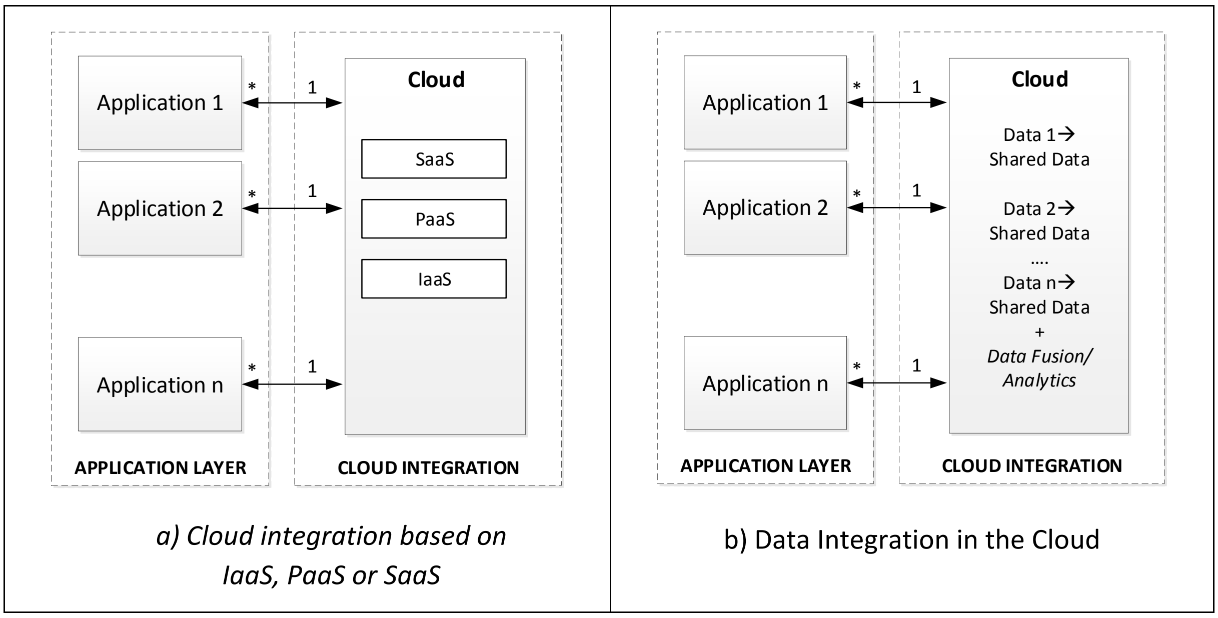


Figure 4-6: Cloud-based integration of different IoT systems

4.3.4 Integration in Application Layer

Besides of integration at the gateway or middleware level we can also achieve the integration of IoT systems at the application layer level. Much has been written about application integration and likewise we will borrow from the earlier concepts to define the integration of IoT systems. In the literature dozens of architecture patterns have been published regarding application integration (Bass et al., 2003; Clements et al., 2010, 2011; Croes, 2015; David Garlan, 1994; Fielding, 2000). In the following we will consider only those patterns that can be directly used for supporting the integration of systems, and in the context of this chapter, in particular the integration of IoT systems.

4.3.4.1 Peer-to-Peer

In the peer-to-peer architectural pattern, peers (IoT Systems) connect to each other directly, and there is no intermediate component between the IoT systems. The conceptual model is shown in Figure 4-7a. The elements in the system are autonomous, equal peers that are both providers and consumers of data and processing power. Further, the primary content is provided by peers, are there are no central components providing content. In addition, peers can be added and removed from the system at any time.

4.3.4.2 Client-Server

The Client-Server architectural pattern is a very common and well-known pattern for network-based applications. The conceptual model this pattern is shown in Figure 4-7b. Hereby some systems play the role of Clients, while other adopts the role of a Server. One or multiple Client components initiate a request to a Server, which then performs some computations and responds to the Clients. Only clients can initiate communication, while servers only respond to requests from clients. If needed server components can be clients to other servers. Clients cannot communicate to each other. As we will see in the later sections this is different from the Event-Based and Streaming invocations since the Client decides itself when to initiate a request.

4.3.4.3 Event-Based

The conceptual diagram of the event-based software architectural pattern is given in Figure 4-7c. This pattern is based on implicit invocations which are induced by events, i.e., when a certain event takes place it triggers the function calls. Event can be defined as a significant change in state. Typically, event-based systems are composed of event producers, event consumers and event channels. The events are sent to the listeners over the network even they are not on the same hardware. So, this pattern is well-suited for real-time applications, message-oriented middleware, and point-to-point communications. Further, the event-based pattern supports parallel execution of tasks and scalability.

4.3.4.4 Publish-Subscribe

This pattern is shown in Figure 4-7d. It consists of mainly three elements including Publishers, Subscribers, and Topics. Publishers write to Topics and Subscribers read from the Topics on which they are registered. One Publisher can write to many Topics and one Subscriber can read from many Topics. Unlike the event-based pattern described above, the subscribers in this pattern are all interested in a type of event happening without knowing the publisher of the event. The adopted communication pattern provides space decoupling, time decoupling and synchronization decoupling (Eugster et al., 2003). The decoupling of producer and consumer participants increases scalability by removing explicit independencies between communicating parties. Removing these dependencies together with the asynchronous communication feature of this infrastructure makes this pattern well-suited for even large scale IoT systems.

4.3.4.5 Service-Oriented Architecture (SOA)

The service-oriented architecture deals with composing applications by integrating distributed, separately maintained components aiming vendor and technology in-dependence. This integration composed of three essential loosely coupled parts which are registry, service providers, and service requestors as shown in Figure 4-7e. In this integration type, service publishes its description to the service registry that keeps the list of all services with their locations and functionalities. When a service requester requires a service, it gets the required information from the registry and communicates with the requested service over a standardized communication.

This type of decoupled integration is especially suitable for heterogeneous distributed systems supporting evolvability and interoperability. The disadvantage of this integration pattern is the complexity.

4.3.4.6 Pipes and Filters

This pattern is composed of two basic elements: pipes and filters as shown in Figure 4-7f. Filters are connected to each other by pipes. Filters transform the data received from another filter into a new form and output this transformed data to the following filter. Pipes are the routes of data streams. Although filters are independent of each other and might execute parallel, they must use the data type agreed with pipe in order to communication takes place. This simple communication mechanism makes the pattern scalable and reusable supporting evolvability. On the other hand, this batch-type data processing cannot handle interactivity well and latency causes performance degradation.

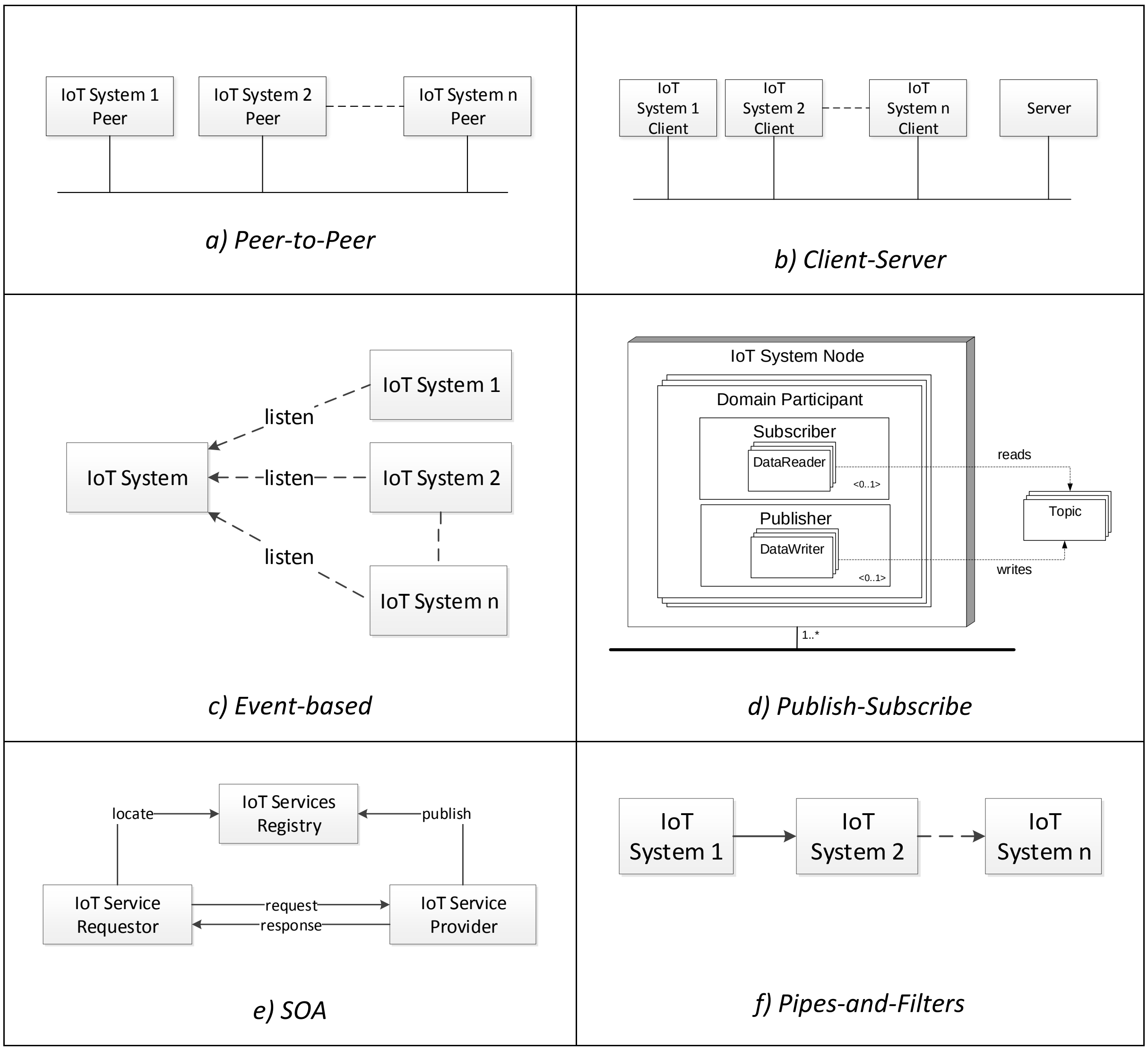


Figure 4-7: Patterns for Integration at the Application Layer

4.4 OVERALL APPROACH

Table 4-2 shows the summary of the previously defined patterns that can be used to support integration of the concerns in the IoT system. Obviously, it is clear that for integrating multiple IoT systems many different issues need to be taken into account. To guide and support the integration of the IoT systems we propose the process as shown in Figure 4-8.

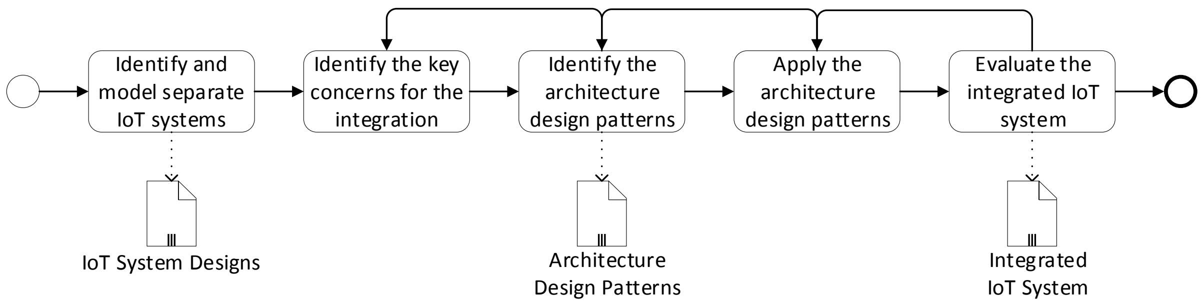

Identify and model separate IoT systems
Identify the key concerns for the integration
Identify the architecture design patterns
Apply the architecture design patterns
Evaluate the integrated IoT system
IoT System Designs
Architecture Design Patterns
Integrated IoT System

Figure 4-8: Patterns for Integration at the Application Layer

The first step in the process is the identification of the individual IoT systems that need to be integrated. The second step in the process includes the identification of the concerns for the integration. This will require checking the needs and the overall purpose for the SoS. Hereby it is important to describe the added value that is created using the integration of these systems. In the third step we identify the patterns that can be used for the integration. These will include the patterns that we have described in the previous section. For this we will adopt the criteria and consider the constraints, the advantages and disadvantages of the corresponding patterns. Once the patterns have been identified we apply and compose the patterns. In principle, more than one pattern can be applied which will require design decision for the composition. In the final step we evaluate the overall architecture of the SoS with respect to the initial objective and the stakeholder concerns in the SoS.

Table 4-2: Identified List of Patterns that can be used in the integration process

Layer	Pattern	Integration Approach
Session Layer	Traditional Gateway	Provides translation of a given protocol to a predefined protocol
	Multi-Gateway	Provides multiple gateways each of which can translate to a dedicated protocol
	Web-Service Multi-Protocol	Provides a single gateway that can provide the translation of the protocols to a common protocol through a central web server
	Intelligent Gateway	Provides a gateway that includes the required functionality for translating to different protocols.
	Middleware	Connects devices within or across IoT systems and provides additional services (e.g. naming and directory, quality of service, etc.)
Cloud Layer	SaaS based	Multiple IoT systems are tenants of the cloud and use the SaaS
	PaaS based	Multiple IoT systems are tenants of the cloud and use the PaaS
	IaaS based	Multiple IoT systems are tenants of the cloud and use IaaS
	Data Integration	Multiple IoT systems use data fusion and analytics as cloud services
Application Layer	Peer-to-Peer	Entities within an IoT system or across IoT systems communicate as peers, that is, autonomously as data providers and consumers
	Client Server	One IoT system or another system is defined as a server which is used by multiple other IoT systems.
	Event-Based	IoT elements listen to other IoT elements. In case of changes events are triggered to the coupled IoT elements (system or devices)
	Publish-Subscribe	Multiple IoT systems communicate as participants that are interested in defined topics. If topics change, the loosely coupled IoT systems are notified which can take further actions.
	SOA	IoT service providers define and register their services to the IoT Service Registry. The IoT service requestor can request and use the registered IoT services.
	Pipes & Filters	Every IoT system is considered as a black box component that gets as input data, which is then processed by the IoT system, and further provided to the output. IoT systems can use data from other IoT systems and/or provide data to other IoT systems. IoT systems can be configured in multiple different ways but there is no shared state.

4.5 INTEGRATING THE SMART CITY ENGINEERING SYSTEMS

In order to illustrate our approach, we will consider the smart city engineering case study as defined in Section 2. The provided solution is given in Figure 4-9. Here it is assumed that Air Quality System and Weather Monitoring System reside at the same location, which are integrated using a smart gateway that realizes the translation of the adopted different protocols in these systems. The Smart Building and Smart Office are also considered in the same location. Hereby, a multi-protocol gateway solution has been used in which multiple gateways for different protocol translations are adopted. The Smart Traffic System, Smart Lighting System, and City Energy Consumption system are considered to be connected over a local area network and communicate through a middleware platform. The middleware provides the translation services and additional network and communication services. All the systems are integrated in the City cloud in which all the cloud integration patterns including

SaaS, PaaS, IaaS and data integration is used. This is one solution in which different patterns have been applied to meet the requirements. For different other requirements other patterns can be used to integrate the IoT systems.

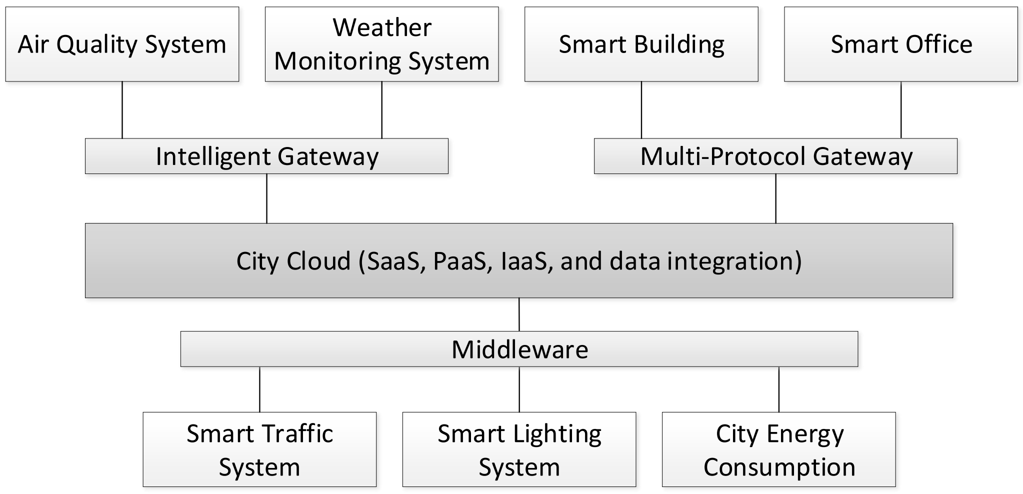


Figure 4-9: Example pattern-based integration of smart city engineering systems

4.6 CONCLUSION

One of the key challenges in IoT is coping with the heterogeneous set of systems and the integration of these systems in the same communication network. Based on a layered reference architecture for IoT we have indicated that the integration can be at different layers including session layer, cloud layer and application layer. Further we have shown that the integration is typically carried out based on well-defined patterns, that is, generic solutions structures for recurring problems. We have not provided any new integration solution but rather systematically compiled and structured the integration patterns as defined in the literature. Our study has resulted in 15 different patterns which can be used in different combinations. To guide the application of the patterns we have provided a general process represented using the BPMN. The process and the patterns have been successfully applied to a smart city case study. Hence, we have shown that the systematic structuring of the integration patterns is useful for developing IoT systems that need to integrate heterogeneous elements. Although we have identified and described the key patterns in the literature, this study could be further extended by considering other patterns. In our future work we will consider other type of IoT reference architectures and based on these, enhance the set of patterns that we have described in this chapter. Further, we will also consider IoT patterns beyond the integration concern such as security and safety patterns.

5

OBSTACLES IN DATA DISTRIBUTION SERVICE MIDDLEWARE[1]

[1] This chapter is based on the following published papers:

- Ö. Köksal and B. Tekinerdogan, “Obstacles in Data Distribution Service Middleware: A Systematic Review,” Future Generation Computer Systems, vol. 68, pp. 191–210, 2017.
- Ö. Köksal and M. Akyuz, “Aspect Oriented Approach for Cross-Cutting Concerns in Data Distribution Service Based Systems,” Journal of Science and Engineering, vol. 19, no. 55, pp. 43–56, Jan. 2017.

Abstract

Data Distribution Service (DDS) is a standard data-centric publish-subscribe programming model and specification for distributed systems. DDS has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. Various papers have been written on the application of DDS, however, there has been no attempt to systematically review and categorize the identified obstacles. The overall objective of this chapter is to identify the state of the art of DDS, and describe the main lessons learned and obstacles in applying DDS. In addition, we aim to identify the important open research issues. In this chapter, a systematic literature review (SLR) is conducted by a multiphase study selection process using the published literature since the introduction of DDS in 2003. We reviewed 468 papers that are discovered using a well-planned review protocol, and 34 of them were assessed as primary studies related to our research questions. We have identified 11 basic categories for describing the identified obstacles and the corresponding research challenges that can be used to depict the state-of-the-art in DDS and provide a vision for further research.

Keywords: Data Distribution Service, Middleware, Systematic Literature Review

5.1 INTRODUCTION

Distributed systems realize the distributed execution of software systems over multiple resources to meet different requirements and quality factors such as performance, interoperation, multi user support. A distributed system consists of multiple software components that are located on networked computers, but act and run as a single system. The computers that are in a distributed system can be connected by a local network and be physically close to each other, or they can be connected in a wide area network and geographically distant. Distributed systems offer many benefits over centralized systems, including scalability, concurrency and redundancy.

To reduce the effort for developing distributed systems, common middleware architectures have been introduced that provide common services such as name and directory services, discovery, data exchange, synchronization, transaction services, etc. Middleware can be classified in different ways including the integration type of middleware that defines the different approaches for integrating the components in the system environment.

Based on the integration type middleware has been often classified as procedural middleware, transactional middleware, object-oriented middleware, and message-oriented middleware (Myerson, 2002). Procedure-oriented middleware uses a synchronous communication to integrate the components. Transactional middleware provides transaction capabilities to support the integration of systems. Object-oriented middleware is an object-oriented extension of procedural middleware including additional support for inheritance, object references and exceptions. Examples of object-oriented middleware are OMG Common Object Request Broker Architecture (CORBA), Java RMI and Microsoft COM (Pritchard, 1999). Finally, message-oriented middleware is a middleware that supports the integration of components using messages. Two different types of message-oriented middleware (MOM) can be distinguished: Message Queuing and Message Publish/Subscribe. In the message queuing middleware, the communication among the components happens via a message queue. Hereby, messages are stored in a specific queue upon which the clients can retrieve messages from the queues they are interested in. The publish/subscribe middleware adopts an event-driven approach based on publish/subscribe communication pattern. The publish/subscribe pattern has gained broad attention in the development of loosely coupled, scalable large-scale applications. In distributed systems with the publish/subscribe interaction pattern, so-called subscribers express their interest in an event, or a pattern of events, and are subsequently asynchronously notified of events generated by publishers. An important and popular publish-subscribe middleware is the Data Distribution Service for Real-Time Systems (DDS) that has been defined by the Object Management Group (OMG) to provide a standard data-centric publish-subscribe specification for distributed systems (OMG, n.d.-a).

It appears that DDS has been applied to different domains including development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. In addition, various different DDS implementation approaches have been proposed in the literature. In this context, various papers have been written on the application of DDS each one addressing particular problem. However, there has been no attempt to systematically review and categorize the obstacles with respect to the application of DDS. The overall objective of this chapter is to identify the state of the art of DDS, and describe the main lessons learned and obstacles in applying DDS. In addition, we aim to identify the important open research issues. In this context we have conducted a systematic literature review (SLR) (Kitchenham et al., 2009; Kitchenham & Charters, 2007) by a multiphase study selection process using the published literature since the introduction of DDS in 2003. We reviewed 468 papers that are discovered using a well-planned review protocol, and 34 of them were assessed as primary studies related to our research questions. Our study shows that DDS can provide important benefits for realizing real-time distributed applications in various domains. We have identified 11 basic categories for describing the identified obstacles and the corresponding research challenges that can be used to depict the state-of-the-art in DDS and provide a vision for further research.

The results of our study can be of benefit for both practitioners and researchers. Practitioners who are interested in applying DDS can use the result of the SLR as a roadmap for finding and analyzing the relevant approaches together with the lessons learned and decide about their applicability. For researchers the results of our study provide an overview of the reported DDS approaches together with the lessons learned, obstacles and research challenges in the DDS domain. As such, the information extraction scheme we used to characterize the study context and study findings can be used to guide the research activities of future studies in the DDS domain.

The remainder of this chapter is organized as follows. Section 5.2 provides a short background on DDS middleware. Section 5.3 describes the adopted research method used in this study. Section 5.4 presents the results of the SLR. Section 5.5 presents the discussion and finally section 5.6 concludes this chapter.

5.2 DATA DISTRIBUTION SERVICE

In this section we describe the background for understanding and supporting the approach that we present in this chapter. Detailed information on DDS can be found in (OMG, n.d.-a, n.d.-b, n.d.-c). Based on these Figure 5-1 shows the conceptual model for the DDS specification that is adapted from the DDS specification (OMG, n.d.-a).

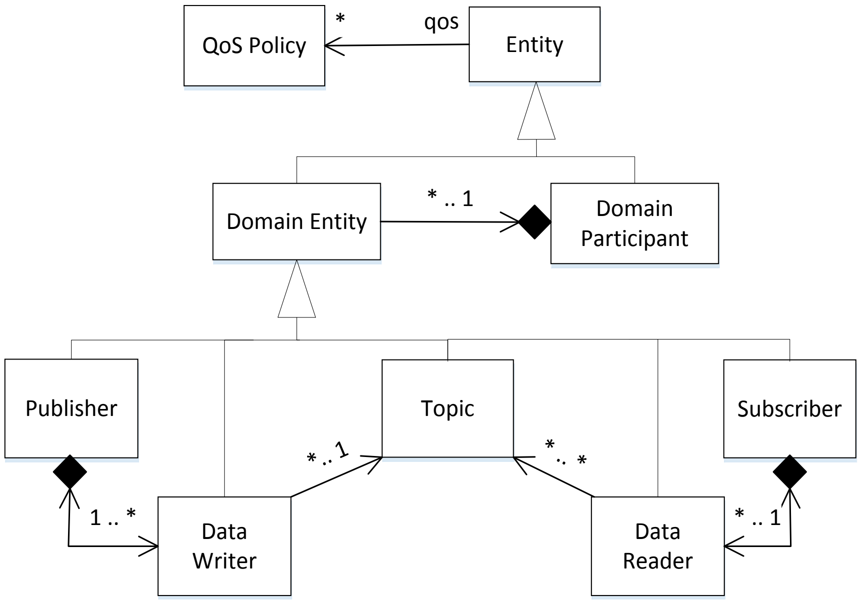


Figure 5-1: Reference architecture for DDS-based systems (adopted from (OMG, 2015b))

In the DDS specification Domain is a logical concept which represents the set of applications that can communicate with each other. Within the same DDS system multiple domains can be defined indicating different sets of applications that communicate with each other. Fig. 1 shows the concepts related to a domain. A domain includes one or more Domain Participants which represent the local membership of the application in the corresponding domain. Domain Participant may participate in more than one domain at the same time. Each Domain Participant may include one Publisher and one Subscriber. Publisher represents the objects responsible for data production and updates. A publisher includes one or more Data Writers that publish data of different data types. Subscriber is responsible of receiving published data and making it available to the participant. A subscriber includes one or more Data Readers to access published data in a type-safe manner. The communication between data readers and data writers is established via Topics. A topic defines a unique name, data type and a set of Quality Services to the published/subscribed data. DDS provides the ability to attach Quality of Service (QoS) parameters to all these entities in order to specify the behavior of a service. Examples of these QoS parameters are the rate of publication, rate of subscription, how long the data is valid, etc. Applications communicate with each other based on topics. Communication between applications can only be realized only if the topic names and the defined QoS parameters match.

The conceptual model of Figure 5-2 defines the so-called Data Centric Publish/Subscribe (DCPS) part of the DDS specification which is mandatory for DDS implementations. In addition to DCPS the DDS specification also defines the Data Local Reconstruction Layer (DLRL) which is an optional layer that may be built on top of the DCPS layer. The purpose of the DLRL layer is to provide a seamless integration with object-oriented language constructs. Finally, an additional specification DDS Interoperability Wire Protocol is provided, which is needed for supporting the interoperability among different DDS implementations. For further details about these specifications we refer to OMG DDS Specifications (OMG, 2015a).

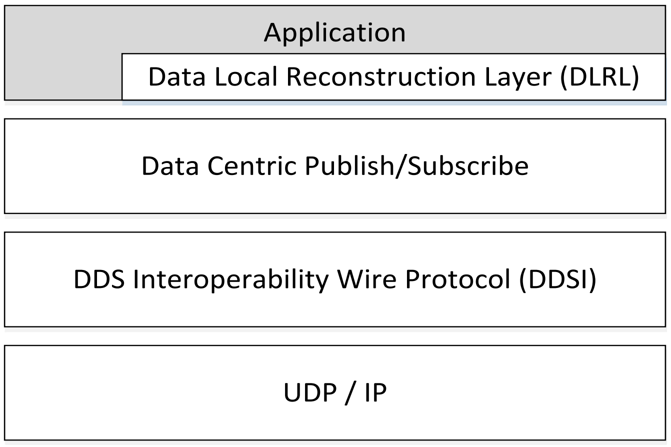

Application
Data Local Reconstruction Layer (DLRL)
Data Centric Publish/Subscribe
DDS Interoperability Wire Protocol (DDSI)
UDP / IP

Figure 5-2: Layered architecture of the DDS with the DDS specifications ((OMG, 2015a))

5.3 Research Method

The overall objective of this chapter is to identify the state of the art of DDS, and describe the main lessons learned and obstacles in applying DDS. For this we will apply a systematic literature review (SLR) or systematic review for short which is a well-defined and rigorous method to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest (Kitchenham et al., 2009; Kitchenham & Charters, 2007). The goal of an SLR is to give a fair, credible and unbiased evaluation of a research topic using a trustworthy, rigorous and auditable method. The inception of systematic reviews is based on the evidence-based concept which is devised in the field of medicine (Dybå et al., 2005). The success of evidence-based medicine has triggered many other disciplines to adopt a similar SLR approach, including for example psychiatry, nursing, social policy, and education. Similarly, the concept of evidence-based software engineering (EBSE) has been introduced together with guidelines for performing systematic literature reviews in software engineering. There are several reasons for undertaking a systematic literature review including summarizing the existing evidence concerning a treatment or technology, identifying any gaps in current research in order to suggest areas for further investigation, providing a framework/background in order to appropriately position new research activities, examining the extent to which empirical evidence supports/contradicts theoretical hypotheses, or assisting in the generation of new hypotheses.

The goal of EBSE is to improve the quality of software-intensive systems and provide insight to stakeholder groups whether practitioners are using best practice or not. In our study we aimed at identifying the obstacles regarding the DDS concepts. Different approaches have been presented in the literature for conducting SLRs in different domains. We followed the complete guidelines for performing SLRs as proposed by Kitchenham and Charters (Kitchenham & Charters, 2007). In the following subsections we discuss the applied research method that is based on an extensive review protocol.

5.3.1 Review Protocol

Before actually conducting the review we first defined the review protocol (Kitchenham et al., 2009). A review protocol describes the methods that will be used to carry out a specific SLR. The basic activities of the adopted review protocol are shown in Figure 5-3.

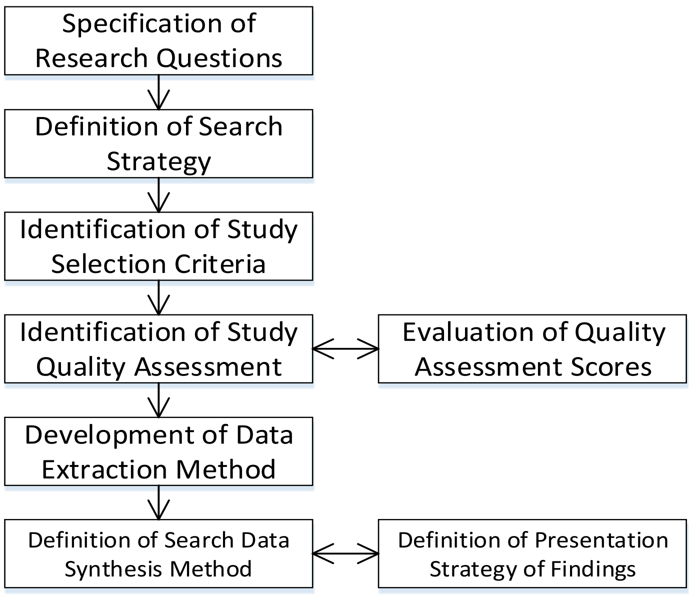

Specification of Research Questions
Definition of Search Strategy
Identification of Study Selection Criteria
Identification of Study Quality Assessment
Evaluation of Quality Assessment Scores
Development of Data Extraction Method
Definition of Search Data Synthesis Method
Definition of Presentation Strategy of Findings

Figure 5-3: Activities under the review protocol

Firstly, we specified our research questions based on the objectives of this systematic review. After this step we defined the search scope and the search strategy. The search scope defines the time span and the venues that we looked at. In the search strategy we devised the search strings that were formed after performing deductive pilot searches. A good search string brings the appropriate search results that will come to a successful conclusion in terms of sensitivity and precision rates. Once the search strategy was defined, we specified the study selection criteria that are used to determine which studies are included in, or excluded from, the systematic review. The selection criteria were piloted on a number of primary studies. We screened the primary studies at all phases on the basis of inclusion and exclusion criteria. Also, peer reviews were performed by the authors throughout the study selection process. The process followed with quality assessment in which the primary studies that resulted from the search process were screened based on quality assessment checklists and procedures. Once the final set of preliminary studies was defined the data extraction strategy was developed which defines how the information required from each study is obtained. For this we developed a data extraction form that was defined after a pilot study. In the final step the data synthesis process takes place in which we present the extracted data and associated results.

5.3.2 Research Questions

The most important part of any systematic review is to clearly and explicitly specify the research questions. Research questions drive the subsequent parts of the systematic review. Hence, asking the right question is crucial to derive the relevant findings properly. The more precise the research questions are, the more accurate the findings will be. In this context, research questions need to be meaningful and important to both practitioners and researchers. As was previously stated no systematic review has been carried out yet on DDS-based systems. Our particular aim in this study is the identification of obstacles and lessons learned, and the future research directions in the domain of DDS. In accordance with these objectives our primary research question can be concretely formulated as follows:

RQ 1. What are the identified obstacles in the DDS domain?

RQ 2. What are the solution directions for the identified obstacles?

5.3.3 Search Strategy

To answer the research question as defined in the previous section we have conducted an extensive search of papers. In the following we describe the scope of the search, the applied research method and the search string.

Our search scope included the papers that were published over the period of January 2003 and December 2015. The main motivation for 2003 was that DDS was introduced by OMG in that year. We searched for papers in selected venues that publish high quality papers. We used the following search databases: IEEE Xplore, ACM Digital Library, Wiley Inter Science, Science Direct and Springer. These venues are listed in Table 5-1.

Table 5-1: Searched publication sources

Characteristics	No. of Included Studies after applying search query	No. of Included Studies after exclusion criterion
IEEE Xplore	299	20
ACM	78	5
Wiley	2	0
Science Direct	87	9
Springer	2	0
Total	**468**	**34**

Our targeted search items were journal papers, conference papers, workshop papers, and white papers. To search the selected databases, we used both manual and automatic search. Automatic search is realized through entering search strings on the search engines of the electronic data source. Manual search is realized through manually browsing the conferences, journals or other important sources.

In our case finding search strings appeared not to be difficult due to the unique concepts in DDS. The search string “DDS” resulted in many different studies that were not relevant to our study (such as digital data synthesizer). As such we chose not to use the acronym but the full name of the DDS standard. Hence, we used the search String “data distribution service” that we actually used for searching in all the listed venues. Since there are no synonyms for data distribution service this search string appeared to be strong enough to identify all the relevant primary studies. In addition to these automated searches we also conducted manual searches both as a preliminary analysis and as a subsequent analysis after having observed the publication channels returned by the search string. The manual searches appeared to be quite useful since we retrieved some good-quality articles that an automatic search could not reveal. The result of the overall search process after applying the search queries and the manual search is shown in the second column of Table 5-1. As it can be seen from the table we could identify 468 papers at this stage of the search process.

5.3.4 Study Selection Criteria

In accordance with the SLR guidelines (Kitchenham & Charters, 2007) we further applied exclusion criteria on the large-sized sample of papers in the first stage. The overall exclusion criteria we used are as follows:

- EC 1: Abstracts or titles that do not mainly discuss the provision of DDS
- EC 2: Abstracts or titles that do not propose an approach to DDS
- EC 3: Papers where the full text is not available
- EC 4: Duplicate publications found in different search sources
- EC 5: Papers written in different language than English
- EC 6: Papers that do not explicitly relate to or discuss DSS
- EC 7: Papers which are experience and survey papers
- EC 8: Papers that discuss only the application of DDS and do not critically reflect on the DDS concepts

The exclusion criteria were checked by a manual analysis. After applying the exclusion criteria 34 papers of the 468 papers remained.

5.3.5 Study Quality Assessment

In addition to adopting general inclusion and exclusion criteria we also assessed the quality of the resulting primary studies. Study quality has no widely-accepted definition, and usually the quality evaluation approach consists of a set of questions for assessing the quality of the selected primary studies. In this context, we adopted the summary quality checklists that are proposed in (Kitchenham & Charters, 2007). We thoroughly reviewed the list of questions in the context of our review and selected the ones that are aligned with our research questions. The quality checklist is shown in Table 5-2. The quality items in the instrument are deployed on a numerical scale because we intended to rank and classify the studies with respect to an overall quality score. Therefore, we preferably employed a three-point scale (i.e. “yes” = 1, “somewhat” = 0.5, “no” = 0) during the assessment. We selected this approach in alignment with the review protocol of Kitchenham et al (Kitchenham et al., 2009) and similar SLRs. The result of the assessment is provided in Appendix B – Assessment of Primary Studies. We used the outcomes of quality assessment stage in order to assist data analysis and synthesis. We examined whether quality differences are correlated with the results reported in different kinds of primary studies.

Table 5-2: Quality Checklist

	Questions
Q1	Are the aims of study clearly defined?
Q2	Are the scope, the context and the experimental design of the study clearly stated?
Q3	Does the report have implications for research and/or practice?
Q4	Are the variables used in the evaluation likely to be valid and reliable?
Q5	Are the measures used in the study quite explicit & aligned with the research aims?
Q6	Is the research process documented adequately?
Q7	Are the main findings stated clearly in terms of validity and reliability?
Q8	Is there an explicit statement of the limitations?

5.3.6 Data Extraction

In order to precisely extract and record the data retrieved from each of the 34 primary studies both authors read the full-texts of these studies. The information needed to address our research question and study quality criteria was collected by means of a data extraction form. Actually, when the study review protocol became definite, the data extraction form was composed in order to reduce the tendency to bias. Since we considered the quality assessment stage as part of the data analysis, the information collected for both the quality criteria and the review data was kept in the same form. The data extraction form was piloted by both of the two researchers in consensus meetings so as to be consistent in subsequent analysis. After independent data extraction, data from both researchers were compared and disagreements were resolved by consensus. Basically, the data extraction form first included standard information such as name of the reviewer, date of data extraction, study ID, title, authors, journal, publication details, a brief summary and space for additional notes. Later on, the form was extended to cover the data directly related to answering the research question and for supporting the search for obstacles of applying DDS. Some of the fields were: publication details, study aim, targeted domain, study settings, DDS solution used, research method used, assessment approach, findings, constraints/limitations, implications for future research and major conclusions. We recorded the places where the extracted information existed within the primary studies in spread sheets. In order to support the process of synthesizing the extracted data, the form was developed in a progressive way so that the transition was performed seamlessly.

5.3.7 Data Synthesis

Data synthesis is the process of collating and summarizing the extracted data in a manner suitable for answering the questions that an SLR seeks to answer. At this stage, we performed a qualitative and quantitative analysis separately on the data extracted from the reviewed papers. We investigated whether the qualitative results can lead us to explain quantitative results. For example, a primary study involving an assessment of DDS technology could help interpret other solutions quantitatively. We made use of tabular representation of the data when feasible, and it enabled us to make comparisons across studies. Also, using the quantitative summaries of the results, we inferred the implications for future search, and consequently the existing research directions within the DDS domain.

5.4 Results

5.4.1 Overview of Selected Studies

The list of primary studies that we have identified is listed in Appendix A – Primary Studies for Deriving Characteristics for DDS.

An overview of the primary studies according to publication channel is shown in Table 5-3. The table shows the publication channels, the types of articles and the number of studies that fall into the channels accordingly. The majority of the papers have been published in conference papers. The journals, Journal of Systems and Software and Computer Standards and Interfaces appeared to be the journals with the highest number of papers.

Table 5-3: Distribution of studies in terms of publication channel and occurrence

No	Publication channel	Type	No. of studies
1	Journal of Systems and Software	Journal	4
2	Computer Standards and Interfaces	Journal	3
3	Int. Symposium on Computer and Communications	Symposium	3
4	Distributed Event Based Systems	Conference	2
5	IEEE Consumer Communications and Networking Conference	Conference	2
6	Intelligent Solutions in Embedded Systems	Workshop	2
7	Int. Conference on Big Data and Smart Computing	Conference	2
8	Int. Conference on Generative Programming & Component Engineering	Conference	2
9	Conference on Information Networking	Conference	1
10	Distributed Simulation and Real-Time Applications	Conference	1
11	EUROMICRO Conf. on Software Engineering & Advanced Applications	Conference	1
12	European Conference on Software Architecture Workshops	Workshop	1
13	IEEE Conference on Emerging Technologies and Factory Automation	Conference	1
14	IEEE Congress on Services	Conference	1
15	IEEE Int. Conference on Distributed Computing Systems Workshops	Conference	1
16	IEEE Int. Symposium on Parallel and Distributed Processing	Symposium	1
17	Int. Conference on Systems and Informatics	Conference	1
18	Int. Symposium on Dependable Computing	Symposium	1
19	Int. Workshop on Future Trends of Distributed Computing Systems	Workshop	1
20	Journal of Parallel and Distributed Computing	Journal	1
21	Space Mission Challenges for Information Technology	Conference	1
22	The Journal of China Universities of Posts and Telecommunications	Journal	1

In Figure 5-4 we present the year-wise distribution of the primary studies along with the venues that they were published in.

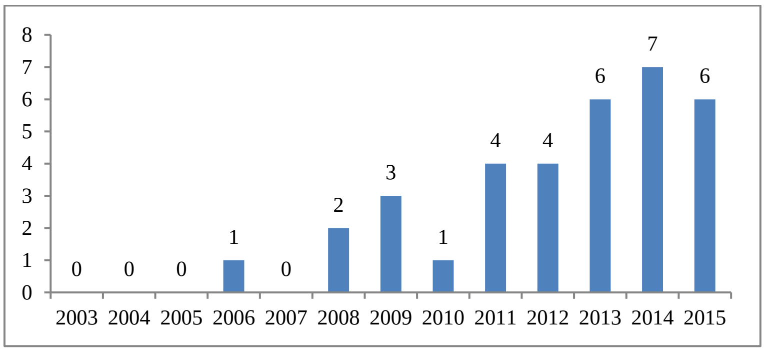


Figure 5-4: Year-wise distribution of number of primary studies

5.4.2 Research Methods

It is very important that the primary studies explicitly define the used research methodology. By analyzing and assessing the studies' reported approaches, we can draw conclusions about the strength of evidence within them. Table 5-4 provides the list of research methods used in the selected 34 primary studies. There are 6 types of research methods that we looked for in the review. The numbers in the table reveal that almost all of the primary studies are based on single case study except studies B and R. Study [B] performed a benchmark for 3 different DDS vendors including RTI, OpenSplice and OpenDDS, the authors only published the results of RTI DDS (Connext). Study [R] compares just OpenSplice and RTI DDS.

Table 5-4: Studies by research methods

Research Method	Studies	Number	Percent (%)
Not described/Descriptive	-	0	0
Single case	A, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, S, T, U, V, W, X, Y, Z, AA, AB, AC, AD, AE, AF, AG, AH	32	94.1
Multiple case	-	0	0
Survey	-	0	0
Experiment	-	0	0
Benchmarking	B, R	2	5,9

5.4.3 Methodological Quality

It should be noted that a systematic literature review is a methodologically rigorous review of research results. For this purpose, using the quality checklist as defined in Table 5-2, we tried to address methodological quality in terms of rigor, credibility and relevance together with reporting quality. All in all, we dedicated the first two questions of the table for the quality of reporting, the third and fourth question for relevance, the fifth and sixth questions for rigor and the last two questions for assessing the credibility of evidence. Figure 5-5 shows the histogram of reporting quality results that has been defined based on the first four questions and the values in Appendix B – Assessment of Primary Studies. The figure indicates that most of the primary studies (88.2%) are good according to reporting quality.

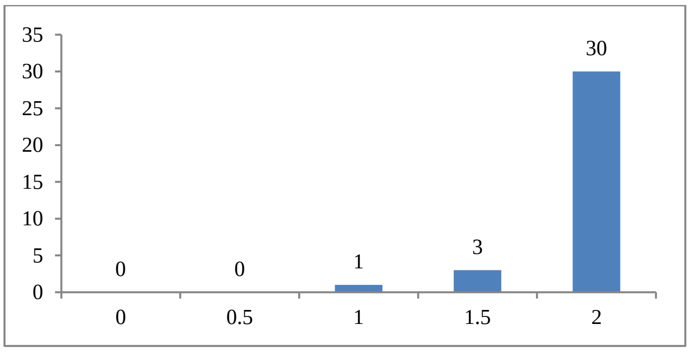


Figure 5-5: Reference Reporting quality of the primary studies

Figure 5-6 shows the relevance quality scores that are based on the evaluation of the third and fourth questions focusing on the assessment of the relevance of our primary studies.

58.8% of the studies (20 studies) were found to be directly relevant to the field, and 23.5% (8 studies) of them were considered relevant to some extent.

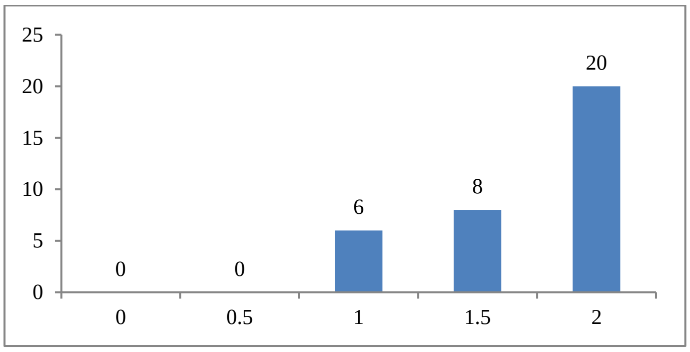


Figure 5-6: Relevance quality of the primary studies

We also assessed the rigor of studies and the trustworthiness of the findings. Figure 5-7 denotes the rigor of the research methods employed on a scale from 0 to 2. Considering the scores 1.5 and 2 as first-rates, 28 of the primary studies (%82.4) established the validity of their findings in a proper form. Eight studies are of top quality in terms of rigor.

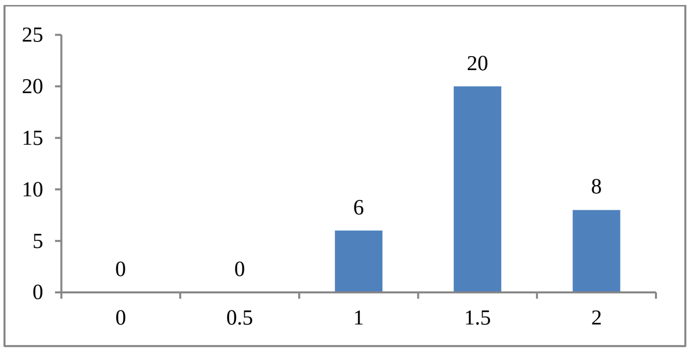


Figure 5-7: Rigor quality of the primary studies

Our last two criteria were intended for the credibility of evidence that is the extent to which the findings and the major conclusions of the primary studies are profoundly clear, valid and suggestive. Figure 5-8 shows the histogram of quality scores based on credibility of evidence. Five studies given in Figure 5-8 got the highest score in this rating scale, having reasonably valid and meaningful findings and corresponding conclusions.

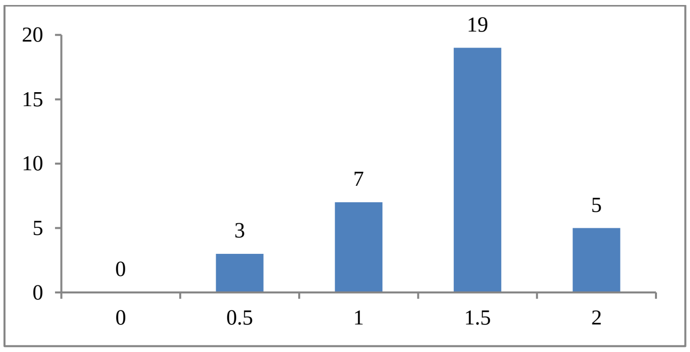


Figure 5-8: Credibility of evidence of the primary studies

Consequently, we can now finalize the overall methodological quality scores. Figure 5-9 shows the total of quality scores in terms of four criteria: reporting, relevance, rigor and credibility of evidence. 27 of the studies (79.4%) having scores equals or greater than 6 are relatively good, and two studies are at the head of this group being high quality. 7 studies having scores less than 6.0 are considered to be of poor quality. In conclusion, the histogram shows that the majority of the primary studies were assessed to be good to a large extent.

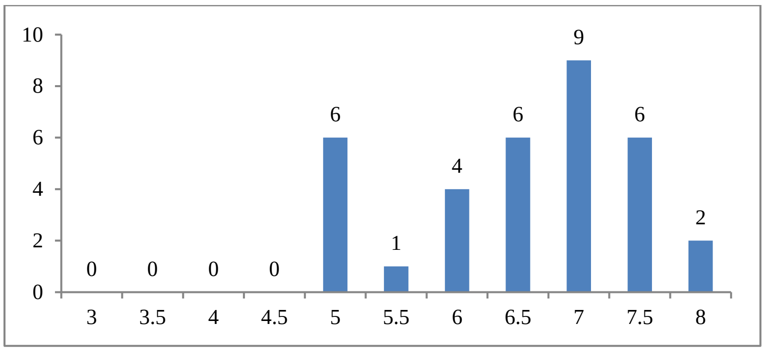


Figure 5-9: Overall quality of the primary studies

5.4.4 Systems Investigated

This section outlines the results we extracted related to three main research questions. We present the data extracted from the primary studies in the form of findings, separately for each research question.

1) RQ 1. What are the identified obstacles in the DDS domain?

The names of the identified problems are given in Table 5-5. In the following Table 5-6 we discuss the problems and solutions that we derived from the selected primary studies. The overview of the identified 11 problems is given in. The first column presents the identified studies, the second column the date of publication of the primary study; the remaining columns refer to the identified problems (P1 to P11). The description of each problem is shown at the right of the Table 5-5.

Table 5-5: Identified Problems

Name of the Problems	
P1	Complexity of DDS configuration
P2	Performance prediction, measurement & optimization
P3	Implementing DDS
P4	DDS integration over WAN
P5	DDS using wireless networks & mobile computing
P6	Interoperability among DDS vendor implementations
P7	Data consistency in DDS
P8	Reliability in DDS
P9	Scalability in DDS
P10	Security
P11	Integration with Event Based Systems

Table 5-6: Primary studies with identified problems of DDS

Study	Year	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
A	2006							X				
B	2008		X									
C	2008					X						
D	2009								X			
E	2009					X						
F	2009					X	X					
G	2010								X	X		
H	2011								X			
I	2011		X				X			X		
J	2011	X		X								
K	2011	X		X								
L	2012					X						
M	2012		X			X						
N	2012					X						
O	2012		X									
P	2013				X		X					
Q	2013							X				
R	2013		X									
S	2013	X	X									
T	2013		X		X							
U	2013	X		X								
V	2014		X				X			X		
W	2014	X										
X	2014		X				X			X		
Y	2014				X							
Z	2014				X							
AA	2014				X							
AB	2014										X	
AC	2015		X									
AD	2015									X		X
AE	2015		X				X			X		
AF	2015		X									
AG	2015					X						
AH	2015											X
Total: 52		**5**	**12**	**3**	**5**	**7**	**6**	**2**	**3**	**6**	**1**	**2**

To discuss each identified problem in detail we further adopt feature diagrams (Czarnecki, Kim, & Kalleberg, 2006) to provide an overview of the identified sub-problems and the addressed solutions for the given problem category. A feature diagram is a tree with the root representing a concept (e.g., a software system), and its descendent nodes are features. Feature diagrams show both the mandatory and the variant features and, in a sense, can be used to support ontological modeling of a domain. Variant features are usually represented as optional or alternative features. A feature configuration is a set of features which describes a member of an SPL. A feature constraint further restricts the possible selections of features to define configurations. The most common feature constraints are "requires" and "mutex" relations. In our overview the root node represents the problem category, while the features represent the sub-problems, and optionally the sub-sub-features define the possible solutions, if these were described. The overall legend (abstract syntax) to be used for modeling the problem categories together with their sub-problems and possible solution directions is given in Figure 2-3. In the following we discuss each problem separately.

P1. Complexity of DDS configuration

Although there exists an OMG's specification to deploy and configure DDS based systems (OMG's Deployment & Configuration Specification (OMG, 2006a)) several authors have indicated the difficulty of configuring DDS before it can be installed and used. The configuration is usually not trivial and requires lots of time and effort due to various reasons such as the many involved parameters, the complex interactions among the parameters and the need for writing glue code. In fact, it has been shown that about 80% of DDS-related code is associated with configuring the middleware. According to Otto et al. (Otte, Gokhale, Schmidt, & Willemsen, 2011) over half of the DDS API that the developers must learn is configuration related. Different solutions have been provided to cope with the complexity and support the configuration of DDS.

In [J] the authors propose the automatic configuration of DDS middleware to reduce time and effort of the configuration. For this, they propose a component-based approach and separate concerns between configuration-based aspects of DDS application development and configuration aspects. In this way the configuration of the DDS is not limited to source code level configuration but can also be applied at deployment time. To realize this, the authors provide an implementation of the OMG's DDS4CCM (OMG, 2006b) which integrates component-based DDS application development with the OMG's Lightweight CORBA Component Model (LwCCM) (OMG, 2012b). They use C++ templates to generate the configuration and the glue code.

Study [S] proposes to adopt model-driven engineering and generative programming to reduce the manual effort in generating a large number of relevant QoS configurations that can be deployed and tested on a cloud platform. The study proposes a domain specific modeling language (DSML) that supports modeling a DDS application for emulation and testing its performance for various combinations of DDS QoS policies.

Study [K] provides the so-called ServiceDDS a DDS-based framework that combines different standard technologies to allow real-time heterogeneous participants to interact dynamically in distributed peer-to-peer architectures. ServiceDDS uses DDS to support dynamic distributed interaction, XMPPP to provide Web access, and RTSJ as a real-time platform. The framework uses service-topics as an abstraction and refinement of the standard DDS entities model. Service-topics can interact with the DDS standard topics and because of the higher abstraction level the complexity is better managed and the configuration and development time can be reduced.

Study [U] presents the integrated CCM (iCCM), a framework for integrating DDS into the CORBA Component Model (CCM), which is a standard-based programming model for implementing component-based DRE systems. The framework does not need any modifications to either the CCM or DDS specifications. Using the framework, the system developers adopt a component-based development and are able to abstract away the low-level implementation details of DDS. In this way reuse is promoted and the deployment and configuration complexities are managed.

Study [W] provides the tool Deploy-DDS that supports the deployment configuration of DDS modules to the physical resources. The tool supports the selection and generation of deployment architectures of DDS based systems, and as such can be used to perform an evaluation during the design phase.

To sum up, the corresponding solutions to the complexity of DDS configuration those are proposed in the identified primary studies include separation of configuration concerns from application concerns, abstraction of lower level source code to components or services, and the use of model-driven development approaches to automatically generate the parameter values and the code.

Figure 5-10 shows the problem feature diagram for this configuration of DDS problem.

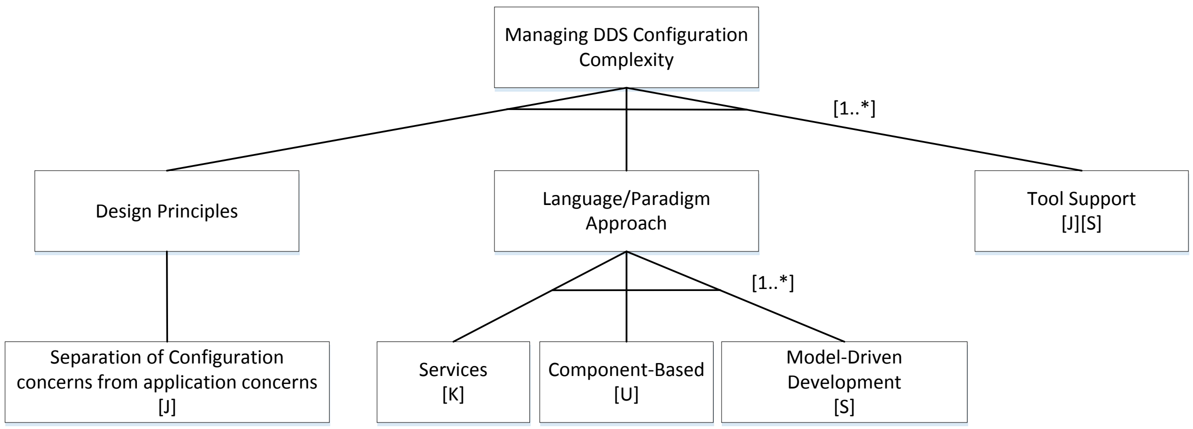

Managing DDS Configuration Complexity
[1..*]
Design Principles
Language/Paradigm Approach
Tool Support [J][S]
[1..*]
Separation of Configuration concerns from application concerns [J]
Services [K]
Component-Based [U]
Model-Driven Development [S]

Figure 5-10: Feature diagram for P1 – Managing DDS configuration complexity

P2. Performance prediction, measurement and optimization

Performance is one of the important quality factors for DDS based systems that can be addressed at different levels including design, implementation and operation of the system. An important issue is to predict the performance of the DDS based system before its implementation. This is because after the DDS has been implemented it is very difficult or costly to adapt the system. After the implementation of the DDS measurement of the performance is considered important to meet the quality of service requirements.

Study [S] focuses on the problem of performance prediction in the presence of diverse combinations of QoS configurations. The authors claim that existing design-time formal methods are limited in providing sufficient accuracy in prediction, tool support and the understandability of the adopted design formalisms. They propose an approach in which the system behavior is emulated and data on the QoS parameters is gathered by experimentation. They provide a model-based performance testing framework to generate a large number of QoS configurations that can be deployed and tested on a cloud platform.

Study [I] offers a bloom filter (BF) algorithm as an alternative to the Simple Discovery Protocol (SDP) algorithm, which is the default DDS network discovery protocol for Real-Time Publish

Subscribe (DDS-RTPS). In the SDP each participant sends its endpoint data to every participant in the domain, and on its turn receives endpoint data from all the other participants. In case of a large network with thousands of endpoints, numerous discovery messages from all participants in the same domain will be required. However, not all date is needed, and the messages sent to uninterested endpoints would be wasted. Moreover, this process will also require a large amount of memory to store all of the data in such cases. The adoption of the BF algorithm helps to optimize the network discovery process and likewise decreases the network and resource consumptions.

An extension of the Study [I] is Study [X] which proposes the usage of the SDPBloom algorithm as an alternative to BF algorithm to further optimize the network discovery process. The SDPBloom algorithm eliminates the duplicate data in Bloom Filters and ensures that each data has only one representative key. The proposed algorithm provides better results than BF. Although both Study [X] and Study [I] optimize the network discovery process these can impede the interoperability since it deviates from the default implementation, that is, SDP of the RTPS. This issue is considered as further research.

Study [AE] is another extension of Study [I]. This work uses another node discovery scheme based on Parallel Dynamic Bloom Filters, namely SDP-Parallel DBF for the same purpose. The proposed algorithm provides better delay time, number of messages and message size in network by reducing unnecessary delay time spent restructuring Bloom Filters. Furthermore, SDP-Parallel DBF offers computational speed-up through parallelization.

Study [M] provides an evaluation of the performance of the DDS-based middleware in the wireless environment. For this several experiments have been carried out in the wired and wireless LAN. Based on the experimental results a new bandwidth-aware design scheme is proposed in which separate uplink and downlink communication model is used depending on the number of Pubs and Subs. In this way, the performance of the DDS communication in WLAN is enhanced.

Study [B] presents an assessment of the strengths and weaknesses of four commercial DDS implementations deployed on an unmanaged setting as that in an enterprise setting that is often characterized by an inter-administrative geographic scale, shared network channels, and heterogeneous with unpredictable quality parameters such as end-to-end latency and load. The study shows that if the application manages a small number of homogeneous resources, this middleware performs timely and reliably. However, in a more general setting with fragmentation and heterogeneous resources, reliability and timeliness rapidly degenerate. The authors suggest self-adopting and self-configuration capability, efficient event routing primitives, and management of heterogeneous resources.

Study [T] addresses the problem of assuring end-to-end quality-of-service (QoS) in Wide Area Network (WAN) based distributed real time (DRE) applications. This is hard because the end-system QoS mechanisms must work across different access points, inter-domain links, and within network domains. Although DDS is widely used in the design of real time distributed systems because of its explicit consideration of QoS parameters it does not provide any mechanism for assuring end-to-end QoS of DRE systems. Moreover, it lacks mechanisms that holistically schedule different resources to realize end-to-end QoS. To address these problems, the paper presents an approach to enhance the DDS by providing (1) an approach for analyzing DDS scheduling capabilities to deliver DDS samples on an end-system using a performance model, and (2) a policy-based framework called Velox to provide end-to-end QoS provisioning for DDS based applications by controlling network resources, such as a bandwidth and end-to-end delay.

Study [O] analyses DDS from the schedulability point of view. The authors focus on how DDS aims to guarantee the real time behavior through the mechanisms of the standard. For the analysis concepts defined in the Modeling and Analysis of Real-Time and Embedded systems (MARTE) standard (OMG, 2008) has been into DDS. The authors conclude that the DDS is suitable to perform the schedulability analysis but does not deal with issues to develop DRE systems such as thread scheduling, which are left to the implementation. Based on the analysis the authors propose extensions to DDS including the addition of an interface to select among different scheduling policies available in the system as well as the definition of new QoS parameters to allow the configuration of schedulable entities through the assignment of specific scheduling parameters.

Study [R] provides a performance comparison of two different DDS implementations including OpenSplice and RTI. The study reports on the advantages and disadvantages of both implementations in terms of data delivery performance, CPU usage, and memory resource usage. Based on the experimental study the authors provide guidelines related to the performance issues in centralized, decentralized and federated DDS implementations. The centralized appears to have the simplest architecture but has to cope with the risk of single point of failure. Both OpenSplice and RTI use a distributed implementation, whereby OpenSplice uses a federated approach and RTI a decentralized approach. The study indicates that federated approaches perform better if the data size is small, but for larger data sizes the decentralized approach seems to have better results with respect to CPU and memory usage. Likewise, it is concluded that decentralized DDS approaches provide better scalability in case of increased data sizes.

In Study [V] the authors focus on the standard “wire protocol” that allows DDS implementations from multiple vendors to interoperate. The authors claim that the adopted Simple Discovery Protocol (SDP) in the standard wire protocol (OMG, 2014) is resource consuming for large scale systems. As such they propose the so-called Content-based Filtering Discovery Protocol (CFDP) for which they also describe a prototype implementation. Furthermore, empirical studies are presented which show that, compared to SDP, the proposed CFDP is more efficient in large scale systems in terms of computing, CPU and network usage.

Study [AC] provides a simulator for performance evaluation of large scale network systems using DDS. The DDS simulator enables developers to measure basic metrics such as discovery completion time, end-to-end message transfer delay, the number of data messages per Domain Participant, and user data processing time. The simulator supports the analysis of the strengths and weaknesses of DDS implementations from different vendors.

Study [AF] proposes “Rateless Code based Reliable Multicast (RCRM)” scheme instead of “Automatic Repeat reQuest (ARQ) based error control in DDS. The proposed RCRM scheme provides higher reliable multicast efficiency since ARQ based error control is effective if only channel condition is moderately good. Furthermore, RCRM enhances the rateless coding performance by using a heuristic algorithm to reduce the computational complexity. This heuristic algorithm encodes transmit packages of publishers with a rateless code improving throughout performance. When using rateless code, the subscribers send only one feedback instead of sending ACK for each encoded packet. This approach is very efficient with respect to ARQ based error control in which simultaneous feedback consumes network resources quickly.

In sum, the identified primary studies indicate that it is difficult to predict and measure the performance of DDS-based systems due to the many different QoS parameters and related to this the large number of possible configurations. For performance prediction the studies propose to adopt emulation and experimentation to predict the QoS values for different configurations. For performance measurement the identified solutions include the adoption of explicit frameworks, benchmarking and tool support. Finally, for performance optimization the identified solutions include the adoption of new discovery protocol algorithms, explicit scheduling mechanisms, and new communication model.

The problem feature diagram for this problem is shown in Figure 5-11.

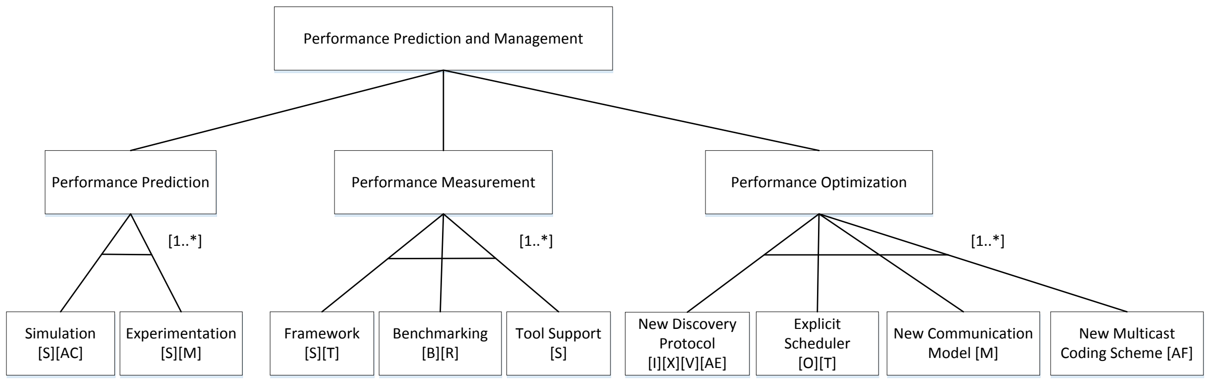


Figure 5-11: Feature diagram for P2 – Performance prediction, measurement and optimization

P3. Implementing DDS

DDS is a specification that was introduced after CORBA. Similar to CORBA it deliberately does not provide an implementation, and likewise the implementation of the DDS is left to the vendors. The DDS specification includes different compliance profiles including minimum profile, content-subscription profile, persistence profile, ownership profile, and object model profile. The minimum profile contains just the mandatory features of the DCPS layer. The content-subscription profile, the persistence profile and the ownership profile include optional features for the DCPS layer. The object model profile includes feature for the Data Local Reconstruction Layer (DLRL).

Different vendors have provided different profile implementations of the DDS. In addition, several studies have discussed the challenges and the approaches for providing the implementations of DDS. Table 5-7 shows the DDS implementation profiles of selected DDS vendors. These vendors have been published on the OMG DDS portal site (OMG, n.d.-c). Four of these including OpenDDS of OCI, Vortex (Open Splice) of Prism Tech, Connext of RTI, and CoreDX of Twin Oaks are also referred to in the primary studies that we have identified. The table characterizes the vendors with respect to the features that are defined by different OMG DDS specifications.

Table 5-7: Implemented Profiles with respect to the selected DDS VENDORS

DDS Vendors	**DDS Product**	**Type**	**Compliance to DDS Specification**				
			DDS Profile	**DDSI**	**Web**	**Security**	**X-Types**
Gallium	Intercom DDS	Commercial	Minimum Profile	No	No	No	No
MilSOFT	MILSOFT DDS	Commercial	Full DCPS	Yes	No	No	No
Ocera Orte	Ocera	Commercial	No	Yes	No	No	No
OCI	OpenDDS	Open Source	Full DCPS	Yes	No	No	No
PrismTech	Vortex	Commercial	Full DCPS + DLRL	Yes	Yes	Yes	Yes
RTI	Connext DDS	Commercial	Full DCPS	Yes	Yes	Yes	Yes
Twin Oaks	CoreDX	Commercial	Minimum Profile & Ownership Profile	Yes	No	No	No

In Study [J] the authors propose to apply a component-based implementation of DDS to increase the abstraction level of the implementation and for addressing the deployment and configuration requirements of modern distributed real-time and embedded systems (DRE). In this way the authors aim to reduce the need for implementing large amounts of boilerplate glue code that is necessary for the configuration of the DDS. They adopt the OMG's DDS for Lightweight Common Component Model CCM (DDS4CCM) specification and describe the design and implementation of the so-called DDS4CIAO which addresses a number of inherent and accidental complexities in the DDS4CCM standard. To address the accidental complexities of the implementation of DDS4CCM they make use of several approaches including extensible interface patterns in the form of mixins, template-driven code generation techniques, and customization techniques.

Study [U] presents the integrated CCM (iCCM), a framework for integrating DDS into the CORBA Component Model (CCM), which is a standard-based programming model for implementing component-based DRE systems. The framework does not need any modifications to either the CCM or DDS specifications. Using the framework, the system developers adopt a component-based development and are able to abstract away the low-level implementation details of DDS. Likewise, reuse is promoted and developers can focus more on the business-logic of the application.

As stated in P1, Study [K] provides ServiceDDS which supports the implementation of DDS services using so-called service-topics that provide an abstraction and refinement of the standard DDS entities model. The abstraction using service topics eases the implementation of the DDS and enables the additions of new features. In this context the authors claim that implemented service topics can interact with the DDS standard topics combining different standard technologies (such as web-access, scheduling support and real time performance.

To sum up, realizing the implementation of DDS is handled in both the identified primary studies and the proposed DDS specifications. The primary studies provide novel solutions which have not been (completely) addressed by the DDS specifications. These solutions apply service-oriented development, component-based development or the current general-purpose programming languages. In addition to these, DDS profile implementations and additional specifications are provided. The feature diagram for this problem is shown in Figure 5-12.

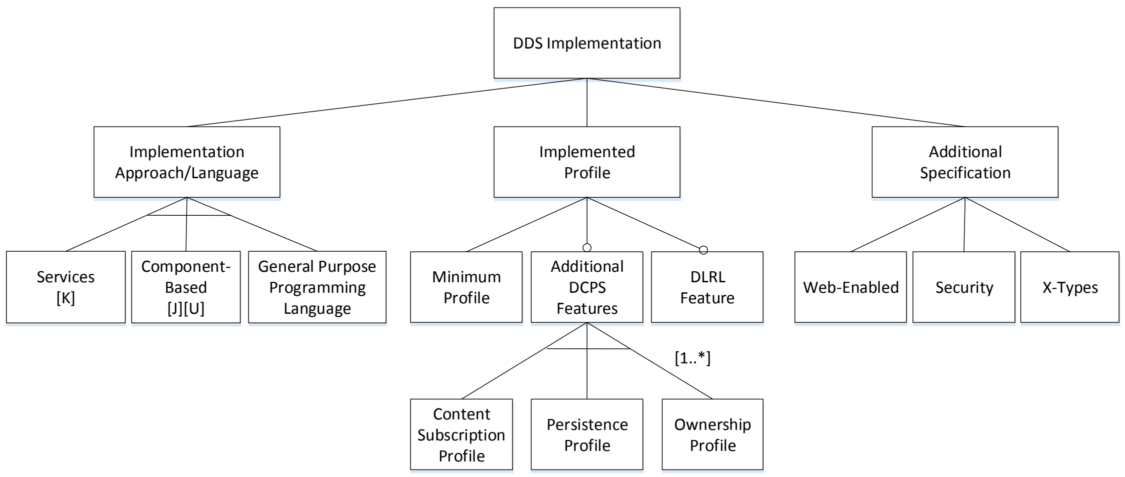


Figure 5-12: Feature diagram for P3 – DDS implementation

P4. DDS integration over WAN

DDS has been mainly designed for Local Area Networks (LAN). However, as more systems become geographically distributed and consist of multiple autonomous systems it has now become necessary for DDS to operate over Wide-Area Networks (WAN). Several studies have discussed the challenges for implementing DDS based systems over WAN.

Study [P] describes the requirements for DDS data-spaces interconnection and presents an architecture that aims to realize these requirements. In particular, the study proposes a DDS interconnection service (DDS-IS) capable of bridging DDS domains as well as adapting between different data schemas. The approach is compliant with the latest OMG specifications and as such does not require any modifications to DDS applications. Experimental results gathered on a prototype implementation have shown that the impact of the service on the communications performance is within the acceptable limits for most real-world uses of DDS (latency overhead is of the order of hundreds of microseconds). Further, the provided service seems to interconnect remote data-spaces efficiently and reduce the network traffic almost N times, with N being the number of final data subscribers.

Study [Y] indicates that most Wide Area Networks (WAN) do not support multicast and UDP transports, which may lead to difficulties when using DDS in WAN. This is because most ISPs in WAN do not allow multicast and UDP flows. The study proposes the use of DDS routers for preserving the semantics of the DDS in the context of WAN and providing an efficient data distribution over WAN. The authors claim that the use of the proposed DDS router outperforms the legacy unicast based communication in terms of scalability and robustness.

Study [K] provides the ServiceDDS a DDS-based framework for supporting the integration and dynamic interaction of real-time heterogeneous participants in distributed peer-to-peer architectures. ServiceDDS participants are able to participate in a global data space using different interactions mechanisms based on DDS or the Extensible Messaging and Presence Protocol XMP (IETF, 2011). DDS offers a data-centric publish/subscribe model for real-time distributed communications. XMPP is a communications protocol for message-oriented middleware based on XML. To meet the demands of real-time systems in the Java programming language RTSJ is adopted.

As stated in problem P2 Study [T] describes the problems with respect to the development of distributed real time (DRE) applications over the Wide Area Network (WAN) using DDS. These DRE systems are becoming more dynamic, larger in topology scope and data volume, and more sensitive to end-to-end latencies, and security threats. Although DDS provides mechanisms for imposing QoS between publisher and subscribers, it does not provide a standard QoS enforcement in the context of WAN, and the required end-to-end QoS support. For DRE system that spans multiple different interconnected networks, assuring end-to-end quality-of-service (QoS) must be defined across different access points, inter-domain links, and within network domains. For the integration over WAN the authors provide the Velox framework to provide end-to-end QoS provisioning for DDS based applications. To support the integration over WAN, the Velox framework implements an end-to-end path abstraction using a Multi-Protocol Label Switching (MPLS) tunnel (Awduche, 1999). This tunnel enables aggregating and merging different autonomous systems from one network domain to another so that data crosses core domains more transparently. In case of large data rates/sizes the network capacity can be easily overwhelmed and applications will not achieve their desired QoS properties, despite the underlying QoS-related resource reservations. To cope with this the Velox framework provides a specific Signaling and Service Negotiation (SSN) capability.

Study [Z] describes Proxy DDS that bridges multiple, isolated DDS domains over WANs and describes the NetQSIP framework that combines DDS, Session Initiation Protocol (SIP) and IP DiffServ to support end-to-end QoS over WANs. The authors claim that, the Velox framework that is presented in their previous work (Study [T]) conducts QoS negotiation and resource reservation in WANs to meet scheduling requirements. However, the Velox framework cannot recover from failures dynamically and does not support dynamic QoS reconfigurations. As such they propose combining Proxy DDS and NetQSIP. Unlike Velox, this solution does not introduce new capabilities at the network layer but uses NetQSIP framework to provide dynamic QoS management and Proxy DDS that can communicate with other proxies without using any tunneling. The experimental results described in the study revealed significant improvement in dynamic resource reservation and effective end-to-end QoS management.

Study [AA] addresses the problem of on-demand dynamic assignment of QoS parameters to DDS distribution services. This is important to avoid an over provisioned network which results in unnecessary network resource wastage. To this end the authors propose a communication architecture that combines DDS with Software Defined Networks (SDN).

SDNs separate the network control and forwarding functions, enabling the fine grained and on demand programming and reprogramming of network behavior. This also allows the abstraction of the underlying infrastructure from applications and network services.

The feature diagram for this problem is shown in Figure 5-13. In sum, although DDS is designed to operate in LAN, one of the most popular DDS research topic is the DDS Integration in WAN. The standard DDS specification uses multicast UDP protocol which is not supported in a WAN. To solve these WAN related DDS problems two basic solution directions have been provided including integration approaches and solutions for quality related concerns.

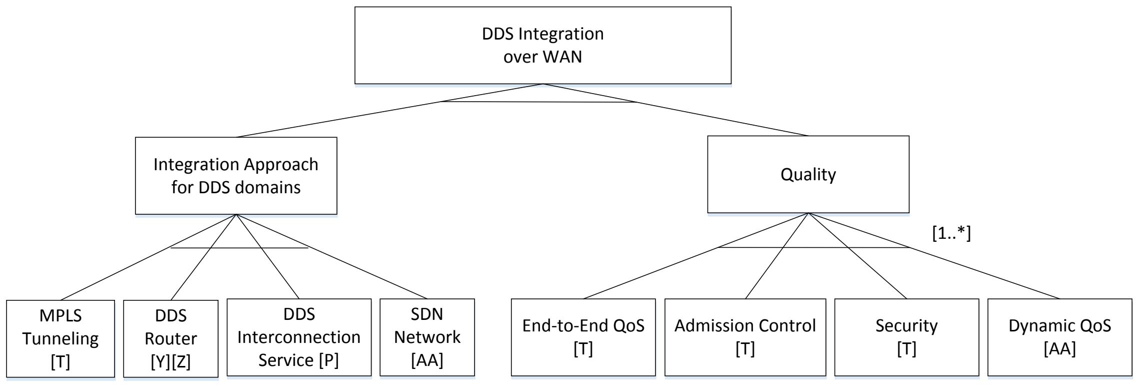


Figure 5-13: Feature diagram of P4 – DDS integration over WAN

P5. DDS using wireless networks and mobile computing

As it is stated in P4, DDS has been mainly designed for Local Area Networks (LAN). On the other hand, similar to Wide Area Networks (WAN), there is a great trend to use DDS in both wireless networks and mobile computing. Several challenges are addressed for integrating and applying DDS in wireless networks and mobile computing.

Study [L] provides an approach for implementing DDS in Wireless Sensor Networks (WSNs) based on the Sensor-Network Publish-Subscribe protocol. This is provided as an alternative to the conventional Real Time Publish-Subscribe protocol that is used in the mainstream implementations of DDS. SNPS seems to perform better compared to resource usage of RTPS implementations. SNPS has been implemented for several wireless and wired network protocols such as (ZigBee, 6LoWPAN, and Ethernet/UDP/IP) on diverse embedded sensor node and PC platforms.

As described before, Study [M] proposes a new bandwidth-aware design scheme for the usage of DDS in Wireless Local Area Network (WLAN). Hereby, it is suggested to use separate uplink and downlink communication model depending on the number of Pubs and Subs. In this way, the performance of the DDS communication in WLAN is enhanced.

Study [N] proposes a novel cloud monitoring and management architecture based on the DDS standard. The focus of the study is the integration of mobile devices in the cloud. These devices are characterized by limited resources and are typically focused on optimizing energy usage. Hence, applications that require large amounts of processing power and resources cannot be easily ported to and deployed on these mobile devices. By moving resource-demanding tasks from the mobile device to the cloud computing infrastructure the problem can be solved to some extent. However, this task-oriented scenario requires that mobile offloaded tasks require high resource usage for a relatively short amount of time. This is in contrast to the current typical service-oriented scenario of the cloud, which service instances over long-lasting processes. To overcome these problems, the study describes a proposal based on the DDS architecture to support the task-oriented and decentralized cloud management scenarios.

Study [E] describes a lightweight and efficient DDS implementation, called TinyDDS, for WSNs. The study focuses on the inherent trade-offs among conflicting objectives such as data yield, data fidelity and power efficiency in the pub/sub scheme in WSNs. To address these issues TinyDDS adaptively performs event publication by balancing conflicting objectives according to dynamic network conditions such as noise level and dynamic node addition. TinyDDS uses its self-configuring event routing protocol and a multi-objective optimization mechanism to perform the adaptation of event publications according to the dynamic network conditions.

Study [F], proposes an interoperable publish/subscribe communication in WSNs based on the TinyDDS that is presented in Study [E]. TinyDDS simplifies the development of publish/subscribe applications and provides an efficient implementation with respect to memory footprint and power consumption in WSNs.

Similar to Study [E], Study [AG] proposes a customizable DDS implementation (sDDS) for WSNs and embedded systems with limited resources. In order to make DDS applicable to resource limited environments the authors used a model-driven software development (MDSD) process to tailor and minimize the middleware functionality for each sensor node, depending on the purpose of the node in the network, the resource capabilities of the hardware and the deployment structure. The authors claim that sDDS can be used for a wide range of target systems from 8 bit to 32 bit controllers. Furthermore, it is also ported to different embedded platforms such as RIOT-OS, FreeRTOS, Contiki and etc.

Study [C] addresses the need to satisfy various constraints such as efficiency, memory footprint and power consumption in WSNs. This often leads to tightly coupled designs and likewise WSN applications lack reuse and are difficult to adapt the non-functional properties (e.g., data routing, concurrency, data aggregation and event filtering). The study presents the pluggable framework in TinyDDS which decouples various non-functional properties and enables the development of flexible and re-usable WSNs applications. TinyDDS adopts the Layer design pattern to separate and modularize the different functionalities into different layers.

The feature diagram for this problem is shown in Figure 5-14. As explained above, DDS has been primarily designed to operate in LAN. Similar to adopting DDS in a WAN context, adopting DDS in wireless networks and mobile computing is a relevant research topic in DDS. We can identify the following solution directions: Sensor Network Pub-Sub Protocol (Study [L]), Cloud Management Architecture (Study [N]), Lightweight DDS Implementation (Study [E, F, AG]), Bandwidth Aware Networking (Study [M]), and Pluggable Framework (Study [C]).

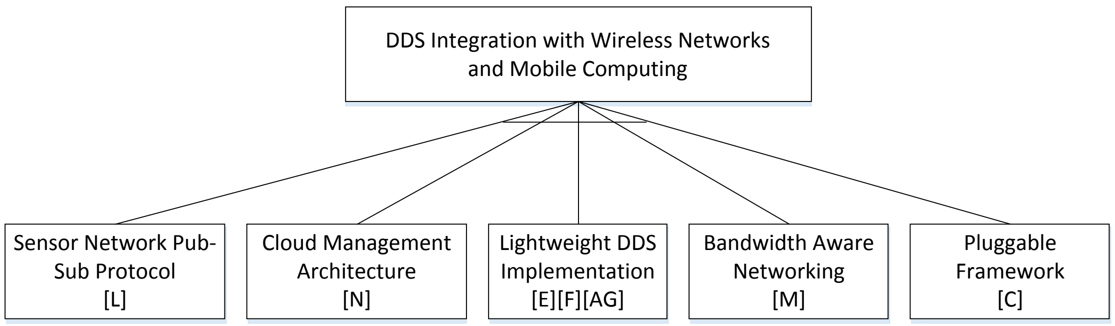

DDS Integration with Wireless Networks and Mobile Computing
Sensor Network Pub-Sub Protocol [L]
Cloud Management Architecture [N]
Lightweight DDS Implementation [E][F][AG]
Bandwidth Aware Networking [M]
Pluggable Framework [C]

Figure 5-14: Feature diagram for P5 – DDS wireless networks and mobile computing

P6. Interoperability among DDS vendor implementations

The OMG's DDS standard provides both programming language interoperability and protocol interoperability (OMG, 2014). Programming language interoperability is the ability to interoperate applications written in different programming languages. Protocol interoperability is the ability to interoperate applications and access network applications with different network protocols. Different studies have focused on tackling interoperability challenges in DDS.

Study [F] focuses on the integration of wireless sensor networks (WSNs) in which the interoperability has not yet been fully addressed. The authors propose an implementation of the DDS, called TinyDDS which customizes standard data types, data representation and session protocols to realize programming language interoperability and protocol interoperability. TinyDDS supports programming language interoperability by implementing a mapping of the OMG IDL (Interface Definition Language) to nesC and provides a set of DDS APIs in nesC. This allows different applications to use different languages with the same DDS APIs for event subscription and publication. TinyDDS also supports protocol interoperability by making publish/subscribe communication interoperable between WSNs and access networks.

As stated before, Study [P] proposes a DDS interconnection service (DDS-IS) capable of bridging DDS domains as well as adapting between different data schemas. DDS-IS provides data model compatibility and confidentiality communication between data models with dissimilar data models (different topic names or data types). The study also addresses the need for establishing QoS requirements for bridged data-spaces. The study claims that the approach guarantees data delivery between different data spaces with the required QoS.

The OMG's DDS standard provides protocol interoperability among different DDS implementations by introducing the real-time publish/subscribe (RTPS) protocol. All vendors must implement at least the Simple Discovery Protocol (SDP) to support RTPS. SDP enables each participant to send its endpoint data to each participant and receive endpoint data from all other participant. In this context, Study [I], Study [X] and Study [V] propose to adopt alternative algorithms to the standard SDP in the RTPS protocol. Study [I] proposes the Bloom Filter algorithm instead of SDP. Study [X] extends this work and proposes the Modified Counted Bloom Filter Algorithm. Similarly, Study [AE] proposes Parallel Dynamic Bloom Filters (SDP-Parallel DBF). Study [V] proposes the Content-based Filtering Discovery Protocol. In all of these studies, the authors claim that SDP is inefficient and according to their test results the newly proposed algorithms outbound SDP.

Figure 5-15 shows the feature diagram with the identified obstacles for DDS Interoperability. Summarizing, although OMG's RTPS (DDSI) specification provides interoperability between the implementations of different DDS vendors, there are still open challenges which need attention. The solution directions have focuses on programming language interoperability, protocol interoperability, and data schema interoperability.

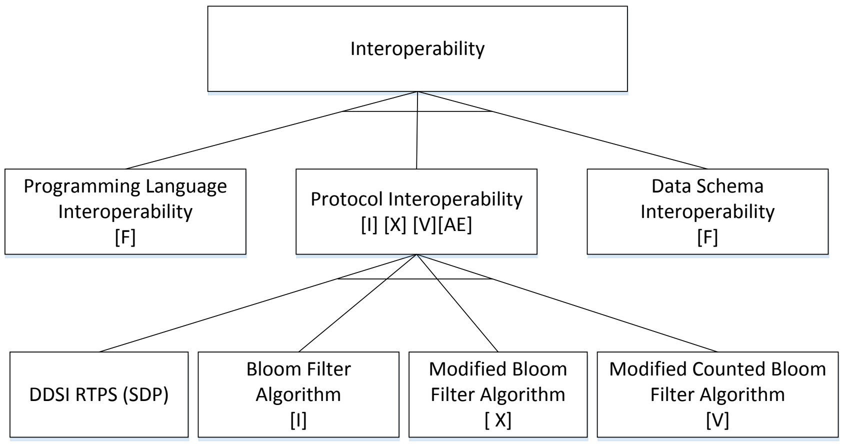

Interoperability
Programming Language Interoperability [F]
Protocol Interoperability [I] [X] [V][AE]
Data Schema Interoperability [F]
DDSI RTPS (SDP)
Bloom Filter Algorithm [I]
Modified Bloom Filter Algorithm [X]
Modified Counted Bloom Filter Algorithm [V]

Figure 5-15: Feature diagram for P6 – DDS interoperability

P7. Data consistency in DDS

In distributed computing, one of the important challenges is ensuring integrity and consistency of data under hard real time constraints. Several studies have discussed the challenges related to data consistency within DDS. In principle we can identify data consistency approaches applied at (re)configuration time and during operation time.

Study [A] provides an analysis of the DDS specification with respect to its support for the correctness preservation during reconfiguration of DDS-based systems. The analysis discusses three aspects of correctness including structural integrity, mutually consistent state and application state invariants. The study concludes that the DDS architecture and the QoS-controlled behavior automatically ensure correctness preservation during reconfiguration.

Study [Q] addresses the problem of data inconsistency at operation time in the context of distributed data consistency management. Hereby no center node is present that is responsible for forwarding data packets and maintaining the communication data. Instead a data exchange model is adopted whereby multiple nodes can write data to the same topic. When a new data reader joins the reader set it should include the data that is consistent with the other nodes in the reader set. However, in case of node failures the requested data might not be delivered and as such the data consistency cannot be ensured. To overcome this problem study [Q] implements a real-time service bus (RTSB) using the so-called Paxos algorithm to solve the data consistency problem in DDS.

Summarizing, the solution directions for the Data Consistency obstacle focus on correctness preservation during reconfiguration, and lack of center node as shown in feature diagram for this obstacle in Figure 5-16.

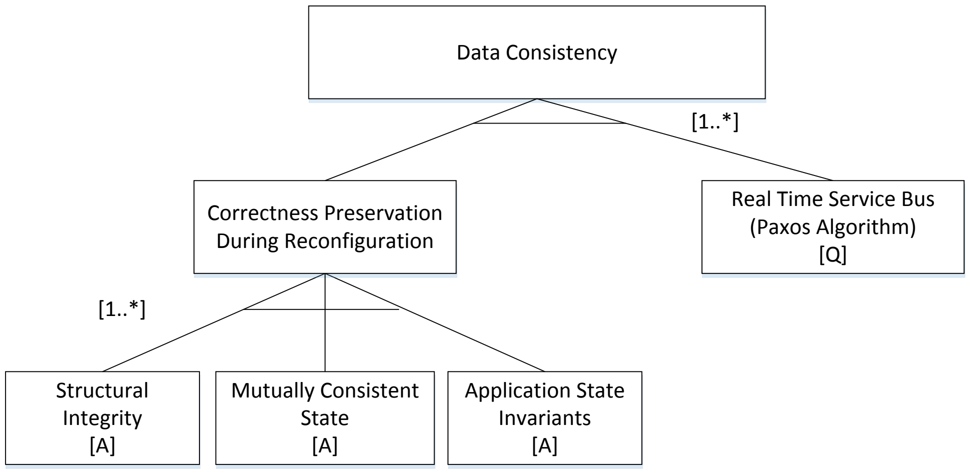

Data Consistency
[1..*]
Correctness Preservation During Reconfiguration
Real Time Service Bus (Paxos Algorithm) [Q]
[1..*]
Structural Integrity [A]
Mutually Consistent State [A]
Application State Invariants [A]

Figure 5-16: Feature diagram for P7 – Data consistency in DDS

P8. Reliability in DDS

Within DDS a reliability protocol is defined that can be tuned for optimum performance on a per data stream basis (OMG, n.d.-c). The reliability protocol needs to be configured and tuned using QoS policies including Reliability, History, Resource Limits, DataWriter Protocol, and DataReader Protocol. It is expected that the particular reliability requirements for this parameter need to be provided in the implementation of DDS based systems. In general, reliability is realized through fault prevention, fault detection, fault tolerance, and fault forecasting. Several studies have discussed the challenges and proposed approaches to support reliability in DDS related to reliability.

Study [D] proposes a DDS-compliant data dispatching infrastructure to reliably disseminate events and to balance data distribution load. The dispatch mechanism puts a routing substrate between publishers and subscribers to detect possible faults in other peers, and to dynamically recover and reconfigure the system when a peer crashes or a new peer arrives. Since the proposed solution for fault-tolerance is fully compliant with the DDS standard, it can be deployed over already installed DDS systems.

Study [H] and Study [G] aim to evaluate the robustness of DDS middleware using robustness testing and fault injections in the implemented DDS. A tool JFIT (Java Fault Injection Tool) is provided that can automatically inject external faults to DDS API functions without altering the source code but modifying the system’s state. The tool has been proposed to accelerate tests execution and to analyze the tests outcomes.

The feature diagram for this problem is shown in Figure 5-17. In sum, although DDS provides some reliability related QoS policies, these are not adequate to address all reliability related problems in DDS based systems. In this context, solutions can relate to fault prevention, fault detection, and fault tolerance issues in DDS.

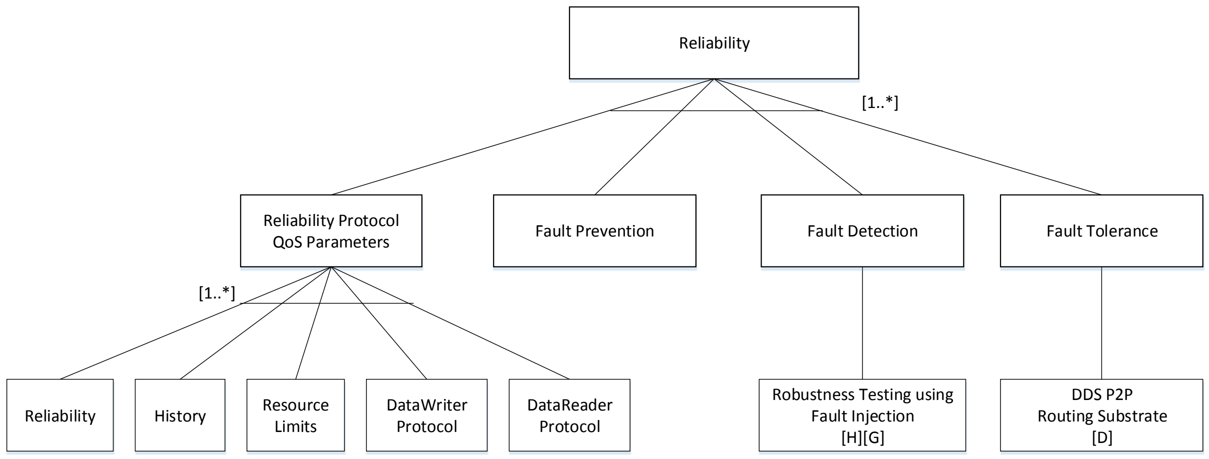


Figure 5-17: Feature diagram for P8 – Reliability in DDS

P9. Scalability in DDS

Scalability defines in general how well a solution to some problem will work when the size of the problem increases. Within the context of distributed systems scalability of a distributed software system indicates whether it is still efficient in case the system load increases. Hereby, the system load can be considered as the number of participating nodes. If the solution is still suitable, efficient and practical after adding resources, the software system can be said as scalable. From this point of view scalability is closely engaged with fault tolerance and maintainability. There are several studies that discuss scalability issues in DDS.

As stated above, several studies criticize the Simple Discovery Protocol (SDP) of the DDS wire protocol DDSI (OMG, 2014). In this context, study [I], introduces the use of Bloom Filter (BF) Algorithm to increase DDS scalability. SPD is used as the standard algorithm for node discovery in DDSI. The authors claim that the SDP is not scalable in case the number of DDS end-points increases. They provide analytical and experimental studies to compare BF and SDP showing that BF is more scalable. Similarly, Study [X], Study [AE] and Study [V] propose the Modified Counted Bloom Filter, Parallel Dynamic Bloom Filter and Content-Based Filtering Discovery Protocol algorithms respectively, all of which appear to be more scalable than the SDP algorithm in DDSI.

In Study [G] authors claim that design techniques for scalable DDS deployments, especially for mobile data intensive applications are still missing. So, they offer two solutions: P2P routing substrate and Relay-based DDS. As it is mentioned above in Study [D], P2P routing substrate provides fault-tolerance to the DDS based systems. Using the routing substrates within DDS domains and connecting these domains via DDS relay components provide scalability to the fault-tolerance issue even in Wide Area Networks (WAN).

The feature diagram for this problem is shown in Figure 5-18. In sum, scalability is another software quality factor which has not been directly addressed by the OMG DDS specifications. We can identify two basic solution directions for scalability problems: scalability in node discovery and scalability in fault-tolerance.

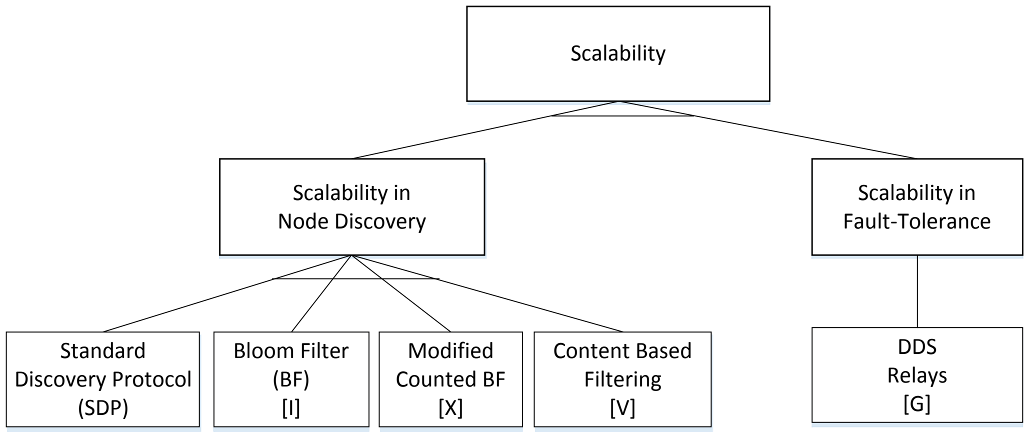

Scalability
Scalability in Node Discovery
Scalability in Fault-Tolerance
Standard Discovery Protocol (SDP)
Bloom Filter (BF) [I]
Modified Counted BF [X]
Content Based Filtering [V]
DDS Relays [G]

Figure 5-18: Feature diagram for P9 – Scalability in DDS

P10. Security

The analysis of the selected primary studies showed the increasing interest in addressing security issues while integrating DDS on WAN, wireless networks and mobile devices. Security concerns appear to be a lesser concern for LAN but when exceeding the boundaries of the LAN new security threats can occur through WAN and mobile computing.

As stated in P4, Study [T] and Study [Z] propose solutions to enforce realization of QoS when a DDS based system is running over WAN. The studies claim also that security policies are required to enhance information dissemination and hence their future work will be on developing security policies to allow authentication, authorization, access control and secure transport.

Study [AB] has directly addressed security as an important issue and provides a solution approach. The study criticizes the fact that information partitioning in current DDS practices are not based on security classifications although this is vital for many systems. As such, they propose a transport mechanism called secured-transport that provides information partitioning enforcing multi-level security (MLS). Furthermore, the study also presents a novel secure discovery mechanism that enables using the secured-transport mechanism with existing DDS implementations.

Besides of the primary studies we can also identify the recently published OMG specification about security (OMG, 2016). The specification defines the Security Model and Service Plugin Interface (SPI) architecture for compliant DDS implementations. The DDS Security Model is enforced by the invocation of these SPIs by the DDS implementation. This specification defines five SPIs that when combined together provide Information Assurance to DDS systems: Authentication Service Plugin, Access Control Service Plugin, Cryptographic Service Plugin, Logging Service Plugin, and Data Tagging Service Plugin. Figure 5-19 shows the conceptual diagram indicating the place of the SPI.

The feature diagram for this problem is shown in Figure 5-19. Several solutions have been proposed to cope with security problems in DDS. A security specification for DDS has been proposed by OMG. Further solutions have been provided for the transport mechanism (study

[AB]), and node discovery mechanism (study [AB]). Finally, new security policies have been prepared (study [T], [Z])

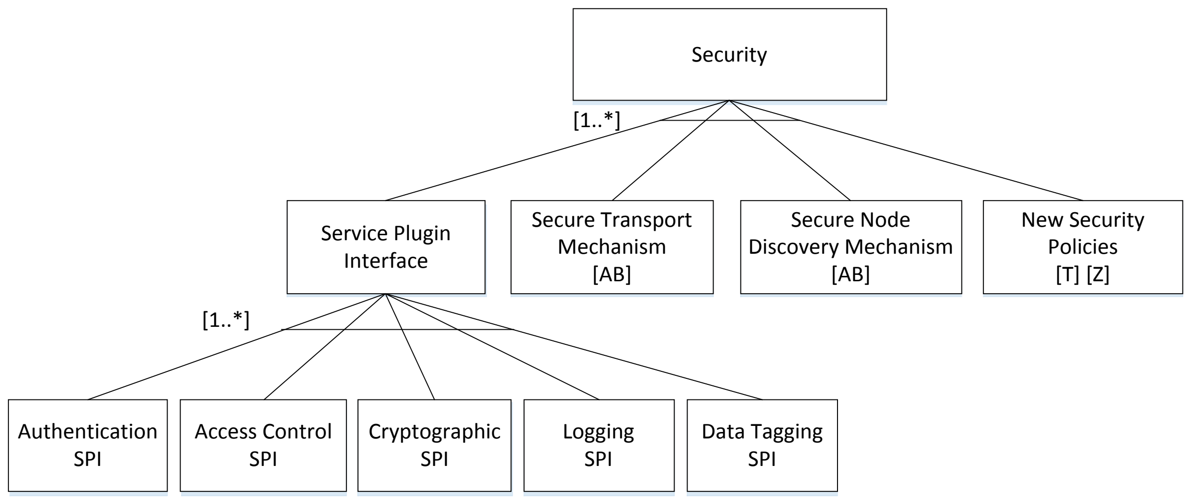


Figure 5-19: Feature diagram for P10 – Security

P11. Integration with Event Based Systems

In traditional imperative programming, the program is modeled as a series of operations and statements are used the change the program's state. This programming model is also referred as sequential or procedural programming. In contrast, reactive programming languages provide dedicated abstraction for time changing values (signals or behaviors). Reactive Programming propagates changes and re-evaluates dependent variables as the signals/behaviors values are updated. One of the application domains of event-based programming is the processing of real-time sensor generated data (data stream) which must be processed in an event-based, distributed, and parallel manner.

In order to build reactive and high-performance stream processing applications, study [AE] investigates the benefits of introducing DDS blending with reactive programming. Although DSS has powerful data delivery mechanisms it lacks of data processing APIs and abstractions to develop event-driven applications. In other words, DDS API is not designed for retrieving individual updates about an object but the state of an object. As such, study [AE] focuses on integrating DDS with event-based programming to unify the local and distributed stream processing aspects under a common dataflow programming model. The authors claim that this approach can be also used in industrial internet of things (IIoT) systems since IIOT can be expressed as a distributed asynchronous dataflow.

Similarly, study [AG] discusses the use of DDS in event-based systems. As stated above the DDS standard does not directly address how to guarantee end-to-end response times to support the implementation of event-based systems. As such, this study investigates how to ensure real-time behavior by applying DDS to event-driven systems within the context of the OMG MARTE Standard (OMG, 2008). The provided solution of this study includes modeling the QoS entities to enable the usage of these DDS features in real-time applications. In order to facilitate the integration of DDS with model driven development processes the authors propose a set of transformations among the QoS settings and the end-to-end flow entities defined by the MARTE modeling standard. From this point of view this study is the extension of Study [O].

The feature diagram for this problem is shown in Figure 5-20. Here, we can identify two basic approaches including reactive programming with DDS, and modeling QoS Entities with transformations to MARTE.

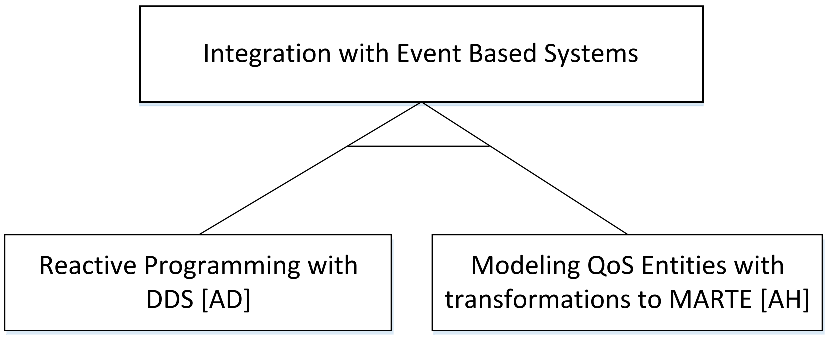

Integration with Event Based Systems
Reactive Programming with DDS [AD]
Modeling QoS Entities with transformations to MARTE [AH]

Figure 5-20: Feature Diagram for P11 – Integration with Event Based Systems

2) RQ 2. What are solution directions for the identified obstacles?

When addressing an obstacle of DDS, the primary studies usually also provide the corresponding solution directions. Table 5-8 provides a summary of the solution directions for each of the identified problem that were defined in Table 5-6 and discussed before. These solution directions are proposed by the authors of the selected primary studies. As we can observe from the table, based on the identified obstacle the solution directions are diverse in nature. Solution directions include design heuristics and design abstractions, adoption of different paradigms, refinement of the DDS concepts, novel introduction and implementation of algorithms, integration with other paradigms, and solutions for realizing system-wide quality management. Although we can observe several obstacles in DDS, the following table shows also promising solution directions.

Table 5-8: Solution Directions for the Identified Obstacles in DDS

No	Primary study	Solution Direction	Study
P1	Complexity of DDS configuration	• Separation of configuration concerns from application concerns • Abstraction of lower level source code to components or services • Use of model-driven development approaches to automatically generate the parameter values and the code • Component Based Development	J K S U
P2	Performance prediction, measurement & optimization	• For performance prediction adopt emulation and experimentation to predict the QoS values for different configurations • For performance measurement adoption of explicit frameworks, benchmarking and tool support • For performance optimization the identified solutions include the adoption of new discovery protocol algorithms, explicit scheduling mechanisms, and new communication model	S, M, AC B, R, S, T I, M, O, T, X, V, AE, AF
P3	Implementing DDS	• Component Based Development • Service Oriented Development	J, U K
P4	DDS integration over WAN	• MPLS Tunneling • DDS Router • SDN Network • Bridging DDS domains • End-to-End QoS • Admission Control • Security • Dynamic QoS	T Y, Z AA P T T T T
P5	DDS using wireless networks and mobile computing	• Sensor Network Pub-Sub Protocol • Cloud Management Architecture • Lightweight DDS implementation • Interoperable Lightweight DDS implementation • Band-width aware Networking • Pluggable Framework • Embedded DDS implementation	L N E F M C AG
P6	Interoperability among DDS vendor implementations	• Programming Language Interoperability • Data Schema Interoperability • Protocol Interoperability: ○ Standard Discovery Protocol (SDP) ○ Bloom Filter Algorithm ○ Modified Counted Bloom Filter Algorithm ○ Parallel Dynamic Bloom Filter Algorithm	F F I, V, X, AE I X AE
P7	Data consistency in DDS	• Correctness Preservation During Reconfiguration • Correctness Maintenance during operation time	A Q
P8	Reliability in DDS	• Robustness Testing • P2P Routing Substrate	G, H D
P9	Scalability in DDS	• DDS Relays • Scalability in Node Discovery: ○ Standard Discovery Protocol (SDP) ○ Bloom Filter Algorithm ○ Modified Counted Bloom Filter Algorithm ○ Parallel Dynamic Bloom Filter Algorithm	G I, V, X, AE I X AE
P10	Security	• Secure Transport Mechanism • Secure Node Discovery	AB AB
P11	Integration with Event-based Sys.	• Reactive Programming with DDS • Modeling QoS Entities with transformations to MARTE	AD AH

5.5 Discussion

We have applied a meticulous systematic literature review based on the proven protocol of Kitchenham et al (Kitchenham et al., 2009). The SLR included 468 papers in from which we selected 34 as primary studies. We have carefully devised and applied our selection and elimination criteria in order not to miss any relevant primary study. The primary goal of SLR was to identify the relevant obstacles and the corresponding solution directions. Based on our thorough study we could identify 11 problem categories. It appears that different studies have focused on different problems and solutions and as such we could observe an uneven distribution. We consider this fact also as the result of our study since this highlights the important obstacles as well as the obstacles which have not yet been fully explored. On its turn this provides a broader vision on DDS and also paves the way for further research.

Our research methodology however is a systematic literature review that focuses on the analysis of existing primary studies in the literature. Hence, we can only report on the problems that were identified in these primary studies. From our SLR we can observe that implementing a DDS is one of the obstacles (P3). The corresponding primary studies report on the obstacles of using component-based development and service-oriented development for implementing DDS. It should be noted that several other more detailed but unreported problems could exist that are directly related to implementation of DDS-based systems. The identification of these problems would require an in-depth study to the corresponding implementations of the DDSs. We consider this however out of scope of our study since these are not within the scope of an SLR study.

We could identify 11 different problem categories. An important number of the problems relate to quality concerns such as reliability, scalability and security. We have described the problems related to specific quality concerns if these were also the topic and focus of the identified primary studies. Quality concerns which were not explicitly reported were not included as obstacles. Our study could on the one hand be used to highlight the relevance of the quality concerns in DDS based systems. On the other hand, our SLR shows also which quality concerns have not been explicitly discussed. This only implies that no in-depth research has been carried out for these quality concerns, and not necessarily that these quality concerns are not relevant for DDS. On its turn this observation can trigger further research on quality concerns in DDS.

Our SLR has also resulted in a set of feature diagrams that summarize the reported obstacles and the solution directions. The feature models that we have developed can be also used pave the way and support the development of a DDS ontology. We consider this as a possible future work.

From our SLR we could also observe that for some of the identified problem categories have been also considered in some of the proposed extensions to the OMG's DDS specification. We have listed these extensions in Table 5-7. Also, we have referred to the OMG's DDS specifications in the problem categories P1, P3, P4, P6, P7, and P10. While describing the problems and the corresponding solutions we could identify that some researchers claim that the offered solution in the primary study is better than the one used of the DDS specification (such as using Bloom Filters instead of Standard Discovery Protocol in DDS RTPS (DDSI) specification). Further, other primary studies handle topics that are not (completely) covered by the provided DDS specification yet (such as reliability problems in Problem 7. Finally, it should be noted that some DDS specifications are still in beta version and have not been finalized yet.

The main threats to validity (Dybå & Dingsøyr, 2008) of this review are publication and selection bias, and data extraction and classification.

The publication bias indicates the case in which researchers are more likely to publish positive results and refrain from publishing studies that have negative results. To cope with this publication bias Kitchenham et al. (Kitchenham & Charters, 2007) recommend searching also company journals, grey literature, conference proceedings and the internet. We have applied this approach which indeed led us to new papers that we could not identify in our regular search. We performed the inclusion/exclusion procedures on a well-established screening of primary studies and included both qualitative and quantitative studies. The inclusion and exclusion criteria are selected by the researchers who performed the systematic literature review. A subjective approach towards defining the selection criteria and selecting the primary studies for further consideration, can introduce a threat to validity in this study. For reducing the bias with respect to the definition of the selection criteria we used the quasi-gold standard approach as defined by Zhang et al. (H. Zhang, Babar, & Tell, 2011). Hereby, we first picked a random set of 10 studies and each of the researchers defined the selection criteria. These criteria were validated together and the final set of exclusion/inclusion criteria was defined.

For reducing the selection bias for selecting the primary studies, the evaluation and the selection of the primary studies were performed separately by the two researchers. Each researcher recorded also the reasons of acceptance or rejection for all the considered studies. Later on, the evaluated list of primary studies of each researcher was compared with that of the other researchers. In case of differences we discussed the paper in detail and came with the final decision. H. Zhang and A. Babar (H. Zhang et al., 2011) provide an enhancement to the SLR protocol of Kitchenham (Kitchenham et al., 2009). In their method the so-called d QGS-based systematic search approach for devising and testing search strategies is applied. For our study we did not consider this directly but for devising the search strings we indeed first read a couple of relevant papers to define and justify our search strings. We have applied both automated searches and manual searches both as a preliminary analysis and as a subsequent analysis after having observed the publication channels returned by the relevant search strings. With our search strings we think that we have identified all the papers that are directly related to DDS.

After the primary studies have been evaluated and selected the relevant data must be extracted for deriving the review results. Hereby defining the data extraction criteria and classification model is very important. To define the data extraction model, we first read a set of randomly selected primary study papers. Each of use defined an initial data model based on the research questions that we had defined. Later on, we compared the different data extraction models, discussed the differences and decided on the data extraction model. After that we applied the data extraction model to a set of primary studies and checked whether we could derive the answers to the research questions with the adopted data extraction model. We applied this several times and after a number of iterations and discussions we decided on the final data extraction model. We can state that the problem categories that we have identified cover the main problems. However, some problems could be also seen as sub-categories of these basic categories. To highlight these, we have adopted feature models.

5.6 Conclusion

In this chapter we have provided a systematic literature review to describe the state of the art of the Data Distribution Service (DDS) middleware and identify the obstacles in applying DDS. We have considered the published literature since the introduction of DDS in 2003, and among 468 papers that were discovered we identified 34 of them as primary studies related to our research questions. First of all, we can state that the application of DDS has been increasingly popular and has been used in various application domains such as defense, finance, and medical domain. In addition to its basic application we can also identify the application and integration of DDS to solve problems in technical domains such as cloud computing, component-oriented development, mobile computing, and wide area network. Our study shows that DDS provides indeed important benefits for realizing real-time distributed applications. Our focus in this chapter was mainly about the obstacles that are encountered when applying DDS. Using the SLR we identified 11 basic categories of problems that were discussed in the identified primary studies. We have described each problem in detail by referring to the papers in which these were discussed. The identified problems included Complexity of DDS configuration, Performance prediction, measurement and optimization, Implementing DDS, DDS integration over WAN, DDS using wireless networks and mobile computing, Interoperability among DDS vendor implementations, Data consistency in DDS, Reliability in DDS, Scalability in DDS, Security, and Integration with Event Based Systems. We have adopted feature diagrams to summarize and provide an overview of the identified problem and its solutions as defined in the primary studies.

Obviously, in addition to the benefits there are still many obstacles to be solved to further support the adoption of DDS. We believe that the results of this chapter pave the way for further research in DDS. The obstacles can be adopted to trigger new research questions. The proposed solutions can be used to further enhance the DDS specification and support the practitioners in their decision making while applying DDS. The description of the obstacles in this chapter provides an overall perspective that could help to synthesize the different solutions. In our future work we plan to address selected research topics based on the categories that we have defined.

6

DATA DISTRIBUTION SERVICE BASED ARCHITECTURE DESIGN FOR THE INTERNET OF THINGS SYSTEMS[1]

[1] This chapter is based on the following published book chapter:

- B. Tekinerdogan, Ö. Köksal, and T. Çelik, "Data Distribution Service-Based Architecture Design for the Internet of Things Systems," in Connected Environments for the Internet of Things, Springer, Cham, 2017, pp. 269–285.

Abstract

The Internet of Things (IoT) is the internetworking of people and physical devices that enable the collection and exchange of data. The number of connections between people and things as well as the volume of data that is generated is dramatically increasing. Hereby, various kinds of data are generated by multiple kinds of devices, which are processed in different ways, and used by different applications. To realize the distributed execution of IoT systems over multiple resources different requirements and quality factors must be satisfied. Traditionally, to reduce the effort for developing distributed systems, middleware architectures have been introduced that provide common services such as name and directory services, discovery, data exchange, synchronization, transaction services, etc. To address the needs and integration of IoT systems the adoption of middleware seems to be a feasible solution. A middleware that is directly related to data-intensive systems in which quality of service is important is the Data Distribution Service (DDS). The DDS is a standard data-centric publish-subscribe programming model and specification for distributed systems that has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. In this chapter, we explore and propose the adoption of DDS as a middleware platform for IoT systems. For this, we first describe the requirements for IoT systems and present the IoT reference architecture. Subsequently we provide a DDS-based architecture for IoT systems based on the Views and Beyond Approach. We illustrate our approach for the architecture design of IoT-based smart city engineering.

Keywords: Data Distribution Service, Internet of Things, Software Architecture

6.1 INTRODUCTION

The Internet of things is the internetworking of people and physical devices that enable the collection and exchange of data. The number of connections be-tween people and things as well as the volume of data that is generated is dramatically increasing. Hereby, various kinds of data are generated by multiple kinds of devices, which are processed in different ways, and used by different applications. To realize the distributed execution of IoT systems over multiple resources different requirements and quality factors must be satisfied.

Traditionally, to reduce the effort for developing distributed systems, middle-ware architectures have been introduced that provide common services such as name and directory services, discovery, data exchange, synchronization, transaction services, etc. To address the needs and integration of IoT systems the adoption of middleware seems to be a feasible solution. A middleware that is directly related to data-intensive systems in which quality of service is important is the Data Distribution Service (DDS) (Angelo Corsaro, n.d.). The DDS is a standard data-centric publish-subscribe programming model and specification for distributed systems that has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains.

In this chapter, we explore and propose the adoption of DDS as a middleware platform for IoT systems. For this, we first describe the requirements for IoT systems and present the IoT reference architecture. Subsequently we provide a DDS-based architecture for IoT systems based on the Views and Beyond Approach. We illustrate our approach for the architecture design of IoT-based smart city engineering.

The remainder of the chapter is organized as follows. In section 6.2 we provide the background on software architecture modeling which is necessary for understanding the architecture views in subsequent sections. In section 6.3 we describe the IoT architecture using selected viewpoints. Section 6.4 presents the architecture models for DDS. Based on the architecture models from section 6.3 and section 6.4 we present the DDS-based IoT architecture in section 6.5. Finally, section 6.6 concludes this chapter.

6.2 Software Architecture Modeling

Architectural drivers define the concerns of the stakeholders which shape the architecture. A stakeholder is defined as an individual, team, or organization with interests in, or concerns relative to, a system. Each of the stakeholders' concerns impacts the early design decisions that the architect makes. A common practice is to model and document different architectural views for describing the architecture according to the stakeholders' concerns. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. Obviously, the notion of viewpoint now plays an important role in modeling and documenting architectures. So far most architectural viewpoints seem to have been primarily used either to support the communication among stakeholders, or at the best to provide a blueprint for the detailed design.

In this chapter, we will use the Views and Beyond framework in which predefined viewpoints are organized into three categories including module styles, component-and-connector styles and allocation styles. Module styles are used to show how the system is structured as a set of implementation units. Component and connector styles are used to show how the system is structures as a set of runtime elements. Allocation styles are used to show how the software elements are mapped to non-software elements in its environment. We will adopt three view-points for our purposes including layered viewpoint and deployment viewpoint.

The Layered viewpoint reflects the division of software modules called layers. In a layered architecture, the system is depicted as a set of layers which are stacked on top of each other. Hereby a layer can only access the next lower layer and call backs from lower layers to higher layers are not allowed. In the following sections, we will see that both IoT and DDS systems include a layered architecture.

In addition to the layered viewpoint we will also apply the deployment view-point, which is used to show how the software elements are allocated to hardware of a computing platform. It is useful for analyzing and tuning certain quality at-tributes of the system such as performance, reliability and security.

6.3 INTERNET OF THINGS ARCHITECTURE

6.3.1 Conceptual Model

Figure 6-1 provides a conceptual model including the relations among the basic IoT concepts. The model has been adopted from the Alliance of IoT Innovation (AIOTI) Domain Model (AIOTI WG03 2015) (Alliance for IoT Innovation, 2015). The domain model represents the basic concepts and relationships in the domain at the highest level. In the model, User interacts with a physical entity of the physical world, a Thing. The User can be a human person or a software agent that has a goal, for the completion of which the interaction with the physical environment must be performed through the mediation of the IoT. A thing is a discrete, identifiable part of the physical environment that can be of interest to the User for the completion of his goal. Things can be any physical entity such as humans, cars, animals, or computers.

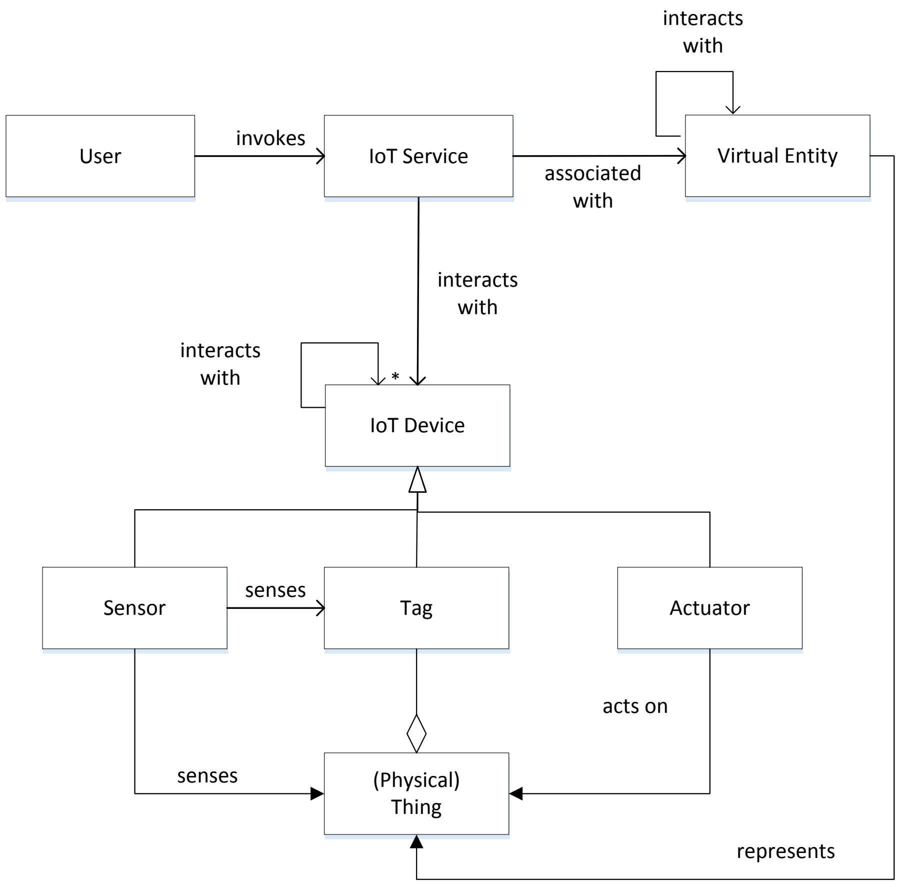


Figure 6-1: Conceptual model for the IoT

The interaction between a User and Thing is mediated by an IoT Service which is associated with a Virtual Entity, a digital representation of the physical entity. A Thing can be represented in the digital world by a Virtual Entity. Different kinds of digital representations of Things can be used such as objects, 3D models, avatars, objects or even a social network account. Some Virtual Entities can also interact with other Virtual Entities to fulfill their goal.

An important aspect in IoT is that changes in the properties of a Thing and its corresponding Virtual Entity needs to be synchronized. This is usually realized by an IoT Device that is embedding into, attached to or simply placed in close vicinity of the Thing. In principle, we can identify three devices including Sensors, Tags and Actuators. Sensors are used to measure the state of things they monitor. Essentially, sensors take a mechanical, optical, magnetic or thermal signal and convert this into voltage and current. This provided data can then be processed and used to define the required action. Tags are devices to support the identification process typically using specialized Sensors called readers. The identification process can be different including optical as in the case of barcodes and QR-code, or RF-based. Actuators are employed to change or affect the things.

6.3.2 Feature Model

In this section, we provide a feature-driven overview of IoT and its “Session Layer” protocols. A feature diagram is a tree with the root and descendent nodes. The root represents a concept and nodes are the features. Feature diagrams might show mandatory features as well as variant features which can be represented as optional or alternative features. A feature configuration is a set of features which describes a member of the represented concept. A feature constraint restricts the possible selections of features to define configurations. The legend (abstract syntax) used for the feature diagrams is given in Figure 2-3.

The top-level feature diagram of the IoT is given in Figure 2-4. This diagram is similar to the layer diagram of the IoT given in the next section.

Session layer is responsible for setting up and taking down of the association between the IoT connection points. The session layer provides services related is-sues of the session such as initiation, maintenance, and disconnection. As such, frequency and duration of various types of sessions are related with the session layer. Selection of the session layer protocol depends on many factors such as data size, number of devices to be connected, latency, etc. Depending on the application requirements different session layer protocols might be used in session layer of the IoT application. Focusing on the session protocols, we have derived the feature diagram given in Figure 2-5.

The mandatory features in the feature diagram are protocol type, source-target, transport type and architecture. Please note that, although transport type belongs to the network layer, it is shown as a mandatory feature in Figure 2-5 since it is closely related with the protocol characteristics.

Widely-used session layer protocol types are given below:

- Message Queuing Telemetry Transport (MQTT): One of the most popular protocols to collect device data and communicate with servers (OASIS, 2014).
- Extensible Messaging and Presence Protocol (XMPP): is based on ex-changes of XML messages in real time that is defined to connect devices to servers (IETF, 2011).
- Advanced Message Queuing Protocol (AMQP): A queuing system de-signed to connect servers to each other (OASIS, 2011).
- Data Distribution Service (DDS): A fast data bus for integrating devices and systems (OMG, 2015b).

- The Constrained Application Protocol (CoAP): A specialized web-based protocol to be used in constrained nodes and constrained networks (IETF, 2013).

The focus of this chapter is the application of the DDS protocol.

There are three types of source-target relations available in session layer proto-cols: Device-to-Device (D2D), Device-to-Server (D2S), and Server-to-Server (S2S) as given in Figure 2-5. Some references these features are also named Machine-to-Machine (M2M), Machine-to-Cloud (M2C), and Cloud-to-Cloud (C2C) respectively. DDS and CoAP are used for M2M communication, whereas MQTT and XMPP are used for M2C and AMQP is used for S2S communication.

Session layer protocols are closely related with the transport type. Session layer protocols use either UDP or TCP for the transport. DDS and CoAP support both UDP and TCP.

6.3.3 Layered View

Various reference architectures have been provided for the IoT which is usually represented as a layered architecture with various set of layers. Hereby, a layer simply represents a grouping of modules that offers a cohesive set of services. Based on the literature we provide the reference architecture as shown in Figure 2-1.

The reference architecture consists of four layers including device/datalink layer, network layer, session layer, and application layer. The device layer includes the capabilities for the things in the network. The network layer provides functionality for networking connectivity and transport capabilities. The IoT layer consists of functionality for generic support capabilities (such as data processing or data storage), and specific support capabilities for the particular applications. The application layer contains the IoT application.

The Security layer is a side-car layer relating to the other four layers, and pro-vides the security functionality. Finally, the management layer supports capabilities such as device management, local network topology management, and traffic and congestion management.

6.3.4 Deployment View

Figure 6-2 shows the deployment view of IoT-based systems. In essence we can identify two distinct nodes, the IoT node and the Product Cloud node. The IoT Node includes modules for sensors, actuators, smart UI and applications. Within the IoT network multiple IoT nodes can exist which is shown with the asterisk symbol (*). The cloud node includes functionality for data storage, application platform, the analytics engine and the cloud applications. Again, we could have more than one cloud node.

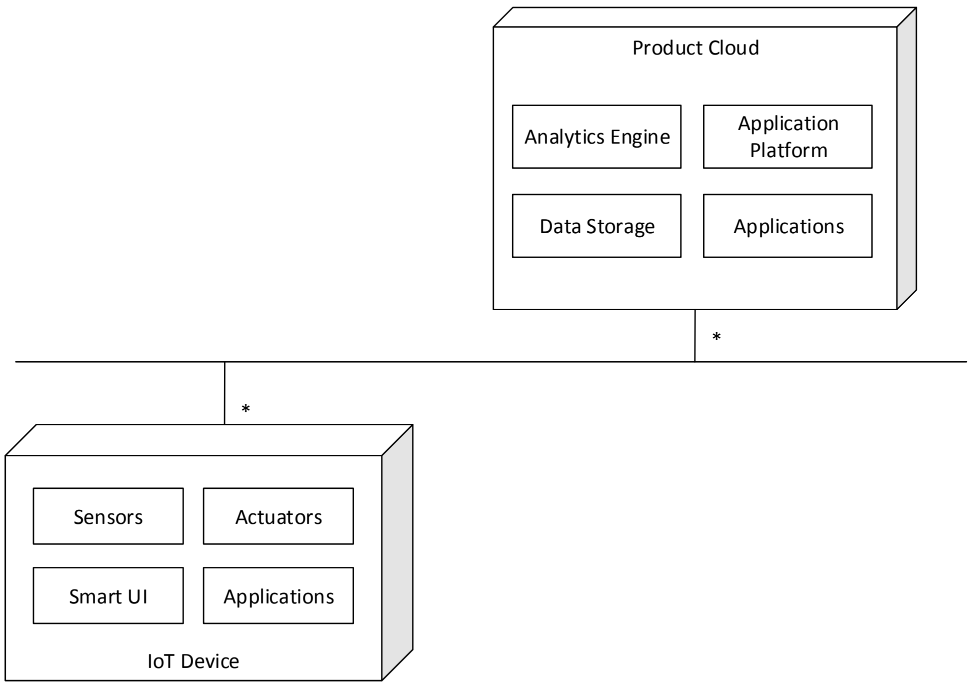


Figure 6-2: Deployment view of the IoT architecture

6.4 DATA DISTRIBUTION SERVICE

Data Distribution Service for Real Time System (DDS) is standardized by Object Management Group (OMG) (OMG, n.d.-b) in 2004 and the latest release is submitted in 2015 (OMG, 2015b). DDS is a data centric middleware for high performance ma-chine-to-machine communications. In this section, we describe the basic back-ground information for Data Distribution Service (DDS). Detailed information about DDS can be found in many references in the literature (OMG, n.d.-c, 2014, 2015b).

6.4.1 Conceptual View

Figure 5-1 shows the conceptual model for DDS middleware. In the figure, the concept Domain is a logical concept which represents the set of applications that can communicate with each other. Several domains can be defined within the same DDS system in order to indicate different set of applications communications with each other. One or more domain participants might exist in each do-main. Domain participants represent the local membership of the application to the assigned domain. Publishers are responsible from data production and up-dates. Publishers include one or more Data Writers that publish different type of data. Similarly, subscribers are responsible of receiving published data and making it available to the participant. A subscriber includes one or more Data Readers to access published data in a type-safe manner. Domain participants might include one publisher and one subscriber at most. The communication between da-ta readers and data writers is established via Topics. A topic defines a unique name, data type and a set of Quality Services to the published/subscribed data. Publishers write the data to the topics and subscribers read the data in topics.

Communication between applications can only be realized only if the topic names and the defined Quality of Service (QoS) parameters match. DDS pro-vides the ability to attach QoS parameters to all these entities in order to specify the behavior of a service such as rate of publication, rate of subscription, how long the data is valid, etc. QoS are also useful for several quality factors such as reliability, durability and scalability which simplify complex network programming.

6.4.2 Feature Model

Based on a thorough domain analysis to DDS middleware systems we have derived a feature model that is shown in Figure 6-3. The figure represents the feature model for Publish-Subscribe Systems. The DDS concepts are shown in bold. In general, publish-subscribe middleware systems can be distinguished based on the type and the service model. Regarding the type, we can identify data-centric, message-centric or object-centric approaches. In the message-centric approach, the middleware is not aware of the content of the data; it is just responsible for transmitting the messages among participants. In data-centric approach, the middleware is aware of the content and can impose quality of service parameter values on the data. In object-centric approaches the middleware is responsible of transmitting objects among participants. As shown in the figure DDS is a data-centric approach.

The service model of publish-subscribe middleware can be characterized based on (1) Communications Model, and (2) Architecture Model. Communication Model defines communication approach that is applied by the participants. The communication approach on its turn can be based on data distribution, shared data, queuing, and remote procedure call. The Architecture Model of a middle-ware can be either centralized or decentralized denoting whether the data flows through a central unit or not. Further, the architecture model can include a broker that manages the data flow. The architecture can be unbrokered, i.e. there is no broker defined, or multi-brokered, whereby multiple brokers manage the data flow. As shown in the figure, the architecture model for DDS is decentralized and unbrokered.

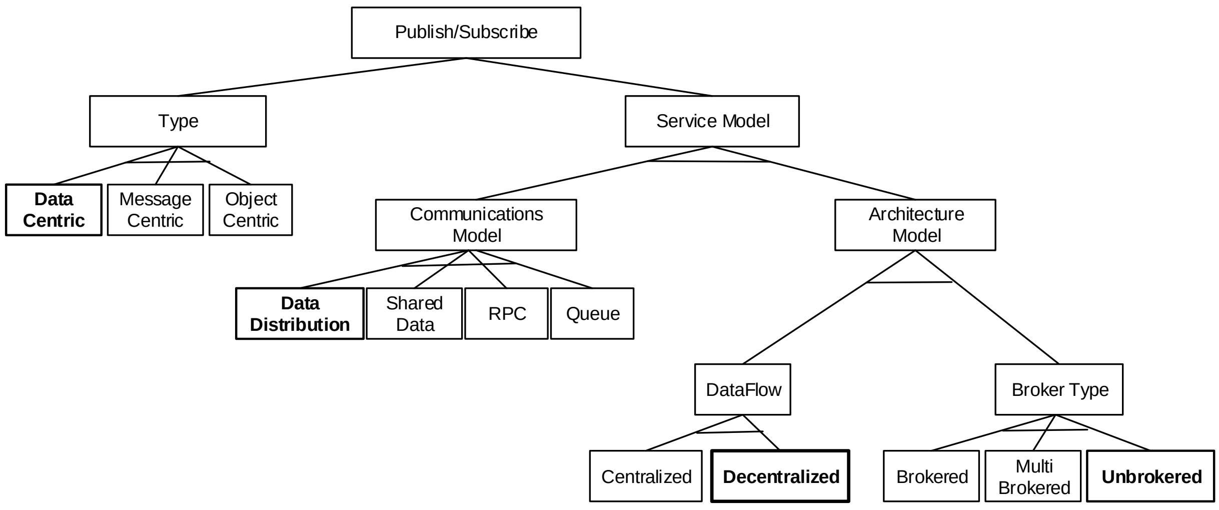


Figure 6-3: Feature model of Publish-Subscribe systems (DDS components highlighted)

6.4.3 Layered View

The DDS can be modeled as a three-layer structure as shown in Figure 5-2. Data Centric Publish Subscribe (DCPS) layer provides efficient delivery of the shared information to the related recipients. DCPS layer in the specification and it is mandatory for the DDS implementations.

Optional Data Local Reconstruction Layer (DLRL) enables simple integration of the services defined in DCPS layer into the application layer. The aim of this is to provide a seamless integration with object-oriented language constructs.

Finally, an additional specification DDS Interoperability Wire Protocol is pro-vided, which is needed for supporting the interoperability among different DDS implementations.

The last layer shown in the Figure 5-2 is related to the transport. DDS might use both UDP and TCP in the transport layer. But DDS also supports UDP and multicast UDP. In fact, one of the powerful features of the DDS is supporting multicast UDP that enables high performance Machine-to-machine communication. On the other hand, since multicast and UDP transports are not supported by many Wide Area Networks (WAN), some additional concepts like Interconnection Services or Routers shall be used in DDS systems to assure end-to-end QoS in WANs (Köksal & Tekinerdogan, 2017b). For further details about these specifications we refer to OMG DDS Specifications (OMG, 2015b).

6.4.4 Deployment View

A typical DDS based system is deployed on a number of Application Nodes. As stated before, publish-subscribe interaction pattern has been applied in several applications and infrastructures, which share similar structure and concepts. Figure 6-4 shows the result of a domain analysis to publish-subscribe systems and represents the deployment view of DDS based systems. Please refer to section “1.4.1 DDS conceptual model” for detailed information about DDS concepts (such as publishers, subscribers, topics, etc.).

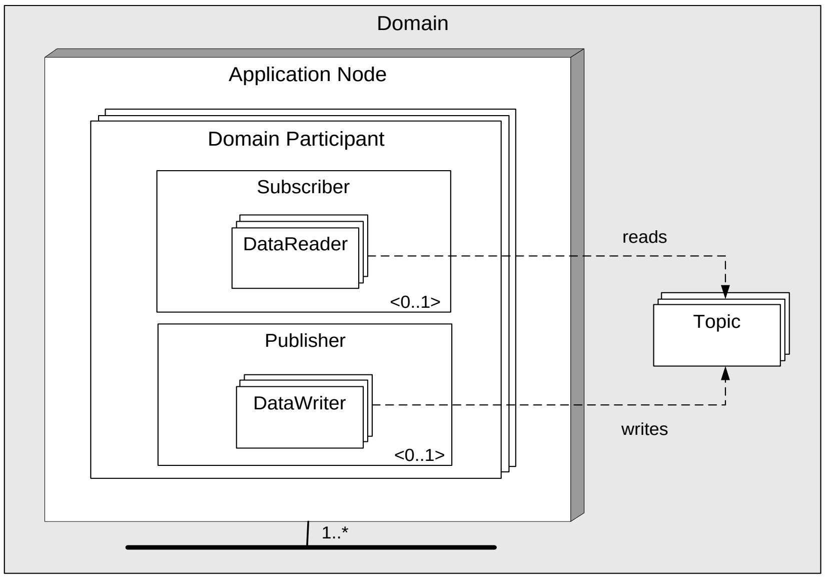


Figure 6-4: Deployment view for DDS-based systems

Defining the deployment view of a DDS based system is a crucial step in de-sign. The deployment model defined determines the allocation of domain participant instances through-out the available physical resources such as available memory and computing power. Although many different deployment alternatives can be defined readily, designing the deployment extremely affects the performance of the overall system.

Sometimes, it is possible to deploy all domain participants (publishers and sub-scribers) to the same node. But such a deployment design cancels the benefits of distributed computing causing single point of failure. On the other extreme, deploying domain participants has many side-effects such as increasing communication overhead and inefficient use of resources. So, it is always advised to analyze the domain participants' communication structure through topics and designing the deployment model accordingly.

6.5 DDS-BASED IOT ARCHITECTURE

In this section, we will present the architecture for DDS-based IoT systems. For this, in section 1.5.1 we will first present the conceptual model that shows the integration of the earlier conceptual models for DDS and IoT. Subsequently, we will present the layered view in section 1.5.2 and deployment view in section 1.5.3.

6.5.1 Conceptual model

Figure 6-5 shows the conceptual model for the DDS-based IoT architecture. Similar to the IoT conceptual model as shown in Figure 6-1 the concept IoT Device can be a Sensor, Tag or Actuator which observe, identify or act on an IoT Thing. A thing has a virtual representation. The DDS concepts Publisher, Subscriber, DataWriter and DataReader are located in the Virtual Entity. Services, that is, Topics in DDS are thus associated with these elements. Domain Participants can include a number of Virtual Entities. Similar to DDS a DDS Entity can specify QoS parameters.

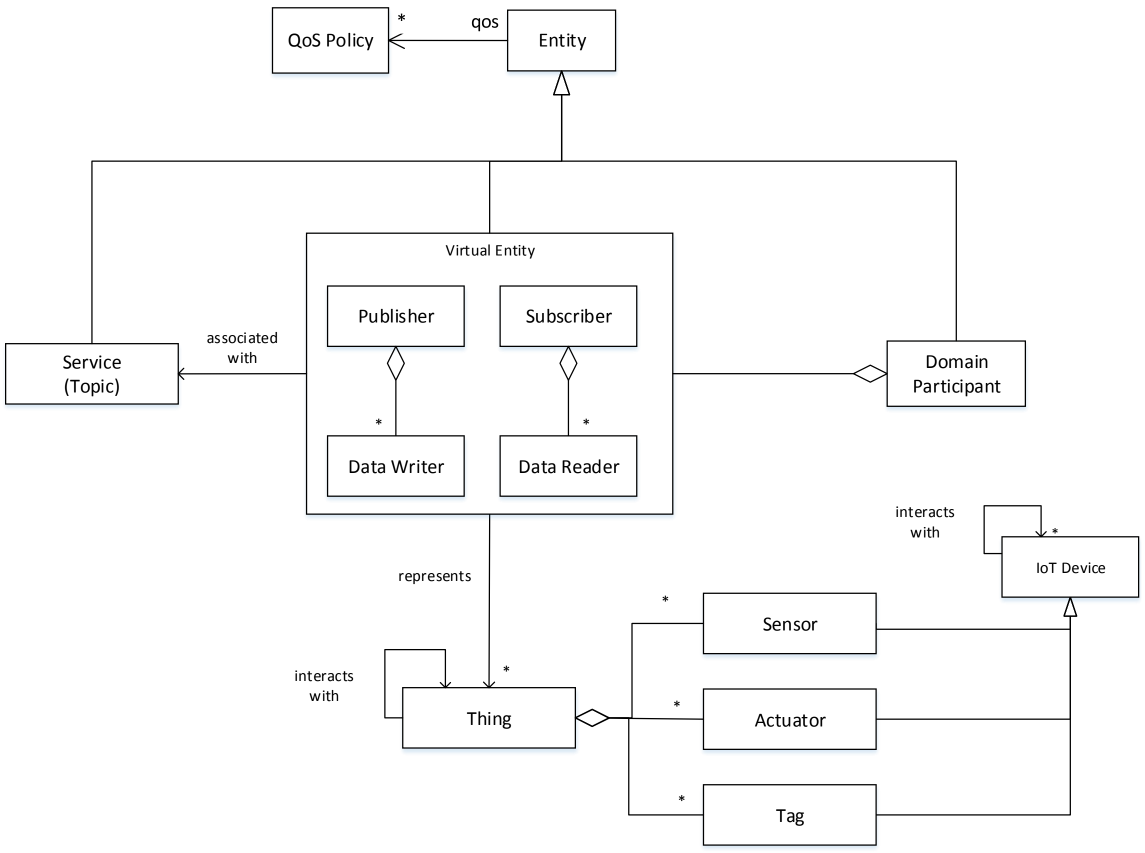


Figure 6-5: Conceptual model for Publish-Subscribe based IoT systems

6.5.2 Layered View

Figure 6-6 shows the layered view that combines the layered view of DDS with that of IoT. The dominant decomposition is taken from the IoT reference architecture as defined earlier in Figure 2-1. Hence the layers are similar to the IoT layers. What is specific is the Session Layer which now includes the concepts of DDS including DLRL, Data Centric Publish Subscribe, and DDSI (OMG, 2014).

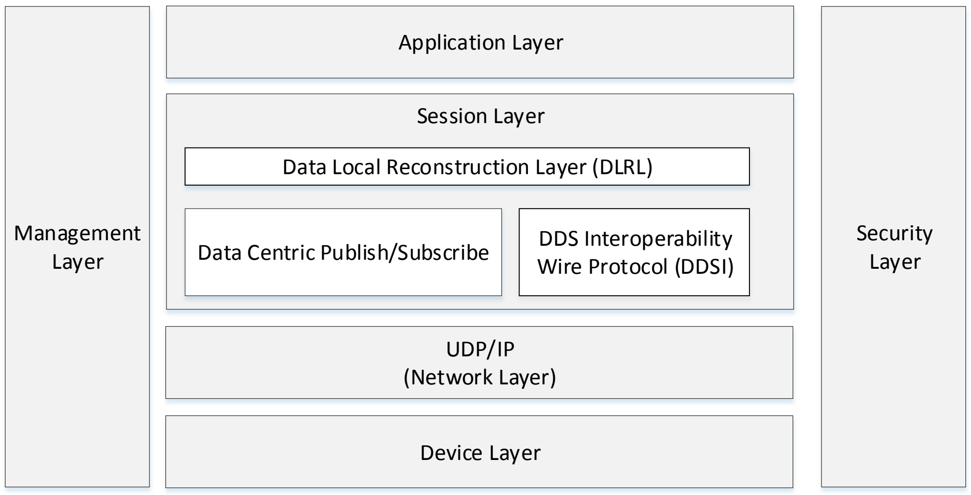

Management Layer
Application Layer
Session Layer
Data Local Reconstruction Layer (DLRL)
Data Centric Publish/Subscribe
DDS Interoperability Wire Protocol (DDSI)
UDP/IP (Network Layer)
Device Layer
Security Layer

Figure 6-6: Layered view for DDS-IoT systems

6.5.3 Deployment View

Figure 6-7 shows the layered view for the DDS-IoT system. In essence, it defines two different nodes, that is, the IoT Node and the Product Cloud Node. The IoT Node will now communicate using the DDS. Hence it includes an Application module that realizes the DDS concepts. That it, it includes the domain participants and herewith the subscribers and publishers. The Product Cloud Nodes is similar to the IoT deployment model.

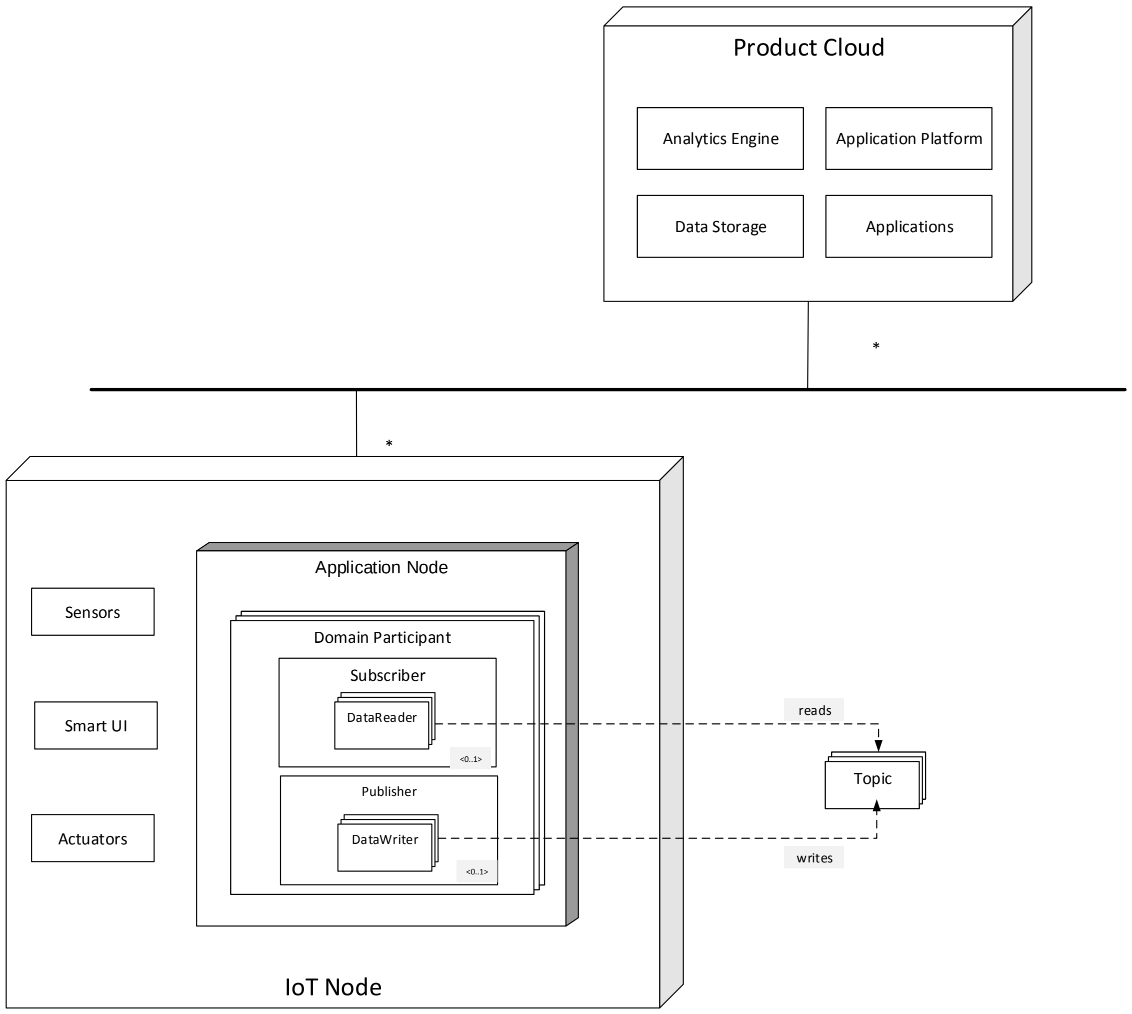


Figure 6-7: Deployment view for DDS-IoT systems

6.6 CONCLUSION

The IoT has now become an important paradigm that is invasive in different application domains. One of the important issues for the IoT is the management of communication and distribution aspects. To support the communication among the different DDS nodes it is important to adopt a feasible middleware. In this context, the DDS is considered as a potential middleware for IoT because of its focus on event-driven communication in which quality of service is also explicitly defined. Research on both paradigms, that is IoT and DDS, have so far been carried almost independently. In recent years, we now observe a growing interest in the application of DDS for IoT.

The results of our study can be considered from this perspective. Our main focus in this chapter was on the architecture design of DDS-based IoT systems. So far, no systematic approach has been provided yet to model the architecture for DDS-based IoT. We have performed a systematic approach in which we adopted architecture viewpoints for modeling DDS, IoT and finally DDS-based IoT systems. Since both the DDS and IoT are often represented as layered structures we have applied the layered viewpoint to represent the DDS-based IoT. Further we have also defined the deployment view for DDS-IoT. We can state that we succeeded to integrate and represent the architecture models that can be used to model DDS-based IoT systems for various application domains. In our future work, we will enhance our study for adopting other architecture viewpoints. In addition, we will adopt the viewpoints for real world industrial IoT projects in which DDS is applied.

7

DERIVING DATA DISTRIBUTION SERVICE BASED FEASIBLE CONFIGURATION ALTERNATIVES[1]

[1] This chapter is based on the following published papers:

- B. Tekinerdogan, T. Çelik, and Ö. Köksal, "Generation of feasible deployment configuration alternatives for Data Distribution Service based systems," Computer Standards and Interfaces, vol. 58, pp. 126–145, May 2018.
- T. Celik, Ö. Köksal, and B. Tekinerdogan, "Deploy-DDS: Tool Framework for Supporting Deployment Architecture of Data Distribution Service based Systems" in Proceedings of the 2014 European Conference on Software Architecture Workshops - ECSAW '14, 2014, pp. 1–5.

Abstract

Data Distribution Service (DDS) has been defined by the OMG to provide a standard data-centric publish-subscribe programming model and specification for distributed systems. DDS has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. To support the analysis and design of a DDS-based distributed system, the OMG has proposed the DDS UML Profile. A DDS-based system usually consists of multiple participant applications each of which has different responsibilities in the system. These participants can be allocated in different ways to the available resources, which leads to different configuration alternatives. Usually, each configuration alternative will perform differently with respect to the execution and communication cost of the overall system. In general, the deployment configuration is selected manually based on expert knowledge. This approach is suitable for small to medium scale applications but for larger applications this is not tractable. In this chapter, we provide a systematic approach for deriving feasible deployment alternatives based on the application design and the available physical resources. The application design includes the design for DDS topics, publishers and subscribers. For supporting the application design, we propose a DDS UML profile. Based on the application design and the physical resources, the feasible deployment alternatives can be algorithmically derived and automatically generated using the developed tools. We illustrate the approach for deriving feasible deployment alternatives of smart city parking system.

***Keywords**:* Data Distribution Service (DDS), Software Architecture Analysis, Design Optimization, Model-Driven Development, Feasible Deployment, Middleware, Research Tool.

7.1 INTRODUCTION

Distributed systems realize the distributed execution of software systems over multiple resources to meet different requirements and quality factors such as performance, interoperation, and multi user support. To reduce the effort for developing distributed systems, common architectures have been introduced including OMG Common Object Request Broker Architecture (CORBA) (OMG, 2012a), Java Message Service (JMS) (Juneau, 2013), and OMG Data Distribution Service (DDS) (OMG, 2015b). These middleware architectures provide common services such as name and directory services, discovery, data exchange, synchronization, and transaction services.

Data Distribution Service (DDS) has been defined by the OMG to provide a standard data-centric publish-subscribe programming model and specification for distributed systems. DDS has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. A DDS-based system usually consists of several applications having different responsibilities in the system. These participants can be allocated in different ways to the available resources, which leads to different configuration alternatives. Usually, each configuration alternative will perform differently with respect to the execution and communication cost of the overall system. In general, deployment configuration is selected manually which is suitable for small to medium scale applications but for larger applications this is not tractable. The OMG DDS specification does not provide an explicit approach to guide the distribution and allocation of the participants to optimize the deployment configuration with respect to performance. Deployment configuration is usually selected manually which is suitable for small to medium scale applications but gets intractable when larger applications are considered.

In this chapter, we provide a systematic approach for deriving feasible deployment alternatives based on the application design, the available physical resources and the execution configuration parameters. The application design includes the identified topics, the number and type of DDS publishers and subscribers. In the approach, first the application design including DDS topics, publishers and subscribers as well as the available physical resources are designed. The application design of these elements is supported by the DDS UML profile that we have extended to support the generation of feasible deployment alternatives.

The resulting design is used to define alternative execution configurations that refine the number and parameters of the corresponding design elements. Based on the application design, available physical resources and the execution configuration, feasible deployment alternatives can be algorithmically derived. The presented approach is supported by corresponding tools that support the application design, the execution configuration definition and the automatic generation of feasible deployment alternatives using model-driven development techniques. We illustrate the approach for deriving feasible deployment alternatives of a smart city parking system.

The remainder of the chapter is organized as follows. In section 7.2, we provide the background on architecture of DDS and designing DDS based systems. Section 7.3 defines the DDS UML profile. Section 7.4 defines the problem statement. Section 7.5 presents the approach for evaluating alternative design options with the adopted models and algorithmic solutions for the approach. Section 7.6 presents the tools that support the approach. Section 7.7 provides the evaluation of the outputs of our approach. Section 7.8 provides discussion. Section 7.9 describes the related work and finally we conclude this chapter in section 7.10.

7.2 Background and Content

In this section, we describe the background for understanding and supporting the approach that we present in this chapter. In section 7.2.1 we present the deployment view for DDS-based systems, followed by a discussion in section 7.3 on the proposed DDS UML profile.

7.2.1 Deployment View for DDS-based Systems

Based on DDS specification (OMG, 2015b), we could derive the deployment view for DDS based systems, as shown in Figure 6-4. A DDS system consisting of several DDS applications is called a Domain. A typical DDS based system is deployed on a number of Application Nodes. Each Application Node includes one or more Domain Participants, which are applications that together form the system execution. Each Domain Participant may include one Publisher that represents the objects responsible for data production and updates. A publisher includes one or more Data Writers that publish data of different data types. Domain Participant may also include one Subscriber that is responsible for receiving published data and making it available to the participant. A subscriber includes one or more Data Readers to access published data in a type-safe manner. Interaction between data reader and data writers is established via Topics. A topic defines a unique name, data type and a set of Quality Services to the published/subscribed data (OMG, 2015b). Note that Domain is a logical concept and a Domain Participant may participate in more than one domain at the same time.

The DDS specification defines two layers: (1) A lower level layer, which provides efficient delivery of the shared information to the related recipients. This layer is named Data Centric Publish Subscribe (DCPS) in the specification and it is mandatory for the DDS implementations. (2) A higher layer that enables simple integration of the services defined in DCPS layer into the application layer. This layer is named Data Local Reconstruction Layer (DLRL) in the specification and it is optional to be provided by the DDS implementations.

DDS provides the ability to specify various parameters like the rate of publication, rate of subscription, how long the data is valid, and many others. These Quality of Service (QoS) parameters allow system designers to construct distributed applications based on the requirements for, and availability of, each specific piece of data. Selected QoS parameters affect the performance of the overall system drastically, and therefore finding the feasible values for the QoS parameters for a system is important for successful development of the target system.

7.3 DDS UML PROFILE

To support the analysis and design of object-oriented systems using DDS technology, the OMG has specified the UML Profile for Data Distribution Specification (OMG, 2010). The profile enables definition of all DDS artifacts defined in the view given in Figure 6-4. This profile also enables the definition of DDS data types which topics will be built on. The profile separates DDS artifacts in three packages including DCPS, DLRL, and DDS Common. The DCPS defines the mandatory part of the DDS specification used to provide the functionality required for an application to publish and subscribe to the values of data objects. The DLRL is the optional portion of the DDS specification used to provide the functionality required for an application for direct access to data exchanged at the DCPS layer. The DDS Common package defines the distributed data communications specification that allows Quality of Service policies to be specified for data timeliness and reliability. The dependencies between the packages are shown in Figure 7-1. The figure indicates that the DCPS and DLRL packages depend on DDS Common. Several tools that implement the draft specification of the above UML Profile for Data Distribution Specification are already available and ready to be used such as Enterprise Architect (Sparx Systems, n.d.).

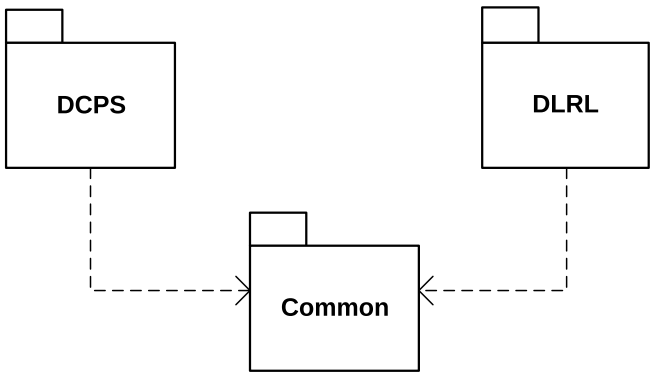


Figure 7-1: Top-Level DDS package structure of the proposed OMG UML Profile

In this section, we define a case study that will be used to illustrate the problem statement and the approach in further sections. The case study that we consider is within the context of smart city engineering (Yoshikawa et al., 2012). For the near future, it is expected that a big part of the world population will live in urban areas. This will have a huge impact on future personal lives and mobility. A smart city uses information and communication technology (ICT) to enhance the quality and performance of urban services, to reduce costs and resource consumption, and to engage more effectively and actively with its citizens (Iijima, 2012; Yoshikawa et al., 2012). Sectors that have been developing smart city technology include government services, transport and traffic management, water and waste, health care, and energy. Smart city applications are developed with the goal to improve the management of urban flows and allowing for real time responses to challenges. One of the important applications in smart city engineering includes the development of smart traffic system (STS). Traffic is already a large problem in many cities and this problem will be even bigger in the future. Many people spend a considerable amount of time in traffic, which leads to unnecessary waste of human resource, time and increase of CO2 emissions. STS provides different capabilities such as traffic light management, congestion detection, traffic regulation, shared parking platform, etc. For example, shared parking platform optimizes the search for finding a suitable parking slot by guiding the drivers to the available nearest parking spots in real-time.

The high-level reference architecture of STS is depicted in Figure 7-2. STS consists primarily of sensors and vehicles. Sensors are the devices that monitor the environment and provide the corresponding data. Vehicles use the sensor data and publish their position and other relevant information to the STS. Within the case study we distinguish between the following sensor types: Traffic Light, Incident Detector, Congestion Detector, Speed Camera, Parking Detection Sensor, Bicycle Station, Parking Lot, and Weather Sensor. Vehicles can be of the following types: Car, Truck, Ambulance, Taxi, Bicycle, and Bus. The sensors and control units are thin clients which do not contain any business logic. In this case, all the STS elements can communicate with the STS.

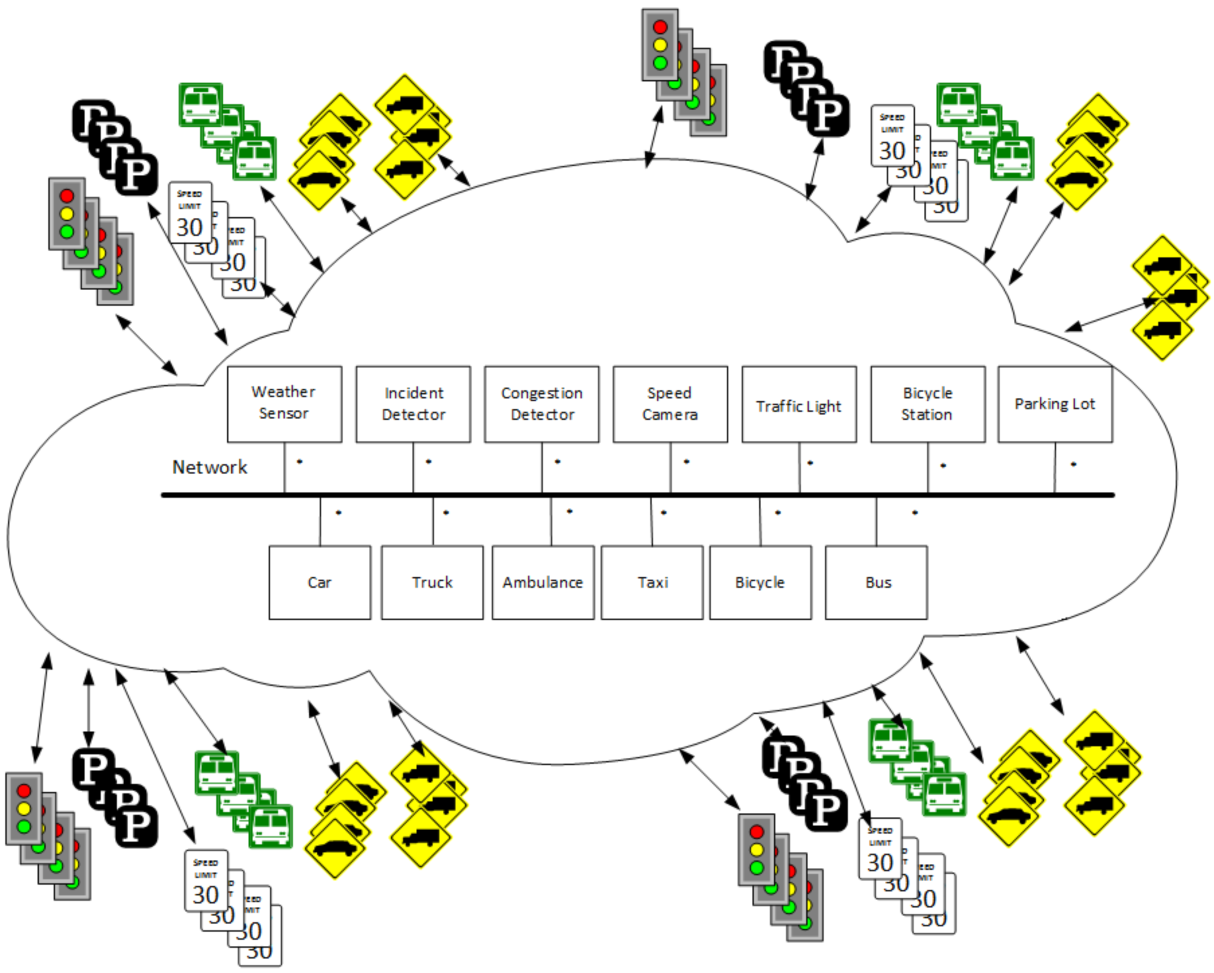


Figure 7-2: High Level Reference Architecture of the Smart City case study

STS is in essence a data-intensive system with stringent demands for QoS parameters. As stated before, the OMG's DDS Middleware explicitly considers QoS properties and as such is very suitable to realize the STS system. In order to implement STS using DDS we need to map the application domain (smart city) concepts to the DDS concepts, that is, domain, the domain participants, the publishers, the subscribers, and the topics in the STS case study. The DDS concept domain is here the Smart City Traffic Domain. Domain participants might be grouped as vehicles, sensors and managers. Managers define the domain participants that include the communication and business logic necessary for executing the required services. As stated before each domain participant can have zero or one publisher and zero or one subscriber. The subscribers and publishers for each domain participant are given in Table 7-1.

Table 7-1: Corresponding DDS Names for Application Domain Participants for STS

Application Domain Name	DDS Name	Publisher	Subscriber
Ambulance	dpAmbulance	dpAmbulancePub	dpAmbulanceSub
Bicycle	dpBicycle	dpBicyclePub	dpBicycleSub
Bus	dpBus	dpBusPub	dpBusSub
Car	dpCar	dpCarPub	dpCarSub
Taxi	dpTaxi	dpTaxiPub	dpTaxiSub
Truck	dpTruck	dpTruckPub	dpTruckSub
Bicycle Station	dpBicycleStation	dpBicycleStationPub	-
Congestion Sensor	dpCongestionSensor	dpCongestionSensorPub	-
Incident Sensor	dpIncidentSensor	dpIncidentSensorPub	-
Parking Lot	dpParkingLot	dpParkingLotPub	-
Speed Camera	dpSpeedCamera	dpSpeedCameraPub	-
Traffic Light	dpTrafficLight	dpTrafficLightPub	-
Weather Sensor	dpWeatherSensor	dpWeatherSensorPub	-
Incident Manager	dpIncidentManager	dpIncidentManagerPub	dpIncidentManagerSub
Logger Manager	dpLoggerManager	-	dpLoggerManagerSub
Parking Manager	dpParkingManager	dpParkingManagerPub	dpParkingManagerSub
Ticket Manager	dpTicketManager	dpTicketManagerPub	dpTicketManagerSub
Traffic Manager	dpTrafficManager	dpTrafficManagerPub	dpTrafficManagerSub
Vehicle Manager	dpVehicleManager	dpVehicleManagerPub	dpVehicleManagerSub
Weather Manager	dpWeatherManager	dpWeatherManagerPub	dpWeatherManagerSub

For example, the entity Car has a corresponding domain participant dpCar, which as a publisher dpCarPub and a subscriber dpCarSub. In a similar sense the subscribers for each domain participant are defined. Finally, we have defined eight different topics for the case study (Table 7-2). In this table we can, for example, see that publisher SpeedCameraPub publishes data in the topic Ticket Info Topic with publish frequency rate 5Hz. Similarly, the two subscribers TicketManSub and VehicleManSub read the published data. Table 7-3 shows an example scenario for STS including the defined number of instances per domain participant.

Table 7-2: Topics of Sample Scenario for Smart Parking System (STS)

Topic Name	Publisher	Publisher Rate [Hz]	Subscriber
Vehicle Info Topic	dpCarPub dpBusPub dpTruckPub dpAmbulancePub dpTaxiPub dpBicyclePub	5 5 5 5 5 5	VehicleManSub
Global Info Topic	VehicleManPub	20	dpCarSub dpBusSub dpTruckSub dpAmbulanceSub dpTaxiSub dpBicycleSub
Traffic Info Topic	TraficLightPub CongestionSensorPub	10 10	TrafficManSub VehicleManSub
Ticket Info Topic	SpeedCameraPub	10	TicketManSub VehicleManSub
Weather Info Topic	WeatherSensorPub	10	WeatherManSub TrafficManSub VehicleManSub
Parking Info Topic	BicycleStationPub ParkingLotPub	10	ParkingManSub VehicleManSub
Incident Info Topic	IncidentSensorPub	10	IncidentManSub VehicleManSub
Logger Topic	TrafficManPub TicketManPub WeatherManPub ParkingManPub IncidentManPub	20 20 20 20 20	LoggerManSub

7.4 PROBLEM STATEMENT

An important step of designing a DDS-based application is to define the deployment model of the system. The deployment model defines the allocation of domain participant instances (e.g. scenario of Table 7-3) to the available physical resources, and largely influences the performance of the overall system.

In principle many different deployment alternatives can be defined. For example, a deployment alternative of the STS can be defined with three nodes in which all vehicle instances are deployed on the first node, sensor instances are on the second node and manager instances are deployed on the third node as given in Figure 7-3.

Table 7-3: Example scenario for STS with defined no. of instances per domain participant

Domain Participant Name	**Number of Instances**
Ambulance	17
Bicycle	184
Bus	46
Car	1435
Taxi	124
Truck	28
Bicycle Station	23
Congestion Sensor	62
Incident Sensor	29
Parking Lot	33
Speed Camera	48
Traffic Light	125
Weather Sensor	16
Incident Manager	15
Logger Manager	15
Parking Manager	15
Ticket Manager	15
Traffic Manager	15
Vehicle Manager	15
Weather Manager	15
TOTAL	**2275**

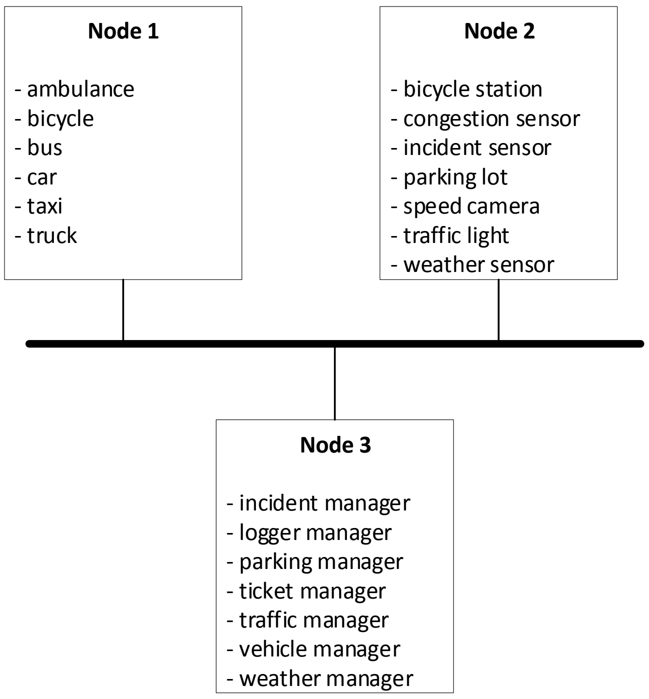


Figure 7-3: Deployment by grouping domain participants

Actually, this alternative follows the conceptual separation of concerns in which a separate node is logically defined almost for each participant type. Further, the communication overhead among the same participant types such as the communication between Vehicle Manager and Traffic Manager are minimized because of being deployed on the same node.

Although this alternative is easy to understand because of the logical separation of participants, it does not always have good time performance because separately deployed participants such as parking sensors and vehicles need to interact very frequently with each other.

A second deployment alternative example is shown in Figure 7-4. Hereby, the 20 participants have been distributed equally over the existing 10 nodes. That is, each node has been allocated two types of participants. This deployment alternative is simple but might not be feasible to minimize the network communication in case participants need to communicate with participants in other nodes.

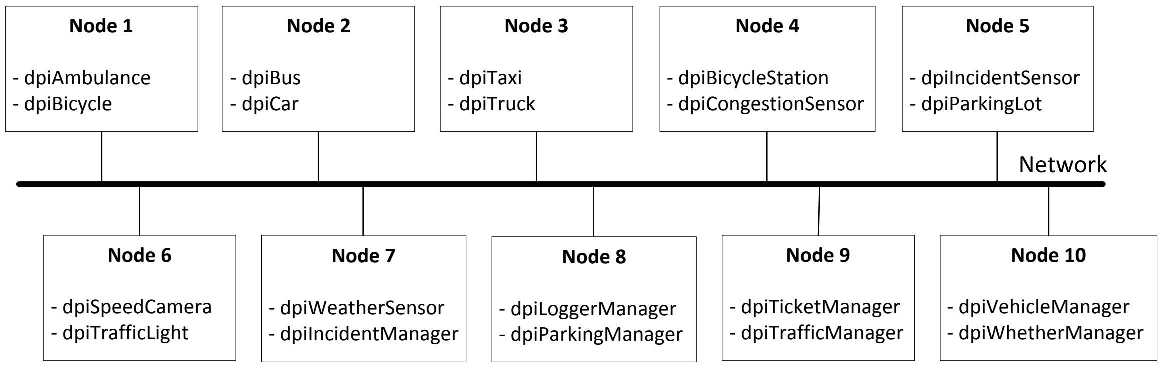


Figure 7-4: Deployment by distributing domain participants over nodes.

We can derive many more different deployment alternatives that may differ with respect to the number of deployment nodes and the mapping of participants to the nodes. Apparently, the number of deployment alternatives is very large and each deployment alternative will perform differently with respect to different quality considerations such as logical separation for understandability, optimizing communication overhead, enhancing utilization of physical resources, etc. Obviously, a more systematic and formal approach is required to guide the search for the feasible deployment alternatives. The OMG DDS specification does not provide an explicit approach to guide the distribution and allocation of the participants to optimize the deployment model with respect to performance in the design phase. Moreover, currently there is no adequate approach and tool support yet to enable the selection of deployment alternatives in the literature. In the following sections, we will provide an approach and tool framework for designing the DDS-based application and deriving feasible deployment alternatives.

7.5 Approach For Generating DDS Deployment Configuration Alternatives

In this section, we provide a systematic process for defining and evaluating feasible deployment alternatives of a DDS-based distributed system. The presented approach will be used in the design phase of the DDS-based system where the development of the system is not started yet, and the system code is not available. The approach is represented as an activity diagram as shown in Figure 7-5. The approach consists of the two basic phases "Architecture Design" and "Feasible Deployment Generation".

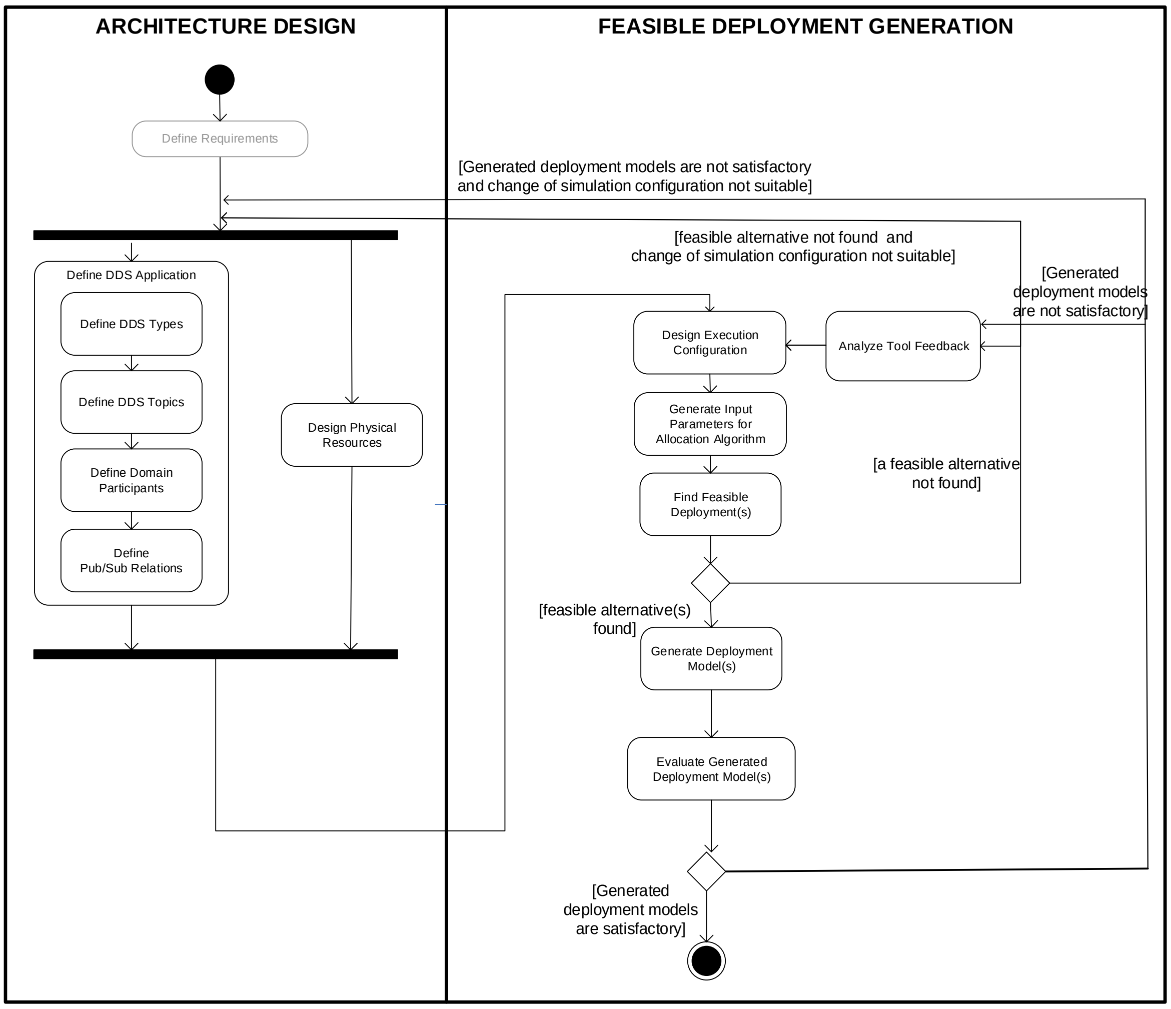

ARCHITECTURE DESIGN
FEASIBLE DEPLOYMENT GENERATION
Define Requirements
[Generated deployment models are not satisfactory and change of simulation configuration not suitable]
[feasible alternative not found and change of simulation configuration not suitable]
[Generated deployment models are not satisfactory]
Define DDS Application
Define DDS Types
Define DDS Topics
Define Domain Participants
Define Pub/Sub Relations
Design Physical Resources
Design Execution Configuration
Analyze Tool Feedback
Generate Input Parameters for Allocation Algorithm
Find Feasible Deployment(s)
[a feasible alternative not found]
[feasible alternative(s) found]
Generate Deployment Model(s)
Evaluate Generated Deployment Model(s)
[Generated deployment models are satisfactory]

Figure 7-5: Activity flow of alternative design evaluation and deriving feasible deployment

Typically, the architecture design phase follows the requirements analysis process. We assume that the requirements analysis phase is performed using the approaches as defined in the literature (e.g. see Rational Unified Process (Kruchten, 2000)) and provides the input for the DDS-based system architecture.

The architecture of the DDS application is designed using the DDS UML Profile that has been defined in section 2.2. This includes the definition of the DDS Types, the DDS Topics, the Domain Participants and the Publish/Subscribe Relations. The DDS application will be deployed on the target environment, which consists of physical resources on which the DDS domain participants will execute. The design of the physical resources is defined in parallel to the DDS application design.

After the architectural system design phase is completed, the feasible deployment model generation phase starts with the definition of the execution configuration. The execution configuration defines the number of each DDS domain participant and update rate for each publication by using the artifacts defined in architecture design phase. From an abstract point of view, the feasible deployment models of a system with several sub-components can be derived by using task assignment algorithms defined in the literature (Aleti, Grunske, Meedeniya, & Moser, 2009; Malek, Medvidovic, & Mikic-Rakic, 2012). For using the task assignment algorithms, the required input parameters need first to be defined. These input parameters are extracted from the design including available resources, execution cost of each task, and communication cost among tasks. After the necessary input parameters are extracted, the feasible deployment models are defined and the deployment models are generated. Subsequently, the feasibility of the generated deployment models is evaluated in the following step. If the generated deployment models are not satisfactory, an iteration step will be required to analyze the system design and refine it according to the provided feedback by the corresponding tool. Here a satisfactory alternative defines a deployment alternative that meets the expected improvement rate of the costs (e.g. communication and execution costs) for the deployment model. Finding feasible deployment models may require several iterations of the process steps. The initial deployment model is realized and verified in development and integration/test activities, and the results are fed back to the designer until a satisfactory alternative is derived.

In the following subsections we will explain the concrete activities that we have defined to realize our approach. Each section also defines the metamodels that are used for modeling the related artifacts of the corresponding step.

7.5.1 Define DDS Application

OMG's UML Profile for Data Distribution Specification already defines necessary metamodel for defining a DDS application, so we did not define a new metamodel for DDS Application definition. The approach defined in this chapter extends and realizes the OMG UML Profile for Data Distribution Specification (OMG, 2010). Our modeling tool realizes necessary parts of UML Profile for Data Distribution Specification to define the DDS types, the DDS topics, the Domain Participants, and the Publish/Subscribe Relations. For example, the relationship among Domain, Domain Participant, Publisher, Subscriber, Data Reader, and Data Writer artifacts are shown in Figure 7-6.

The model implies that a DDS application may consist of one or more Domains, a Domain Participant can be member of one or more Domains, a Domain Participant may contain zero or one Publisher/Subscriber, and so on. The attributes of metamodel classes are not shown for the sake of simplicity and can be inspected from the specification (OMG, 2010).

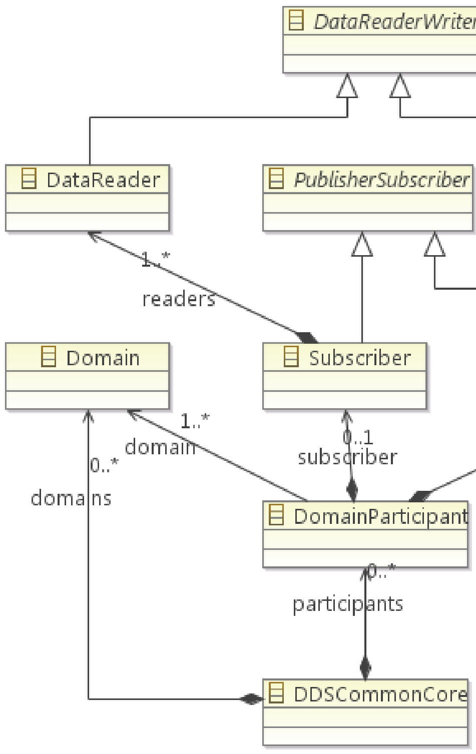


Figure 7-6: Metamodel for DDS UML Profile/DCPS/DCPS Package

7.5.2 Design Physical Resources

Parallel to the activity Define DDS Application, the activity Design Physical Resources defines the available nodes together with their processing power and memory capacity, as well as the network connections among the nodes. For example, one may decide to adopt 25 nodes on which the participants need to be deployed. As an example configuration, it could be decided that each node has a memory capacity of 12280 MB and contains two processing units with four cores at the frequency of 2.3 MHz. Equally, the nodes could also have different memory capacity and computation power.

The physical resource metamodel has not been defined in the UML Profile for Data Distribution Specification. As such, we have developed the metamodel in Figure 7-7 to support the process in Figure 7-5. The Physical Resource Metamodel given in Figure 7-7 can be used to represent the artifacts for modeling the available physical resources.

PhysicalResourceModel is the root class of the metamodel that defines a physical resource model. There can be one or more Nodes in a physical resource model, which represents computation resources. Each node has a name attribute that identifies the node. The powerFactor attribute defines the computation power of the node relative to other nodes. A node can have one or more processors, one or more custom node properties, and memory capacity. Processor defines properties of a processing unit using the attributes name, frequency and coreCount. The attribute name is the symbolic name of the processor like "Intel Core I7". The attribute coreCount defines the number of cores that the processor has. The attribute frequency defines the frequency of the processor in Mhz. MemoryCapacity has a value attribute that represents the memory capacity of the node in terms of megabytes.

CustomNodeProperty can be used to define additional properties for the node. The properties are defined as name-value pairs. For example, one may decide to include a specific property diskCapacity with value 340 Gb.

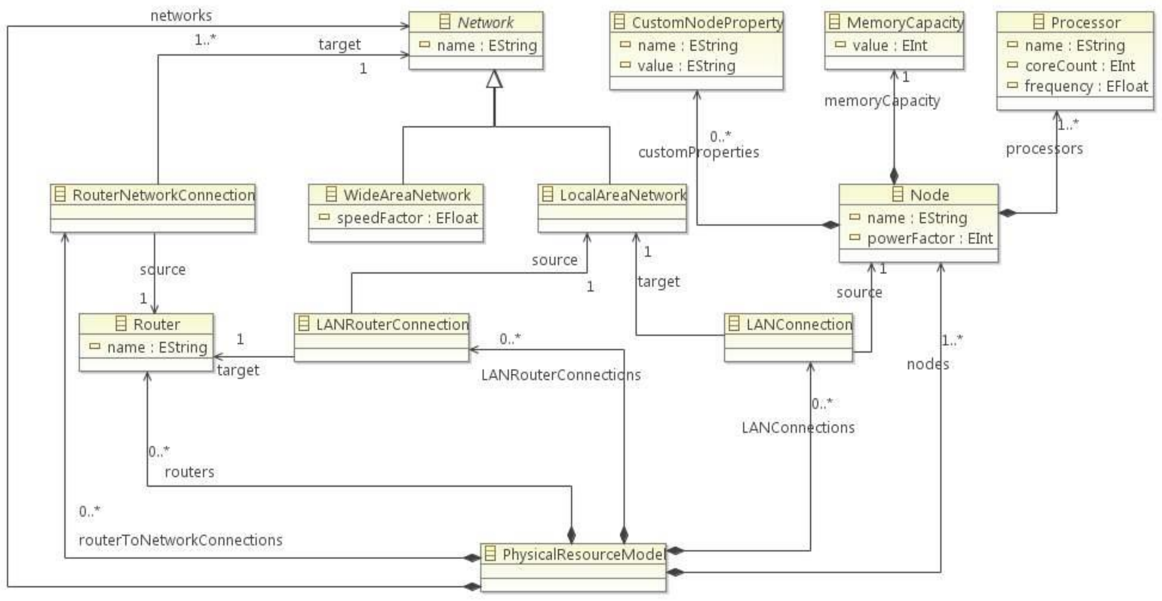


Figure 7-7: Physical Resource Metamodel

There can be one or more networks in a physical resource model. The Network class is the abstract base class for LocalAreaNetwork (LAN) and WideAreaNetwork (WAN) classes. The name attribute of the Network class is the symbolic name of the network. WideAreaNetwork class has speedFactor attribute that defines the speed of the network in comparison with a LAN. LANConnection represents the connection of a node to a LAN. Router represents routers for connecting networks with each other. The name attribute of the Router class is the symbolic name of the router. LANRouterConnection class represents connection of a LAN to a router while the RouterNetworkConnection class represents connection of a router to a network.

7.5.3 Design Execution Configuration

The Execution Configuration Metamodel is used to define the artifacts to model the execution configuration shown in Figure 7-8. ExecutionConfiguration class defines an execution configuration which contains elements of Metadata and DomainParticipantInstance. Metadata defines name, creation date, creator, and version of the execution configuration. DomainParticipantInstance represents an instance of a Domain Participant that is defined in the DDS Application Definition Metamodel.

Each Domain Participant instance can have a different execution cost for different nodes. For this, DomainParticipantInstance contains a list of ExecutionCost that define estimated execution cost for each node which the Domain Participant instance can execute. Note that the execution cost is dependent on the selected execution configuration. For example, the execution cost of a Mobile Client Subscriber model changes according to existing Parking

Detection Sensors in the execution configuration. The execution cost is a scaled value that shows the execution cost of a Domain Participant Instance in comparison with other Domain Participant Instances in the execution configuration. For example, the execution cost for each Parking Detection Sensor domain participant is defined using scaled value and defined as 7 over 20 for one node, 14 over 20 for another node, etc. The execution costs of modules are influenced by the processor's powerFactor and memoryCapacity attributes. In a similar sense, the communication costs among modules are influenced by the networks speedFactor attribute. Since the execution and communication costs of domain participants can only be exactly measured after the system is developed, during design time their values can only be estimated. This estimation can be conducted by using, for example, design phase complexity calculation methods such as proposed by (Prismtech, n.d.-b) or prototyping.

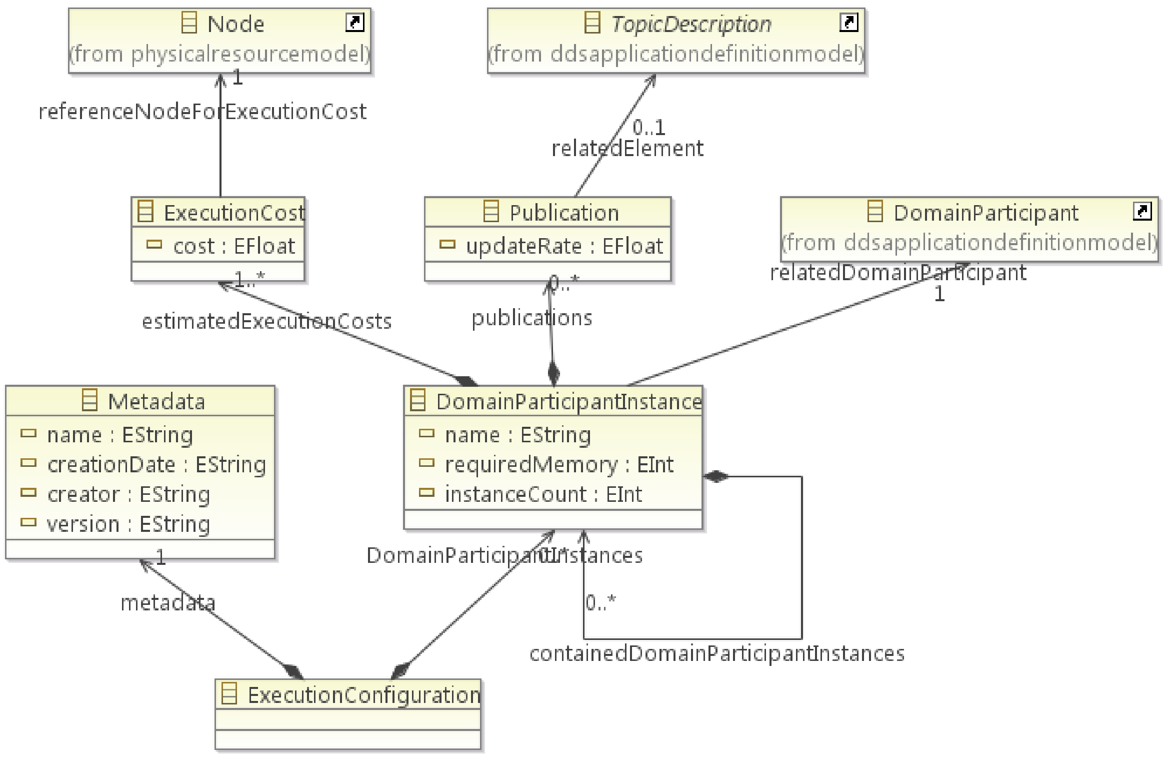


Figure 7-8: Execution Configuration Metamodel

The attribute requiredMemory of DomainParticipantInstance represents the estimated memory amount that the domain participant will require during execution. Similar to the execution cost, this parameter can be estimated in the design phase. The attribute instanceCount defines the number of Domain Participant Instances in the execution configuration. This attribute is added because there may be multiple instances of the same Domain Participant in an execution configuration. For example, in a large Smart Parking System scenario, there can be hundreds of Parking Detection Sensors and it is not feasible to add one domain participant for each of them to the execution configuration separately.

The relation relatedDomainParticipant associates a DomainParticipantInstance with a DomainParticipoant that is defined in the activity Define DDS Application. DomainParticipantInstance can have zero or more Publications that represent the update rate and the related element from DDS Topic definition. Each publication is associated with an TopicDescription defined in “Define DDS Application” step.

The updateRate attribute shows how many times a Domain Participant instance will update a Topic in a second. For example, we could decide to have 2000 Parking Detection Sensor domain participants where each of them publishes a Sensor object with update rate of 2 times per minute.

7.5.4 Generate Input Parameters for Allocation Algorithm

Once the parameters for the physical resources and execution configurations have been defined we can start the search for the feasible deployment alternatives. In principle, this can be carried out in different ways in which multiple different approaches and algorithms can be identified. The allocation could be, for example, based on one of the following heuristics:

1. minimizing the number of the nodes to which the tasks are allocated
2. uniform distribution of tasks over the nodes
3. random allocation of tasks over the nodes
4. minimizing the overall communication costs

The presented approach is generic and does not hardwire a particular heuristic approach. If needed, in addition to the above heuristics we could also identify other heuristics. In the next section, we will discuss each of these approaches in the implementation of the tool and the overall evaluation.

Besides the heuristics, we could also adopt a more formal and systematic algorithm for the deployment process. In this chapter, we will adopt the so-called Multi-Processor Task Assignment (MPTA) problem (Malek et al., 2012; Ucar, Aykanat, Kaya, & Ikinci, 2006). For this problem, the following parameters can be defined:

- T, set of m tasks = {t1, t2, ..., tm}
- P, set of n processors {p1, p2, ..., pn}
- Mp, memory capacity of processor p
- mi, amount of memory needed for task i
- Xiq, cost of executing ti task on pq processor.
- E, set of communication between tasks, whereby each communicating task combination (i, j) has a communication cost cij if tasks ti and tj are assigned to different processors. Communication cost is negligible if two tasks are assigned to same processor.

The objective in our problem is to minimize the sum of total execution cost and total communication cost (among domain participants) while not exceeding the memory capacity of each node. Based on the above definitions we can formulate our objective as follows (Malek et al., 2012; Ucar et al., 2006):

Assign tasks to processors to minimize the sum

$$\sum_{i=1}^{m} \sum_{p=1}^{n} a_{ip} x_{ip} + \sum_{(i,j)\in E} \sum_{p=1}^{n} a_{ip}(1-a_{jp})c_{ij}$$

Subject to

$$\sum_{p=1}^{n} a_{ip} = 1, i \in T$$

$$\sum_{i} m_i a_{ip} \le M_p$$

$$a_{ip} = \{0,1\}, p \in P, i \in T$$

(aip = 1, if task i is assigned to processor p, 0 otherwise)

In fact, the required parameters of the MTPA problem can be extracted from the system design that has been defined in the previous activities. In Table 7-4 we explain for each parameter how it is extracted from the design.

Table 7-4: Extracting MPTA parameters from the design

MPTA Parameter	Extraction from Design
T	Each domain participant instance will be mapped to a Task, so T is list of domain participant instances defined in Execution Configuration Design activity.
P	Each node defined in Physical Resource Design activity
M_p	*memoryCapacity* attribute of node defined in Physical Resource Design activity.
m_i	*requiredMemory* attribute of *DomainParticipantInstance* defined in Execution Configuration Development activity
X_{iq}	*nodeExecutionCostTable* attribute of *DomainParticipantInstance* defined in Execution Configuration Development activity
C_{ij}	Calculated by using: - Publications defined in Execution Configuration Design activity, - Subscriptions defined in Publish/Subscribe Relations of Domain Participants Design activity, - Data Types and Topics defined in DDS Application Design activity

7.5.5 Find Feasible Deployment Configuration

The activity Find Feasible Deployment takes as input the parameter values of the previous activity and executes an algorithm that computes a feasible deployment alternative, if one is available. Different algorithms in the literature can be used to solve the MPTA problem. Please note that we do not focus on a particular algorithm but recommend using a practical one for the corresponding case. In our case, we could for example use the MPTA algorithm as defined by Mehrabi et al. (Mehrabi, Mehrabi, & Mehrabi, 2009) because it adopts the parameters of execution cost, communication cost and memory requirements. If a feasible deployment is found, the output of this activity is a table that represents the mapping of tasks (domain participants) to processors (nodes). If the algorithm was not successful in finding a feasible solution the process returns to the activity Design Execution Configuration. This can be repeated several times until a feasible deployment is found. If it appears that a feasible deployment cannot be found by changing just the execution configuration, then the designer can decide to return to the beginning of step 3 to refine/update the design.

7.5.6 Generate Deployment Configuration

The Deployment Metamodel is used to describe the deployment model in the "Generate Deployment Model(s)" activity shown in Figure 7-9. The deployment metamodel contains Members and Nodes. Each Member is deployed on one of the Nodes defined in Physical Resource Model. One or more Domain Participant Instances can be deployed on a Member.

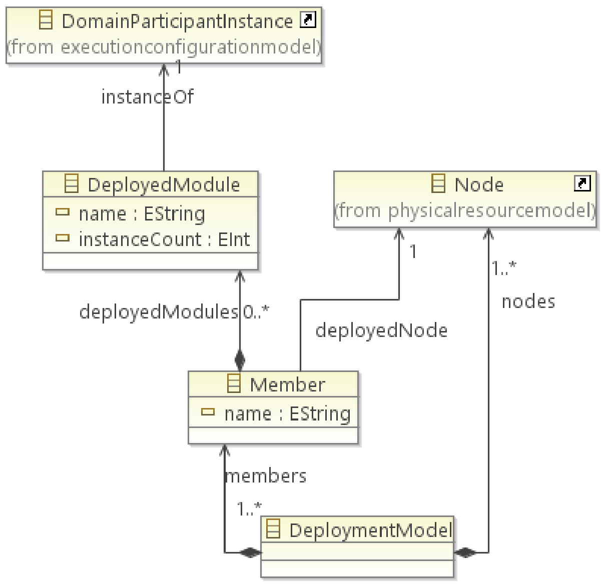


Figure 7-9: Deployment Configuration Metamodel

7.6 Tools and Applying the Approach to the Case Study

In this section, we present the tool Deploy-DDS that provides an integrated development environment for supporting the activities of the approach described in the previous section. Deploy-DDS is built on the Eclipse platform and is implemented as a set of plug-ins. The developed plug-ins are built on other Eclipse frameworks including Eclipse Modeling Framework (EMF) (Steinberg, 2009), and Graphical Modeling Framework (GMF) (Voelter, Kolb, Efftinge, & Haase, 2006). EMF is a modeling framework and code generation facility that we use to develop the metamodels. GMF is a generative component and runtime infrastructure that we use for developing graphical editors for the developed metamodels. Further, we use Emfatic (Daly, 2004), which provides a text editor and a language for editing EMF models. In addition, we use EuGENia GMF tool (Kolovos et al., 2010) that provides mechanisms for abstracting away the complexity of GMF and for easier development of GMF editors. EuGENia tool is a part of Epsilon project (Kolovos, Paige, & Polack, 2006).

In the following subsections, we describe the top-level tool architecture in section 7.6.1. In section 7.6.2 we show the application of Deploy-DDS for designing the DDS Application, Physical Resources, and Execution Configuration for the case study.

7.6.1 Tool Architecture

The Deploy-DDS tool provides an integrated environment for modeling DDS based applications, generating and analyzing deployment models. Deploy-DDS tool is built on the Eclipse platform and is implemented as a set of plug-ins. The developed plug-ins are built on other Eclipse framework plug-ins including Eclipse Modeling Framework (EMF) (Budinsky, Steinberg, Merks, Ellersick, & Grose, 2003), Graphical Editing Framework (GEF) (Moore, Dean, Gerber, Wagenknecht, & Vanderheyden, 2004), and Graphical Modeling Framework (GMF) (Voelter et al., 2006). EMF is a modeling framework and code generation facility that we use to develop the metamodels.

GEF is a framework that is used for generating rich graphical editors and views. GMF is a generative component and runtime infrastructure that we use for developing graphical editors for the developed metamodels. Further, we use Emfatic (Daly, 2004), which provides a text editor and a language for editing EMF models. In addition, we use EuGENia (Kolovos et al., 2010) GMF tool that provides mechanisms for abstracting away the complexity of GMF and for easier development of GMF editors. EuGENia tool is a part of Epsilon project (Kolovos et al., 2006). The layered tool architecture of the Deploy-DDS is given in Figure 7-10. Deploy-DDS consists of five different tools.

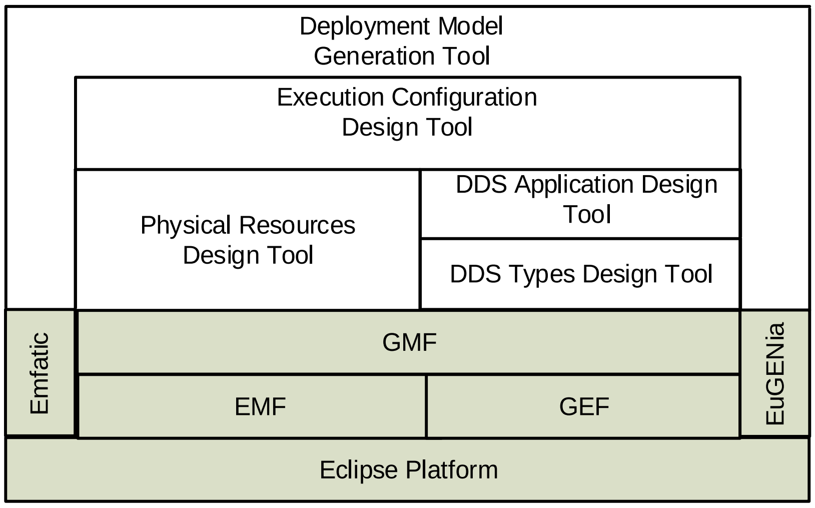

Deployment Model Generation Tool
Execution Configuration Design Tool
Physical Resources Design Tool
DDS Application Design Tool
DDS Types Design Tool
Emfatic
GMF
EMF
GEF
EuGENia
Eclipse Platform

Figure 7-10: Layered Architecture of S-IDE environment

The common perspective of Deploy-DDS is given in Figure 7-11. The left pane includes the Model Explorer View that shows the available models and their elements. The Editing pane in the middle provides the main drawing area for the DDS based application design. The Properties Editor View at the bottom provides an editing area for the attributes of the design model elements that are selected from the Editing Pane or the Model Explorer.

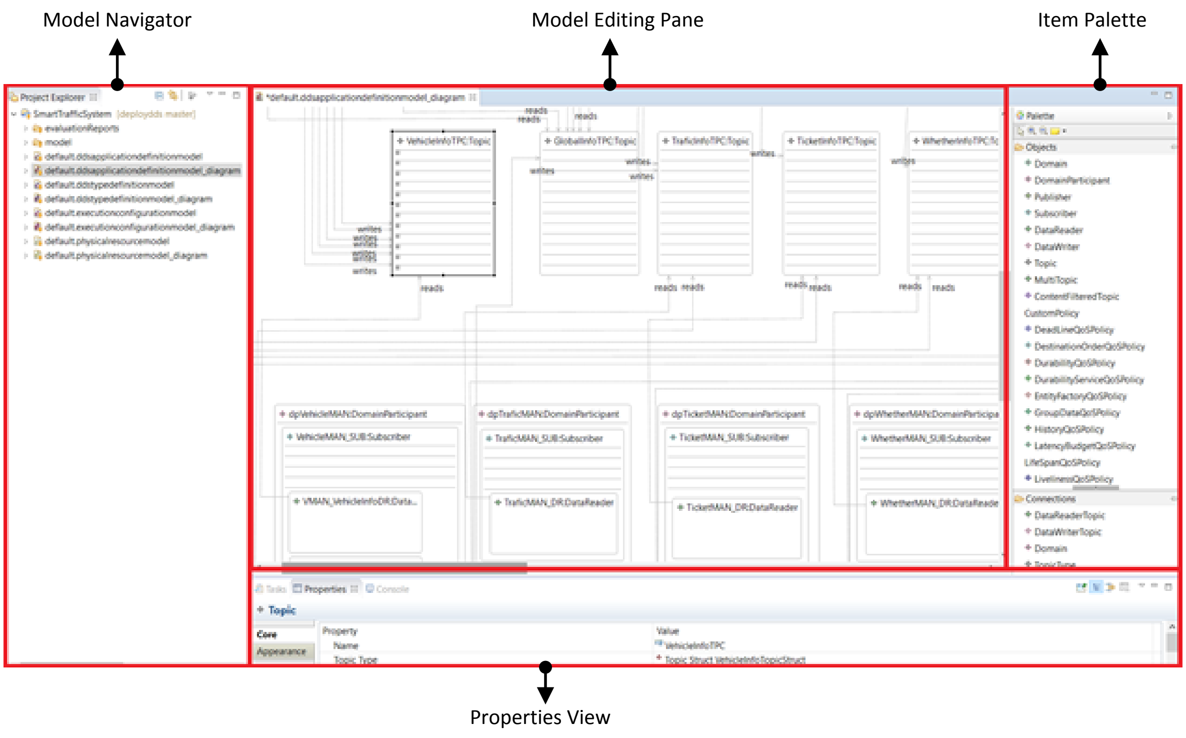


Figure 7-11: General Perspective of Deploy-DDS tool

Deploy-DDS supports different activities in the approach; the dependencies between these activities are shown in Figure 7-12. The meaning of the adopted symbols in the diagram are shown in the legend below the diagram. The activities result in artifacts which are denoted using the stereotype <<Artifact>>. The circles with numbers denote the control flow among the activities.

The DDS Type Repository Definition results in the DDS Type Repository, which is provided as an input to the DDS Topics & Participants Definition activity, and Execution Configuration Definition activity. The DDS Topics & Participants Definition activity is used to produce the DDS Topics, Domain Participants, and Pub/Sub Definitions which is also an input to the Execution Configuration Definition activity. The Physical Resources Design activity is used to define the Physical Resource Model, which is an input to Execution Configuration Definition activity and the Deployment Model Generation activity. The Execution Configuration Definition activity is used to define the Execution Configuration, which is provided as an input to the Deployment Model Generation activity that on its turn generates the Deployment Model. In the following subsections, we describe each activity in more detail using the Smart Parking System (STS) case study defined in Section 3.

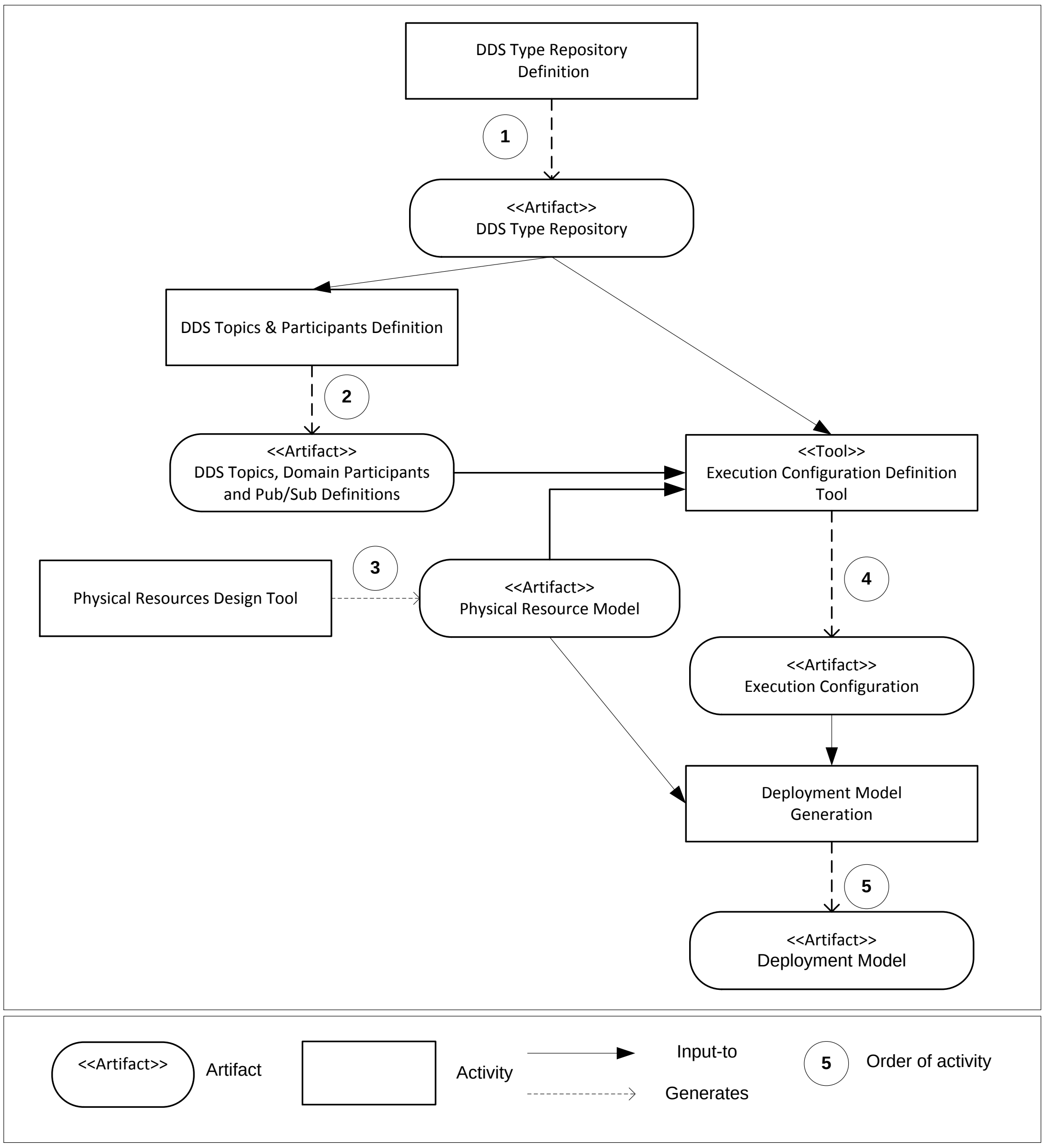

DDS Type Repository Definition
1
<<Artifact>> DDS Type Repository
DDS Topics & Participants Definition
2
<<Artifact>> DDS Topics, Domain Participants and Pub/Sub Definitions
<<Tool>> Execution Configuration Definition Tool
Physical Resources Design Tool
3
<<Artifact>> Physical Resource Model
4
<<Artifact>> Execution Configuration
Deployment Model Generation
5
<<Artifact>> Deployment Model
<<Artifact>>
Artifact
Activity
Input-to
Generates
5
Order of activity

Figure 7-12: Dependency Graph of Activities

7.6.2 Using Deploy-DDS to design DDS Application Models for the Case Study

As stated before, using the tool the activities DDS Application Design, Type Repository Definition, Physical Resources Design and Execution Configuration Design define the corresponding modeling tools. Figure 7-13 shows a part of the DDS Type Repository of the case that has been developed using the DDS Type Repository Definition activity. As stated before, publishers and subscribers communicate via topics. Hereby, publishers write data fields in the topic and subscribers read data fields in the topic. Type Definitions of the topics are given in Figure 7-13. For example, in this diagram we defined a topic VehicleInfo. In this topic we have four data fields. The vehicleID field shows the unique ID of the related vehicle. The speed field shows the speed of the vehicle. Finally, latitude and longitude fields show the geographic position of the vehicle.

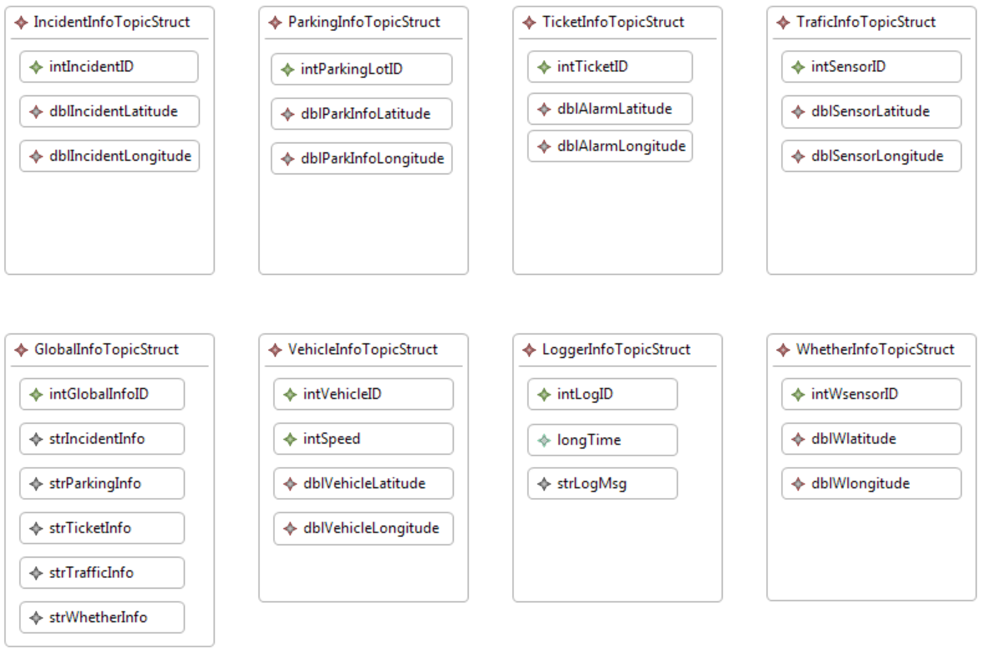


Figure 7-13: Type Definition Model of the case study

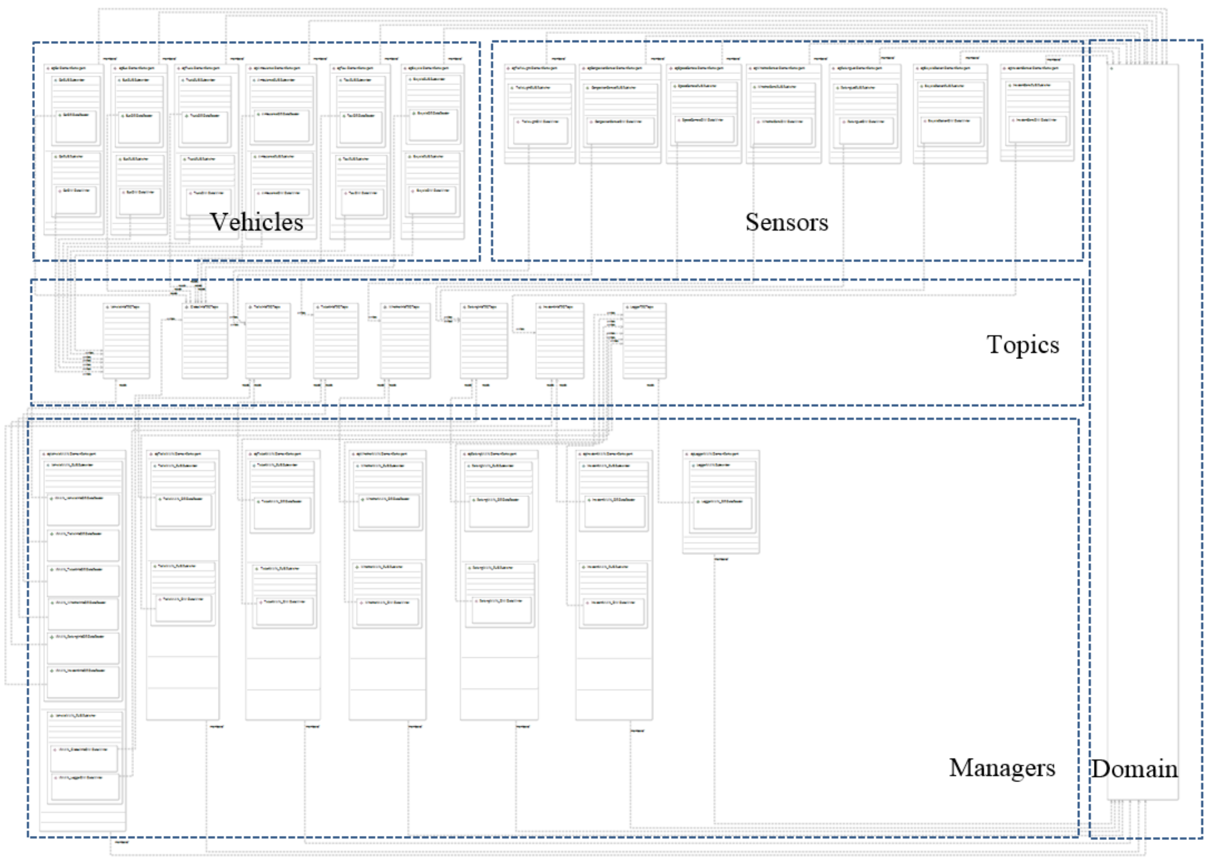


Figure 7-14: Application Definition Model of the case study

Figure 7-14 shows the application definition model of the case study. The domain participants in this figure are all in the "Smart City Traffic" (STS) domain. In this diagram, we can classify domain participants mainly in three categories: Vehicles, Sensors and Managers. Sensors just have data publishers to publish related sensor information. This information is read by related managers and vehicles via the defined topics. Vehicles have both publishers and subscribers. They publish their id, speed and position information basically. This information is read by managers. Similar to vehicles, managers have publishers and subscribers except Logger Manager which has nothing to publish in the STS domain and just a subscriber. Managers combine this information with information coming from sensors. Managers might communicate with other software modules such as database and cloud modules (for simplicity this part was excluded in the model). Resulting information combined in managers are published into vehicles again and drivers might have broad information about the details of the city traffic such as accident information and congestion information so that they can use less dense roads or they can arrive at proper parking places with less travel.

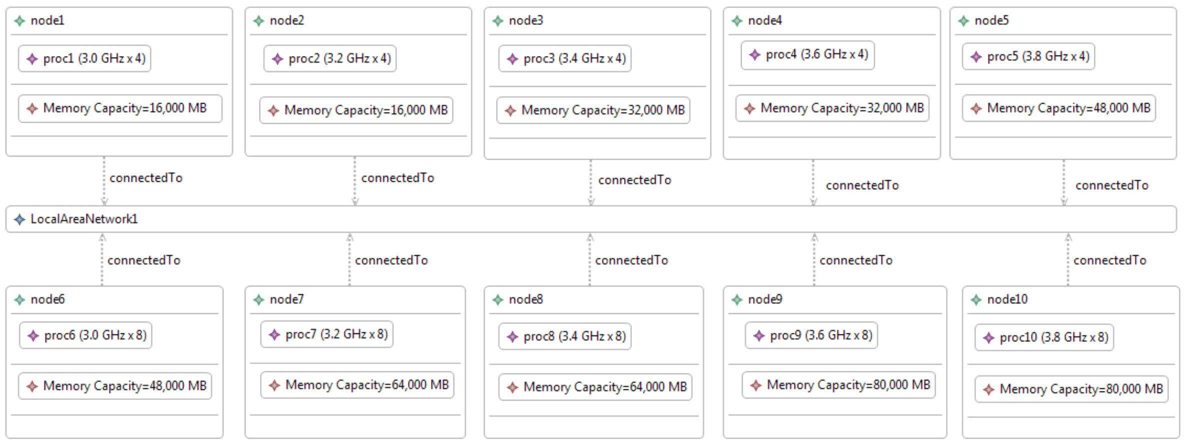


Figure 7-15: Physical Resource Model for Case Study with Ten Nodes

Figure 7-15 shows the Physical Resource Model Diagram of the case study. In this case, we have 10 nodes (computers) with different number of processors and different memory capacities. The processor capacity ranges from 3.0 GHz to 3.8 GHz, while the memory capacities range from 16.000 MB to 80.000 MB (less readable in the figure). This heterogeneity makes obtaining a feasible solution more difficult. Figure 7-16 shows the execution configuration model of vehicle participants of the case study.

Hereby, as an example, Vehicle publishers publish data at 5 Hz, with different execution costs for different nodes. For this chapter, we assume that the proper execution costs are provided. These could be typically obtained experimentally or based on expert knowledge. The more precise the values of the execution costs the more effective the tool will be to derive the feasible deployment alternatives.

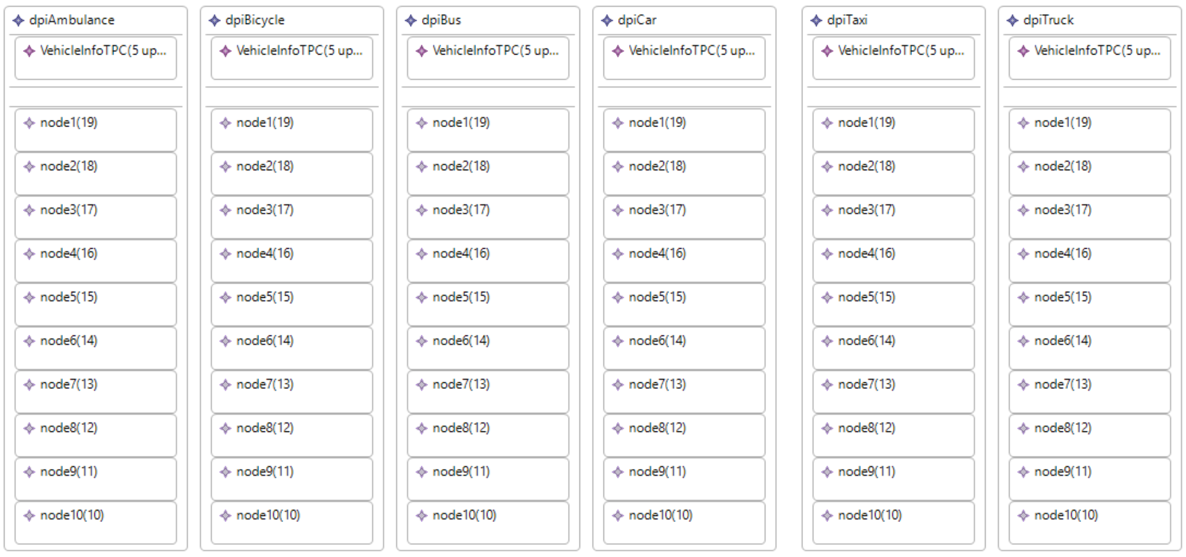
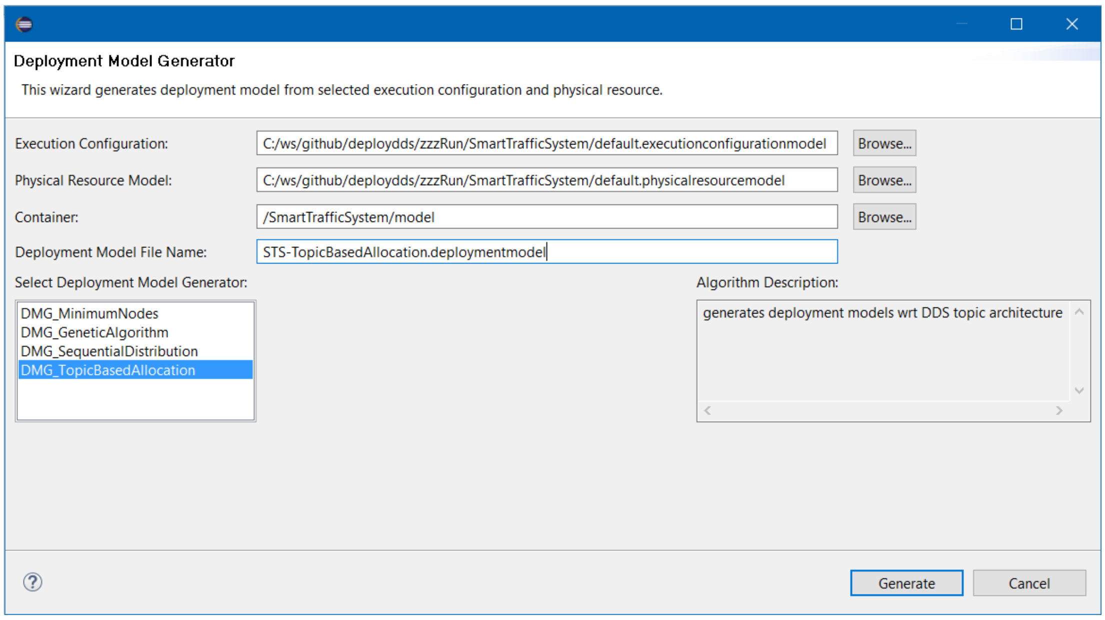


Figure 7-16: Partial view of the Execution Configuration Diagram for the Case Study

7.7 EVALUATION

In the previous sub-sections, we have described the development of the physical configuration model, the type definition model, the application definition model and the execution configuration model. Given these models we can now generate the possible deployment alternatives. The corresponding snapshot of the tool is shown in Figure 7-17. As it can be seen in the figure the execution configuration, and the physical resource model can be provided as an input to the tool. The field Container, defines the folder in which the results are stored. In principle, the deployment generation can be realized using multiple different alternative algorithms. The user can select one of the implemented deployment model generators.



Deployment Model Generator

This wizard generates deployment model from selected execution configuration and physical resource.

Execution Configuration:	C:/ws/github/deploydds/zzzRun/SmartTrafficSystem/default.executionconfigurationmodel
Physical Resource Model:	C:/ws/github/deploydds/zzzRun/SmartTrafficSystem/default.physicalresourcemodel
Container:	/SmartTrafficSystem/model
Deployment Model File Name:	STS-TopicBasedAllocation.deploymentmodel

Select Deployment Model Generator:

DMG_MinimumNodes
DMG_GeneticAlgorithm
DMG_SequentialDistribution
DMG_TopicBasedAllocation

Algorithm Description:

generates deployment models wrt DDS topic architecture

Figure 7-17: Algorithms used to find deployment alternatives in DeployDDS tool

In DeployDDS tool, we have selected five different deployment model generators (DMG) to obtain the deployment models including DMG_TopicBasedAllocation, DMG_GeneticAlgorithm, DMG_SequentialAllocation, and DMG_MinimumNodeAllocation. Each of these algorithms has been implemented in the tool and provides a solution for the MPTA problem as discussed in section 5.4. In the following, we shortly describe the algorithms that we have implemented:

DMG_TopicBasedAllocation aims to find feasible deployment models with minimum communication cost. This logic is implemented by a Greedy Algorithm which allocates the publishers and subscribers of the same topic into the same node. If the node does not have adequate memory for the publishers and the subscribers, only appropriate number of publishers and subscribers will be allocated to that node. The number of nodes that cannot be allocated to the same node because of the lack of memory will be allocated to the next nodes. As stated above, the publishers and the subscribers that cannot be allocated to the same node will cause communication cost. So, if there is enough memory to allocate all communicating publishers and the subscribers into the same node, this DMG will result in zero communication cost.

DMG_GeneticAlgorithms uses a genetic algorithm-based solver to find feasible deployment models.

DMG_SequentialAllocation, allocates domain participant instances into the available nodes. It starts with the first domain participant and allocates sufficient number of domain participants into the first node. It will allocate sufficient number of participants to the first available nodes and then switches to the second node. Note that, if the memory available in the first node is sufficient to allocate all participants, then this DMG will result in the same deployment model with DMG_MinimumNodeAllocations.

DMG_MinimumNodeAllocation aims to find feasible deployment models using the minimum number of nodes. If possible, this DMG allocates all tasks to the same node which will result in zero communication cost. In order to allocate all participants to the same node, this DMG starts from the node that has maximum memory available. If the memory required to allocate all tasks to the same node is not available in a single node, then more nodes will be allocated. The resulting deployment model will be using the minimum number of nodes. At the end, many nodes might become unused.

By selecting one of these generators the feasible deployment alternative can be automatically generated using the selected deployment generator. If necessary, the user of the tool can implement another algorithm and deploy it in the tool. In principle, each newly defined algorithm will follow the steps of the common pseudo-code as shown in Figure 7-18. As shown in line 1, the algorithm GENERATE_FEASIBLE_DEPLOYMENT takes two input parameters: a physical resource model and an execution configuration as defined, for example, in Figure 7-15 and Figure 7-16, respectively. Line 2 extracts processors from the physical resource model by calling EXTRACT_PROCESSORS in which a processor is created for each node in the physical resource model. In Line 3, tasks are extracted from the execution configuration by calling EXTRACT_TASKS in which a task is created for each domain participant and execution cost among tasks is calculated. In Line 4, the actual MPTA algorithm is executed by calling EXECUTE_MPTA. The result of this is stored in assignment_table that includes the assignment of tasks to the processors. Likewise, assignment_table defines an abstract specification of the feasible deployment alternative. In Line 5, the deployment is actually generated by calling CREATE_DEPLOYMENT_MODEL with the parameter assignment_table.

```
1.  GENERATE_FEASIBLE_DEPLOYMENT (phy_resources, exec_config)
2.      processors ← EXTRACT_PROCESSORS (phy_resources)
3.      tasks ← EXTRACT_TASKS (exec_config)
4.      assignment_table ← EXECUTE_MPTA (tasks, processors)
5.      CREATE_DEPLOYMENT_MODEL (assignment_table)
```

Figure 7-18: Pseudo-code for generating feasible deployment alternative

Figure 7-19 shows the generated deployment alternatives for the case study using the DMG_TopicBasedAllocation (Mehrabi et al., 2009).

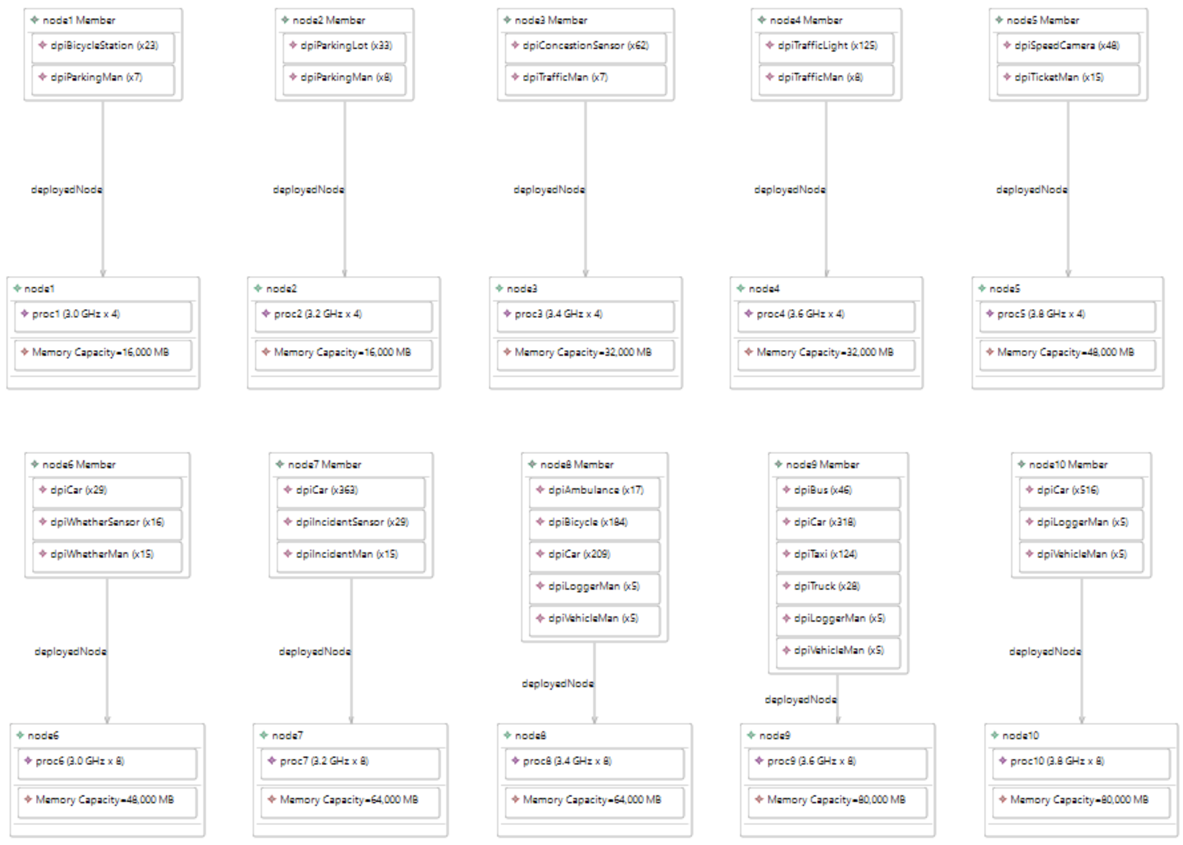


Figure 7-19: Generated Feasible Deployment Alternative including 2275 Tasks with DMG_TopicBasedAllocation

The generation algorithm is implemented in Java and executed on a quad-core Intel I-5 2.70 GHz 64-Bit computer with 4 GB of RAM. The figure is not mentioned to be completely readable. What we can state is that the resulting deployment model includes 10 nodes as given before in the physical resource definition model in Figure 7-15. Further, the execution configuration model as partially defined in Figure 7-16 has been deployed to the physical nodes to optimize the values for the metrics execution cost, communication cost and memory requirements. A close analysis of the generated alternative of Figure 7-19 shows that the total memory requirements of domain participant instances that are deployed on each node do not exceed the memory capacity of the corresponding nodes. Further, based on the adopted genetic algorithm, it appears that domain participant instances that interact frequently and which have high communication costs, are as much as possible co-located on the same node. The domain participant instances that are remaining and which would exceed the memory capacity of Node-1 are deployed to other nodes in a similar manner. Overall, the feasibility of the generated deployment alternative is based on the MPTA algorithm that we have used, and which has been validated in earlier studies (Mehrabi et al., 2009).

The generated deployment diagram can soon become too large to view in a single diagram. For this we can also show the results in Table 7-5. The results for the selection on the other deployment generator algorithms are shown in Table 7-6 (DMG_GeneticAlgorithm), Table 7-7 (DMG_SequentialAlgorithm) and Table 7-8 (DMG_MinimumNodeAllocation).

Table 7-5: Deployment results for DMG_TopicBasedAllocation

Instance Name	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**	**N9**	**N10**	**TOTAL**
dpiAmbulance									15	2	17
dpiBicycle										184	184
dpiBus										46	46
dpiCar					320	67	426	426		196	1435
dpiTaxi						124					124
dpiTruck						28					28
dpiBicycleStation									23		23
dpiCongestionSensor									62		62
dpiIncidentSensor									29		29
dpiParkingLot									33		33
dpiSpeedCamera									48		48
dpiTrafficLight									47	78	125
dpiWeatherSensor									15	1	16
dpiIncidentMan									15		15
dpiLoggerMan									15		15
dpiTicketMan									15		15
dpiTrafficMan									15		15
dpiParkingMan									15		15
dpiVehivleMan									15		15
dpiWeatherMan									15		15
TOTAL	**0**	**0**	**0**	**0**	**320**	**219**	**426**	**426**	**377**	**507**	**2275**

Table 7-6: Deployment results for DMG_GeneticAlgorithm

Instance Name	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**	**N9**	**N10**	**TOTAL**
dpiAmbulance	2			2	7	1	2	1	2		17
dpiBicycle	24	19	14	19	23	21	16	17	14	17	184
dpiBus	6	3	8	3	4	3	4	4	4	7	46
dpiCar	74	83	155	136	162	155	186	173	148	163	1435
dpiTaxi			9	9	39	10	14	14	19	10	124
dpiTruck				5	5	4	3	4	4	3	28
dpiBicycleStation				2	5	3	1	1	8	3	23
dpiCongestionSensor					16	8	10	10	9	9	62
dpiIncidentSensor					11	4	4	5	2	3	29
dpiParkingLot					8	6	4	6	4	5	33
dpiSpeedCamera					9	6	7	14	4	8	48
dpiTrafficLight		1	13		4	36	17	13	19	22	125
dpiWeatherSensor			1		2	1	1	6	1	4	16
dpiIncidentMan			2		4		5	1	2	1	15
dpiLoggerMan			3				1	4	4	3	15
dpiTicketMan				6		1	1	3	2	2	15
dpiTrafficMan				2			5	2	1	5	15
dpiParkingMan				1			8	2	2	2	15
dpiVehivleMan				5			2	2	4	2	15
dpiWeatherMan				1			6	4	1	3	15
TOTAL	**106**	**106**	**205**	**191**	**299**	**259**	**297**	**286**	**254**	**272**	**2275**

Table 7-7: Deployment results for DMG_SequentialAllocation

Instance Name	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**	**N9**	**N10**	**TOTAL**
dpiAmbulance	2	2	2	2	2	2	2	1	1	1	17
dpiBicycle	19	18	18	18	18	18	18	19	19	19	184
dpiBus	4	5	5	5	5	5	5	4	4	4	46
dpiCar	81	81	188	188	179	143	143	144	144	144	1435
dpiTaxi					63	13	12	12	12	12	124
dpiTruck					14	2	3	3	3	3	28
dpiBicycleStation					11	3	3	2	2	2	23
dpiCongestionSensor					18	18	6	7	7	6	62
dpiIncidentSensor						18	3	3	2	3	29
dpiParkingLot						19	3	3	4	4	33
dpiSpeedCamera						29	5	5	5	4	48
dpiTrafficLight						15	73	12	12	13	125
dpiWeatherSensor							10	2	2	2	16
dpiIncidentMan							12	1	1	1	15
dpiLoggerMan							9	2	2	2	15
dpiTicketMan							12	1	1	1	15
dpiTrafficMan							9	2	2	2	15
dpiParkingMan							12	1	1	1	15
dpiVehivleMan							9	2	2	2	15
dpiWeatherMan								13	1	1	15
TOTAL	**106**	**106**	**213**	**213**	**310**	**285**	**349**	**239**	**227**	**227**	**2275**

Table 7-8: Deployment results for DMG_MinimumNodeAllocation

Instance Name	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**	**N9**	**N10**	**TOTAL**
dpiAmbulance									1	16	17
dpiBicycle										184	184
dpiBus										46	46
dpiCar					320	67	426	426		196	1435
dpiTaxi						124					124
dpiTruck						28					28
dpiBicycleStation									23		23
dpiCongestionSensor									62		62
dpiIncidentSensor									29		29
dpiParkingLot									33		33
dpiSpeedCamera									48		48
dpiTrafficLight									73	52	125
dpiWeatherSensor										16	16
dpiIncidentMan									15		15
dpiLoggerMan									15		15
dpiTicketMan									15		15
dpiTrafficMan									15		15
dpiParkingMan									15		15
dpiVehivleMan									15		15
dpiWeatherMan									15		15
TOTAL	**0**	**0**	**0**	**0**	**320**	**219**	**426**	**426**	**374**	**510**	**2275**

Each of these deployment generators will perform differently. To validate each algorithm, we adopt the Communication Cost and Execution Cost metrics. Communication Cost defines the overall communication costs of the required communication tasks in the generated deployment alternative. The execution cost metric defines the overall cost of the required tasks on the required number of processors. We have calculated the communication costs and execution costs for the selected deployment generators applied to the case study. The results are shown in Table 7-9. The unit of the communication costs is Mbytes/s; execution cost is a relative unit.

Table 7-9: The communication and execution costs values for the deployment generators

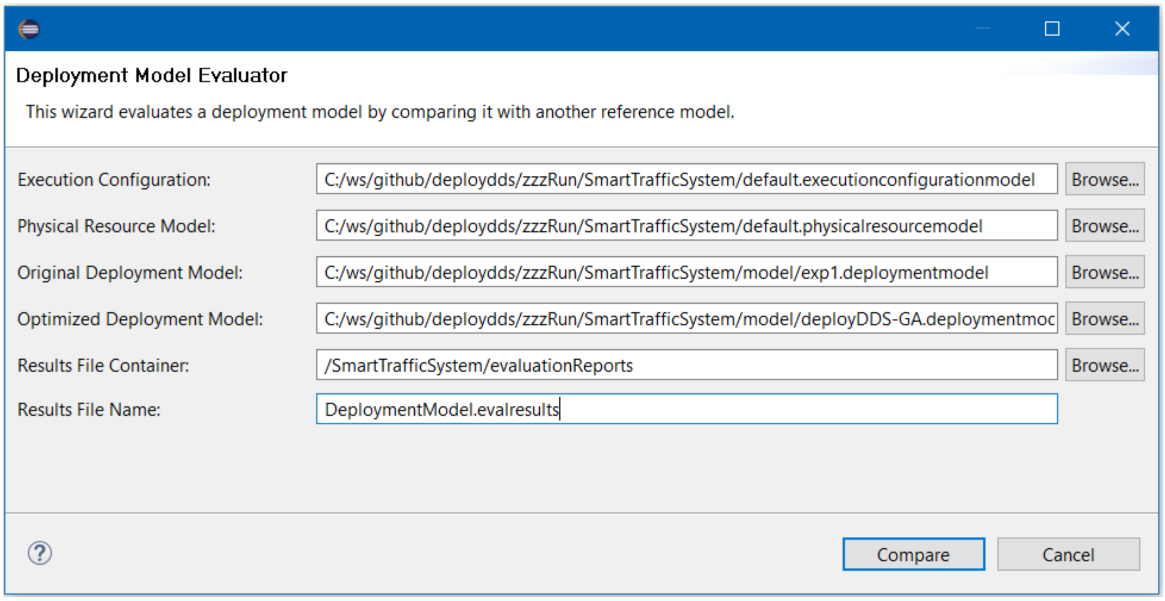
Deployment Generator	**Communication Cost [Mbytes/s]**	**Execution Cost**
DMG_TopicBasedAllocation	4.05	22773
DMG_GeneticAlgorithm	5.00	31381
DMG_MinimumNodeAllocation	4.02	27730
DMG_SequentialAllocation	4.76	31763

As we can observe in Table 7-9, for both metrics the deployment generator *DMG_MinimumNodeAllocation* performs the best, while *DMG_GeneticAlgorithm* has the lowest performance. These values are also shown by the tool and as such provide useful insight for deciding on the proper deployment allocation.

Each of the selected algorithms provides feasible deployment alternatives and can in principle directly be used to implement the system. In order to further analyze the validity of the generated deployment models we use two approaches.

The first approach is intuitive and based on the visual inspection of the generated deployment model alternatives by an expert. Therefore, this approach relies on the expert's experience to provide logical reasoning about the feasibility of the deployment alternative. In addition, the generation of the alternative is done automatically and not performed by the expert. An example reasoning of an expert could be based on the deployment alternative given in Figure 7-19. A close analysis of this generated deployment alternative shows that the total memory requirements of DDS based software system (i.e., STS) does not exceed the capacity of the corresponding nodes. Further, based on the adopted genetic algorithm, it appears that software domain participants that interact frequently and which have high communication costs, are as much as possible co-located on the same node. Apparently, the publishers and subscribers in the system have frequent interactions in publish-subscribe communication via ParkingLot topic and in the deployment model (Figure 7-19). The adopted algorithm has co-located instances of these modules as much as possible to keep the communication cost minimum. The remaining instances, which would exceed the capacity of Node-1, are deployed to other nodes in a similar manner.

The second, more formal approach for evaluating the generated deployment alternative is to compare the generated alternative with another deployment alternative. As shown in Figure 7-20, the DeployDDS tool enables the comparison of two deployment models that were defined before, either generated and/or manually defined. To compare two models, the execution configuration and the physical resource model is provided. Once the Compare button is pressed the output is written to the corresponding result folders.


Deployment Model Evaluator

This wizard evaluates a deployment model by comparing it with another reference model.

Execution Configuration:	C:/ws/github/deploydds/zzzRun/SmartTrafficSystem/default.executionconfigurationmodel
Physical Resource Model:	C:/ws/github/deploydds/zzzRun/SmartTrafficSystem/default.physicalresourcemodel
Original Deployment Model:	C:/ws/github/deploydds/zzzRun/SmartTrafficSystem/model/exp1.deploymentmodel
Optimized Deployment Model:	C:/ws/github/deploydds/zzzRun/SmartTrafficSystem/model/deployDDS-GA.deploymentmoc
Results File Container:	/SmartTrafficSystem/evaluationReports
Results File Name:	DeploymentModel.evalresults

Figure 7-20: Deployment Model Evaluator of DeployDDS tool

The comparison process provided in the DeployDDS is generic and can be applied in a similar way for the alternatives generated with all the defined deployment generators. We show the evaluation of the generated deployment model with a manually generated deployment model that is based on a deployment model that is generated by an expert. We have manually defined the deployment model for the expert judgment deployment alternative in DeployDDS environment. The results of the expert allocation are shown in Table 7-10.

Table 7-10: Deployment results for expert distribution

Instance Name	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**	**N9**	**N10**	**TOTAL**
dpiAmbulance									17		17
dpiBicycle									184		184
dpiBus								16	30		46
dpiCar	17	17	123	123	230	231	338	322	17	17	1435
dpiTaxi	12	13	13	13	13	12	12	12	12	12	124
dpiTruck	3	2	2	3	3	3	3	3	3	3	28
dpiBicycleStation										23	23
dpiCongestionSensor										62	62
dpiIncidentSensor										29	29
dpiParkingLot										33	33
dpiSpeedCamera									6	42	48
dpiTrafficLight									125		125
dpiWeatherSensor									16		16
dpiIncidentMan										15	15
dpiLoggerMan										15	15
dpiTicketMan										15	15
dpiTrafficMan										15	15
dpiParkingMan										15	15
dpiVehivleMan										15	15
dpiWeatherMan										15	15
TOTAL	**32**	**32**	**138**	**139**	**246**	**246**	**353**	**353**	**410**	**326**	**2275**

The expert deployment allocates an equal number of domain participant instances to each node. The expert checks the available memory of the nodes and if the memory is not sufficient for the required number of tasks, he/she tries to allocate the remaining tasks to the other available nodes. The results for this expert deployment allocation are shown in Table 7-10. Figure 7-21 shows the results of the expert deployment for three nodes.

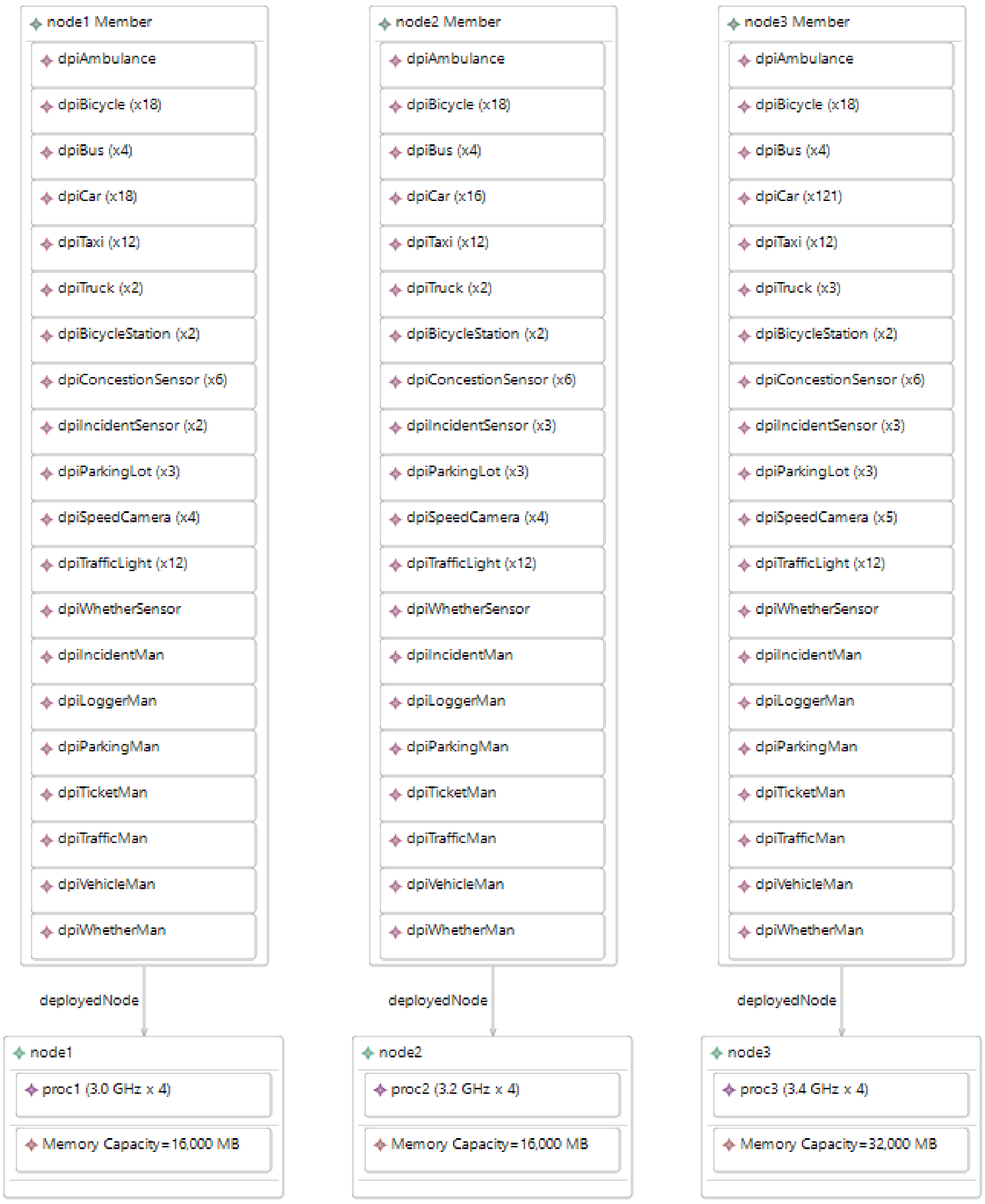


Figure 7-21: Expert Deployment Model for first three nodes

The numbers in each cell defines the number of instances of the participants. For example, dpiBicycleStation (x23) means that 23 instances of BicycleStation is deployed into the node and dpiAmbulance means a single instance of Ambulance is deployed into the corresponding node. The communication and execution costs values for this export deployment are given in Table 7-11.

Table 7-11: The communication and execution costs values for the expert deployment

Deployment Generator	Communication Cost [Mbytes/s]	Execution Cost
Expert Deployment	4.37	29483

If we compare the expert-based deployment with that of the earlier defined DMG_TopicBasedAllocation in Figure 7-19 we can conclude that both the communication costs and execution costs metric values are slightly better for the DMG_TopicBasedAllocation. DMG_TopicBasedAllocation tries to define the deployment such that the participants and the subscribers of the same topic are located into the same node. This strategy minimizes the communication cost in the deployment model. For example, the participant BicycleStation publishes the data which is subscribed by ParkingManager. As such, these two participants are located into the same node. Similarly, ParkingLot and ParkingManager are located into the second node and ConcestionSensor and the TrafficMan participants are deployed into the third node.

Similar to the comparison with DMG_TopicBasedAllocation we can compare the expert deployment also with the results of the other deployment generators. The comparison results are shown in Table 7-12. Here we have set the communication cost and execution cost of the expert deployment to 100%. The other percentages define the percentage in relation to the expert's results. From this table, we can conclude that the defined deployment generators perform in general better than the expert deployment. When execution costs are compared the DMG_MinimumNodeAllocation seems to perform the best. Based on these results a given deployment generator could be selected. Note that the results of the algorithms can be different for different execution configuration models and the physical resource models. The approach and the tool can be used to assist in selecting the most feasible deployment model. As stated before, if needed, new deployment generators can be easily defined to optimize the results even further.

Table 7-12: Comparison of Expert Deployment models with respect to Deployment Model Generators

Deployment Generator	Communication Cost [%]	Execution Cost [%]
DMG_TopicBasedAllocation	92.8	94.1
DMG_GeneticAlgorithm	114.4	106.4
DMG_MinimumNodeAllocation	92.1	94.1
DMG_SequesntialAllocation	108.9	107.7
Expert Deployment Model	100	100

7.8 DISCUSSION

The Data Distribution Service (DDS) is now a popular and recognized data-centric publish-subscribe programming model and specification for distributed systems. It has been applied in many different application domains which have resulted in several lessons learned. One of the important issues is the support for modeling and design abstractions in DDS based systems. OMG has provided the DDS UML Profile to support the analysis and design of a DDS-based distributed system. The focus of this chapter has been mainly on deriving configuration alternatives. This is an important and relevant problem for many DDS-based systems which consist usually of multiple participant applications each of which has different responsibilities in the system. The potential configuration space is in general too large and not tractable for the human system engineer and a systematic approach with automated support is necessary.

We have provided both a systematic approach with the related toolset that can be used for a broad range of DDS-based systems to derive feasible configuration alternatives to meet the functional and quality concerns given the available resources. The approach has also been illustrated for a relevant case study on smart city engineering which has been used to illustrate both the problem and the approach.

The approach adopted the UML profile to complement the existing work. The UML profile appeared to be very useful in preparing and supporting the analysis and design of the DDS system for deriving design alternatives. It should be noted that the OMG's DDS UML Profile is a specification and no realization of it was present yet. As such, one of the supporting contributions in this chapter is also the realization of this profile in the Eclipse Modeling Framework (EMF). To support the systematic approach for generating the design alternatives we had to enhance the profile further (e.g. physical resource modeling, and execution configuration model).

Based on the modeled system design and physical resources using the realized DDS's UML Profile, the feasible deployment alternatives could be algorithmically derived and automatically generated using the developed tools. In the toolset, we have implemented different algorithms for deriving feasible deployment alternatives. Yet the approach does not mandate the usage of a particular algorithm but provides the required input values for these algorithms. The focus of the chapter is not the design of algorithms but the overall system engineering approach for selecting a feasible alternative in a large configuration space. The algorithms that we used were justified in the literature for solving the MPTA problem as discussed in section 7.5. The correctness of these algorithms has been discussed in the corresponding papers and based on this we can assume that a feasible solution is derived. In addition, depending on the state of the system different MPTA algorithm implementations may be used to optimize the values for the metrics. For comparison of the algorithms, we refer to, for example, (Ucar et al., 2006).

Both the approach and the tools assist the designer to derive a feasible deployment model. We do not maintain a claim that the tool is a replacement for the human expert. In fact, the tool can be a complementary and supporting alternative for the human expert who can design, generate and evaluate the derived alternatives. After deriving the deployment alternative, if necessary, expert judgment can be further used to refine the deployment alternative.

One of the important benefits of the approach is also the early analysis of the system and the generation of the feasible deployment model at design time. Deferring the definition of the deployment to the development phase might in practice easily lead to non-feasible implementations which will require iterating the design and the related project lifecycle artifacts such as detailed design, implementation, test artifacts, documentation, etc.

The identified deployment model may be refined and optimized if more accurate information is available in subsequent phases of the project lifecycle. The approach itself can actually be used at any time during the project life cycle and, if possible, even after the system has been developed. In the latter case, the measured run-time parameter values can be used, instead of estimated values, to define the optimal deployment model. The runtime parameter values can be collected by using tools that collect activities (e.g. topic updates) of domain participants.

7.9 Related Work

The allocation of software units on computing systems has applications in different computing domains such as embedded systems, local/wide area distributed systems, parallel and distributed simulations, etc. In our earlier work, we have carried out a systematic review to identify the obstacles of DDS based systems (Köksal & Tekinerdogan, 2017b). One of the identified key obstacles that we derived from the systematic review was indeed the task allocation problem. In this chapter, we have provided a systematic approach with the corresponding toolset to tackle this task allocation problem.

In the literature, we can observe that some of the studies propose concrete approaches for specific domains (such as Parallel and Distributed Systems or DDS) while others provide more generic approaches that can be configured to use in different domains. In this work, we have focused on the allocation problem that is directly focused on the DDS domain. In our early work, we have provided an approach for deriving feasible deployment alternatives for parallel and distributed simulation systems in (Celik, Tekinerdogan, & İmre, 2013) and tool support for the approach in (Celik & Tekinerdogan, 2013). From a generic perspective, the solved allocation problem is characterized as the Multi-Processor Task Assignment (MPTA) problem (Stone, 1977) which is a general problem that can be applied to different domains. Each MPTA problem however requires a specific approach and dedicated steps to solve the allocation problem. In this study, it is clear that the domain of DDS is specific and different and provides additional challenges including modeling the DDS system, the individual steps of the overall approach and the corresponding toolset. Our work could be further specialized by considering specific QoS parameters (such as reliability) to derive feasible alternatives. We consider this as a further complementary research.

Several generic approaches can be identified in the literature to provide a solution for the allocation problem. For example, in (Koziolek & Reussner, 2011), the authors introduced a generic quality optimization framework that can be used for different component based models. Similarly, in (Malek et al., 2012) the authors propose an extensible framework that supports formal modeling of distributed software systems. The study provides a set of tailorable algorithms for finding optimized deployment architectures with respect to multiple, possibly conflicting QoS (Quality of Service) dimensions. The study also provides a visual deployment architecture modeling and analysis environment for the framework. Similar to our work, the authors evaluated the framework with simulated distributed system scenarios.

In (Svogor & Carlson, 2016), the authors use heuristics and Analytic Hierarchy Process (AHP) (Saaty, 1988) for weighted multi-objective design space exploration. The main objective of the study is to support systems architects in complex allocation decisions in the early design phases. In (Aleti, Grunske, et al., 2009) the authors use constructive algorithms for deployment optimization of embedded systems. Hereby, an Ant Colony Optimization (ACO) is used as a constructive multi-objective optimization strategy which is compared with a Genetic Algorithm (GA) based iterative approach. The authors conclude that they observed that constructive and iterative approaches performed similarly in their experiments.

In (Kruchten, 2000), the authors define an approach to optimize the task placement and the signal to message mapping in hard-real time distributed systems domain. Authors use a mixed integer linear optimization framework to automate the assignment of priorities to tasks and messages. The optimization process aims to meet end-to-end deadline constraints and minimize latencies by leveraging worst case response time analysis. Authors validate the developed approach by applying it to an automotive system case study.

In (Islam, Lindstrom, & Suri, 2006), authors focused on reducing error propagation while allocating software components to distributed embedded hardware nodes. The study presents a systematic resource allocation approach for the consolidated mapping of safety critical and non-safety critical applications onto a distributed platform with consideration of dependability and real-time requirements as primary drivers. The approach focuses on finding a feasible solution satisfying multiple concurrent constraints such as ensuring criticality partitioning, avoiding error propagation and reducing interactions across components. The authors applied the approach to an actual automotive case study to prove its feasibility.

In (Martens, Koziolek, Becker, & Reussner, 2010), an approach is defined for automatically improving software architecture models for performance, reliability, and cost using evolutionary algorithms. Starting with a given initial architectural model, the approach iteratively modifies and evaluates architectural models by using a multi-criteria genetic algorithm based on Palladio Component Model (Becker, Koziolek, & Reussner, 2009). The approach supports quantitative performance, reliability, and cost prediction of software architectures. The approach is validated by automatically investigating more than 1200 alternative design candidates for a component-based business information system and analyzing quality criteria trade-offs.

The detailed literature study showed us that the task allocation is a well-known and still widely studied research area with a large application domain spreading from embedded systems to wide area distributed systems. Our approach is complementary to these approaches and is specific since it provides and integrates the necessary modeling abstractions using the extended DDS UML profile, the systematic approach, and the tool environment which can be extended for different additional functionality including different algorithms.

It should be noted that besides the academic papers we can also identify several interesting tools on the task allocation problem, which have been provided by several vendors or which have been presented in various papers. In (Aleti, Björnander, Grunske, & Meedeniya, 2009), the authors define an extensible tool for Architecture Optimization of AADL (Architecture Analysis and Description Language) Models (Feiler, Gluch, & Hudak, 2006) with name of ArcheOpterix. The study provides a framework to identify optimal and near optimal deployment architectures with respect to multiple quality objectives and design constraints.

The existing DDS design tools focus in general on designing the DDS application and code generation. Prismtech (Prismtech, n.d.-a), a well-known DDS infrastructure vendor, provides

Vortex OpenSplice Modeler (Prismtech, n.d.-b) that is a domain specific model driven development tool. Vortex OpenSplice Modeler enables definition of topics (information modeling), DDS entities such as publishers/writers and subscribers/readers (application design). These capabilities are similar to our tool framework, but Vortex OpenSplice Modeler enables Java/C++ code generation which our tool does not provide. On the other hand, our tool framework provides an automated deployment design optimization approach which is not provided by Vortex OpenSplice Modeler. Another major DDS vendor is RTI (RTI, n.d.), which provides UML based modeling environment for DDS. Sparx Systems also provide a UML based DDS modeling environment (OMG, 2010) as a plug-in for Enterprise Architect application. Another DDS vendor, MilSOFT, also provides a modeling & code generation tool for DDS (Milsoft, n.d.). All these tools provide modeling support and to some extent code generation for DDS topics and applications, but they do not provide explicit support for the deployment optimization which is the main contribution of our research. In principle, our systematic approach could be also integrated with the existing tools.

7.10 Conclusion

An increasing number of systems are data-intensive and rely on the publish-subscribe programming model to realize the distribution aspects. The Data Distribution Service (DDS) provides a standard data-centric publish-subscribe programming model and specification for distributed systems. In addition, the OMG has provided the DDS UML Profile to support the modeling of the DDS applications. These are important developments but they do not consider the design aspects explicitly. An important design concern is of course the selection of the feasible deployment alternative given the application model, the physical resources, and the execution configurations. So far, this problem has not been explicitly addressed in the DDS literature. We have provided a systematic approach by extending the DDS UML profile and an extensible tool framework.

We have developed a tool framework, Deploy-DDS that provides an integrated development environment for deriving a feasible deployment alternative. The tool framework consists of several tools for modeling, generating and analyzing of the deployment alternatives.

Furthermore, we have evaluated the approach for a relevant IoT case study on smart city engineering. The approach showed to be useful in the modeling, the design and the evaluation of the DDS deployment alternatives. The adoption of different algorithms and the ability to add new algorithms can support the system architect also in the experimentation of the different algorithms. Since in practice the task allocation problem and the selection of the feasible design alternatives are not tractable we believe that the approach and the toolset that we have provided is necessary.

In our future work, we will do research on further extension and specialization of the approach. In this context, we will consider the adoption of specific quality criteria such as reliability and further focus on the trade-off analysis using multiple quality criteria. Further, we will also consider the analysis and comparison of various algorithm implementations to further optimize the approach.

8

GENERAL DISCUSSION

8.1 INTRODUCTION

In this thesis, our main objective was to analyze and design integrated IoT systems. To this end, we have started with first defining the characteristic features of IoT systems. We have described a layered reference architecture for deriving IoT system architectures. Further, we have presented the common and variant features of the most commonly used session layer protocols. Subsequently, we have provided a systematic approach to architect the design of IoT-based FMISs whereby we have adopted a feature-driven domain analysis approach. Further, we have provided several design patterns to integrate IoT-based systems at different layers of the IoT reference architecture. Among the session layer protocols of the IoT reference architecture, we have focused on the DDS middleware that is mainly used for machine-to-machine communication in IIoT. We have used a systematic review approach to identify the obstacles of DDS middleware and to provide the corresponding solution directions. Using the IoT reference architecture and the DDS architecture we have designed the architecture for DDS-based IoT systems. Finally, we have provided a systematic approach to find feasible deployment alternatives for DDS-based systems by extending the DDS UML profile and developing the Deploy-DDS research tool. In the next sections, we will provide the details on how we have addressed the defined research questions and the corresponding contributions in the thesis.

8.2 ADDRESSING RESEARCH QUESTIONS

As stated above, the main objective of this thesis is to analyze and design integrated IoT systems. In this context, we defined the following research questions:

RQ1. What are the characteristic features of the IoT systems?

RQ2. How to design the architecture for an IoT-based system?

RQ3. What are the identified obstacles of the DDS middleware?

RQ4. What are the solution directions for the identified obstacles of DDS?

RQ5. What are the approaches for integrating multiple IoT-based systems?

RQ6. How to design a DDS-based IoT system?

RQ7. How to derive feasible deployment alternatives for DDS-based systems?

In order to answer these research questions, three different research methodologies were used: Systematic Literature Review, Design Science Research, and Case Study Research.

8.2.1 RQ1-What are the characteristics features of the IoT systems?

Before dealing with the integration of IoT devices it is important to identify the current state of the IoT and identify the IoT features. We have applied a feature-based domain analysis approach and identified the common and variant features of the IoT systems. We have developed feature diagrams to model the IoT-based systems. Further, we developed a reference architecture including multiple different layers for the IoT systems. Among these layers, we have particularly focused on session layer that which is responsible for setting up and taking down of the association between the IoT connection points. Further we have defined the criteria for selecting the identified communication protocols for the different conditions. Adopting a feature-driven domain modeling approach appeared to be useful in identifying the key features of IoT systems. Hereby, we have not adopted a heavy systematic literature review approach but selected relevant primary studies from which we could derive the IoT features. We believe that the feature model includes the necessary features of IoT system and it appeared to be very useful for the subsequent activities in our overall study.

8.2.2 RQ2-How to design the architecture for an IoT-based system?

In order to design the architecture for an IoT-based system we have presented a systematic approach to guide the architect. Again, we have adopted a feature-driven domain analysis approach to model the common and variant precision farming features. Further, based on FMIS and IoT reference architectures we have described the steps and the modeling approaches for designing the IoT-based FMIS architecture.

The method that we have discussed can be adopted for deriving IoT-based FMIS architecture for multiple different systems. Hence, we focus on the whole product family of IoT-based FMIS systems rather than on a single system. The notion of product families or product line engineering and the corresponding systematic reuse is discussed in detail in the product line engineering community (Clements, 2006). Our method is inspired and customizes the product line engineering approach in which reference models are developed and applications are developed by reusing these reference models. The reference feature diagram that we have shown aims to target and integrate the domains of IoT and FMIS illustrating the overall method. The feature diagrams as well as the reference architecture design could be easily extended. We have discussed the architectures for IoT and FMIS separately and illustrated the integration of both for supporting IoT-based FMIS systems. The architecture can be extended in two ways. First of all, we could of course detail the different views to provide an even more comprehensive result. This would require for example to further detail the modules that are needed in the decomposition view. Secondly, we could extend the architecture representations with other architecture views. We have chosen three architecture views including decomposition view, layered view and deployment view. If needed other architecture views in the architecture documentation process could be used as well. Although, we have showed our approach for two important case studies in the smart agri-food sector the method can be actually applied for the development of other FMISs.

8.2.3 RQ3- What are the identified obstacles of the DDS middleware?

One of the session layer protocols of the IoT is the DDS which is a standard data-centric publish-subscribe programming model. We have shown that DDS can be used for integrating IoT systems. We have applied a meticulous systematic literature review based on the proven protocol of Kitchenham et al (Kitchenham et al., 2009). The SLR included 468 papers in from which we selected 34 as primary studies. We have carefully devised and applied our selection and elimination criteria in order not to miss any relevant primary study. The primary goal of SLR was to identify the relevant obstacles and the corresponding solution directions. Based on our thorough study we could identify 11 problem categories. It appears that different studies have focused on different problems and solutions and as such we could observe an uneven distribution. We consider this fact also as the result of our study since this highlights the important obstacles as well as the obstacles which have not yet been fully explored. On its turn this provides a broader vision on DDS and also paves the way for further research.

We could identify 11 different problem categories. An important number of the problems relate to quality concerns such as reliability, scalability and security. We have described the problems related to specific quality concerns if these were also the topic and focus of the identified primary studies. Quality concerns which were not explicitly reported were not included as obstacles. Our study could on the one hand be used to highlight the relevance of the quality concerns in DDS based systems. On the other hand, our SLR shows also which quality concerns have not been explicitly discussed. This only implies that no in-depth research has been carried out for these quality concerns, and not necessarily that these quality concerns are not relevant for DDS. On its turn this observation can trigger further research on quality concerns in DDS.

Our research methodology however is a systematic literature review that focuses on the analysis of existing primary studies in the literature. Hence, we can only report on the problems that were identified in these primary studies. From our SLR we can observe that implementing a DDS is one of the obstacles. This is an important observation when adopting DDS for integrating IoT systems. In this thesis we have tackled this problem by showing the DDS-based IoT architecture which was discussed in chapter 6. It should be noted that several other more detailed but unreported problems could exist that are directly related to implementation of DDS-based systems. The identification of these problems would require an in-depth study to the corresponding implementations of the DDSs. The identification of DDS obstacles is also important beyond the context of IoT. Further research could be carried out in this context.

8.2.4 RQ4-What are the solution directions for the identified obstacles of DDS?

As stated above, based on the SLR we have identified 11 basic categories of obstacles for the DDS middleware. Our SLR has resulted in a set of feature diagrams that summarize the reported obstacles as well as the solution directions. The feature models that we have developed can be also used to pave the way and support the development of a DDS ontology. These solution directions are proposed by the authors of the selected primary studies. For each identified obstacle the solution directions appeared to be diverse in nature. Solution directions include design heuristics and design abstractions, adoption of different paradigms, refinement of the DDS concepts, novel introduction and implementation of algorithms, integration with other paradigms, and solutions for realizing system-wide quality management.

From our SLR we could also observe that for some of the identified problem categories have been also considered in some of the proposed extensions to the OMG's DDS specification. While describing the problems and the corresponding solutions we could identify that some researchers claim that the offered solution in the primary study is better than the one used of the DDS specification. Further, other primary studies handle topics that are not (completely) covered by the provided DDS specification yet. Finally, it should be noted that some DDS specifications are still in beta version and have not been finalized yet.

8.2.5 RQ5-What are the approaches for integrating multiple IoT-based systems?

One of the key challenges in IoT is coping with the heterogeneous set of systems and the integration of these systems in the same communication network. Based on a layered reference architecture for IoT we have indicated that the integration can be at different layers including session layer, cloud layer and application layer. Further we have shown that the integration is typically carried out based on well-defined patterns, that is, generic solutions structures for recurring problems. We have not provided any new integration solution but rather systematically compiled and structured the integration patterns as defined in the literature. Our study has resulted in 15 different patterns which can be used in different combinations. To guide the application of the patterns we have provided a general process represented using the BPMN. The process and the patterns have been successfully applied to a smart city case study. Hence, we have shown that the systematic structuring of the integration patterns is useful for developing IoT systems that need to integrate heterogeneous elements.

8.2.6 RQ6-How to design a DDS-based IoT system?

Based on the IoT reference architecture and the DDS reference architecture that we have developed, we have provided a DDS-based architecture for IoT systems. We have adopted architecture viewpoints for modeling DDS, IoT and finally DDS-based IoT systems using a systematic approach. Further, we have applied the layered viewpoint to represent the DDS-based IoT since both the DDS and IoT are often represented as layered structures. We have also defined the deployment view for DDS-IoT. We can state that we succeeded to integrate and represent the architecture models that can be used to model DDS-based IoT systems for various application domains.

8.2.7 RQ7-How to derive feasible deployment alternatives for DDS-based systems?

We have provided both a systematic approach and a research toolset that can be used for a broad range of DDS-based systems to derive feasible configuration alternatives to meet the functional and quality concerns given the available resources. The approach has also been illustrated for a relevant case study on smart city engineering which has been used to illustrate both the problem and the approach.

The approach adopted the UML profile to complement the existing work. The UML profile appeared to be very useful in preparing and supporting the analysis and design of the DDS system for deriving design alternatives. It should be noted that the OMG's DDS UML Profile is a specification and no realization of it was present yet. As such, one of the supporting contributions is the realization of this profile in the Eclipse Modeling Framework (EMF). To support the systematic approach for generating the design alternatives we had to enhance the profile further (e.g. physical resource modeling, and execution configuration model).

Based on the modeled system design and physical resources using the realized DDS's UML Profile, the feasible deployment alternatives could be using the developed Deploy-DDS tool. In this toolset, we have implemented different algorithms for deriving feasible deployment alternatives. Our focus here is not the design of algorithms but the overall system engineering approach for selecting a feasible alternative in a large configuration space. Both the approach and the tools assist the designer to derive a feasible deployment model. One of the important benefits of the approach is also the early analysis of the system and the generation of the feasible deployment model at design time.

8.3 FUTURE RESEARCH

In this thesis we have presented several novel approaches to analyze and design integrated IoT systems.

First, we have defined a layered reference architecture for IoT systems and modeled the common and variant features of the most commonly used session layer protocols. Researchers can use the results of this study to elaborate on further research on session layer protocols. Also, our future work for this part is to focus on the performance evaluations of these session layer protocols under different environments (LAN, WAN, etc.) and work conditions (high-performance computing, constrained environments, etc.).

Second, we have presented a systematic approach to guide the architect in designing IoT-based FMIS. Based on FMIS and IoT reference architectures we have described the steps and the modeling approaches for designing the IoT-based FMIS architecture. We have used prospective and retrospective case studies to illustrate our approach. Although, we have showed our approach for two important case studies in the smart agri-food sector the method can be actually applied for the development of other FMISs. We have not focused on the implementation of these systems. For the prospective case study, it is decided to develop first a simulation system to evaluate the outcome of the method. We consider this as part of our future work. Although our method has illustrated the development of IoT-based FMIS systems we could even use the method for developing traditional FMIS systems. In that case we would omit the IoT architecture part and just focus on the development of reference models for FMIS. This part was also considered as another future work.

Third, we provide several integration patterns that can be used to integrate IoT-based systems via different layers of the IoT reference architecture as stated above. This study could be further extended by considering other patterns. Hereby, other type of IoT reference architectures could be considered and based on these the set of patterns that we have described in this chapter could perhaps be enhanced. Further, IoT patterns beyond the integration concern such as security and safety patterns could be identified in the future work.

Fourth, among the session layer protocols of the IoT, we focused on the DDS which is mainly used for machine-to-machine communication in IIoT. We have also shown that DDS can be used for integrating IoT systems. Based on the SLR we have performed we have identified 11 problem categories for DDS. An important number of the problems relate to quality concerns such as reliability, scalability and security. On its turn this observation can trigger further research on quality concerns in DDS. Also, the identification of DDS obstacles is also important beyond the context of IoT. Further research could be carried out in this context. Apart from these, our SLR has resulted in a set of feature diagrams that summarize the reported obstacles as well as the solution directions. The feature models that we have developed can be used to pave the way and support the development of a DDS ontology. We consider this as another possible future work.

Fifth, we have defined a DDS-based architecture for IoT systems. We have adopted architecture viewpoints for modeling DDS, IoT and finally DDS-based IoT systems using a systematic approach. Further, we have applied the layered viewpoint to represent the DDS-based IoT since both the DDS and IoT are often represented as layered structures. As a future work, other architecture viewpoints could be used. In addition, we will adopt the viewpoints for real world industrial IoT projects in which DDS is applied.

Sixth, we have provided a systematic approach to find feasible deployment alternatives for DDS-based systems by extending the DDS UML profile and developing the Deploy-DDS research tool. In our future work, we will do research on further extension and specialization of the approach. In this context, we will consider the adoption of specific quality criteria such as reliability and further focus on the trade-off analysis using multiple quality criteria. Further, we will also consider the analysis and comparison of various algorithm implementations to further optimize the approach. Further, as the future work of our research tool, we will add different algorithms to the Deploy-DDS tool to find feasible deployment alternatives for the DDS-based systems and evaluate and compare the solutions with respect to the existing algorithms.

APPENDICES

APPENDIX A – PRIMARY STUDIES FOR DERIVING CHARACTERISTICS OF DDS

A. B. Zieba, and M. van Sinderen, ‘‘Preservation of Correctness during System Reconfiguration in Data Distribution Service for Real-Time Systems’’, IEEE International Conference on Distributed Computing Systems Workshops, pp. 30, 2006.

B. R. Baldoni, L. Querzoni, and S. Scipioni, ‘‘Event-Based Data Dissemination on Inter-Administrative Domains: Is it Viable?’’, International Workshop on Future Trends of Distributed Computing Systems, pp. 44–50, 2008.

C. P. Boonma, and J. Suzuki, ‘‘Middleware Support for Pluggable Non-Functional Properties in Wireless Sensor Networks’’, IEEE Congress on Services, pp. 360–367, 2008.

D. Corradi, and L. Foschini, ‘‘A DDS-Compliant P2P Infrastructure for Reliable and QoS-Enabled Data Dissemination’’, IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8, 2009.

E. P. Boonma, and J. Suzuki, ‘‘Self-Configurable Publish/Subscribe Middleware for Wireless Sensor Networks’’, IEEE Consumer Communications and Networking Conference, pp. 1–8, 2009.

F. P. Boonma, and J. Suzuki, ‘‘Toward Interoperable Publish/Subscribe Communication between Wireless Sensor Networks and Access Networks’’, IEEE Consumer Communications and Networking Conference, pp. 1–6, 2009.

G. Corradi, L. Foschini, and Luca Nardelli, ‘‘A DDS-Compliant Infrastructure for Fault-Tolerant and Scalable Data Dissemination’’, International Symposium on Computer and Communications, pp. 489–495, 2010.

H. Napolitano, G. Carrozza, A. Bovenzi, and C. Esposito, ‘‘Automatic Robustness Assessment of DDS-Compliant Middleware’’, International Symposium on Dependable Computing, pp. 302–307, 2011.

I. I. S. Monedero, J. P. Molina, J. M. L. Vega, and J. M. L. Soler, ‘‘Bloom filter-based discovery protocol for DDS middleware’’, Journal of Parallel and Distributed Computing, pp. 1305–1317, 2011.

J. W. R. Otte, A. Gokhale, D. C. Schmidt, and J. Willemsen, ‘‘Infrastructure for Component-Based DDS Application Development’’, International Conference on Generative Programming and Component Engineering, pp. 53–62, 2011.

K. J.A. Dianes, M. Diaz, and B. Rubio, ‘‘Using standards to integrate soft real-time components into dynamic distributed architectures’’, Computer Standards and Interfaces, pp. 238–262, 2011.

L. K. Beckmann, and M. Thoss, ‘‘A Wireless Sensor Network Protocol for the OMG Data Distribution Service’’, Intelligent Solutions in Embedded Systems, pp. 45–50, 2012.

M. H. S. Park, J. H. Jang, J. D. Kim, H. I. Jung, and S. H. Lee, ‘‘Bandwidth-Aware DDS Communication in WLAN’’, International Conference on Systems and Informatics, pp. 1542–1546, 2012.

N. Corradi, L. Foschini, J. P. Molina, J. M. L. Soler, ‘‘DDSEnabled Cloud Management Support for Fast Task Offloading’’, International Symposium on Computer and Communications, pp. 67–74, 2012.

O. H. P. Tijero, and J. J. Gutierrez, ‘‘On the schedulability of a data-centric real-time distribution middleware’’, Computer Standards and Interfaces, pp. 203–211, 2012.

P. J. M. L. Vega, J. P. Molina, G. P. Castellote, and J. M. L. Soler, ‘‘A content-aware bridging service for publish/subscribe environments’’, Journal of Systems and Software, pp. 108–124, 2013.

Q. Zhi, W. Z. Shun, D. G. Lan, and D. F. Jun, ‘‘Data management solutions based on the data distribution service communication model’’, The Journal of China Universities of Posts and Telecommunications, pp. 127–132, 2013.

R. P. Bellavista, A. Corradi, L. Foschini, A. Pernafini, ‘‘Data Distribution Service: A Performance Comparison of OpenSplice & RTI Implementations’’, International Symposium on Computer and Communications, pp. 377–383, 2013.

S. K. An, T. Kuroda, A. Gokhale, S. Tambe, and A. Sorbini, ‘‘Modeldriven Generative Framework for Automated OMG DDS Performance Testing in the Cloud’’, International Conference on Generative Programming and Component Engineering, pp. 179–182, 2013.

T. Hakiri, P. Berthou, A. Gokhale, D. C. Schmidt, and T. Gayraud, ‘‘Supporting end-to-end quality of service properties in OMG data distribution service publish/subscribe middleware over wide area networks’’, Journal of Systems and Software, pp. 2574–2593, 2013.

U. D. Feiock, and J. H. Hill, ‘‘Using Component-based Middleware to Design and Implement Data Distribution Servise Systems’’, EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 208–211, 2013.

V. K. An, A. Gokhale, D. C. Schmidt, S. Tambe, P. Pazandak, and G. P. Castellote, ‘‘Content-based Filtering Discovery Protocol: Scalable and Efficient OMG DDS Discovery Protocol’’, Distributed Event Based Systems, pp. 130–141, 2014.

W. T. Celik, O. Koksal, and B. Tekinerdogan, ‘‘Deploy-DDS: Tool Framework for Supporting Deployment Architecture of Data Distribution Service based Systems’’, European Conference on Software Architecture Workshops, pp. Article No: 35, 2014.

X. H. A. Putra, and D. S. Kim, ‘‘Node discovery scheme of DDS combat management system’’, Computer Standards and Interfaces, pp. 20–28, 2014.

Y. K. H. Lee, C. K. Kim, K. T. Kim, and W. T. Kim, ‘‘Router Design for DDS: architecture and performance evaluation’’, International Conference on Big Data and Smart Computing, pp. 250–254, 2014.

Z. Hakiri, P. Berthou, A. Gokhale, D. C. Schmidt, and T. Gayraud, ‘‘Supporting SIP-based end-to-end Data Distribution Service QoS in WANs’’, Journal of Systems and Software, pp. 100–121, 2014.

AA. L. Bertaux, A. Hakiri, S. Medjiah, P. Berthou, and S. Abdellatif, ‘‘A DDS/SDN Based Communication System for Efficient Support of Dynamic Distributed Real-Time Applications’’, Distributed Simulation and Real Time Applications, pp. 77–84, 2014.

AB. S. Pradhan, W. Emfinger, A. Dubey, W. R. Otte, D. Balasubramanian, and A. Gokhale, ‘‘Establishing Secure Interactions Across Distributed Applications in Satellite Clusters’’, Space Mission Challenges for Information Technology, pp. 67–74, 2014.

AC. H. Kim, G. Yoon, W. Lee, J. Park, and H. Choi, ‘‘A Performance Simulator for DDS Networks’’, Conference on Information Networking, pp. 122–126, 2015.

AD. S. Khare, K. An, A. Gokhale, S. Tambe, and A. Meena, ‘‘Reactive Stream Processing for Data Centric Publish/Subscribe’’, Proceedings of the 9th ACM International Conference on Distributed Event Based Systems, pp. 234–245, 2015.

AE. M.R. Khaefi, J. Y. Im, and D. S. Kim, ‘‘An Efficient DDS Node Discovery Scheme for Naval Combat System’’, Emerging Technologies & Factory Automation, IEEE 20th Conference on, pp. 1–8, 2015.

AF. K. H. Lee, C.K. Kim, S. H. Lim, and W. T. Kim, ‘‘Rateless Code Based Reliable Multicast for DDS’’, Big Data and Smart Computing, International Conference on, pp. 150–156, 2015.

AG. B. Kai, and D. Olga, ‘‘sDDS: A portable DDS implementation for WSN & IoT Platforms’’, Intelligent Solutions in Embedded Systems, 12th International Workshop on, pp. 115–120, 2015.

AH. Hector Perez, and J. Javier Gutierrez, ‘‘Modeling the QoS parameters of DDS for event-driven RT Application’’, Journal of Systems and Software, Volume 104, pp. 126–140, 2015.

APPENDIX B – ASSESSMENT OF PRIMARY STUDIES FOR DDS

Table 10-1: Definition of Assessment Questions

Criteria	Q	Definition
Reporting	Q1	Are the aims of study clearly defined?
	Q2	Are the scope, the context and the experimental design of the study clearly stated?
Relevance	Q3	Does the report have implications for research and/or practice?
	Q4	Are the variables used in the evaluation likely to be valid and reliable?
Rigor	Q5	Are the measures used in the study quite explicit & aligned with the research aims?
	Q6	Is the research process documented adequately?
Credibility	Q7	Are the main findings stated clearly in terms of validity and reliability?
	Q8	Is there an explicit statement of the limitations?

Table 10-2: Assessment of Primary Studies

	Reporting		Relevance		Rigor		Credibility		
Primary Study	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Total
A	1.0	0.5	0.5	0.5	0.5	1.0	0.5	0.5	5.0
B	1.0	1.0	0.5	0.5	1.0	0.5	1.0	1.0	6.5
C	1.0	1.0	0.5	0.5	0.5	1.0	0.5	0.0	5.0
D	1.0	1.0	0.5	1.0	1.0	0.5	1.0	0.5	6.5
E	1.0	1.0	0.5	0.5	1.0	0.5	0.5	0.0	5.0
F	1.0	1.0	0.5	0.5	1.0	0.5	0.5	0.0	5.0
G	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.5	7.0
H	1.0	1.0	0.5	1.0	1.0	0.5	1.0	0.5	6.5
I	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.5	7.0
J	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
K	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.5	7.0
L	1.0	1.0	0.5	0.5	0.0	1.0	0.5	0.5	5.0
M	1.0	1.0	1.0	0.5	0.5	0.5	1.0	1.0	6.5
N	1.0	1.0	1.0	0.5	0.5	1.0	0.5	0.5	6.0
O	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
P	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	7.5
Q	0.5	0.5	1.0	1.0	1.0	0.5	0.5	0.5	5.5
R	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.5	7.0
S	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.5	7.0
T	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
U	1.0	1.0	1.0	1.0	0.5	0.5	0.5	0.5	6.0
V	1.0	1.0	1.0	0.5	0.5	0.5	1.0	0.5	6.0
W	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.5	7.0
X	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	7.5
Y	1.0	1.0	1.0	0.5	1.0	0.5	1.0	0.5	6.5
Z	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
AA	1.0	0.5	1.0	0.5	0.5	1.0	1.0	0.5	6.0
AB	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.5	7.0
AC	1.0	0.5	0.5	1.0	0.5	0.5	1.0	0.0	5.0
AD	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
AE	1.0	1.0	1.0	1.0	0.5	0.5	1.0	0.5	6.5
AF	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.5	7.0
AG	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.5	7.0
AH	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5

APPENDIX C – PRIMARY STUDIES FOR DERIVING CHARACTERISTICS OF IOT

A. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys and Tutorials, 17(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

B. Gazis, V., Gortz, M., Huber, M., Leonardi, A., Mathioudakis, K., Wiesmaier, A., … Vasilomanolakis, E. (2015). A survey of technologies for the internet of things. In IWCMC 2015 - 11th International Wireless Communications and Mobile Computing Conference (pp. 1090–1095). https://doi.org/10.1109/IWCMC.2015.7289234

C. Gilchrist, A. (2016). Industry 4.0: the industrial internet of things. Apress.

D. IETF. (2011). XMPP Main. Retrieved March 19, 2018, from https://xmpp.org/

E. IETF. (2013). CoAP. Retrieved from http://coap.technology/

F. Köksal, Ö., & Tekinerdogan, B. (2017). Feature-driven domain analysis of session layer protocols of internet of things. In Proceedings - 2017 IEEE 2nd International Congress on Internet of Things, ICIOT 2017 (pp. 105–112). https://doi.org/10.1109/IEEE.ICIOT.2017.19

G. OASIS. (2011). AMQP. Retrieved February 18, 2018, from http://www.amqp.org/specification/1.0/amqp-org-download

H. OASIS. (2014). MQTT. Retrieved February 18, 2018, from http://mqtt.org/2014/11/mqtt-v3-1-1-now-an-oasis-standard

I. OMG. (2015). DDS Specification V 1.4, 180. Retrieved from http://www.omg.org/spec/DDS/1.4/

J. Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., & Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. IEEE Communications Surveys and Tutorials, 15(3), 1389–1406. https://doi.org/10.1109/SURV.2012.111412.00158

K. Pandya, H. B., & Champaneria, T. A. (2015). Internet of things: Survey and case studies. 2015 International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO), 1–6. https://doi.org/10.1109/EESCO.2015.7253713

L. Schneider, S. (2016). Leading Applications & Architecture for the Industrial IoT. Retrieved from https://www.rti.com/leading-applications-ebook

M. Sheng, Z., Yang, S., Yu, Y., Vasilakos, A., McCann, J., & Leung, K. (2013). A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities. IEEE Wireless Communications, 20(6), 91–98. https://doi.org/10.1109/MWC.2013.6704479

Appendix D – Primary Studies for Deriving Characteristics of FMIS

A. Adamchuk, V. I., Hummel, J. W., Morgan, M. T., & Upadhyaya, S. K. (2004). On-the-go soil sensors for precision agriculture. Computers and Electronics in Agriculture. https://doi.org/10.1016/j.compag.2004.03.002

B. Fountas, S., Carli, G., Sørensen, C. G., Tsiropoulos, Z., Cavalaris, C., Vatsanidou, A., … Tisserye, B. (2015). Farm management information systems: Current situation and future perspectives. Computers and Electronics in Agriculture, 115, 40–50. https://doi.org/10.1016/j.compag.2015.05.011

C. Kaloxylos, A., Eigenmann, R., Teye, F., Politopoulou, Z., Wolfert, S., Shrank, C., … Kormentzas, G. (2012). Farm management systems and the Future Internet era. Computers and Electronics in Agriculture, 89, 130–144. https://doi.org/10.1016/j.compag.2012.09.002

D. Murakami, E., Saraiva, A. M., Ribeiro, L. C. M., Cugnasca, C. E., Hirakawa, A. R., & Correa, P. L. P. (2007). An infrastructure for the development of distributed service-oriented information systems for precision agriculture. Computers and Electronics in Agriculture, 58(1), 37–48. https://doi.org/10.1016/j.compag.2006.12.010

E. Nikkilä, R., Seilonen, I., & Koskinen, K. (2010). Software architecture for farm management information systems in precision agriculture. Computers and Electronics in Agriculture, 70(2), 328–336. https://doi.org/10.1016/j.compag.2009.08.013

F. Rains, G. C., & Thomas, D. L. (2009). Precision farming: an introduction. University of Georgia. Retrieved from https://athenaeum.libs.uga.edu/handle/10724/12223?show=full

G. Schmitz, M., Martini, D., Kunisch, M., & Mösinger, H. J. (2009). AgroXML enabling standardized, platform-independent internet data exchange in farm management information systems. In Metadata and Semantics (pp. 463–468). https://doi.org/10.1007/978-0-387-77745-0_45

H. Shibusawa, S. (2001). Precision Farming Approaches for Small Scale Farms. IFAC Proceedings Volumes, 34(11), 22–27. https://doi.org/10.1016/S1474-6670(17)34099-5

I. Sørensen, C. G., Fountas, S., Nash, E., Pesonen, L., Bochtis, D., Pedersen, S. M., … Blackmore, S. B. (2010). Conceptual model of a future farm management information system. Computers and Electronics in Agriculture, 72(1), 37–47. https://doi.org/10.1016/j.compag.2010.02.003

J. Sørensen, C. G., Pesonen, L., Bochtis, D. D., Vougioukas, S. G., & Suomi, P. (2011). Functional requirements for a future farm management information system. Computers and Electronics in Agriculture, 76(2), 266–276. https://doi.org/10.1016/j.compag.2011.02.005

K. Steinberger, G., Rothmund, M., & Auernhammer, H. (2009). Mobile farm equipment as a data source in an agricultural service architecture. Computers and Electronics in Agriculture, 65(2), 238–246. https://doi.org/10.1016/j.compag.2008.10.005

L. Wang, N., Zhang, N., & Wang, M. (2006). Wireless sensors in agriculture and food industry—Recent development and future perspective. Computers and Electronics in Agriculture, 50(1), 1–14. https://doi.org/10.1016/J.COMPAG.2005.09.003

M. Zhang, N., Wang, M., & Wang, N. (2002). Precision agriculture - A worldwide overview. Computers and Electronics in Agriculture, 36(2–3), 113–132. https://doi.org/10.1016/S0168-1699(02)00096-0

REFERENCES

Adamchuk, V. I., Hummel, J. W., Morgan, M. T., & Upadhyaya, S. K. (2004). On-the-go soil sensors for precision agriculture. Computers and Electronics in Agriculture. https://doi.org/10.1016/j.compag.2004.03.002

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys and Tutorials, 17(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Al-Fuqaha, A., Khreishah, A., Guizani, M., Rayes, A., & Mohammadi, M. (2015). Toward better horizontal integration among IoT services. IEEE Communications Magazine, 53(9), 72–79. https://doi.org/10.1109/MCOM.2015.7263375

Aleti, A., Björnander, S., Grunske, L., & Meedeniya, I. (2009). MOMPES: ArcheOpterix: An Extendable Tool for Architecture Optimization of AADL Models. Retrieved from https://researchbank.swinburne.edu.au/file/64734696-2617-4a83-b47e-2d0fc7cadaec/1/PDF %28Published version%29.pdf

Aleti, A., Grunske, L., Meedeniya, I., & Moser, I. (2009). Let the ants deploy your software - An ACO based deployment optimisation strategy. In ASE2009 - 24th IEEE/ACM International Conference on Automated Software Engineering (pp. 505–509). https://doi.org/10.1109/ASE.2009.59

Alliance for IoT Innovation. (2015). AIOTI WG03 Reports. Retrieved October 11, 2016, from https://aioti.eu/aioti-wg03-reports-on-iot-standards/

Angelo Corsaro. (n.d.). The Simple DDS API. Retrieved March 19, 2018, from https://code.google.com/archive/p/simd-cxx/

Angelov, S., Grefen, P., & Greefhorst, D. (2012). A framework for analysis and design of software reference architectures. Information and Software Technology, 54(4), 417–431. https://doi.org/10.1016/j.infsof.2011.11.009

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). Feature-Oriented Software Product Lines. Feature-Oriented Software Product Lines: Concepts and Implementation (Vol. 9783642375). https://doi.org/10.1007/978-3-642-37521-7

Aqeel-Ur-Rehman, Abbasi, A. Z., Islam, N., & Shaikh, Z. A. (2014). A review of wireless sensors and networks' applications in agriculture. Computer Standards and Interfaces. https://doi.org/10.1016/j.csi.2011.03.004

Awduche, D. O. (1999). MPLS and traffic engineering in IP networks. IEEE Communications Magazine, 37(12), 42–47. https://doi.org/10.1109/35.809383

Bass, L., Clements, P., & Kazman, R. (2003). Software Architecture in Practice , Second Edition. Software Architecture. https://doi.org/10.1024/0301-1526.32.1.54

Beck, H. (2001). Agricultural enterprise information management using object databases, Java, and CORBA. Computers and Electronics in Agriculture, 32(2), 119–147. https://doi.org/10.1016/S0168-1699(01)00162-4

Becker, S., Koziolek, H., & Reussner, R. (2009). The Palladio component model for model-driven performance prediction. Journal of Systems and Software, 82(1), 3–22. https://doi.org/10.1016/j.jss.2008.03.066

Botta, A., De Donato, W., Persico, V., & Pescape, A. (2014). On the integration of cloud computing and internet of things. In Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014 (pp. 23–30). IEEE. https://doi.org/10.1109/FiCloud.2014.14

Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., & Grose, T. J. (2003). Eclipse Modeling Framework. Addison-Wesley.

Bushmann, F., Meunier, R., & Rohnert, H. (1996). Pattern-oriented software architecture: A system of patterns. John Wiley&Sons, 1, 476. https://doi.org/10.1192/bjp.108.452.101

Calbimonte, J. P., Sarni, S., Eberle, J., & Aberer, K. (2014). XGSN: An open-source semantic sensing middleware for the web of things. In CEUR Workshop Proceedings (Vol. 1401, pp. 51–66).

Capilla, R., Bosch, J., Trinidad, P., Ruiz-Cortés, A., & Hinchey, M. (2014). An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. Journal of Systems and Software, 91(1), 3–23. https://doi.org/10.1016/j.jss.2013.12.038

Celik, T., & Tekinerdogan, B. (2013). S-IDE: A tool framework for optimizing deployment architecture of High Level Architecture based simulation systems. Journal of Systems and Software, 86(10), 2520–2541. https://doi.org/10.1016/j.jss.2013.03.013

Celik, T., Köksal, Ö., & Tekinerdogan, B. (2014). Deploy-DDS. In Proceedings of the 2014 European Conference on Software Architecture Workshops - ECSAW '14 (pp. 1–5). New York, New York, USA: ACM Press. https://doi.org/10.1145/2642803.2642838

Celik, T., Tekinerdogan, B., & İmre, K. M. (2013). Deriving Feasible Deployment Alternatives for Parallel and Distributed Simulation Systems. ACM Transactions on Modeling and Computer Simulation, 23(3), 1–24. https://doi.org/10.1145/2499913.2499917

Chaudhary, S., Sorathia, V., & Laliwala, Z. (2004). Architecture of sensor based agricultural information system for effective planning of farm activities. In Proceedings - 2004 IEEE International Conference on Services Computing, SCC 2004 (pp. 93–100). https://doi.org/10.1109/SCC.2004.1357994

Clements, P. (2006). Software Product Lines. Software Product Lines, 3714(3), 1–105. https://doi.org/10.1007/11554844

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., … Nord, R. (2011). Documenting Software Architectures: Views and Beyond (2nd ed.). Addison-Wesley. Retrieved from https://www.amazon.com/Documenting-Software-Architectures-Views-Beyond/dp/0321552687

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., … Stafford, J. (2010). Documenting Software Architectures. Style DeKalb IL, 592. https://doi.org/10.1109/ICSE.2003.1201264

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., & Bone, M. (2010). The concept of reference architectures. Systems Engineering, 13(1), 14–27. https://doi.org/10.1002/sys.20129

Croes, E. (2015). Software Architectural Styles in the Internet of Things. Radboud University Nijmegen. Retrieved from http://www.ru.nl/publish/pages/769526/z_evertson_croes_master_thesis_s4241754.pdf

Czarnecki, K., Kim, C. H. P., & Kalleberg, K. T. (2006). Feature models are views on ontologies. 10th International Software Product Line Conference, SPLC 2006, 1, 41–51. https://doi.org/10.1109/SPLINE.2006.1691576

Daly, C. (2004). Emfatic Language Reference. Retrieved March 10, 2018, from http://www.eclipse.org/epsilon/doc/articles/emfatic/

Daniel Kellmereit, D. O. (2013). The Silent Intelligence--The Internet of Things. DnD Ventures. Retrieved from https://books.google.com.tr/books/about/The_Silent_Intelligence.html?id=0SQLnwEACAAJ&redir_esc=y

David Garlan, M. S. (1994). An Introduction to Software Architecture. New Jersey: World Scientific Publishing Company. Retrieved from http://www.cs.cmu.edu/afs/cs/project/able/www/paper_abstracts/intro_softarch.html

Demirli, E., & Tekinerdogan, B. (2011). Software language engineering of architectural viewpoints. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6903 LNCS, pp. 336–343). https://doi.org/10.1007/978-3-642-23798-0_36

Diaz-Cacho, M., Delgado, E., Falcon, P., & Barreiro, A. (2015). IoT integration on industrial environments. In IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS (Vol. 2015–July). https://doi.org/10.1109/WFCS.2015.7160553

Dlodlo, N., & Kalezhi, J. (2015). The internet of things in agriculture for sustainable rural development. In 2015 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC) (pp. 13–18). IEEE. https://doi.org/10.1109/ETNCC.2015.7184801

Dybå, T., & Dingsøyr, T. (2008). Strength of evidence in systematic reviews in software engineering. Proc. Second ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. T, (7465), 178–187. https://doi.org/http://doi.acm.org/10.1145/1414004.1414034

Dybå, T., Kitchenham, B. A., & Jorgensen, M. (2005). Evidence-based software engineering for practitioners. IEEE Software, 22(1), 58–65. https://doi.org/10.1109/MS.2005.6

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. ACM Computing Surveys, 35(2), 114–131. https://doi.org/10.1145/857076.857078

Evans, D. (2011). The Internet of Things How the Next Evolution of the Internet Is Changing Everything. Retrieved from https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). The Architecture Analysis & Design Language (AADL): An Introduction, (February), CMU/SEI-2006-TN-011. Retrieved from http://www.sei.cmu.edu/publications/pubweb.html

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Fountas, S., Carli, G., Sørensen, C. G., Tsiropoulos, Z., Cavalaris, C., Vatsanidou, A., … Tisserye, B. (2015). Farm management information systems: Current situation and future perspectives. Computers and Electronics in Agriculture, 115, 40–50. https://doi.org/10.1016/j.compag.2015.05.011

Fountas, S., Wulfsohn, D., Blackmore, B. S., Jacobsen, H. L., & Pedersen, S. M. (2006). A model of decision-making and information flows for information-intensive agriculture. Agricultural Systems, 87(2), 192–210. https://doi.org/10.1016/j.agsy.2004.12.003

Gazis, V., Gortz, M., Huber, M., Leonardi, A., Mathioudakis, K., Wiesmaier, A., … Vasilomanolakis, E. (2015). A survey of technologies for the internet of things. In IWCMC 2015 - 11th International Wireless Communications and Mobile Computing Conference (pp. 1090–1095). https://doi.org/10.1109/IWCMC.2015.7289234

Gilchrist, A. (2016). Industry 4.0 : the industrial internet of things. Apress.

Griffin, T. W., & Lowenberg-DeBoer, J. (2005). Worldwide adoption and profitability of precision agriculture Implications for Brazil. Revista de Política Agrícola, 14(4), 20–37. Retrieved from https://seer.sede.embrapa.br/index.php/RPA/article/view/549/498

Griffin, T. W., Shockley, J. M., Mark, T. B., Shannon, D. K., Clay, D. E., & Kitchen, N. R. (2018). Economics of Precision Farming. https://doi.org/10.2134/precisionagbasics.2016.0098

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. Scandinavian Journal of Information Systems, 19(2), 87–92. https://doi.org/http://aisel.aisnet.org/sjis/vol19/iss2/4

IETF. (2011). XMPP Main. Retrieved March 19, 2018, from https://xmpp.org/

IETF. (2013). CoAP. Retrieved from http://coap.technology/

IETF. (2017). Internet Engineering Task Force. Retrieved March 10, 2018, from https://www.ietf.org/

Iijima, K. (2012). Systems Development Technology for Public Infrastructure. Hitachi Review, 61(3), 159–166. Retrieved from https://pdfs.semanticscholar.org/e896/8e42869c25662db9ab0fc9512cf052510036.pdf

Islam, S., Lindstrom, R., & Suri, N. (2006). Dependability driven integration of mixed criticality SW components. In Proceedings - Ninth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2006 (pp. 485–495). IEEE. https://doi.org/10.1109/ISORC.2006.

ITU. (2005). The Internet of Things. Itu Internet Report 2005, 212. https://doi.org/10.2139/ssrn.2324902

Jawad, H., Nordin, R., Gharghan, S., Jawad, A., & Ismail, M. (2017). Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review. Sensors, 17(8), 1781. https://doi.org/10.3390/s17081781

Juneau, J. (2013). Java Message Service. Retrieved March 10, 2018, from http://www.oracle.com/technetwork/java/docs-136352.html

Kaloxylos, A., Eigenmann, R., Teye, F., Politopoulou, Z., Wolfert, S., Shrank, C., … Kormentzas, G. (2012). Farm management systems and the Future Internet era. Computers and Electronics in Agriculture, 89, 130–144. https://doi.org/10.1016/j.compag.2012.09.002

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A Survey on Application Layer

Protocols for the Internet of Things. Transaction on IoT and Cloud Computing, 3(1), 11–17. https://doi.org/10.5281/ZENODO.51613

Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012). Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In 2012 10th International Conference on Frontiers of Information Technology (pp. 257–260). IEEE. https://doi.org/10.1109/FIT.2012.53

Kitchenham, B. a, & Pfleeger, S. L. (2002). Principles of Survey Research Part 3: Constructing a Survey Instrument. ACM SIGSOFT Software Engineering Notes, 27(2), 20. https://doi.org/10.1145/511152.511155

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Engineering, 2, 1051. https://doi.org/10.1145/1134285.1134500

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering - A systematic literature review. Information and Software Technology, 51(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

Kolovos, D. S., Paige, R. F., & Polack, F. A. C. (2006). Eclipse Development Tools for Epsilon. Eclipse Modeling Symposium on Eclipse Summit Europe, (October). Retrieved from https://pdfs.semanticscholar.org/8268/b0795e2af075b4d0f78d1cc558dd69b24ea7.pdf

Kolovos, D. S., Rose, L. M., Abid, S. Bin, Paige, R. F., Polack, F. A. C., & Botterweck, G. (2010). Taming EMF and GMF using model transformation. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6394 LNCS, pp. 211–225). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-16145-2_15

Koziolek, A., & Reussner, R. (2011). Towards a generic quality optimisation framework for component-based system models. In Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering - CBSE ’11 (p. 103). New York, New York, USA: ACM Press. https://doi.org/10.1145/2000229.2000244

Köksal, Ö., & Akyuz, M. (2017). Aspect Oriented Approach for Cross-Cutting Concerns in Data Distribution Service Based Systems. Journal of Science and Engineering, 19(55), 43–56. https://doi.org/10.21205/deufmd.2017195528

Köksal, Ö., & Tekinerdogan, B. (2017a). Feature-driven domain analysis of session layer protocols of internet of things. In Proceedings - 2017 IEEE 2nd International Congress on Internet of Things, ICIOT 2017 (pp. 105–112). https://doi.org/10.1109/IEEE.ICIOT.2017.19

Köksal, Ö., & Tekinerdogan, B. (2017b). Obstacles in Data Distribution Service Middleware: A Systematic Review. Future Generation Computer Systems, 68, 191–210. https://doi.org/10.1016/j.future.2016.09.020

Kruchten, P. (2000). The Rational Unified Process: An Introduction. The Rational Unified Process: An Introduction. Addison-Wesley. Retrieved from https://books.google.com.tr/books/about/The_Rational_Unified_Process.html?id=RYCMx6o47pMC&redir_esc=y

Kruize, J. W., Wolfert, J., Scholten, H., Verdouw, C. N., Kassahun, A., & Beulens, A. J. M. (2016). A reference architecture for Farm Software Ecosystems. Computers and Electronics in Agriculture, 125, 12–28. https://doi.org/10.1016/J.COMPAG.2016.04.011

Kutter, T., Tiemann, S., Siebert, R., & Fountas, S. (2011). The role of communication and co-operation in the adoption of precision farming. Precision Agriculture, 12(1), 2–17. https://doi.org/10.1007/s11119-009-9150-0

Linseisen, H. (2001). Development of a Precision Farming Information System. In Proceedings of the Third European Conference on Precision Agriculture (pp. 689–694). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.460.5744

Lomotey, R. K., Pry, J., Sriramoju, S., Kaku, E., & Deters, R. (2017). Middleware Framework for IoT Services Integration. In Proceedings - 2017 IEEE 6th International Conference on AI and Mobile Services, AIMS 2017 (pp. 89–92). IEEE. https://doi.org/10.1109/AIMS.2017.20

Lowenberg-DeBoer, J., Erickson, K., & Vogel, K. A. (2000). Precision farming profitability. Agricultural Research Programs, Purdue University.

Ma, J., Zhou, X., Li, S., & Li, Z. (2011). Connecting Agriculture to the Internet of Things through Sensor Networks. In 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing (pp. 184–187). IEEE. https://doi.org/10.1109/iThings/CPSCom.2011.32

Malek, S., Medvidovic, N., & Mikic-Rakic, M. (2012). An Extensible Framework for Improving a Distributed Software System's Deployment Architecture. IEEE Transactions on Software Engineering, 38(1), 73–100. https://doi.org/10.1109/TSE.2011.3

Martens, A., Koziolek, H., Becker, S., & Reussner, R. (2010). Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering - WOSP/SIPEW '10 (p. 105). New York, New York, USA: ACM Press. https://doi.org/10.1145/1712605.1712624

McEwen, A., & Cassimally, H. (2014). Designing the Internet of Things. Wiley. Retrieved from https://www.wiley.com/en-tr/Designing+the+Internet+of+Things-p-9781118430620

Mehrabi, A., Mehrabi, S., & Mehrabi, A. (2009). An Adaptive Genetic Algorithm for Multiprocessor Task Assignment Problem with Limited Memory. Proceedings of the World Congress on Engineering and Computer Science, II. Retrieved from https://www.researchgate.net/publication/44260490_An_Adaptive_Genetic_Algorithm_for_Multiprocessor_Task_Assignment_Problem_with_Limited_Memory

Milsoft. (n.d.). MilSOFT DDS Middleware. Retrieved March 19, 2018, from http://www.milsoft.com.tr/en/portfolio/mil-dds/

Moore, B., Dean, D., Gerber, A., Wagenknecht, G., & Vanderheyden, P. (2004). Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM, International Technical Support Organization. https://doi.org/10.1147/JRD.2010.2041693

Murakami, E., Saraiva, A. M., Ribeiro, L. C. M., Cugnasca, C. E., Hirakawa, A. R., & Correa, P. L. P. (2007). An infrastructure for the development of distributed service-oriented information systems for precision agriculture. Computers and Electronics in Agriculture, 58(1), 37–48. https://doi.org/10.1016/j.compag.2006.12.010

Myerson, J. (2002). The Complete Book of Middleware. Auerbach. Retrieved from http://books.google.com/books?hl=en&lr=&id=Gc886KgsdcsC&oi=fnd&pg=PR15&dq=The+Complete+Book+of+Middleware&ots=tJvUIis_-B&sig=u38CH5ZT8eUBClg80hN061_WFAg%5Cnhttp://books.google.com/books?hl=en&lr=&id=Gc886KgsdcsC&oi=fnd&pg=PR15&dq=The+complete+book+of+midd

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, Q. Z. (2017). IoT Middleware: A Survey on Issues and Enabling Technologies. IEEE Internet of Things Journal, 4(1), 1–20. https://doi.org/10.1109/JIOT.2016.2615180

Nikkilä, R., Seilonen, I., & Koskinen, K. (2010). Software architecture for farm management information systems in precision agriculture. Computers and Electronics in Agriculture, 70(2), 328–336. https://doi.org/10.1016/j.compag.2009.08.013

OASIS. (2010). OASIS. Retrieved March 10, 2018, from https://www.oasis-open.org/

OASIS. (2011). AMQP. Retrieved February 18, 2018, from http://www.amqp.org/specification/1.0/amqp-org-download

OASIS. (2014). MQTT. Retrieved February 18, 2018, from http://mqtt.org/2014/11/mqtt-v3-1-1-now-an-oasis-standard

Olivieri, A. C., & Rizzo, G. (2015). Scalable Approaches to Integration in Heterogeneous IoT and M2M Scenarios. In Proceedings - 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2015 (pp. 358–363). https://doi.org/10.1109/IMIS.2015.55

OMG. (n.d.-a). Data Distribution Service Specification. Retrieved February 18, 2018, from http://www.omg.org/spec/DDS/

OMG. (n.d.-b). OMG. Retrieved May 17, 2018, from http://www.omg.org

OMG. (n.d.-c). OMG DDS Portal. Retrieved March 10, 2018, from http://portals.omg.org/dds/

OMG. (2006a). Deployment and Configuration of component-based Distributed Applications Specification Version 4.0. Retrieved from https://www.omg.org/spec/DEPL/About-DEPL/

OMG. (2006b). The CORBA Component Model Specification Version 4.0. Retrieved from https://www.omg.org/spec/CCM/About-CCM/

OMG. (2008). OMG Marte Specification - beta2. Retrieved from http://www.omg.org/omgmarte/Specification.htm

OMG. (2010). DDS UML Profile - beta1. Retrieved from http://www.omg.org/cgi-bin/doc?ptc/10-05-17.pdf

OMG. (2012a). CORBA Specification Version 3.3. Retrieved from http://www.omg.org/spec/CORBA/

OMG. (2012b). DDS for Lightweight CCM Specification Version 1.1. Retrieved from http://www.omg.org/spec/DDS4CCM/1.1

OMG. (2014). DDS Interoperability Wire Protocol Specification Version 2.2. Retrieved March 10, 2018, from http://www.omg.org/spec/DDSI-RTPS/2.2/

OMG. (2015a). DDS + DLRL Specification V 1.4. Retrieved from https://www.omg.org/spec/DDS-DLRL/About-DDS-DLRL/

OMG. (2015b). DDS Specification V 1.4, 180. Retrieved from http://www.omg.org/spec/DDS/1.4/

OMG. (2016). DDS Security Specification V 1.0. Retrieved March 10, 2018, from https://www.omg.org/spec/DDS-SECURITY/1.0/

Otte, W. R., Gokhale, A., Schmidt, D. C., & Willemsen, J. (2011). Infrastructure for Component-Based DDS Application Development. In GPCE 11: PROCEEDINGS OF THE TENTH INTERNATIONAL CONFERENCE ON GENERATIVE PROGRAMMING AND COMPONENT ENGINEERING (Vol. 47, pp. 53–61). https://doi.org/10.1145/2047862.2047872

Öztürk, K., & Tekinerdogan, B. (2011). Feature Modeling of Software as a Service Domain to Support Application Architecture Design. Proc. of the Sixth International Conference on Software Engineering Advances (ICSEA 2011), 142–148.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., & Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. IEEE Communications Surveys and Tutorials, 15(3), 1389–1406. https://doi.org/10.1109/SURV.2012.111412.00158

Pandya, H. B., & Champaneria, T. A. (2015). Internet of things: Survey and case studies. 2015 International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO), 1–6. https://doi.org/10.1109/EESCO.2015.7253713

Prismtech. (n.d.-a). PrismTech. Retrieved March 10, 2018, from http://www.prismtech.com/

Prismtech. (n.d.-b). Vortex OpenSplice Modeler. Retrieved March 10, 2018, from http://www.prismtech.com/vortex/vortex-opensplice/tools/modeler

Pritchard, J. (1999). COM and CORBA side by side: Architectures, Strategies, and Implementations. Addison-Wesley.

Rains, G. C., & Thomas, D. L. (2009). Precision farming: an introduction. University of Georgia. Retrieved from https://athenaeum.libs.uga.edu/handle/10724/12223?show=full

RTI. (n.d.). RTI. Retrieved March 10, 2018, from https://www.rti.com/

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering. https://doi.org/10.1007/s10664-008-9102-8

Saaty, T. L. (1988). What is the Analytic Hierarchy Process? In Mathematical Models for Decision Support (pp. 109–121). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-83555-1_5

Salman, T., & Jain, R. (2017). Networking protocols and standards for internet of things. In Internet of Things and Data Analytics Handbook (pp. 215–238). https://doi.org/10.1002/9781119173601.ch13

Schimmelpfennig, D. (2016). Farm Profits and Adoption of Precision Agriculture. Retrieved from https://www.ers.usda.gov/webdocs/publications/80326/err-217.pdf?v=42661

Schmitz, M., Martini, D., Kunisch, M., & Mösinger, H. J. (2009). AgroXML enabling standardized, platform-independent internet data exchange in farm management information systems. In Metadata and Semantics (pp. 463–468). https://doi.org/10.1007/978-0-387-77745-0_45

Schneider, S. (2016). Leading Applications & Architecture for the Industrial IoT. Retrieved from https://www.rti.com/leading-applications-ebook

Seelan, S. K., Laguette, S., Casady, G. M., & Seielstad, G. A. (2003). Remote sensing applications for precision agriculture: A learning community approach. Remote Sensing of Environment. https://doi.org/10.1016/j.rse.2003.04.007

Shelby, Z., Hartke, K., & Bormann, C. (2004). The Constrained Application Protocol (CoAP). Kenchiku Setsubi Iji Hozen Suishin Kyōkai. Retrieved from https://www.ietf.org/rfc/rfc7252.txt

Sheng, Z., Yang, S., Yu, Y., Vasilakos, A., McCann, J., & Leung, K. (2013). A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities. IEEE Wireless Communications, 20(6), 91–98. https://doi.org/10.1109/MWC.2013.6704479

Singh, M., Rajan, M. A., Shivraj, V. L., & Balamuralidhar, P. (2015). Secure MQTT for Internet of Things (IoT). In Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015 (pp. 746–751). https://doi.org/10.1109/CSNT.2015.16

Sørensen, C. G., Fountas, S., Nash, E., Pesonen, L., Bochtis, D., Pedersen, S. M., … Blackmore, S. B. (2010). Conceptual model of a future farm management information system. Computers and Electronics in Agriculture, 72(1), 37–47. https://doi.org/10.1016/j.compag.2010.02.003

Sørensen, C. G., Pesonen, L., Bochtis, D. D., Vougioukas, S. G., & Suomi, P. (2011). Functional requirements for a future farm management information system. Computers and Electronics in Agriculture, 76(2), 266–276. https://doi.org/10.1016/j.compag.2011.02.005

Sparx Systems. (n.d.). Enterprise Architect UML modeling tool. Retrieved March 10, 2018, from http://sparxsystems.com/

Steinberg, D. (2009). EMF: Eclipse Modeling Framework. Addison-Wesley.

Steinberger, G., Rothmund, M., & Auernhammer, H. (2009). Mobile farm equipment as a data source in an agricultural service architecture. Computers and Electronics in Agriculture, 65(2), 238–246. https://doi.org/10.1016/j.compag.2008.10.005

Stone, H. S. (1977). Multiprocessor Scheduling with the Aid of Network Flow Algorithms. IEEE Transactions on Software Engineering, SE-3(1), 85–93. https://doi.org/10.1109/TSE.1977.233840

Svogor, I., & Carlson, J. (2016). SCALL: Software Component Allocator for Heterogeneous Embedded Systems. Proceedings of the 2015 European Conference on Software Architecture Workshops - ECSAW ’15, 1–5. https://doi.org/10.1145/2797433.2797501

Tekinerdogan, B. (2014). Software Architecture. In T. Gonzalez, J. Diaz-Herrera, & A. Tucker (Eds.), Computing Handbook: Computer Science and Software Engineering (p. 2280). Chapman and Hall/CRC . Retrieved from https://www.crcpress.com/Computing-Handbook-Third-Edition-Computer-Science-and-Software-Engineering/Gonzalez-Diaz-Herrera-Tucker/p/book/9781439898529

Tekinerdogan, B., Celik, T., & Köksal, Ö. (2018). Generation of feasible deployment configuration alternatives for Data Distribution Service based systems. Computer Standards & Interfaces, 58, 126–145. https://doi.org/10.1016/J.CSI.2018.01.002

Tekinerdogan, B., & Köksal, Ö. (2018). Pattern Based Integration of Internet of Things Systems. In Internet of Things - ICIOT 2018 (pp. 19–33). Springer, Cham. https://doi.org/10.1007/978-3-319-94370-1_2

Tekinerdogan, B., Köksal, Ö., & Celik, T. (2017). Data Distribution Service-Based Architecture Design for the Internet of Things Systems. In Connected Environments for the Internet of Things (pp. 269–285). Springer, Cham. https://doi.org/10.1007/978-3-319-70102-8_13

Tekinerdogan, B., & Öztürk, K. (2013). Feature-Driven Design of SaaS Architectures. In Zaigham Mahmood &

Saqib Saeed (Eds.), Software Engineering Frameworks for the Cloud Computing Paradigm (pp. 189–212). Springer. https://doi.org/10.1007/978-1-4471-5031-2_9

Tekinerdogan, B., Öztürk, K., & Doğru, A. (2011). Modeling and reasoning about design alternatives of software as a service architectures. In Proceedings - 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011 (pp. 312–319). IEEE. https://doi.org/10.1109/WICSA.2011.49

Tekinerdogan, B., Sozer, H., & Aksit, M. (2012). Feature-Based Rationale Management System for Supporting Software Architecture Adaptation. International Journal of Software Engineering and Knowledge Engineering, 22(7), 945–964. https://doi.org/10.1142/S021819401250026X

Turkish Land Crop Office. (2017). Turkish Grain Report 2016. Retrieved from http://www.tmo.gov.tr/Upload/Document/hububat/hububatraporu2016.pdf

Turkish Ministry of Agriculture. (2018). Herbal Production Data. Retrieved from http://www.tarim.gov.tr/sgb/Belgeler/SagMenuVeriler/BUGEM.pdf

Tüzün, E., Tekinerdogan, B., Kalender, M. E., & Bilgen, S. (2015). Empirical evaluation of a decision support model for adopting software product line engineering. Information and Software Technology, 60, 77–101. https://doi.org/10.1016/j.infsof.2014.12.007

Ucar, B., Aykanat, C., Kaya, K., & Ikinci, M. (2006). Task assignment in heterogeneous computing systems. Journal of Parallel and Distributed Computing, 66(1), 32–46. https://doi.org/10.1016/j.jpdc.2005.06.014

Verdouw, C., Wolfert, J., & Tekinerdogan, B. (2016). Internet of Things in agriculture. CAB Reviews: Perspectives in Agriculture, Veterinary Science, Nutrition and Natural Resources, 11(35). https://doi.org/10.1079/PAVSNNR201611035

Vermesan, O., & Friess, P. (2014). Internet of Things – From Research and Innovation to Market Deployment. River Publishers. https://doi.org/10.1007/s11036-012-0415-x

Voelter, M., Kolb, B., Efftinge, S., & Haase, A. (2006). From Front End To Code - MDSD in Practice. Retrieved March 10, 2018, from http://www.eclipse.org/articles/Article-FromFrontendToCode-MDSDInPractice/article.html

Wolfert, J., Verdouw, C. N., Verloop, C. M., & Beulens, A. J. M. (2010). Organizing information integration in agri-food-A method based on a service-oriented architecture and living lab approach. Computers and Electronics in Agriculture, 70(2), 389–405. https://doi.org/10.1016/j.compag.2009.07.015

Wolfert, S., Ge, L., Verdouw, C., & Bogaardt, M.-J. (2017). Big Data in Smart Farming – A review. Agricultural Systems, 153, 69–80. https://doi.org/10.1016/J.AGSY.2017.01.023

Xi Chen, R. J. (2014). Constrained Application Protocol for Internet of Things. Wireless and Mobile Networking, 857, 1–12. Retrieved from http://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html

Yin, R. K. (2009). Case study research: design and methods / Robert K. Yin. Applied social research methods series: 5. https://doi.org/10.1097/FCH.0b013e31822dda9e

Yoshikawa, Y., Sato, A., Hirasawa, S., Takahashi, M., & Yamamoto, M. (2012). Hitachi's vision of the smart city. Hitachi Review. Retrieved from http://www.hitachi.com/rev/pdf/2012/r2012_03_101.pdf

Zhang, H., Babar, M. A., & Tell, P. (2011). Identifying relevant studies in software engineering. Information and Software Technology, 53(6), 625–637. https://doi.org/10.1016/J.INFSOF.2010.12.010

Zhang, N., Wang, M., & Wang, N. (2002). Precision agriculture - A worldwide overview. Computers and Electronics in Agriculture, 36(2–3), 113–132. https://doi.org/10.1016/S0168-1699(02)00096-0

SUMMARY

IoT (Internet of Things) enables anytime and anyplace connectivity for anything by linking the objects of the real world with the virtual world. In the near future, it is predicted that more than 50 billion of things will be connected to the internet. This will lead to many different IoT-based systems that will have a huge impact on the society. Often, these IoT systems will not be standalone but will be composed with other different systems to create additional value. Hence, with the heterogeneity and the integration of IoT-based systems with other IoT-based or non-IoT-based systems has become an important challenge.

In this thesis, the main objective is to analyze, design and integrate IoT-based systems and to answer the following research questions:

RQ1. What are the characteristic features of IoT systems?

RQ2. How to design the architecture for an IoT-based system?

RQ3. What are the identified obstacles of the data distribution (DDS) middleware?

RQ4. What are the solution directions for the identified obstacles of DDS?

RQ5. What are the approaches for integrating multiple IoT-based systems?

RQ6. How to design a DDS-based IoT system?

RQ7. How to derive feasible deployment alternatives for DDS-based systems?

In order to answer these research questions, three different research methodologies were used: Systematic Literature Review, Design Science Research, and Case Study Research.

In chapter 2, we have applied a feature driven domain analysis of IoT systems. We have presented the reference architecture for IoT and discussed the corresponding layers. Among these layers, we have focused on the session layer of the IoT. The protocols in this layer are related with the communication sessions of the IoT systems and hence determine the communication characteristics of the IoT systems. We have presented the common and variant features of the most commonly used session layer protocols, namely AMQP, CoAP, DDS, MQTT, and XMPP which are used for communication between M2M (machine-to-machine), M2S (machine-to-server), and S2S (server-to-server). Further, we have provided an evaluation framework to compare session layer communication protocols. Among these protocols, we focused on the DDS that is mainly used for M2M communication in Industrial Internet of Things (IIoT).

In chapter 3, we have described an architecture design method for architecting IoT systems for the Farm Management Information Systems (FMIS) domain. Hereby, we have also developed a family feature diagram to represent the common and variant features of IoT-based FMIS. In order to illustrate our approach, we have performed a systematic case study approach including the IoT-based wheat and tomato production with IoT-based FMIS. The case study research showed that the approach was both effective and practical.

In chapter 4, we have presented the method for designing integrated IoT systems. We showed that integration of IoT-based systems can be at different layers including session layer, cloud layer and application layer. Further we have shown that the integration is typically carried out based on well-defined patterns, that is, generic solutions structures for recurring problems. We have systematically compiled and structured the 15 different integration patterns which can be used in different combinations and likewise supporting the composition of different IoT systems. We have illustrated the use of example patterns in a smart city case study and have shown that the systematic structuring of the integration patterns is useful for integrating IoT systems.

A systematic research methodology has been applied in chapter 5 to identify the current obstacles to adopt DDS and their solution directions. We have selected 34 primary studies among the 468 identified papers since the introduction of DDS in 2003. We identified 11 basic categories of problems including complexity of DDS configuration, performance prediction, measurement and optimization, implementing DDS, DDS integration over WAN, DDS using wireless networks and mobile computing, interoperability among DDS vendor implementations, data consistency in DDS, reliability in DDS, scalability in DDS, security, and integration with event-based systems. We have adopted feature diagrams to summarize and provide an overview of the identified problem and their solutions defined in the primary studies.

DDS based architecture design for IoT systems is presented in chapter 6. DDS is considered to be a potential middleware for IoT because of its focus on event-driven communication in which quality of service is also explicitly defined. We provide a systematic approach to model the architecture for DDS-based IoT in which we adopted architecture viewpoints for modeling DDS, IoT and DDS-based IoT systems. We have integrated and represented the architecture models that can be used to model DDS-based IoT systems for various application domains.

When designing DDS-based systems typically multiple different alternatives can be derived. Chapter 7 presents an approach for deriving feasible DDS configuration alternatives. For this we have provided a systematic approach for extending the DDS UML profile and developed an extensible tool framework Deploy-DDS to derive feasible deployment alternatives given the application model, the physical resources, and the execution configurations. The tool framework Deploy-DDS implements a set of predefined algorithms and can be easily extended with new algorithms to support the system architect. We have evaluated the approach and the tool framework for a relevant IoT case study on smart city engineering.

Chapter 8 concludes the thesis by summarizing the contributions.

SAMENVATTING

IoT (Internet of Things) maakt altijd en overal connectiviteit mogelijk van allerdaagse voorwerpen die kunnen communiceren met personen en met andere objecten, en die op grond hiervan autonome beslissingen kunnen nemen. In de nabije toekomst wordt voorspeld dat meer dan 50 miljard entiteiten met internet verbonden zullen zijn. Dit zal leiden tot veel verschillende op IoT gebaseerde systemen die zodoende een grote impact op de samenleving zullen hebben. Vaak zullen deze IoT-systemen niet op zichzelf staan, maar zullen gecombineerd worden met andere verschillende systemen om zodoende additionele waarde te creëren. Bijgevolg is de heterogeniteit en de integratie van IoT systemen met andere IoT systemen een belangrijke uitdaging geworden.

In dit proefschrift is het hoofddoel het analyseren, ontwerpen en integreren van IoT-gebaseerde systemen en hiermee het beantwoorden van de volgende onderzoeksvragen:

1. Wat zijn de karakteristieke kenmerken van IoT-systemen?

2. Hoe ontwerp je de architectuur voor een op IoT gebaseerd systeem?

3. Wat zijn de geïdentificeerde obstakels in het DDS-domein?

4. Wat zijn de oplossingsrichtingen voor de geïdentificeerde obstakels van DDS?

5. Wat zijn de benaderingen voor het integreren van meerdere IoT-gebaseerde systemen?

6. Hoe een DDS-gebaseerd IoT-systeem te ontwerpen?

7. Hoe haalbare haalbaarheidsalternatieven voor DDS-gebaseerde systemen kunnen worden afgeleid?

Om deze onderzoeksvragen te beantwoorden, werden drie verschillende onderzoeksmethoden gebruikt: Systematisch literatuur onderzoek, design science research en case study research.

In hoofdstuk 2 hebben we een feature-driven domeinanalyse van de IoT-systemen toegepast. We hebben de architectuur en de softwarelagen van de IoT-referentiearchitectuur gepresenteerd. Hierbij hebben we ons vooral gericht op de sessielaag van het IoT. De protocollen in deze laag zijn gerelateerd aan de communicatiesessies van de IoT-systemen en bepalen daarom de communicatiekenmerken van de IoT-systemen. We hebben de algemene en variantkenmerken gepresenteerd van de meest gebruikte sessielaagprotocollen, namelijk AMQP, CoAP, DDS, MQTT en XMPP die worden gebruikt voor communicatie tussen M2M (machine-to-machine), M2S (machine-naar-server)) en S2S (server-naar-server). Verder hebben we een evaluatiekader geboden om communicatieprotocollen voor sessielagen te vergelijken. Vervolgens hebben we ons gericht op de DDS die hoofdzakelijk wordt gebruikt voor M2M-communicatie in Industrial Internet of Things (IIoT).

In hoofdstuk 3 hebben we een nieuwe aanpak beschreven voor het ontwerpen van IoT-systemen voor Farm Management Information Systemen (FMIS). Verder hebben we een domein gebaseerde aanpak gebruikt en een feature diagram ontwikkeld met de gemeenschappelijke en variabele eigenschappen van een op IoT-gebaseerde FMIS. Om onze aanpak te illustreren, hebben we een systematische case study-aanpak uitgevoerd, waaronder de op IoT gebaseerde tarwe- en tomatenproductie met IoT-gebaseerde FMIS. Uit het case study-onderzoek bleek dat de aanpak zowel effectief als praktisch was.

Hoofdstuk 4 presenteert het ontwerp van geïntegreerde IoT-systemen. We hebben aangegeven dat integratie van op IoT gebaseerde systemen zich op verschillende niveaus kan bevinden, waaronder sessielaag, cloudlaag en applicatielaag. Verder hebben we laten zien dat de integratie meestal wordt uitgevoerd op basis van goed gedefinieerde patronen, dat wil zeggen generieke oplossingsstructuren voor terugkerende problemen. We hebben systematisch de 15 verschillende integratiepatronen gecompileerd en gestructureerd die in verschillende combinaties kunnen worden gebruikt. We hebben het gebruik van patronen in een Smart City-casestudy geïllustreerd en hebben aangetoond dat de systematische structurering van de integratiepatronen nuttig is voor het ontwikkelen van IoT-systemen.

In hoofdstuk 5 is een systematische onderzoeksmethode toegepast om de huidige obstakels voor het adopteren van DDS en hun oplossingsrichtingen te identificeren. We hebben 34 primaire onderzoeken geselecteerd uit de 468 geïdentificeerde artikelen sinds de introductie van DDS in 2003. We identificeerden 11 basiscategorieën van problemen, waaronder de complexiteit van DDS, prestatievoorspelling, meting en optimalisatie, implementatie van DDS, DDS-integratie via WAN, DDS met behulp van draadloos netwerken en mobiel computergebruik, interoperabiliteit tussen implementaties van DDS-leveranciers, gegevensconsistentie in DDS, betrouwbaarheid in DDS, schaalbaarheid in DDS, beveiliging en integratie met op events gebaseerde systemen. We hebben de feature diagrammen gebruikt om een overzicht te geven van de geïdentificeerde problemen en hun oplossingen.

In hoofdstuk 6 wordt een op DDS gebaseerd architectuurontwerp voor IOT-systemen gepresenteerd. DDS wordt beschouwd als een potentiële middleware voor IOT vanwege de focus op event-based communicatie. We bieden een systematische aanpak om de architectuur van DDS-gebaseerde IoT-systemen te modelleren. We hebben DDA-gebaseerde IoT-systemen geïntegreerd voor verschillende toepassingsdomeinen.

Bij het ontwerpen van op DDS gebaseerde systemen kunnen typisch meerdere verschillende alternatieven worden afgeleid. Hoofdstuk 7 presenteert een aanpak voor het afleiden van haalbare DDS-configuratie-alternatieven. Het DDS UML-profiel is een uitbreidbaar hulpmiddelraamwerk dat de implementatie van DDS mogelijk maakt. Het hulpmiddelraamwerk Deploy-DDS implementeert een set vooraf gedefinieerde algoritmen en kan eenvoudig worden uitgebreid naar nieuwe systemen. We hebben de aanpak en het tool-framework geëvalueerd voor een relevante case study over smart city engineering.

Hoofdstuk 8 sluit het proefschrift af door de bijdragen samen te vatten.

ACKNOWLEDGEMENTS

I got the opportunity to start a PhD with Professor Bedir Tekinerdogan in 2010. In this journey many people helped me and I wish to express my gratitude to each of them.

First of all, I would like to thank my promotor Bedir Tekinerdogan for his endless patience and support. He always helped me with different aspects and made this PhD degree possible for me. I greatly appreciate all his efforts. I feel very fortunate to have you as my promotor and finalizing this journey was not possible without your commitment.

I would like to thank Wageningen University for accepting me into the PhD program. Also, I would like to thank the information technology group and the secretaries for their support.

My PhD journey started at Bilkent University where I have taken 8 graduate courses and passed a very difficult qualification exam. I would like to thank Bilkent University for accepting me and giving me an opportunity to start PhD in one of the best universities in Turkey.

I would like to thank Doğan Altunbay. It was not possible for me to pass qualification exam in Bilkent University without his tremendous support. Thank you for being my teammate with your right questions and solutions for passing the qualification exam.

I want to thank my coauthors Turgay Çelik and Mirun Akyüz. Turgay encouraged and helped me to improve my papers and writing the thesis. I am very grateful for that. Mirun also supported me in writing our paper and with the lectures that we took together.

I would like to thank my company ASELSAN. They allowed me to continue my PhD study while I was working for ASELSAN. It was a great chance to have such an opportunity.

I would like to thank General Directorate of Agricultural Research and Policies (TAGEM) and especially Dr. Hakan Yıldız and Turgay Polat for their support and evaluation of our work related with precision agriculture. They always showed their interest in my work and were open for nice conversations.

Such a hard work would not have been possible without the support of my family. I would like to thank them for the trust they had in me and all the possible support they have provided during my research. They always encouraged me to challenge myself and, it is lovely to see and feel their genuine expression of pleasure in the completion of this thesis.

Ömer Köksal   Wageningen, July 2018

About the Author

Ömer Köksal was born on the 26th of December 1969 in Ankara, Turkey.

He is graduated from Middle East Technical University where he received his MSc degree (2004) in Computer Engineering. He is working for ASELSAN since 1996 where he is now fulfilling the position of senior lead software design engineer. His main research topics include distributed computing, software architecture design, model-driven development, aspect-oriented software development, design patterns, and software product line engineering.

He has been active in many national and international projects and worked as the software team leader in many application domains including avionic software development, simulator development, and command and control software development for naval and unmanned platforms.

In 2015 Ömer Köksal started his PhD at Wageningen University. In his PhD research he studied Software Architecture Design and Analysis of Data Distribution Service Based Internet of Things Systems and their integration. The graduation ceremony of his PhD was in July 2018.

His research focuses on information management in internet of things and data distribution service domains. His purpose is to improve and facilitate the use of these technologies in the development of enterprise software projects.

Colophon
Financial support from Wageningen University – Information Technology Group printing this thesis is gratefully acknowledged.

Cover design by:
Ebru Zırhlıoglu

Printed by:
Digiforce