QoS-A ware deployment of lighting control behaviors

Zhao, Z.

Award date:
2018

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
System Architecture and Networking Research Group

QoS-Aware Deployment of Lighting Control Behaviors

Master Thesis

Ziyuan Zhao

Supervisors:
dr. Tanir Ozcelebi (TU/e)
dr. Ben Pronk (Signify)
dr. Qingzhi Liu (TU/e)

Committee:
dr. Tanir Ozcelebi (TU/e)
dr. Ben Pronk (Signify)
dr. Qingzhi Liu (TU/e)
dr. Majid Nabi (TU/e)

Eindhoven, September 2018

Abstract

The Internet of Things (IoT) has been called the next generation of the industrial revolution. By blending the physical and digital realms, the IoT is profoundly changing the way we relate to our environment and information. In a smart building, the IoT-based (connected) lighting system is integrated with advanced Things (e.g. sensors, luminaries) and communication channels that help develop new services and application domains. The lighting system as an IoT infrastructure can support IoT applications in many areas.

The rise in connected lighting systems has drawn researchers' attention in determining the optimal deployment of applications in a lighting system under functional and Quality of Service (QoS) requirement. In this thesis, **we propose a methodology for determining the optimal deployment of the application in a lighting system that supports any network topology and network configuration**. Latency and resilience are criteria in this thesis. Though "resilience" is used in many different application domains, the quantification of it has not been done well. Therefore, we also propose **a methodology based on an automatic generation of a fault tree to evaluate the resilience (dependability) of applications in the lighting system,** when permanent faults occur in distributed components in the application.

The proposed methodology was validated by using a Java-based platform developed by TNO, we have done some implementation work on a Java-based platform named Cosim from TNO. We developed a tool to evaluate deployment decisions and can automatically generate the optimal deployment decision regarding requirements. The tool allows users to specify the application via domain specific language (DSL) and manipulate parameters representing network configuration as well as user requirements. The deliverables include a design matrix, an evaluation model and a tool that assists users in making the deployment decision.

Preface

The research "QoS-Aware Deployment of Lighting Control Behaviors" has been conducted to fulfil the graduation requirements of the Master's degree of Embedded Systems at the Eindhoven University of Technology. My research questions have been formulated together with my supervisor, dr. Tanir Ozelebi. The research has been conducted in the R&D department of Signify in Eindhoven between January and August 2018 consisting of a 3-month-long preparation phase and a 5-month-long graduation phase.

First of all, I would like to thank TU/e and Signify for the opportunity to study and work in the field of IoT. I would like to thank dr. Tanir Ozcelebi and dr. Qingzhi Liu for their inspiration and great help. I would like to thank dr. Ben Pronk for his guidance, patience and insight into work, it was a very pleasure to work with him. I would like to thank ir. Jack Sleuters for his ideas and feedbacks in the implementation work. I would like to thank dr. Majid Nabi for being my exam committee.

I would like to thank my colleagues for their kind support. I would like to thank my friends, including but not limited to, ir. Srikanth Sistu, ir. Yongmin Qiu, ir. Wenguang Feng, ir. Adam Zika, ir. Benjamin Feleki, Tianyu Liu and Yuyang Qi for their encouragement and support. Finally, I am particularly indebted to thank my parents for their love.

Contents

List of Figures

List of Tables

Abbreviation

IoT	Internet of Things
DSL	Domain-Specific Language
QoS	Quality of Service
WSN	Wireless Sensor Network
QA	Quality Attribute
FT	Failure Tree
FTA	Failure Tree Analysis
FP	Failure Probability
CDF	Cumulative Distribution Function
MTTF	Mean Time to Failure
LUT	Lookup Table
LCB	Lighting Control Behavior
SPOF	Single Point of Failure as known as One Point of Failure

Notation Table

Table 1: Table of Notation My Research

s_i	:	physical sensor i
i	:	identifier
sc_i	:	the connect between the physical sensor i and the controller
$sobj_i$	:	logical sensor object i
sg_i	:	sensor group i
a_i	:	physical actuator i
ac_i	:	the connect between the physical actuator i and the controller
$aobj_i$	:	logical actuator object i
ag_i	:	actuator group i
L_{s_i}	:	The latency from the sensor i to the controller
L_{a_i}	:	The latency from the controller to the actuator i
L_{rsp}	:	Response time. The end-to-end delay on a critical path
L_{syn}	:	Synchronicity. The maximum difference on latency between actuators
$F_i(t)$	:	The probability that component i would fail in (0, t]
$D_i(t)$	:	The probability that component i continue functioning (doesn't fail)in (0, t].
F_i	:	The probability that component i would fail in (0, unit time]
$MTTF_{disrupt}$	:	Mean time to the system starting disruption
$MTTF_{degrade}$	:	Mean time to the system starting degrading
$F_{degrade}$	:	The probability that the system would start degrading in (0, unit time]
$F_{disrupt}$	:	The probability that the system would start degrading in (0, unit time]
		Decision Variables
$\forall components$ on FTA	=	$\begin{cases} 1, & \text{failure} \\ 0, & \text{not fail} \end{cases}$

Chapter 1

Introduction

It is estimated that there will be 50 billion devices connected to the Internet by 2020[13]. The Internet of Things (IoT) is the next evolution of the internet. An IoT system consists of services distributed over distinct devices. In an IoT-based lighting system, the "Things" are normally luminaires, sensors and dedicated devices, which are mostly constrained in one or more ways: They are limited in resources like CPU, memory and battery. They may be challenged with unreliable or lossy communication, as well as wireless technologies with limited bandwidth. They may be integrated with dedicated functionality as sensing, actuating, data collecting and communicating. They may be placed in diverse settings such as in buildings or outside. In the system, they collaborate with each other to fulfil the common goals of the lighting control but also other IoT applications.

The control behaviour which is the primary application of the IoT-based lighting system is increasingly complicated and prone to updates as new devices are integrated into existing systems. In the current business, a modern professional lighting system for offices is sold as part of "project business" where the final control system installed in a building is almost always unique. It would involve core R&D teams when it faces extension or other changes. To support the increasing scale of business and the demands for extensibility of the lighting system, the definition, creation, modification and maintenance of the lighting system requires a more concise and simple description. Also, the deployment of application components requires more flexibility.

Confronted with the challenges from IoT, Signify has decided to implement a domain-specific language (DSL) together with TNO for the lighting system domain. It aims at describing the lighting system concisely and defining the lighting control behaviour in a technology-independent way. It also aims at supporting users in the domain, who are not experts in software development, to construct appropriate control applications. For example, a building manager without full knowledge of the target platform can easily use the DSL to describe the lighting behaviour. Currently, there is a simulation tool developed by TNO to simulate the lighting system and control behaviours specified by DSL. This work was carried out due to such requirements which is interesting both academically and by Signify.

TU/e defines a generic life cycle model[20] for IoT application shown in Figure1.1. A specific IoT-based system (lighting system) and IoT applications have their instantiation of the generic life cycle. An instantiation includes stakeholders' responsibility for activities in the life cycle and the procedure for realising the life cycle stages. The workflow of the realisation of IoT application consists of such stages as construction, deployment, execution and so on. In the lighting system, the instantiation of those stages include compiling DSL code in the cloud, deploying converted code to Things and executing the code on Things.

Figure 1.2 shows an example workflow of the realisation of the life cycle states including construction, deployment and execution. In the construction stage, the cloud application compiles DSL source code to a specific language code such as C, C++, Java. It may further compile the code into either a machine code, a bytecode or an executable. Those applications are deployed to end devices via the network in the deployment stage. Afterwards, end devices are responsible for installation and the operation. During the execution, end devices upload the configuration information for further use like reconstruction.

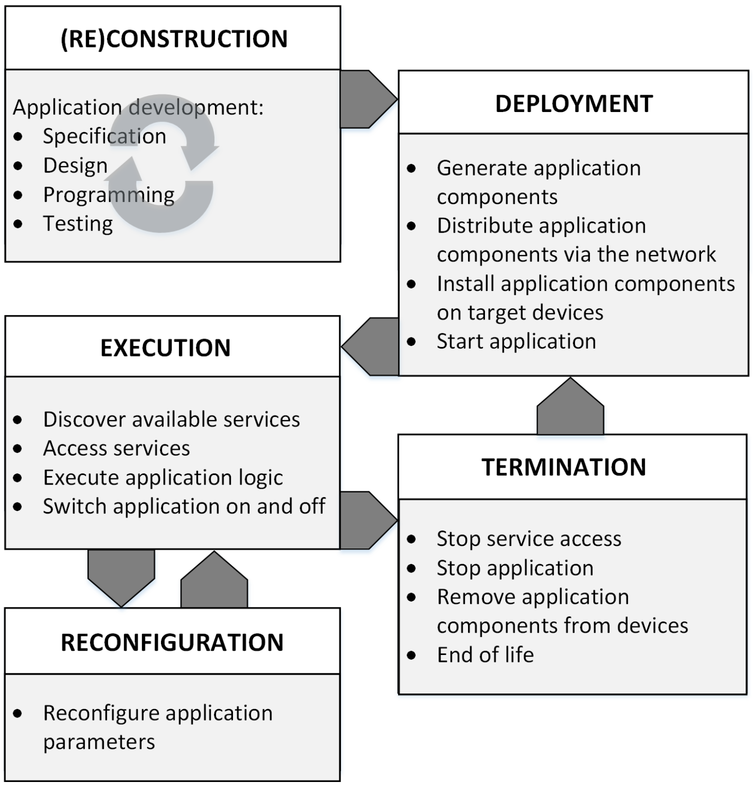

(RE)CONSTRUCTION
Application development:
Specification
Design
Programming
Testing
DEPLOYMENT
Generate application components
Distribute application components via the network
Install application components on target devices
Start application
EXECUTION
Discover available services
Access services
Execute application logic
Switch application on and off
TERMINATION
Stop service access
Stop application
Remove application components from devices
End of life
RECONFIGURATION
Reconfigure application parameters

Figure 1.1: Generic life cycle of IoT Application (Rahman, Ozcelebi, & Lukkien, 2017[20])

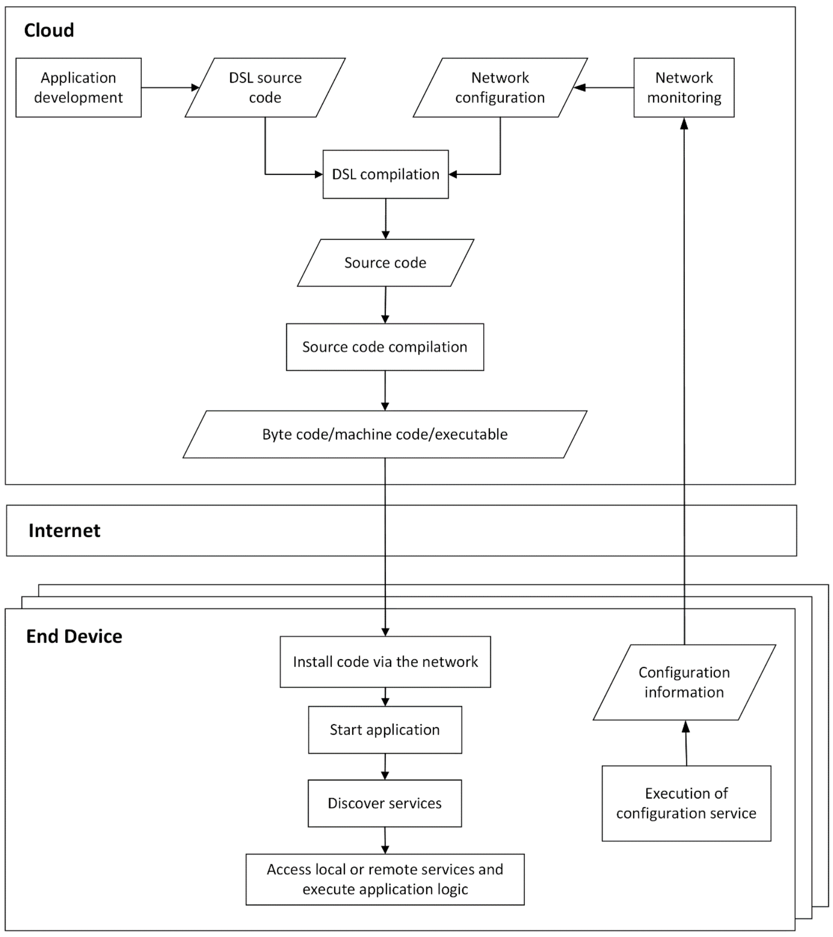


Figure 1.2: A workflow for realising the IoT application lifecycle stages of construction, deployment and execution

1.1 Current Design, Decision-making and Installation Process

The process for the creation of a lighting installation in a building can be divided into several phases. A new lighting installation starts with a lighting design in which the required things (e.g. sensors and luminaires) and their coordinators in the building are defined. This step defines the visual aspects of the installation and considers customer preference, aesthetic considerations and basic regulations. This step is followed by a physical design phase of the installation including the properties (e.g. quantity, capabilities and location) of all infrastructure elements required for the installation. Infrastructure elements are gateways, routers, network cables, power and so on. After manufacturing and delivery, such a system is installed by installers that also installs cabling, power, networks and performs a basic test of connectivity as well as functionality. Finally, a commissioner deploys the system in which the required control behaviour, the grouping of sensor and actuators per area, linking of behaviour between various areas is configured in the nodes. This phase normally ends with the handover and signoff of the installation to the customer this phase normally ends.

1.2 Motivation

Current practices based on experience and heuristics deliver a well-behaved system in most cases. However, there is always a small percentage of systems that display problems after deployment that require the attention of maintenance staff and updates to either the physical installation or the deployment. With the increasing scale of business in connected systems, these still represent an avoidable cost for the business. Also, with increasing deployment of sensors in lighting infrastructure the communication load on the lighting system grows, potentially interfering with the basic control functions and increasing the number of issues. Finally, it is unclear if the systems that are already in the field now are optimal from a performance point of view. With increasing size, complexity and scale of connected systems simple over-dimensioning of the system is not a permanent solution. Because of these trends, Signify is interested in a more solid upfront design of the lighting network and deployment. Deployment tooling that can adequately predict the performance and other Quality of Service (QoS) parameters like the reliability for a certain deployment is one of the research areas explored. The long-time vision being an automatic generation of the optimal deployment for a given installation.

From an academic perspective, this master thesis intends to make the deployment stage in Figure1.1 more concrete by looking into "Distribute application components via the network". We explore means to identify the optimal deployment decision, to model the deployment, to evaluate deployment, and to abstract attributes from the application description and network configuration to support this process. Different deployment decisions can yield different application performance. Metrics and evaluation schemes aim to quantify the application's performance for these cases. Multi-objective optimisation is used to facilitate the identification of an optimal deployment decision given multiple, sometimes conflicting, requirements. A tool is needed to provide insights into different configurations and their quality attributes that influence the deployment decision.

1.3 Research Questions

The research of this thesis focuses on determining the optimal deployment decision an IoT-based lighting system that is aware of the functional requirement and QoS requirement, which are latency and resilience in this project. The research questions are listed below:

1. How to make optimal deployment decisions?
    (a) How to identify the deployment and optimal decision?
    (b) What is a suitable decision making process?
    (c) How to compare different deployment decision?
    (d) What are the lighting control behaviour and its functional requirement?
2. Based on question 1, what are the suitable evaluation methods to compare deployment decisions?
    (a) What are the sub-criteria and their metrics?
    (b) What is a mathematical model for computing QoS quantitatively?
    (c) What information provided by the application specification and network configuration can be extracted as factors in the calculation?

1.4 Outline

In chapter 2, we present a survey of existing research on deployments, network configuration models, QoS (latency and resilience) evaluation methods and optimisation. It points to the gap of work on deployments and resilience evaluation.

In chapter 3, we introduce the operational concept of lighting control behaviours, networks in lighting systems, existing DSLs used in lighting systems, sub-criteria for QoS and a simulation platform. A detailed problem statement is described in this chapter.

In chapter 4, the infrastructure, application as well as input of each evaluation function are mathematically defined.

In chapter 5, we present the decision-making matrix that includes criteria, metrics and a decision process. Focus gradient in this project is described.

In chapter 6, we introduce the evaluation methods for performance yield by the deployment. Evaluation methods used are aware of factors retrieved from application specification, the network configuration and metrics of criteria.

In chapter 7, we introduce the implementation work and simulation environment.

In chapter 8, the evaluation of the tool is done, and the results are described

In chapter 9, future work and final conclusions are discussed.

Chapter 2

Related Work

In this chapter, we review existing work on network abstraction, IoT applications deployments, QoS evaluation, multi-objective optimisation problem.

2.1 Network Abstraction

With a given topology and attributes assigned on nodes and links, many routing techniques can find an optimal routing, build a virtual network or mask connectivity details in a physical network. The attributes can be latency, bandwidth, reliability or other quality profiles. [3] presents a survey of state-of-the-art routing techniques in WSNs.

In the context of IoT, formal modelling approaches have been recently proposed to achieve connectivity and coverage optimisation of WSNs. [16] has proposed an algorithm to map virtual network to the components of a physical network and [11] proposes an algorithm on node mapping and link mapping. The two approaches in [16] and [11] study network virtualisation and can offer a communication link between two endpoints with quality profiles assigned. Furthermore, [9] has done some work on network abstraction from the type of connection technologies employed at the wireless sensor network (Bluetooth, Zigbee, RFID, etc.)

Based on those studies, qualities of a communication link such as latency, bandwidth, reliability between two endpoints can be extracted from WSNs with any configuration in the media layers denoted in the Open Systems Interconnection model (OSI model). This thesis will use a network model, which stores the extracted information, to describe the networks in lighting systems.

2.2 QoS-aware Deployment of IoT Applications

There are existing approaches to model infrastructure, application and evaluation of deployment. [9] proposes a model to support the QoS-aware deployment of multicomponent IoT application to Fog infrastructure. This approach describes the network configuration in the infrastructure, interactions among application components and Things. It defines a Fog infrastructure as a 4-tuple $\langle\ T, F, C, L\ \rangle$, where T is a set of Things with functionalities, F is a set of Fog nodes with software and hardware capabilities, C is a set of Cloud data centres and L is a set of available communication links between nodes. It further defines an application as a triple $\langle\ \Gamma, \Lambda, T\ \rangle$ where Γ is a set of software components, Λ denotes interaction among components and T is a set of Things requested by the application. Then it formalises the deployment procedure and offers algorithms to find available deployments.

[30] formalises the QoS-aware deployment as a maximum weighted bipartite problem. Given

the user preferences and the matching score of applications on Things, it assigns weights to different deployment and presents an algorithm based on the Integer Linear Programming model to find the optimal deployment.

[26] models and formulates the problem that function placement influences the latency, availability or other qualities on a service chain, where "service chain" refers to how functions compose a service. It uses accumulation to calculate the end-to-end latency and availability on a service chain. Furthermore, the paper presents two QoS-aware deployment strategies that are based on Integer Linear Programming and an efficient heuristic respectively to obtain the optimal deployment.

2.3 The DSLs in Lighting Systems

The DSL describes the infrastructure and IoT applications in the lighting system. DSLs in this context are formal languages with formal syntax that can generate executable models (simulators) for a particular application domain and thereby allowing static and dynamic validation of system instances. TNO and Signify have created following DSLs before the start of this project:

1. Building DSL, describing a building and its physical components. The components have such configuration parameters as name, type and coordinates.
2. Template DSL, describing the control functionality for a room or area that can be visualised as a state machine. This describes the behaviour of logical objects.
3. Control DSL, mapping the logical objects required by Template DSL onto Building DSL's physical components.

The lighting system's behaviour in physical space is described by Control DSLs, which combines one or multiple Template DSLs that each independently describes a distinct control behaviour and together yield a more complex control behaviour for the complete lighting system. A Template DSL describes the lighting system's behaviour for a control area abstractly without coupling the control behaviour to an actual building topology.

These DSLs are coupled, and the simulators generated from the DSL's can be executed in a co-simulation framework developed by TNO, which addresses the timeliness of execution, synchronicity, data exchange and coherency of simulation. These DSLs specify the function and component mapping as shown in the operational concept 3.1. The simulation environment will be further explained in this report. The objective of each DSL will be discussed in chapter 3

2.4 Latency Evaluation

OpenAIS[1] evaluates latency by taking the following aspects into account, which are the configuration parameter of physical layers:

- Bandwidth of the physical layer.
- Number of hops between nodes for multi-hop networks.
- Duty cycling delay.
- Average media access time.
- Average packet loss rate.

It further gives the formula to calculate the communication cost based on the listed configuration attributes (quality profiles). Similarly, [26] has shown that the end-to-end latency is the accumulation of communication cost on links and nodes on the communication path.

2.5 Scenarios and Communication Load

Communication load (message load) plays an important role in latency evaluation. The Messages sent over the network in lighting systems are generally generated by their sensors when events happen or controllers when they act on actuators. The latter depends on the event from sensors and the control logic defined by Template DSL

Given a Template DSL and statistics of events, it is feasible to estimate the communication load over the network. The master thesis [31] studies the relationship of distribution of occupancy event and network traffic in lighting systems given a control logic.

A scenario describes events with time stamps that happened on physical sensors in an area and every event has its type (occupancy, vacancy or daylight). By following the consequent messages delivered through the network, we can estimate the total communication load and further evaluate the latency for different controller allocations.

2.6 Multi-objective Optimization Problem

Multiple criteria affect the decision on deployments. In addition, several criteria may be in conflict with each other. [17] summarizes the methods of multi-criteria optimization into two fundamental approaches: Single-objective methods and Pareto optimality. The former is to convert multi-criteria back to single-criterion optimization. Weighted Sum is one of those methods. It requires given information from the user to assign importance on every criterion.

Finding the Pareto Frontier is one of the latter (Pareto Approaches), which includes one or more optimal solutions. A solution belongs to Pareto Frontier if no other solution can dominant it regarding all criteria. This approach can provide all possible optimised solutions of all criteria simultaneously, which can be studied and compared in depth. Using Pareto Frontier can filter solutions to save workload for further process.

[17] found that there is no single approach is superior. The selection of specific methods depends on the type of information provided, users' preference, requirements on solutions and the availability of software.

2.7 Resilience

Recently resilience has been studied in the field of communication networks[24]. Resilient communication networks aim to provide and maintain service in view of the following occurrence of faults:

- Enable user and applications to access information when needed such as sensor monitoring
- Maintain end-to-end communication association
- Provide distributed processing and networked storage

In this section, we review the fundamental concepts of fault-tolerant systems, the definition of dependability as well as its sub-criteria and metrics. Then we address the similarity in problem space between the lighting system and other projects. Finally, we address the current gap in dependability evaluation in lighting systems. The resilience of an application in a lighting system must be evaluated because of its physical network and logical objects. Most studies on resilience are analysing networks configuration rather than considering both application's properties and networks configuration.

2.7.1 Dependability, failures, failure probability, faults and time to failure

[5] introduces dependability as a system property that integrates attributes such as reliability, availability, safety, security, survivability, maintainability. It says that dependability consists of three parts: the threat to, the attributes of, and the means by which dependability is attained, as shown in Figure 2.1. The **threats, availability, reliability** and **fault forecasting** are of our interests. The following attributes are defined as in[5]:

- **Dependability** of a computing system is the ability to deliver service that can justifiably be trusted.
- The **service** delivered by a system is its behaviour as its user perceives it. **Correct service** is delivered when the service implements the system function.
- A **user** is another system that interacts with the service.
- The **function** of a system is what the system is intended to do, and is described by the functional specification.
- A **system failure** is an event that occurs when the delivered service deviates from correct service.
- A **failure** is a transition from the right service to an incorrect service. It is an alternate definition of **dependability**.
- A **fault** is the adjudged or hypothesized cause of an **error**. An **error** is a part of the system state that may cause a subsequent failure.

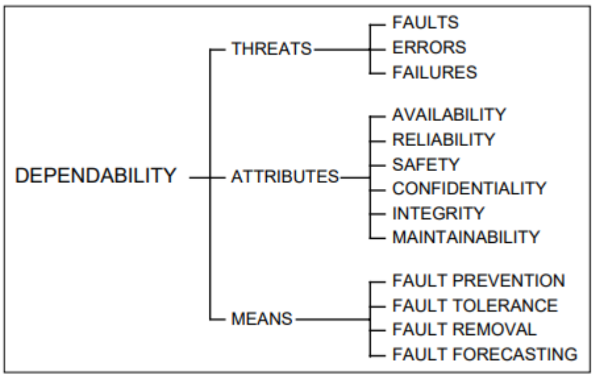


Figure 2.1: Dependability Tree

Mean Time to Failure

Mean time to failure (MTTF) is one of the basic measures of dependability, which is the expected value of the failure density function, and the mean time to repair (MTTR) is the expected value of the repair density function.

[24] describes that dependability consists of two major aspects that are reliability and availability. Availability is the probability that service remains operable when needed, which requires the knowledge of MTTF and MTTR. Reliability is a continuity of service, which is used to characterise if a component/system/service remains operable for a specific period. [23] define reliability as the probability that a component does not fail in the time interval (0, t]. Considering that the time to failure of components, T, is a random variable defined by a cumulative distribution function F(t) (CDF), the reliability R(t) is given by:

	Property	**RoSES Graceful Degradation**
Fault Model	Fault Duration	Permanent
	Fault Manifestation	fail silent components, potentially correlated
	Fault Source	All non-malicious sources
	Granularity	Component failure in distributed embedded systems
	Fault Profile Expectations	Random; arbitrary
System Response	Fault Detection	State variable staleness
	Degradation	fail-operational; Maximize system utility
	Fault Response	Reconfigure SW based on data and control flow graphs
	Recovery	Reconfigure SW & reboot system
	Time constants	Long time between failures;Can handle multiple failures
	Assurance	Future work; reliability-driven

Table 2.1: Problem Spaces and it addressed by the RoSES project

$$R(t) = Pr[T > t] = 1 - F(t)$$

which is the probability of no failure in [0, t]. We use this probability to represent dependability and to evaluate the resilience of the service.

Fault Forecasting

[5] Fault forecasting is one of the means to attain dependability, which aims at estimating the present number and the likely consequence of faults. It is conducted by evaluating the system behaviour concerning fault occurrence. Evaluation has two aspects:

- **Qualitative Evaluation** which aims to identify, classify, rank the failure modes, or the event combinations (component failures or environmental conditions) that would lead to system failures.
- **Quantitative Evaluation** which aims to evaluate, concerning probabilities, the extent to which some of the attributes of dependability are satisfied. those attributes are then viewed as measures of dependability

2.7.2 Self-Healing System

One of the potential approaches to achieving dependable system operation is to incorporate so-called "self-healing" mechanisms into system architectures and implementations. The lighting system has similar properties to a self-healing system that has similar goals to the general area of dependable computing systems, and many "self-healing techniques" ultimately are dependable computing techniques.

[15] proposes a taxonomy for describing the problem space of self-healing systems including fault models and system response listed in Table2.1. This paper also points out that any fault-tolerant system should have a specified fault model. Furthermore, three projects are described in this paper, and one of them is RoSES, which is a project that is exploring graceful degradation as a means to achieve dependable systems. We find many similarities between RoSES and a lighting system, the properties and description listed in Table2.1 helps identify faults and system's response in a lighting system.

2.7.3 Error handling, graceful degradation and redundancy

[1] describes error handling as an essential property of software-intensive systems as software and systems are never error-free and hardware, as well as communication malfunction, happens

sometimes. Degradation on performance happens when a system does not restore complete functionality after a fault[15]. The degree of degraded operation provided by a self-healing system is its resilience to damage.

Regarding the description in [1], in a lighting system, malfunction of a logical object should leave all functionality not involving that object operational. For example, the group behaviour should continue if one of the actuators (sensors) belonging to the group fails. Though one luminaire fails to act according to the control logic, the rest continues their group activities. This is considered as graceful degradation[10].

There are several graceful degradation measures, for instance, compensation, redundancy and buffering. We focus on redundancy, which is a technique by masking faults to hide the occurrence of failures by making a system fault tolerant. With physical redundancy[25], extra equipment or processors are added to make it possible for the system as a whole to tolerant the loss or malfunctioning of some components. Extra processors can be added to the system so that if a small number of the crash, the system can still function correctly.

2.7.4 Analysis methods of dependability

Fault tree (FT) and Markov chain are two common analysis methods used in modelling fault-tolerant systems. [6] gives a comparison of the two. The two analysis methods are used in two approaches to modelling complex system, which are structural decomposition and behavioural decomposition. The first one divides the system into smaller subsystems, analysing the dependability and then combining the subsystem solution to obtain the system solution. The second decomposes fault-occurrence/repair behaviour into relatively infrequent events.

[19] discusses the dependability evaluation methods of fault tree analysis (FTA) and the effect of active redundancy on the dependability of a system with calculation. The redundancy of units in this paper is similar to the duplication of controller introduced before.

[6] points out that a FT representation of the system is often more concise than the corresponding Markov chain, but a Markov chain can model more complex behaviours. [21] has introduced the FT as a graphical model that represents the combination of events that lead to a system failure. The model uses a tree-like structure composed of events and logic gates, where events represent either normal or faulty conditions and gates represent the relationship among events. The inputs of these gates are single or combination of events resulted from the output of other gates. The process of building a FT is performed deductively and starts by defining the *TOP event* representing the system failure condition. From the TOP event, the possible root causes are identified by proceeding backwards. Events at the bottom are referred to as *basic events*. There are several types of gates, such as *AND* and *OR*, shown in Figure2.2[21].

From a probabilistic point of view, the assessment of a FT consists of calculating the probability of the TOP event starting from the probabilities of the basic events. Assume a gate with n independent inputs, where the occurrence of event i is described by means of a cumulative distribution function $F_i(t)$, then the gate output $F_fc(t)$ is shown in Figure2.2.

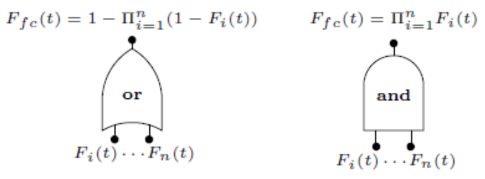

$F_{fc}(t) = 1 - \Pi_{i=1}^{n}(1 - F_i(t))$
or
$F_i(t) \cdots F_n(t)$
$F_{fc}(t) = \Pi_{i=1}^{n} F_i(t)$
and
$F_i(t) \cdots F_n(t)$

Figure 2.2: Cumulative Distribution Function of Logic Combination

2.7.5 System response, resilience curve and metrics

A lot of work has been on done describing several stages of a system's response when facing faults in engineering, social science and ecology. The stages and related performance usually include:

1. Reliable state: The system operates normally.
2. Unreliable state: Performance degrades
3. Disrupted state: Performance reaches its bottom
4. Recovery state: Recovery procedure starts
5. Recovered steady state: Recovery completes

Those stages in different domains are discussed in [8] [27] [29]. Those studies illustrate the stages shown in the figures2.3 2.4 2.5. The illustrations show the similarity of system responses in different domains. The vertical axis shows the performance of a system when the horizontal axis denotes the timeline. Each 't' on the timeline represent the moment when the system state changes. These papers point out that those moments and duration of states can indicate the dependability of a system.

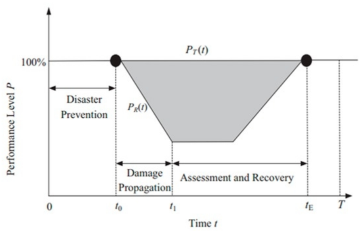


Figure 2.3: Urban Resilience

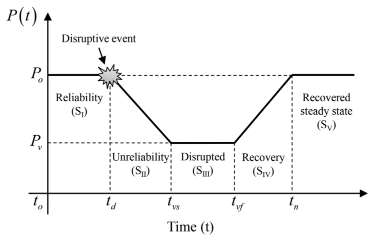


Figure 2.4: Engineering resilience quantification

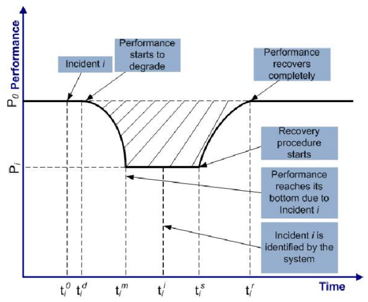


Figure 2.5: Resilience in industrial control systems

2.7.6 Dependability in the lighting system

Threats for the dependability in lighting system come from either the physical network layer or the application layer. [2] Failure report for OpenAIS uses failure modes as states to denote the manner where an item fails, which tells in which way an item is no longer able to fulfil a required function.[2] states that for each level of item or function analysed, the failure modes should be identified and analysed. Also, an effect of a failure mode at a lower level may become a cause of failure in the next higher level.

There is substantial research on the dependability of networks. [18] discusses common methods and proposes a method that is based on event-tree to consider node failures in network-dependability calculation.[23] proposed a methodology using FTA evaluate dependability of WSN, which supports different levels of network configuration and arbitrary failure condition

At a low level, failure can come from hardware malfunction, network function loss and software failure. [2] lists such examples of failure modes of a sensor as wearing out, wrong sensitivity, failing to trigger, false positive, etc. However, a calculation of dependability, which concerns node failure at the application level or system response that fits the IoT lighting system domain, has not been found.

Chapter 3

System Overview and Problem Statement

This chapter introduces the domain knowledge including the operational concept of the lighting control behaviour, the domain specific language (DSL), the network configuration in the lighting system and QoS. It also introduces the simulation platform and the problem statement for this project.

3.1 Domain Knowledge

This section introduces existing DSLs, which are used to specify the application, the local networks and QoS. It helps readers understand the functional requirement by introducing the application components constituting the lighting control behaviour (LCB) and its provided service.

3.1.1 Operational Concept and Lighting Control Application

The operational concept of lighting control behaviour is based on the process workflow shown in Figure 3.1. Basic components in the workflow comprise a sensor, controller and actuator. The definition and functionality of those components vary slightly for different views.

In the physical view, a sensor signals the controller after detecting a physical effect. Then the controller may send signals to the actuator according to its control logic. The actuator then acts physically by enabling the light. Sensors in the IoT lighting system are generally buttons, occupancy sensors and daylight sensors while actuators are usually luminaries. A simple example of control logic is that a sensor detects an occupancy in a room, then the lighting system responds by illuminating the luminaires in the room.

The logical view presents the functional decomposition of the workflow into various logical objects. There is a group object situated between the control object and sensor objects. The group object collects all information from sensor objects and sends to the control object. Every sensor object sends a message to the sensor group object when there is an event as input. Those multiple messages are aggregated at the sensor group object to generate one event message at the control object. After processing it, the control object sends an action message to the actuator group object that distributes the action message to multiple actuator objects. Sensors or actuators mapped to the same group are assumed to be homogeneous in their behaviours.

The integration view represents the Things in the lighting infrastructure. They are networked devices integrating functionality as sensors, actuators (luminaires) and controllers. In many cases, a Thing is capable of multiple functionalities shown in Figure 3.1.

Dotted lines in the figure denote the mapping of the relationship. Objects in the logical view are finally be mapped to devices in the integration view via components in the physical view. The main question for making a deployment decision is in which device in the integration the controller should be allocated. A physical sensor can become a controller when the control function is deployed on it.

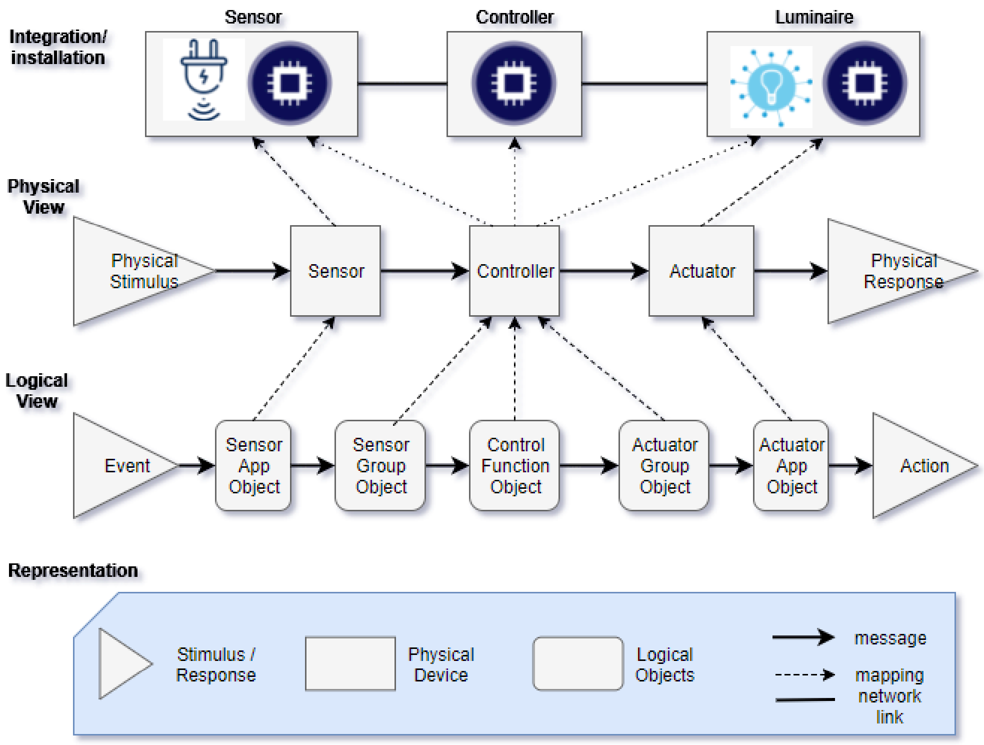


Figure 3.1: Operational Concept

3.1.2 Domain Specific Language (DSL)

Signify and TNO have created some DSLs to specify lighting systems and lighting control behaviours [12]. Building DSL, Template DSL and Control DSL are studied in this thesis which are studied below.

The Building DSL describes the topology of a building by indicating the perimeter of the building, its floors and rooms on every floor. It indicates the type of physical devices and the coordinates of their installation. We use “area” to refer to a physical space in a building, for instance, a room, a corridor or a corner.

In the Template DSL, sensors or actuators are combined in application groups, where those sensors and actuators are assumed to be homogeneous for their behaviours. Actuation specifications specify the actuator group to be controlled and its setting depending on a condition. Transitions indicate what events cause a state transition and what action is taken depending on current state. The structure of Template DSL consists of three parts shown in 3.2: required application groups and variables declaration, actuation specification, and the transitions that can be visualized as state machines.

The Control DSL deploys template DSLs onto Building DSL, which maps logical objects required by template DSL onto physical Things described by Building DSL.

Example of the DSLs in Figure3.2, there are two states in the state machine specified by Template DSL, which can be "Vacant Area" and "Occupied Area" respectively. Event 1 and Event 2 triggering transitions can be "Occupancy Detected" and "Vacancy Detected" that are sensed by occupancy sensors "Sensor Group 1". Action 1 is switching the light on, which is applied on luminaires "Actuator Group 1" while Action 2 is switching the light off. Control DSL relate "things" in building to logical objects required by DSL concerning types and coordinates of things.

In this context, the term "deployment" is used to describe the manner of mapping the control function, which executes the state machine, on to available "things" in the building. To support the deployment, an abstract network model describing the configuration of connectivity between "Things" is required as well.

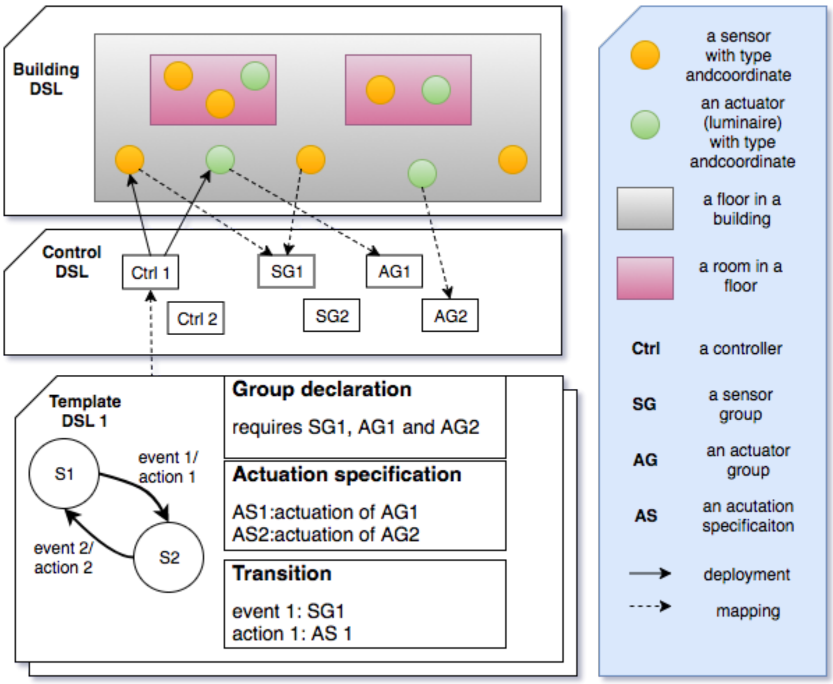


Figure 3.2: Template DSL, Building DSL, Control DSL and Deployment

3.1.3 Networks

Lighting systems support both wired and wireless network. Typical physical components connected by the network include luminaires, sensors, area (floor, building) controllers, IT-infrastructure components, cloud computing and management systems. Figure3.3 shows an example physical view of an OpenAIS [1] system with luminaires and sensors that are connected to a local field network using wired and wireless networks. In this project, we are mainly interested in the local wireless network from Figure3.3, which connects to the backbone network through a border router. Within a local wireless network, all devices use the same network technology and cannot be separated geographically.

Local networks in the lighting system are sensor-actuator networks (SANETs)[7] which is a new generation of the wireless-sensor network (WSN). Some nodes implement sensor functionality sensing the environment while some implement actuator functionality was acting on the environment. Nodes with only a network function (e.g. specialised gateway and router) are obscured from the network view as well as our consideration in this thesis.

Local network in this project is considered to be static with only high-level information on nodes, links and routings. Abstraction of hardware capabilities and link qualities are of interest while network function, protocols and communication technologies are not taken into account to keep the QoS evaluation simple. The deployment decision shall be independent of communication technologies. Thus we focus on the configuration in media layers (physical layer, data link layer and network layer) referring to the forementioned OSI model.

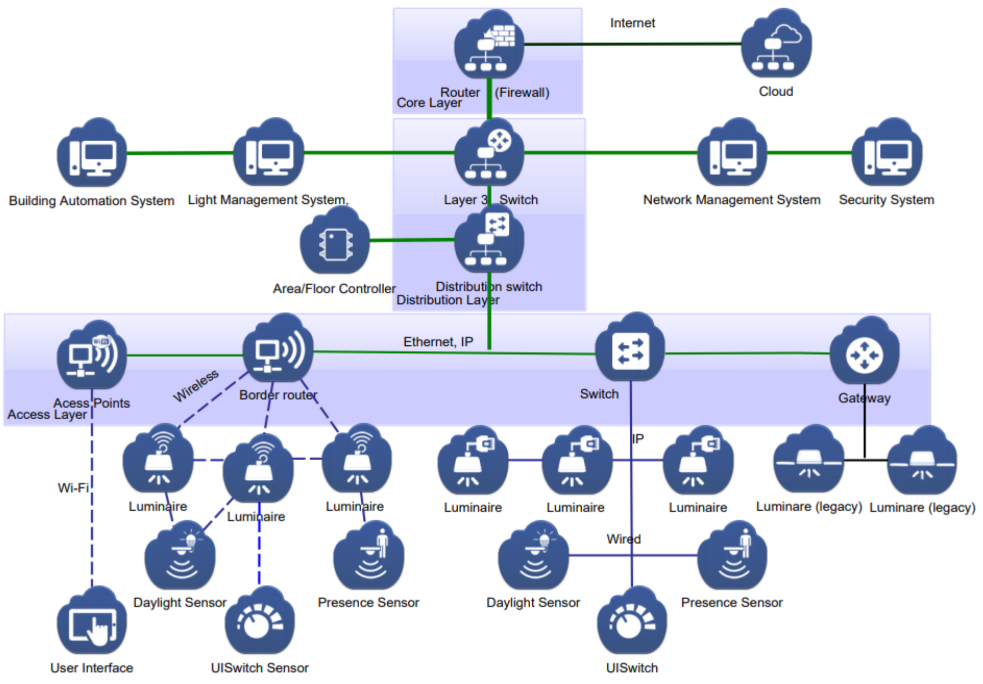


Figure 3.3: Physical View of the Network [1]

3.1.4 Quality of Service

Quality of services define the criteria for applications' performance. Among the many possible criteria, latency and resilience are the most interesting indicators to evaluate the application's performance in a lighting system, and therefore these will be considered in this project. The approach in this project is generic, and more criteria may be added to the process in the future. Considering the performance of lighting systems, users should see reliable and well synchronised visible action of lights in a reasonably short time. Furthermore, the waiting time before visible action should also be consistent and short [1].

Originally, the term "resilience" was studied in the fields of ecology and psychology. A Canadian ecologist[14] first described the concept of it in ecological systems in order to draw attention to trade-offs between constancy and change, or between predictability and unpredictability. Criteria for resilience in lighting systems include:

- the system should work with high availability and continue functioning when faults appear.
- the system should be capable of error-handling.

3.1.5 Centralized and Distributed Deployment

The "deployment" in this context refers to the deployment of the controller from physical view to things in the integration view in Figure3.1. In other words, it refers to the deployment the controller (as denoted by Control DSL) onto Things (as described by Building DSL), where the Template DSL defines the control logic.

In a centralised deployment, there is only one controller (the thing with control logic) that can be the critical component in the lighting control behaviour (LCB). Allocation of the controller would yield different QoS performance, which is the problem in a centralised deployment. Besides allocation, the distribution will influence the QoS of the LCB. There are three approaches to the distribution that we will consider: centralised deployment, decomposition and duplication of the controller.

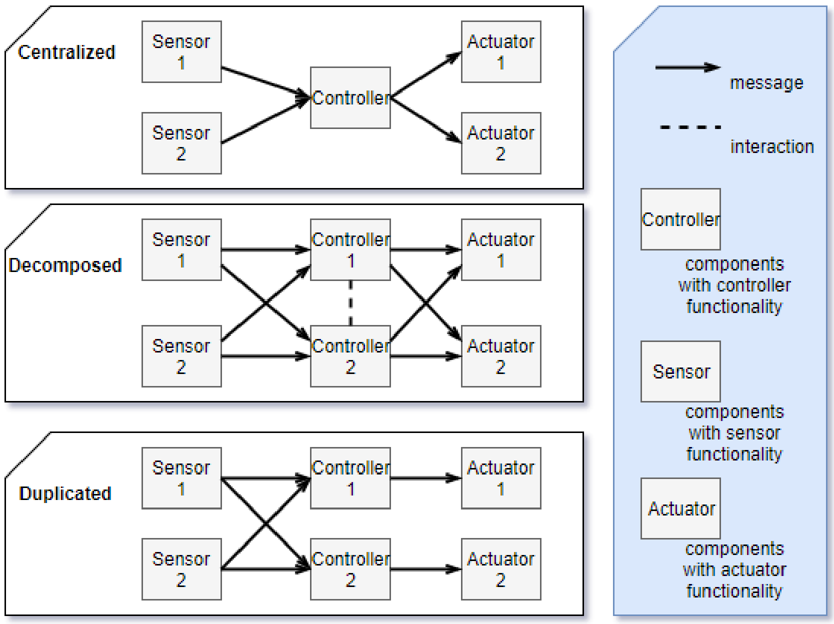


Figure 3.4: Centralized and Distributed Deployment

- In a centralized deployment, the controller receives messages from every sensor and sends messages to every actuator. This is the simplest allocation (deployment). The only variation that still exists is the selection on which of the nodes the controller function is allocated.
- In a decomposed deployment, the control logic is partitioned and distributed to different physical components. This can generate partitioned controllers that each of which consumes fewer resources. However, various parts (distributed control logic) need to interact with each other to fulfil the control logic. The decomposition of DSL is out of the scope for this project. Also, it requires estimating the communication volume caused by interaction between partitioned controllers and a partitioning evaluation function. Mechanisms for such decomposition requires the design of cost function[22] involving negotiation with the DSL-design team, which is not feasible concerning time constraints of the project.
- In a duplicated deployment, each controller has the complete control logic as specified in the template DSL. However, each has been assigned a smaller control scope (less controlled actuators) compared to a centralised controller. Duplication is a feasible approach. Every duplication inherits the control logic from the centralised controller (same template DSLs). The duplication and distribution plan will influence the QoS.

3.2 Simulation Platform

TNO-ESI, a partner of Signify, has developed a simulation tool that can simulate the lighting control behaviour on Things in any given building and validate some critical aspects of the lighting system. It enables users to describe a lighting systems' physical view and its control behaviour by DSLs (see Figure 3.1 and Figure 3.2).

It simulates the workflow and components from the operational concept in Figure 3.1. Users can configure components at different views of the workflow and denote their mapping relationship. The configuration is done by modifying relevant DSL's mentioned earlier. The platform translates the DSL source code and converts the configuration to simulators. It provides a graphical tool where a user can observe the simulation of lighting behaviours for various scenarios.

With the help of the simulation tool, designers can gain an insight into the relationship between different DSLs and pay more attention to aspects which cannot be verified or validated in the designated timeframe. It supports DSL syntax check, object mapping, simulators generation regarding DSLs, group communication simulation, functional simulation of the lighting control behaviour.

By using this tool, our implementation can parse DSL into identified attributes required by evaluation models. Furthermore, we can simulate the lighting control behaviour (LCB) in different scenarios to get stochastic information representing the communication load, which is used for latency evaluation. We will further introduce this simulation tool in chapter 7.

3.3 Problem Statement

In a lighting system, Things (e.g. luminaires, sensors) are often connected by restricted networks. They implement task-engines that enable them to realise the lighting control behaviour (LCB) specified by application specification as known as DSLs. A question raising researchers' interest is how to decide on the optimal deployment of the controller based on the information of the DSL source codes and network configuration.

As a first step, it must be identified that what is an optimal deployment decision. In the System Overview, we have introduced the concept of "deployment" in this context. In the Related Work, there is existing research about deploying applications onto IoT systems, which formalise the property of application, infrastructure and the activity of deployment.

There is also research on finding an optimal decision, which is stated as a multi-objective optimisation problem (MOOP). General approaches are to find a decision belonging to a Pareto frontier or a decision with optimal (maximum or minimum) score.

To select the optimal decision, we have introduced some criteria, metrics and evaluation methods that can be used to calculate and compare different decisions. Then, we have introduced some network models and the DSLs in the domain. What information can be extracted from them as factors in QoS calculation is one of our interests. Furthermore, what information should be provided beside DSL and network configuration is also an interesting aspect.

Though there is a lot of existing work which propose methodologies of deployment, they cannot be directly applied in the lighting system due to the operational concept of LCB in Figure 3.1. In addition, evaluation methods for QoS cannot be directly applied as they should be aware of the concept of LCB as well.

For resilience, some methodologies for identifying, analysing and evaluating it are introduced in the Related Work, which supports most industrial applications and network reliability. However, a quantitative evaluation method that considers network configuration, application specification, and functional requirement of LCB (three views in the Figure 3.1) has not been done well.

The problems to be solved in this master thesis can be summarized as components in the flow diagram in Figure 3.5. Colored blocks are problems solved in this work and explained in next sections. As we have known the optimisation components in the decision-making matrix, we focus on what are the decision-making process, the evaluation methods, the attributes from the input and other influences or configurations. We intend to develop a deployment tooling that allows users to manipulate some parameters and assist users to make deployment decisions.

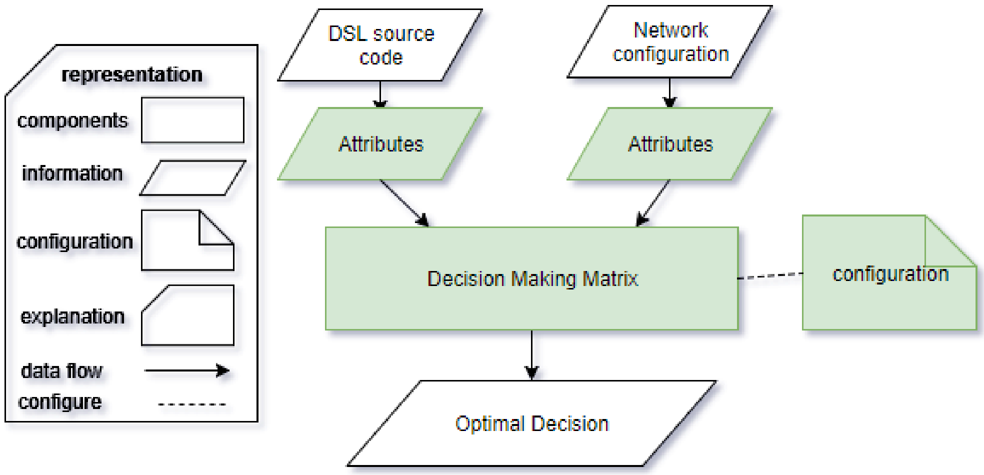


Figure 3.5: Research questions in the flow diagram

Chapter 4

Formalization

In this chapter, we formally define the infrastructure, applications, deployment in lighting systems by considering the aspects considered in work [9].

4.1 Infrastructure in Lighting Systems

The infrastructure in this context refers to Things and networks connecting them. We describe the physical network by using a graph model (i.e. *G=(V,E)*), where V is a set of vertices denoting hardware nodes (end-node devices) in a network and E is a set of edges representing communication links between those nodes. We describe Things T as a set of physical devices with hardware capability and functionality required by lighting control behaviour. Then a Infrastructure is a 4-tuple $\langle$ *V, E, T, L* $\rangle$

- A network node $n = \langle\ i,\ q\ \rangle$ where i is its identifier, q is its network quality.
- An network link e is $e = \langle$ *n1, n2, q* $\rangle$ where (n1, n2)∈(V×V) and q is the quality of the communication link.
- A thing $t = \langle$ *i, ty, H* $\rangle$, where i is its identifier, *ty* is its type and H is its hardware capacity. The type is either an actuator or a sensor.
- l is the end-to-end communication between two t. l∈ L ⊆ {⟨t1, t2, q ⟩ | (t1, t2)∈(T×T) and q is the quality of the communication between the two things }

Assuming each t from T is mapped to a n in V, then the l = ⟨t1, t2, q ⟩ is a combination of e connects *n1* and *n2* directly or indirectly. And the q of the l is a combination of those q on those e.

Some observations result in the choices of attributes made above. First of all, identifiers are needed to distinguish nodes as different physical nodes may be homogeneous at application level (e.g. same application object in the application group). Types help distinguish nodes in aspect of its functionality (e.g. sensors, luminaires or others). Hardware capability of a node decides if the node is one of the alternatives to be deployed of the control function regarding its CPU and memory. The quality profile can be the reliability or communication cost or other parameters related to evaluation.

Second, we hide the type of communication technology (Zigbee, WiFi, and so on) described in the host layer in OSI model, but abstract attributes from the media layers in the OSI model[28]. This abstraction leaves a quality profile assigned to each communication link.

Third, the model does not deliberately bind to any particular standard for hardware and software capabilities specification. The definition of hardware and communication links can be extended by additional attributes such as software capabilities (e.g. platforms) of the nodes and bandwidth of links. Finally, the extension on augments in the definition is relevant to evaluation schemes and stakeholders, which is not bounded to the mathematical model.

4.2 Lighting Control Behaviours

The lighting control behaviour (LCB) is an application consisting of several application components. The components in LCB at the application level are logical objects, groups and the control function, which have been discussed in 3. We define LCB as a 5-tuple = $\langle\ \Gamma,\ B,\ \Lambda,\ \Theta\ \rangle$ where:

- Γ is a set of logical objects, each denoted by γ=$\langle\ i,\ ty\ \rangle$ where i is the identifier and ty is the type of things needed by the γ.
- Θ is a set of logical groups, each denoted by $\theta = \langle\ i,\ \Gamma\ \rangle$. Each group is a aggregation of logical sensors or logical actuators.
- B is a set of controllers, each $b = \langle\ i,\ H\ \rangle$ where i is the identifier and H is the hardware capacity required for deploying controller. The requirement includes CPU and memory.
- Λ is a set of interactions between logical groups. An interaction $\lambda = \langle\ (\theta 1,\theta 2),\ in,\ out\ \rangle$ where $(\theta 1,\theta 2) \in (\ \Theta \times \Theta)$. *in* is an event triggering the interaction and *out* is an output from the interaction.

Template DSL specifies the control logic which is the interaction Λ between sensor groups and actuator groups. Control DSL specifies the application the controller B, components Γ and the aggregation of Γ to Θ.

4.3 Deployments

We can remove this formalises the notion of deployment of control logic over a physical network. Assuming all T are mapped to nodes in N, the deployment is to bind B to T. There are constraints that an eligible deployment must meet.

- checks that hardware of the t satisfy the hardware requirement H of the b.
- ensuring that all logical objects Γ have been bound to T, and there is l connects each of the γ to b.

4.4 QoS

In this section, we describe two QoS parameters that are latency and resilience as follows:

4.4.1 Latency

To compute the latency in this work, we used the latency evaluation scheme as discussed in section 2 of chapter2. The computed latency is L = $\langle\ V,\ E,\ \Gamma,\ \Lambda,\ T,\ L\ \rangle$. It is the accumulation of communication cost by nodes and links in the network. It requires that:

- For every sensor objects and actuator objects belonging to Γ, they should be bound to T that have been bound to V.
- For each $\lambda \in \Lambda$, the interaction should be bound to a L where each l is a combination of one or multiple e. The binding depends on the binding of logical objects to things, and the combination depends on the routing of the network.

For attributes required by the calculation, node quality and link quality are attributes from the physical network configuration. Things representing source and destination of communication in the network are determined through the binding of logical objects and nodes in the network. Furthermore, communication volume is obtained through the statistics of events in of λ happened in selected scenarios.

4.4.2 Resilience

In our work, resilience evaluation depends on the combination of application components and the quality of their bound Things as well as links. Similar to latency, resilience is F = $\langle$ V, E, Γ, T, Θ, L $\rangle$. The aggregation Γ of components can be used to describe the dependency of components in the operational concept and failure conditions of LCB. The quality in Θ as well as L, which is determined by V, E and deployment, can be used to describe the failures of Things and connects which are the basics of the LCB.

Chapter 5

Deployment Decision Matrix

This chapter discuss the solutions to questions addressed in diagram in Figure 3.5 in Problem Statement in chapter 3. Based on the diagram and the formalization in chapter 4, this chapter explains the optimal decision as the output of the decision maker and proposes a decision-making matrix including a decision making process and evaluation components.

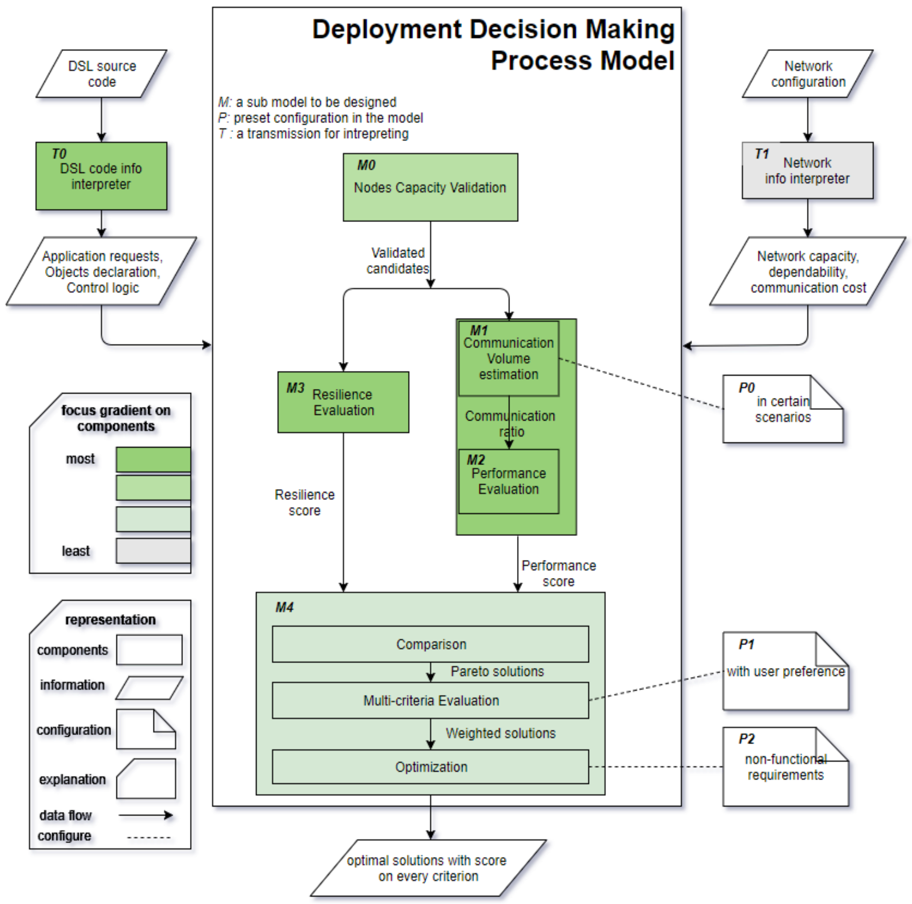

Deployment Decision Making Process Model
M: a sub model to be designed
P: preset configuration in the model
T : a transmission for intrepreting
DSL source code
T0 DSL code info interpreter
Application requests, Objects declaration, Control logic
Network configuration
T1 Network info interpreter
Network capacity, dependability, communication cost
M0 Nodes Capacity Validation
Validated candidates
M3 Resilience Evaluation
Resilience score
M1 Communication Volume estimation
Communication ratio
M2 Performance Evaluation
Performance score
P0 in certain scenarios
M4
Comparison
Pareto solutions
Multi-criteria Evaluation
Weighted solutions
Optimization
P1 with user preference
P2 non-functional requirements
optimal solutions with score on every criterion
focus gradient on components
most
least
representation
components
information
configuration
explanation
data flow
configure

Figure 5.1: Decision Making Process

5.1 Optimal Decision

One aspect (sub-criterion) of the service does not necessarily lead to a decision on deployment. The decision depends on a comprehensive analysis considering all aspects of the service and user preference. The user decides whether there is a priority level between different sub-criteria. If there is, the user shall assign weights to every sub-criterion.

A final score based on the weighted sum of scores for all criteria will guide the decision-making process. In this project, there is no weighting nor constraints on criteria given from the user. The weights and constraints used in this project are not based on research but assumed. The aim is to validate the correctness of the proposed process and evaluation components used in the process.

Without user preference, Pareto efficiency is a sufficient method to find optimal decision. In this project, we focus on decisions belonging to the Pareto frontier, which are equally optimal but would yield different performance to the lighting control behaviour.

5.2 Decision-making Process

Based on related work in chapter 2, we propose a model for the deployment decision-making process shown in Figure 5.1 as an extension of the realization model in Figure 1.2. It guides the deployment design cycle through the process including identifying inputs, evaluating alternatives, comparison of solutions and selecting an optimal solution.

We divide the decision-making mdel into several components (sub-models) that are two interpreters, three criteria evaluation components and one optimal decision computing component:

- *T0* and *T1* are interpreters that extract attributes from the information provided by the network configuration and DSL source code. They transfer this information into the required input for the model.

- *M0* validates if the capacity of any node meets the application requirement on this hardware. This is used to restrict the number of alternatives from candidates in the network. Furthermore, it helps to evaluate the extensibility of deployment concerning the application utilisation ratio within the network.

- *M1* and *M2* together evaluate the timing performance. *M1* estimates the communication load among nodes in the network by executing control logic for selected scenarios (events happening with time stamps). With this communication load and communication cost on network paths, *M2* calculates the end-to-end latency between each pair of sensor and actuators as well as the synchronicity among the actuators.

- *M3* evaluates the resilience (fault tolerance) of the application deployed. Based on the hierarchy of objects declared in the DSL and the dependability of the network, it can evaluate the dependability of the application this deployment.

- *M4* defines the Pareto space for the deployment, scores every alternative deployment regarding its score on criteria and user preference, and then generates an optimal solution satisfying all constraints and non-functional requirements.

Instead of implementing *T1*, we propose a naive network model offering attributes required by the evaluation. The model is in form of a look-up table that is able to be filled qualities with communication cost (latency), link dependability and other attributes. We will further explain this point.

5.3 Criteria

A decision making process for the deployment starts with identifying critical aspects of the service.

In this section, we define each criterion in the context of a lighting system and break down of these into sub-components. We use sub-criteria for those sub-components, where each sub-criterion is rated by its own metric. The weighted sum score of each sub-criterion reflects the the score of actual criteria. The weighted sum will be introduced later.

In this project, the sub-criteria for latency are response time and synchronicity while the sub-criteria for resilience are reliability and degradability. Their metrics are show in Table 5.1.

Besides latency and resilience shown in the table, the footprint of the control function on devices is also a criteria, where the sub-criteria are CPU capability and memory capability. To simplify the the question, we define the metrics is to compare the capability of devices with the requests from control function. The result is either satisfaction or dissatisfaction, which are used to filter deployment decisions instead of being considered in MOOP.

Criteria	Sub-criteria	Metrics
Performance (Latency)	Response time	End-to-end latency of the total chain. (e.g. “press button” to “light on”)
	Synchronicity	Difference on message communication in groups. (e.g. “the first light on” to “the last light on”)
Resilience (fault tolerant)	Reliability	The ability that the application runs without error.
	Degradability	The ability to handle errors until human intervention.

Table 5.1: Sub-criteria and Metrics for latency and resilience

Chapter 6

QoS Evaluation Methods

This chapter introduces the evaluation methods of latency and resilience that are components *M2 and M3* aforementioned in chapter 5. The evaluation methods are originally proposed by existing work cited in chapter 2, but we adapt them to fit the lighting system based on the definition in chapter 4 and metrics selected in chapter 5.

Also, we introduce the assumption on device capacity, network model and communication load, which are components *M0, M1, and T1* for the use of evaluation methods.

6.1 Assumption

In this section, we introduce the assumption we make on hardware capacity, communication load and the network model converted from network configuration.

6.1.1 Hardware Capacity

CPU and Memory capacity of a device should meet the hardware requirement of control function before it to be deployed. If both capacities meet the requirements, a node is one of the alternatives for deployment. In the other case, it will not be processed in further evaluation to reduce computation load.

6.1.2 Communication Load

The number of packets transmitted or received for each message is not equal among sensors and actuators. It would cause different latency for messages with different loads travelling on the same path. Therefore, communication load is one of the attributes in latency evaluation. Predefined configuration and distribution of events in a scenario can yield different communication load among devices. We assume the communication load in the latency evaluation is given and constant.

6.1.3 Naive Network Model

Since we focus on end-to-end connections, we need a network model providing the quality of communication links between any two end-nodes in the network. In this project, we use a look-up table (LUT) based network model to provide attributes required by the latency evaluation and resilience evaluation. The attributes stored in the lookup table can represent communication cost, link dependability and other qualities. The weights represent the quality of connectivity influenced by network configuration listed in chapter 2. Information stored in the naive model can be extracted from a more complicated model that is not considered in this work.

Table6.1 gives an example showing a naive network model with given routing, weights on links, the identity of nodes and the type of each node. In the example, we distinguish the upload links and download links, which includes the case where these are identical. A lookup table is built regarding the network. Nodes listed in the row are the source of messages while those listed in the column are the destination of messages. We can observe that the communication cost from *node 1* to *node 3* is different to the other way round. Node identity of sensors and actuators should also be provided along the LUT.

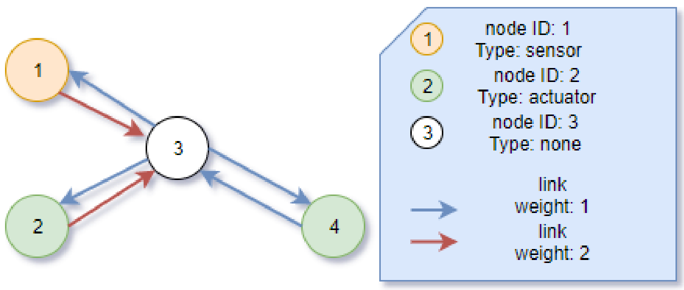


Figure 6.1: Network naive model: 4 nodes

Latency	node 1	node 2	node 3	node 4
node 1	0	3	1	2
node 2	3	0	1	2
node 3	2	2	0	1
node 4	3	3	1	0

Table 6.1: Latency between two end nodes LUT

6.2 Latency

Referring to the definition in [1], latency is the end-to-end delay from the a sensor in a sensor group to the an actuator in an actuator group. Some examples are given on page 102 in OpenAIS D2.7[1], one of which we have discussed in Related Work

Factors in this functions are the number of hops on a network route, bandwidth, physical link quality and node computation power that are attributes assigned to nodes and links in the network model. Source and destination of the route and communication volume can be derived from template DSL and the selected scenario.

The evaluation function is the accumulation of all attributes on the route from source to destination. The result of the latency evaluation function must satisfy the non-functional requirements as following:

Csi	Csci	Cai	Caci	Msi	Mai
Communication cost by sensor i.	Cost by the connect to sensor i.	Cost by actuator i.	Cost by the connect to actuator i.	Message load of sensor i.	Message load of actuator i.

Table 6.2: Communication Cost Symbols

- The latency from sensor 1 to the controller: $L_{s_1} = (C_{s_1} + C_{sc_1}) * M_{s_1}$
- The latency from the controller to the actuator 1: $L_{a_1} = (C_{a_1} + C_{ac_1}) * M_{a_1}$

Response time

We consider the response time on the critical path. It is the sum of the longest path from sensors to the controller and the longest from the controller to actuators. The response time depends on the allocation of Things on the network, the communication cost on path and

The critical response time: $L_{Rsp} = MAX(L_{s_1}, ...L_{s_n}) + MAX(L_{a_1}, ...L_{a_n})$

Synchronicity

We consider the difference between the longest path and shortest path among actuators belonging to a group. The lower the difference obtained, the higher the synchronicity is.

The synchronicity: $L_{Syn} = MAX(L_{a_1}, ...L_{a_n}) - MIN(L_{a_1}, ...L_{a_n})$

6.3 Resilience

The network in a lighting system can be considered to be built up from two layers, where the physical network is considered to be the under-layer network while the logical objects constitute the upper-layer network. We analyse the resilience of an application given the logical (upper) layer, where attributes are based on the configuration of the physical network. Some work has been done on analysis of dependability or fault tolerance of physical networks. We are focusing on the logical network that is defined by DSL, allocation and network configuration. Regarding the comparison and discussion in Related Work, we make the following decisions on assumptions of the problem space and measures of the resilience.:

- Fault: arbitrary permanent failures on components.
- System response: degradation of performance
- Analysis Methods: failure tree analysis for evaluation
- Evaluation Indicators: mean time to start degradation on service and the mean time to the moment when no function in the service works
- Improving technique: redundancy on control function with distributed control scope.

[23] has introduced the definition and calculation of dependability regarding availability and reliability. It also addresses the relationship among dependability, failure probability and means time to failure (MTTF). Based on its research, we have the following definition:

- D(t) = 1 - F(t), D(t) represents the probability of the application delivering correct service without failure in a time interval (0, t], which also reflect the dependability of the lighting control behaviour in this context. F(t) is the cumulative distribution function of failures. They are probabilities with values between 0 and 1.
- The smaller the F(t), the longer the application operating without failure and the better the performance on resilience.
- For simplicity, we use D and F instead of D(t) and F(t) when there is a give *t*. We will define the *t* before starting calculations.

We select two moments mentioned in Chapter 2 to indicate the performance on resilience: the moment when the system changes from a reliable state to unreliable state and the moment when the system changes from the unreliable state. By considering the concept MTTF, we name the first moment as “time to degrade” and the second moment as “time to disrupt” In this section, we explain how to generate an FTA based on the DSL’ specification and the naive network model to evaluate the two moments (two sub-criteria)

6.3.1 Failure Condition of Lighting Control Behaviour

The failure condition for Lighting Control Behavior (LCB) defines which combination of components may lead to further failure. In this methodology, we support any combination that can be expressed using boolean operators (i.e. *AND, OR*). The failure condition associated with logical objects obj_i is defined as group_i. The cause of logical objects failure will be described in the next section. A combination of logical objects lead to a group object failure defined as group_n, where n is the identification of combination and is represented by the boolean *AND* of the failure

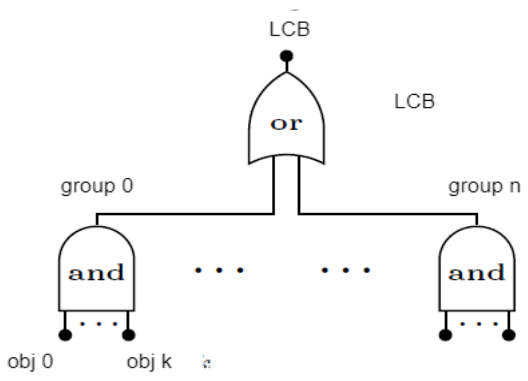


Figure 6.2: Application Failure Condition

6.3.2 Failure Condition of Group Object

Regarding the Operation Concept introduced before, the control function interacts through group objects instead of each application objects individually instead of each application object. A group object is operational as long as there is an application object belonging to the group operates. According to the operational concept, the application objects are identical to the group objects. That is, from the perspective of the controller, sensors (or actuators) are homogeneous on operation when they are mapped to the same group. Objects mapped to the same group are homogeneous and thus redundant to each other.

In the proposed model, components are based on the failure event. The boolean *AND* represents the failure event of a group object whereas a basic failure event represents the failure of an application object.

6.3.3 Failure Condition of Application Object

After obtaining expression of the control behaviour failure condition, it is necessary to define the conditions that may lead to the failure of control behaviour. We consider two possibilities for an application object failure:

- Node Failure: Hardware malfunction or software crash. The device cannot provide service as a sensor or actuator.
- Connectivity failure: There is no path between the object allocated node and the controller.

In the latter case, though the application itself does not fail, it is considered non-operational from a network perspective because it is no longer possible to communicate with the node. As aforementioned, the failure condition of an application object is split into two parts involving hardware and connectivity problems, where the latter is more dependent on the deployment of the control function. If a node along the path fails, the network may have required mechanisms to reconfigure itself to use other paths. Self-healing routing protocols do this type of reconfiguration, and similar mechanisms have been introduced in chapter 2.

6.3.4 Construct the Fault Tree

We will use an example to describe constructing based on the DSLs and the network model from the top to down deductively. Consider the DSLs and the network model in Figure6.3.

1. The Template DSL specifies the control logic. One transition requires sensor group 1 and actuator group 1 (luminaire group 1) whereas the other requires sensor group 2 and actuator group 2. The application failure is a combination, expressed using *OR*, of logical groups and the controller.

2. Control DSL specifies the mapping relationship between physical thing given by Building DSL and logical groups. The network model provides the dependability represented by failure probability of every node and the end-to-end communication between two nodes. The failure of a group objects is represented by the boolean *AND* of the failure condition of logical objects.

3. As mentioned before, the failure of a logical application object can be caused by two possibilities that the hardware failure or the connectivity failure. We use *OR* to represent the failure condition of a logical object. The failure happened on the physical device or connectivity is the basic event in the FTA, where the naive network model provides the means of a cumulative distribution function. The failure of a controller object is only due to the failure of the physical controller as there is no connectivity needed.

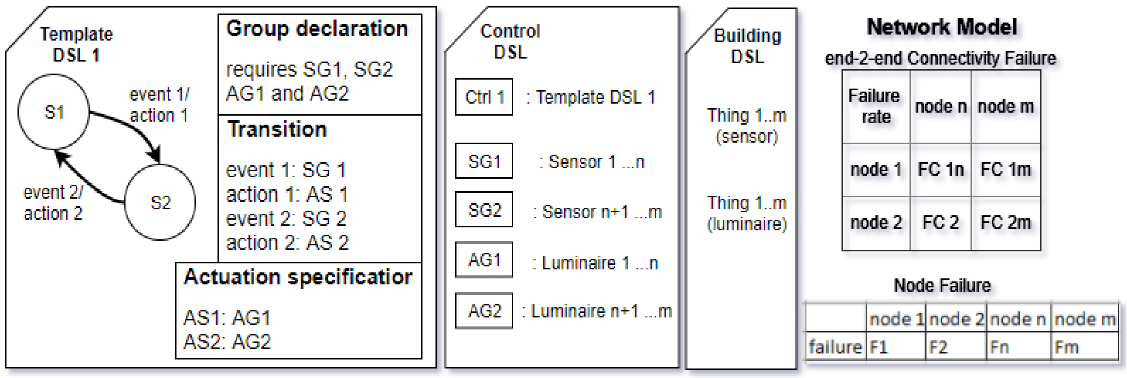


Figure 6.3: DSL and Network Model used by constructing FTA

Figure 6.4 shows the combination of events in LCB. The basic events at the bottom of the tree are failures of Things, and their connects to the controller. The events depending on those are failures of the logical objects bound to the Things. Events at the upper layer are the failures of groups where those logical objects aggregate. The top event is the failure of the application representing the disruption of LCB. The boolean equation describing the event in FTA can be converted to CDF for calculating the failure probability. The equations are shown as following

Fs_1	Fsc_1	Fctrl	Fa_1	Fac_1
F of sensor 1	F of sensor connectivity 1	F of the controller	F of actuator 1	F of actuator connectivity 1

Table 6.3: Failure Symbols

- F of a logical sensor object: $F_{sobj_1} = 1 - (1 - F_{S_1}) * (1 - Fsc_1)$

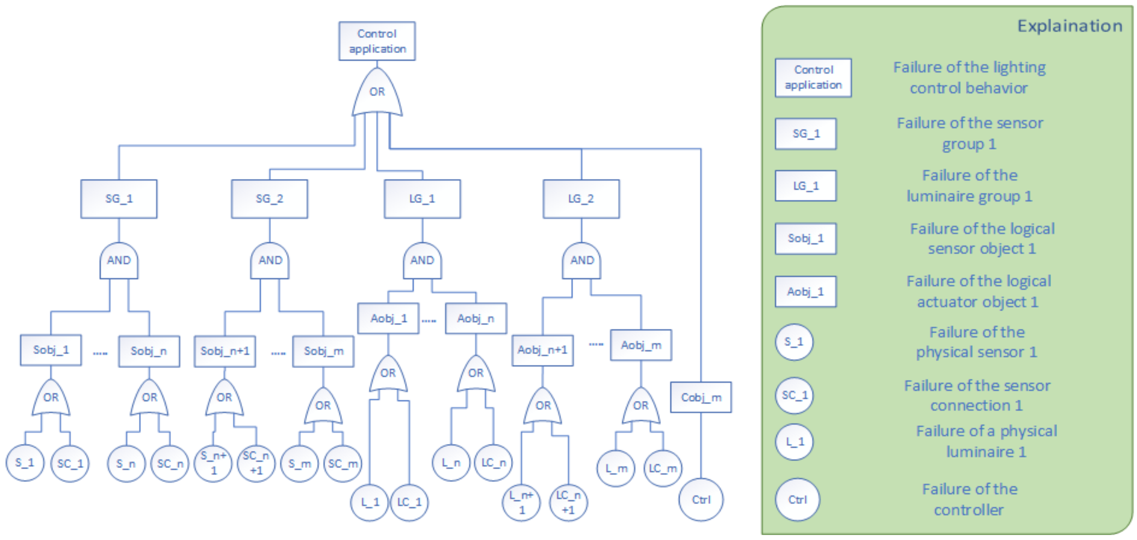

Control application
OR
SG_1
SG_2
LG_1
LG_2
AND
Sobj_1
Sobj_n
Sobj_n+1
Sobj_m
Aobj_1
Aobj_n
Aobj_n+1
Aobj_m
Cobj_m
S_1
SC_1
S_n
SC_n
S_n+1
SC_n+1
S_m
SC_m
L_1
LC_1
L_n
LC_n
L_n+1
LC_n+1
L_m
LC_m
Ctrl
Explaination
Failure of the lighting control behavior
Failure of the sensor group 1
Failure of the luminaire group 1
Failure of the logical sensor object 1
Failure of the logical actuator object 1
Failure of the physical sensor 1
Failure of the sensor connection 1
Failure of a physical luminaire 1
Failure of the controller

Figure 6.4: Failure Tree Analysis, single controller

- F of a sensor group object: $F_{SG_1} = \prod_{i=1}^{n} F_{sobj_i}$
- F of the lighting control behavior: $F_{service} = 1 - \prod_{i=1}^{2}(1 - F_{SG_i}) * \prod_{i=1}^{2}(1 - F_{LG_i}) * F_{Ctrl}$

6.3.5 Duplication, Redundancy

We have introduced redundancy as a technique to achieve fault tolerance (lowering failure probability of the control application). In lighting systems, we adapt redundancy technique by duplicating controllers and distributing actuators to control the scope of each controller whereas they share common sensors. Those controllers are identical and independent units working in parallel and that a single unit is capable of supplying the required service. If a controller fails, actuators belong to the control scope, which connected to the controller, will not act while the rest of the actuators continue.

A simple example of a fault tree with duplicated controllers is illustrated in Figure 6.5. The tree with duplicated controllers is constructed based on the tree with a single controller. The difference is that two more conditions representing the failure of control scope are added to the tree. The failure of the application is a combination of the failures of sensor groups and the control scopes. The failure of a controlled scope is a combination of its controller and actuators controlled by the controller. The CDF converted from the FTA are as follows:

- F of the control scope 1: $F_{CtrlSP_1} = 1 - \prod_{i=1}^{2}(1 - F_{LG_i}) * F_{Ctrl1}$
- F of the control scope 2: $F_{CtrlSP_1} = 1 - \prod_{i=3}^{4}(1 - F_{LG_i}) * F_{Ctrl2}$
- F of the total control scope: $F_{CtrlSP} = \prod_{i=1}^{2} F_{CtrlSpi}$
- F of the lighting control behavior:$F_{service} = 1 - \prod_{i=1}^{2}(1 - F_{SG_i}) * (1 - F_{CtrlSP})$

6.3.6 Resilience Indicators

We have identified the problem space in the lighting system (self-healing system), where faults are permanent, and the system degrades its performance. Since the system recovery mechanism is not in our consideration, we focus on the “reliable state”, “unreliable state” and “disrupted state” of the lighting control behaviour. The mean time from “reliable state” to “unreliable state” as well

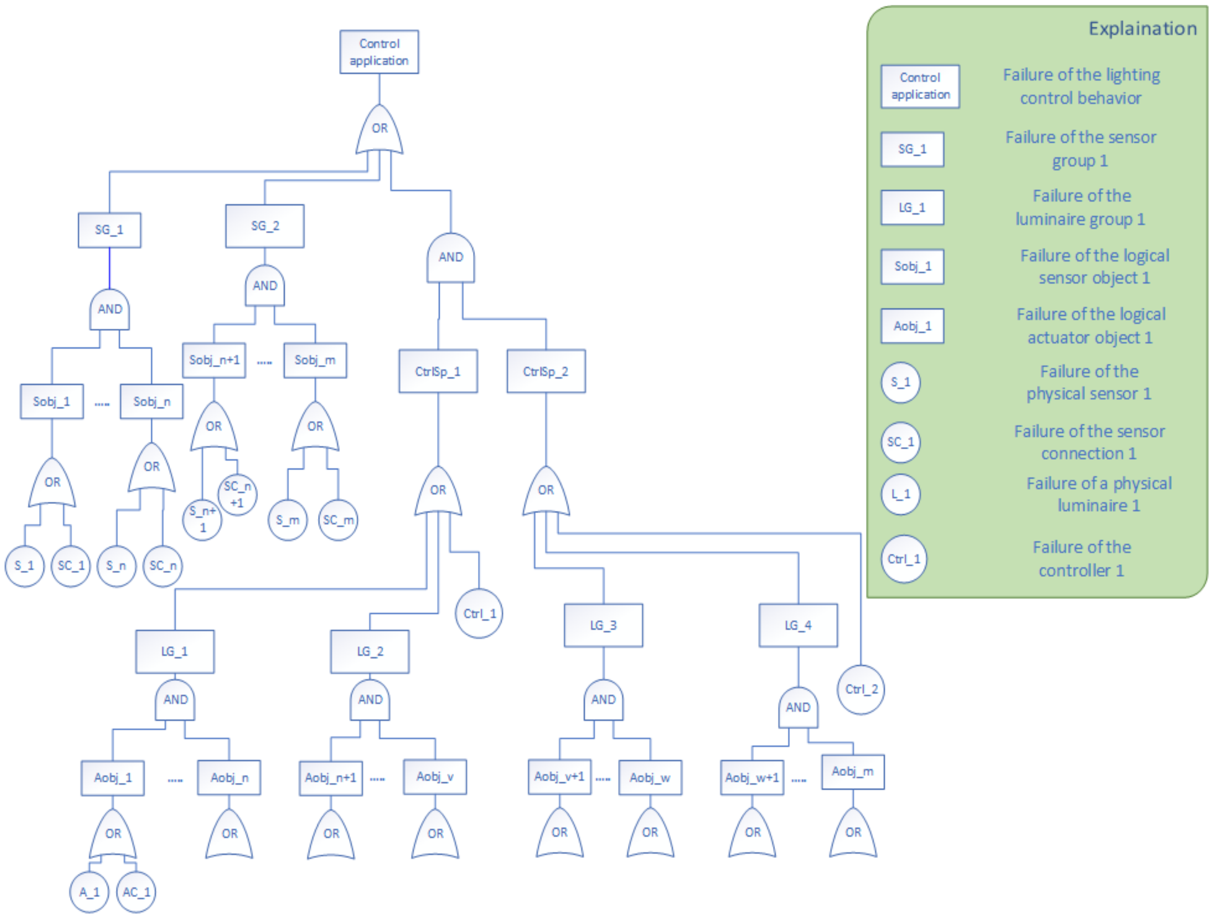

Control application
OR
SG_1
SG_2
AND
Sobj_1
Sobj_n
Sobj_n+1
Sobj_m
S_1
SC_1
S_n
SC_n
S_n+1
SC_n+1
S_m
SC_m
CtrlSp_1
CtrlSp_2
Ctrl_1
Ctrl_2
LG_1
LG_2
LG_3
LG_4
Aobj_1
Aobj_n
Aobj_n+1
Aobj_v
Aobj_v+1
Aobj_w
Aobj_w+1
Aobj_m
A_1
AC_1
Explaination
Failure of the lighting control behavior
Failure of the sensor group 1
Failure of the luminaire group 1
Failure of the logical sensor object 1
Failure of the logical actuator object 1
Failure of the physical sensor 1
Failure of the sensor connection 1
L_1
Failure of a physical luminaire 1
Failure of the controller 1

Figure 6.5: Failure Tree Analysis, duplicated controller

as the mean time from “unreliable state” to “disrupted state” are sub-criteria for the resilience. Failure probability can be converted to represent the two meantime in this context.

Time to degrade

We use Td to represent the moment when a failure occurs after a continuous time without fault of any application component in the workflow. After Td, the performance of service starts to degrade.

Referring to FTA, the performance starts to degrade when there is a basic failure event happens. Failure probability of this condition: $F_{degrade} = 1 - \prod_{i=1}^{n}(1 - F_{S_n}) * \prod_{i=1}^{m}(1 - F_A) * \prod_{i=1}^{n}(1 - F_{SC_n}) * \prod_{i=1}^{m}(1 - F_{AC_m}) * F_{Ctrl}$

Time to disrupt

We use Tb to represent the moment when the service cannot be degraded anymore regarding the severity of service.

Referring to FTA, the performance starts to disrupt when the TOP event happens. Failure probability of this condition: $F_{disrupt}$ is the probability of the top event described by the FTA, which depends on the failure of logical groups.

Chapter 7

Implementation

This chapter introduces the implementation of deployment tooling. It includes deployment DSL, latency evaluation component, resilience evaluation component on the cosim platform and MOOP on matlab.

To assist the decision-making process, the tool has to include the following capabilities:

- gets the configuration of the network and specifications of the DSL as input.
- validates the capabilities of candidates for the deployment.
- generates and evaluate the performance and resilience result from all validated deployment candidates.
- compares evaluation of deployments and deliver the proposed one to users

7.1 Solution

Our solution is an Eclipse-based software application. Users of the tool have to only provide network configuration and some domain knowledge. The network configuration is provided in a form of lookup table concerning the quality of nodes and links among them. The domain knowledge includes control behaviour, object mapping and building topology.

This chapter will introduce the implementation of each component from the model proposed. The implementation concerns interfaces, algorithms and the platform

7.1.1 Algorithms

This section introduces the algorithms used in the implementation. They include the validation of node capacity among deployment candidates, critical latency calculation, maximum differential latency (synchronicity) and failure probability representing the meantime to disruption of the lighting control behaviour.

Node Capacity Validation

The input of the Algorithm1 is the set of nodes **N** and the profile of the controller. Only if both the CPU and memory provided by the nodes are greater than those required by the controller, the node becomes one of the deployment candidates.

Algorithm 1: Capacity Validation

```
  Input: N, controller
  Output: VN Validated Nodes
1 for each node in N do ;
2   if node.CPU > Controller.CPU then ;
3     if node.mem > Controller.mem then ;
4       VN.insert(node) ;
5 return VN ;
```

Latency

L_{SC} is the set of latency from sensors to the controller while L_{AC} is the set of latency from the controller to every actuators. The critical latency is the sum of the maximum values from both sets. The maximum differential latency (synchronicity) is the difference between the maximum and the minimum of L_{AC} where the actuators belong to a same actuator group. AG is the set of actuator groups. L_{SYN} is the set of L_{syn}.

Algorithm 2: Latency Critical Latency

```
  Input: $L_{SC}$, $L_{AC}$
  Output: Response time
1 $L_{rsp} = \max(L_{SC}) + \max(L_{AC})$ ;
2 return $L_{rsp}$ ;
```

Algorithm 3: Latency Differential Latency

```
  Input: $L_{AC}$, AG
  Output: Max Differential Latency
1 for each ag in AG do ;
2   $L_{SYN} = \max(L_{AC})$ - $\min(L_{AC})$ ;
3   $L_{SYN}$.insert($L_{syn}$);
4 return $\max(L_{SYN})$ ;
```

Resilience

There are two types of failure of the lighting control behaviour where the failure probability of degradation (mean time to degradation) is the product of everything and connects that are shown on a network view while failure probability of disruption (mean time to disruption) is shown in Algorithm4.

The input of Algorithm4 is the information of sensor groups (SG), actuator groups (AG), sensor logical objects (Ss), actuator logical objects (As) and the failure probability of the controller. The output is the failure probability of lighting control behaviour. According to the hierarchy of FTA and computation rule of the logical gate, the algorithm calculates the failure probability of each logical groups and the combination of them.

The Algorithm5 calculate the failure probability when there are duplicated controllers connected to the lighting control behaviour. It calculates the failure probability of each controller and its control scope. Then it calculates the failure probability of the lighting control behaviour based on the control scope and sensor groups.

Algorithm 4: Failure Probability of Disruption (Mean Time to Disruption)

```
Input: SG, AG, Ss, As, cs
Output: $F_{disrupt}$
for each sg in SG do ;
  for each in SG do ;
    if ss belongsto sg then ;
      $F_{sg}$.multiply($F_{ss}$) ;
  $D_{SG}$.multiply(1 - $F_{sg}$) ;
for each sg in SG do ;
  for each in SG do ;
    if ss belongsto sg then ;
      $F_{sg}$.multiply($F_{ss}$) ;
  $D_{SG}$.multiply(1 - $F_{sg}$) ;
$D_{disrupt} = D_{SG} * D_{AG} * D_{cs}$ ;
return $F_{disrupt}$ = 1 - $D_{disrupt}$ ;
```

Algorithm 5: Failure Probability of Disruption (Mean Time to Disruption) with duplicated controllers

```
Input: SG, AG, Ss, As, cs
Output: $F_{disrupt}$
for each sg in SG do ;
  for each in SG do ;
    if ss belongsto sg then ;
      $F_{sg}$.multiply($F_{ss}$) ;
  $D_{SG}$.multiply(1 - $F_{sg}$) ;
for each scp in Scope do ;
  for each AG belongto scp do ;
    for each ag in AG do ;
      for each as in AS do ;
        if as belongsto ag then ;
          agf.multiply(asf) ;
    $D_{AG}$.multiply(1 - $F_{sg}$) ;
  scpD = $D_{cs}$*$D_{AG}$;
F(Contro$L_{SC}$ope) = product of each control scope;
$D_{disrupt} = D_{SG}$ * D(Contro$L_{SC}$ope) ;
return $F_{disrupt}$ = 1 - $D_{disrupt}$ ;
```

7.2 Implementing Languages

Since the programming language we use is Java, we develop the DSL in Xtext which has an advanced Java integration. Xtext is a framework for the development of programming languages and domain-specific languages. It covers all aspects of a complete language infrastructure, from parsers, linker, compiler or interpreter to Eclipse IDE integration.

[4] explains how DSL is defined and used and that is illustrated in Figure 7.1. DSL infrastructure is constructed in a meta-model workplace where the syntax and code transformation is defined. Shown in the upper part of Figure7.1, A DSL implementation starts by defining the concrete syntax, Then the Xtext framework is used to generate a parser, an Ecore-based metamodel[4] and a textual editor for Eclipse. The Ecore-based metamodel represents the Abstract Syntax, and the textual editor is using for Textual Input in Figure7.1. Not only the syntax but also transformation is defined in the meta-level workplace. Generated Code (Java in our project) of an instance is generated in the instance-level.

In our implementation, we focus on the concrete syntax (.xtex), code generation (.xtend), textual input(.deploymentDSL) and generated code (.Java) in the the Figure 7.1 that shows the transformation of the implementation work.

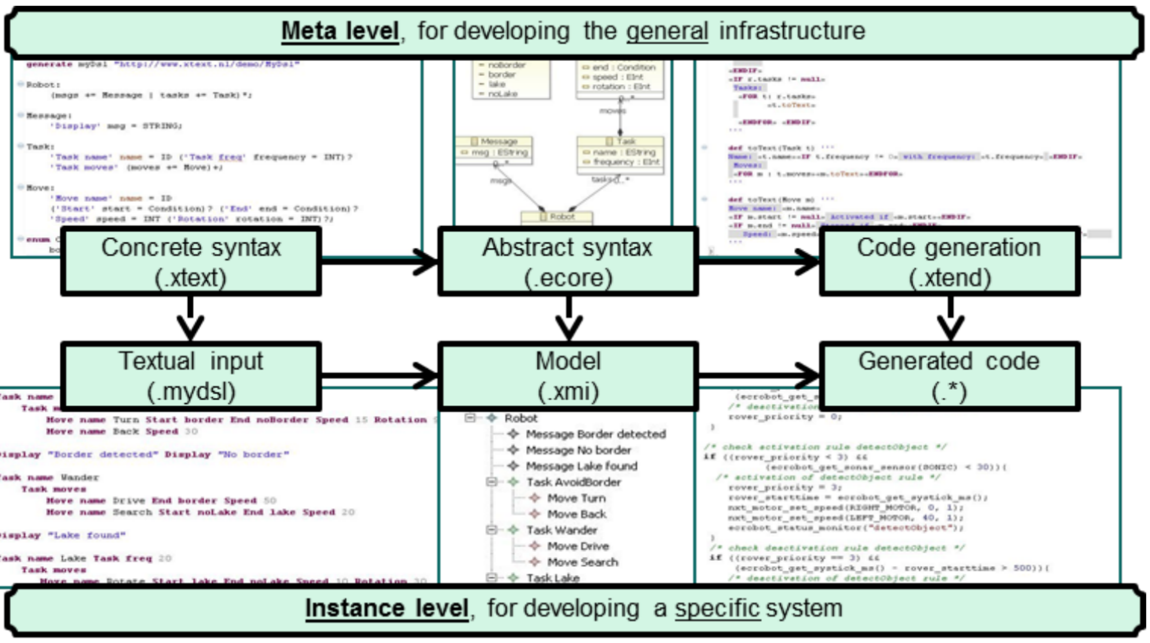


Figure 7.1: DSL Framework[4]

7.3 Cosim and Existing Work

We have introduced following DSLs in this thesis:

- **Building DSL**, to specify a building and its properties including the geometry and location of sensors as well as actuators in the building.
- **Template DSL**, to specify light behaviour consisting of state machines, transitions, actions and related logical objects.

- **Control DSL**, to specify control behaviours by mapping logical objects to devices in the building (Building DSL).

Using these languages helps one develops models that represent an abstract model of a building and its commissioned lighting control behaviour. When this abstract model has been created, it can be used to generate artefacts for many different goals, one of which is the creation of a co-simulation environment for which simulators for sensors, controllers and so on are generated.

7.3.1 Cosim Architecture

TNO-ESI has developed a simulation toolchain called co-simulation framework for simulating and verifying functional behaviour. It can simulate the behaviour of many nodes in any given building layout. Also, it is integrated into a framework allowing: scenario playing and logging, visualisation, integration with network model and extension to energy usage prediction. A Java implementation of the Co-sim framework is used as the simulation environment. The co-simulation environment consists of two parts shown in B.1:

- **The Cosim framework** that provides the timing and PublishSubscribe services to connected simulators. The framework forms the static part of the simulation environment and is used for each abstract lighting system model.
- **Simulators** that interact with the co-simulation framework through publishing messages and being notified by messages they subscribed. A number of the simulators has been created and instantiated:
  - A sensor simulator for each sensor
  - An actuator simulator for each actuator.
  - A sensor group simulator for each sensor group
  - An actuator group simulator for each actuator group
  - A controller simulator for each controller.

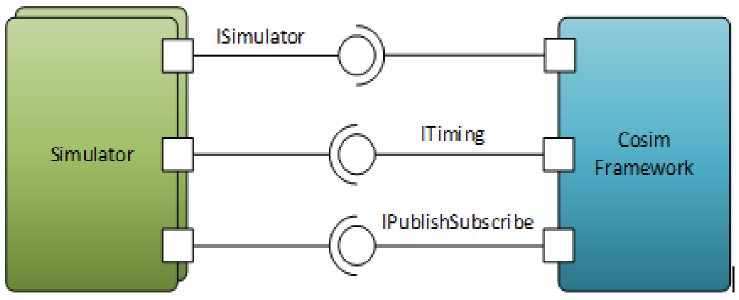


Figure 7.2: co-simulation environment

7.3.2 Simulator Generation

A transformation component, shown as T in figure7.3, has created that uses the abstract model DSL to generate required simulators to simulate the behaviour of the modelled lighting system.

The transformation T has to be developed only once, but can be applied to any Lighting System abstract model.

Such DSLs as Building DSL, Template DSL and Control DSL is essential for the generation of simulators. Building model provides a runtime for the building DSL, same in the cases of Template DSL and Control DSL. The transformation T has to be developed only once and can be applied to any Lighting System abstract model

In Cosim, Building DSL, Template DSL and Control DSL are textual input at Instance level shown in 7.1. The syntax is defined by xtext file and the transformation($\boldsymbol{T}$) is defined in xtend file. The generated simulator are Java file.

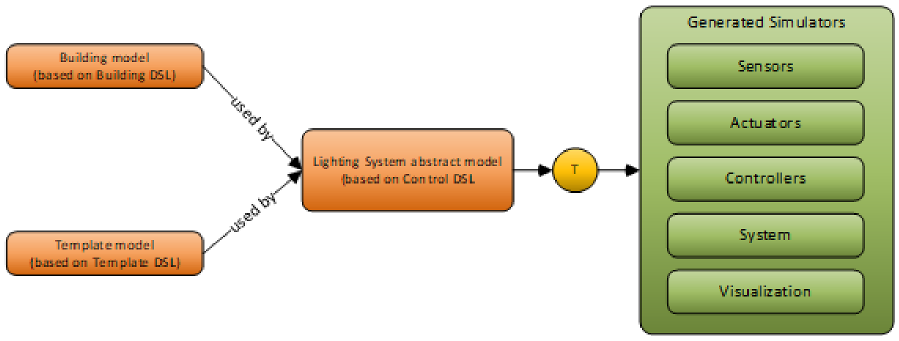


Figure 7.3: Generating simulators from an abstract Lighting System model

7.4 Implementation

In this section, we will introduce the implementation of the proposed decision-making process including evaluation components in CoSim and multi-objective optimisation in Matlab. We have done some extensions on co-simulation environment which are listed in Table 7.1.

Extension Work	File Type	Short Description
Deployment DSL syntax	.xtext	node functionality define; node capacity configure
Deployment Validation	.xtend	functionality duplication check; node capacity check
Pub/sub messages on Cosim	.Java	retrieve NodeID from DSL; add nodeID as an augment to published messages
Communication Monitor Simulator	.Java	counter messages published by sensors and subscribed by actuators
Latency Evaluation	.Java	naive network model (with communication cost); evaluate latency regarding network model, messages counter and deployment
Resilience Analyzer Dynamic	.xtend	retrieve system structure from DSL to generate fault tree
Resilience Analyzer Static	.Java	naive network model (with dependability); evaluate dependability regarding network model, FTA and deployment
Pareto	.matlab	score performance find Pareto frontier from deployment alternatives
Weighted Sum	.matlab	assign weight (user preference) to criteria and determine the optimal deployment
Duplication and Resilience	.matlab	calculate the failure probability when there are multiple controllers

Table 7.1: Implementation Work in This Project

7.4.1 Deployment DSL and node capacity check

To integrate the deployment, we design a simple deployment DSL. The simple deployment DSL is at Instance level of DSL structure in Figure7.1 that denote Functionalities and capacities of each node. Functionalities are presented by the name of sensors and luminaires defined by building DSL. Capacities are CPU and Memory which is used to validate if the node is capable of deployment. Control DSL defines the name and requirement on capacities of the controller.

The syntax used in capacity validation and node functionality assignment are defined in ***deployment.xtext*** File. It checks if the user deploys the functionality on nodes with the correct name. It also checks if the node to be chosen has enough CPU as well as Memory.

The translation syntax is defined in ***deploymentGenerate.xtend***. Every generated luminaire and sensor simulator is assigned the nodeID regarding deployment.dsl. Evaluation simulator is aware of this information.

The result capacity validation reminds the user of the compatibility before generating simulators. It makes sure that all nodes processed later don't have the problems on compatibility. Hence it reduces unnecessary computation.

7.4.2 Latency Evaluation

There are two components in the latency evaluation, which are message estimation and latency evaluation. The interaction style of Cosim is Publish/subscriber. By counting the number of the messages published by sensors and it subscribed by actuators, we can record the network load with given lighting control behaviour and scenario. Applying a typical scenario, we can estimate the communication load yield by the control logic and the use of the building.

The next component is used to calculate the critical latency and the maximum differential latency, which are described in the last chapter. The algorithms are shown below.

The differential value between departure time and arrival time is determined by the weights provided by network LUT.

7.4.3 Resilience Evaluation

As introduced in the evaluation design, FTA and CDF are used to calculate the probability of failure of lighting control behaviour. We have introduced the combination of event based on the property of the operational concept of lighting control behaviour and the generation of CDF from FTA description.

In the resilience evaluation component, following the rule of FTA and specification of control DSL, the order of CDF is generated. The value of each variable in the CDF is assigned referring to the weight in the network configuration LUT.

7.4.4 Multi-objective Optimization

The multi-objective optimisation can be divided into two components that are the Pareto Frontier component and Weighted Sum component. The former works as a filter selecting the non-dominant set while the latter according to user's preference computing the optimal solution.

The inputs of Pareto are quality profiles of each deployment decision. The quality profiles contain the evaluation result of each criterion that lies in four dimensions. By comparing every nodes' quality profiles, the Pareto components output the Pareto Frontier (non-dominant set) where each

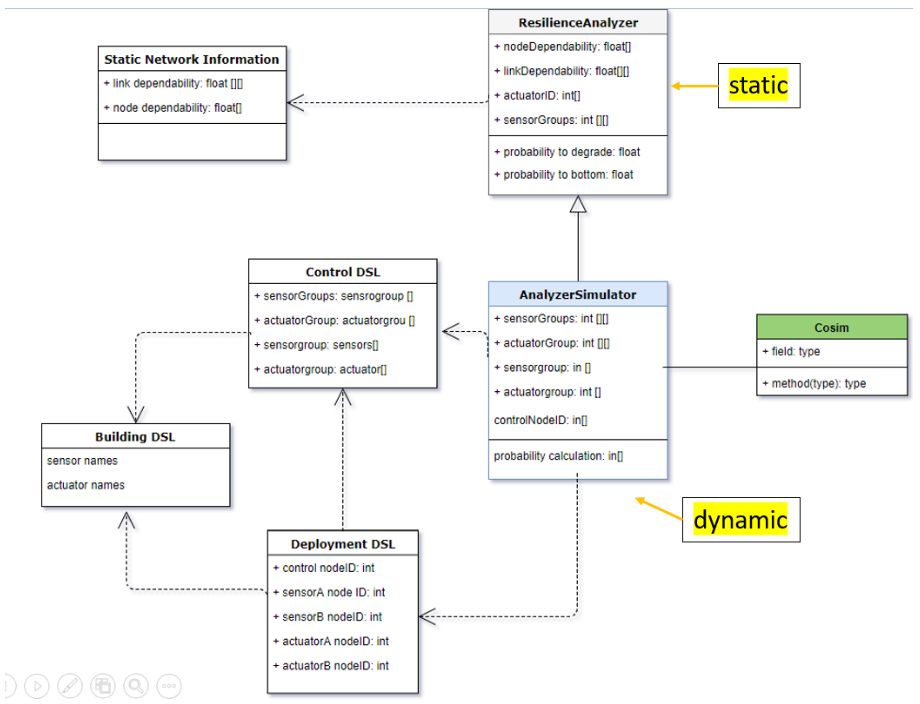


Figure 7.4: Resilience Analyzer Simulator UML

deployment decision is equivalent optimal.

By assigning weight to each quality profile, the evaluation component outputs the decision that yields the most fitting performance.

7.4.5 Message Types

All simulators on Cosim interact through publishing and subscribing messages, names of the message are extended when there are new simulators connected to the Cosim, or there is new topic created.

7.4.6 All Simulators

Our implementation work is integrated with existing work regarding the model in Figure7.5. The user decides the deployment of the controller and denotes the binding of Things to the controller through Deployment DSL. The translator generates augments (i.e. node identity) that are used by simulators. Network configuration is stored in the "Resilience Analyzer" and "Performance Analyzer", which are M1 and M3. Though network topology, routing and link quality and the binding are constant, different deployment decision would yield different paths from the node, where the controller is, to nodes, where Things are.

The MOOP components use the evaluation results calculated by the two evaluation components in Matlab. It compares results yield by all eligible deployments and outputs optimal decisions.

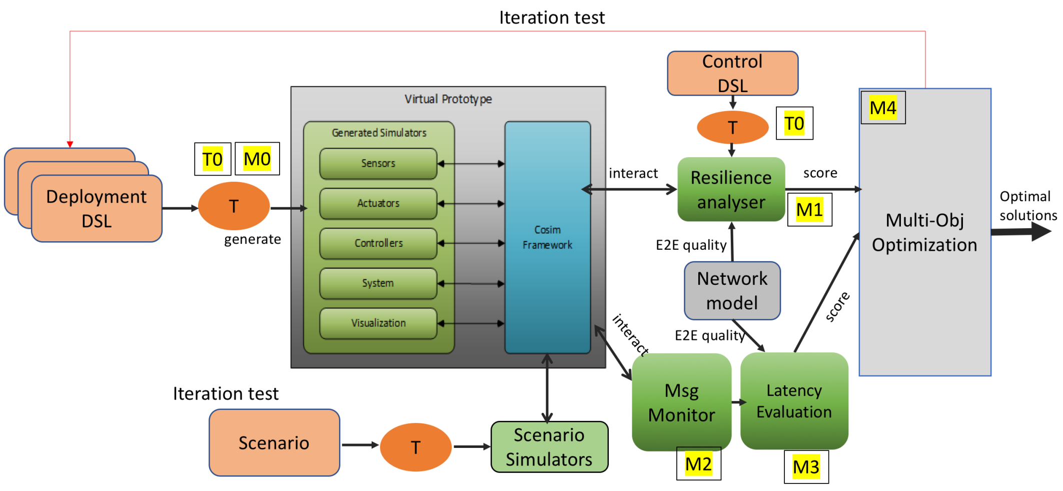


Figure 7.5: All simulators and test plan

Chapter 8

Experiments and Results

This chapter introduces two types of experiment that are optimal decision generation and resilience with the distributed controller. The former is to verify if the proposed evaluation methods can evaluate decisions and if the tool can find the optimal decision. The latter aims at further testing the output from FTA by observing the influence of distributed control on system resilience

We use two sections to introduce the experiments and analyse the results respectively. This chapter explains the experimental methods that were to compare the generated deployment decision from designed decision maker with the manual deployment decision from domain expert as well as to evaluate the FTA.

Our main goal is to highlight some of the capabilities of the proposed methodology including being aware of application specification, network configuration, scenarios, and QoS requirements.

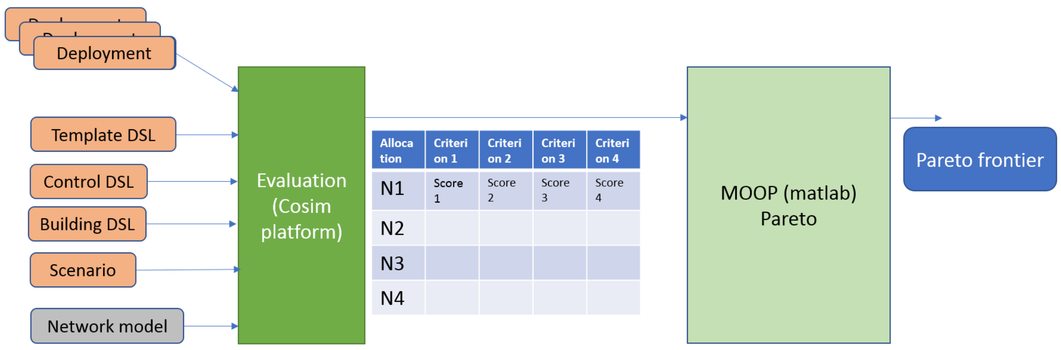
8.1 Experiment One – Optimal Decision Generation

This experiment is to verify the methodology proposed in this paper which is to compare the QoS of different deployment decision and to find the optimal one. To validate if the tool is aware of the network configuration and able to evaluate as well as compare QoS yield by different decisions, we used different network configuration to test with and compare the generated decision with the benchmark. In this section, we will introduce the process of this test, the input material, the benchmark, the platform. Also, we will present some results obtained to compare with the benchmark.

8.1.1 Test Setup

The test setup is shown in Figure 8.1. The input is an abstract model specifying the lighting control behaviour, a scenario describing the events, a network model and deployment DSL designed in this project. Those model are transformed to simulators on the Cosim platform. By executing the simulation and evaluation, QoSs of a deployment decision is obtained. After executing the process iteratively with different deployment decision each time, containing QoS of all available deployment decisions are recorded in a table. Then this table is transferred to the MOOP component on Matlab. The component generates a Pareto frontier set first and then output the optimal one from the set considering the user preference.

In the experiment, we focus on that which deployment decision belongs to the Pareto frontier in different network configurations with a fixed abstract model and scenario.



Allocation	Criterion 1	Criterion 2	Criterion 3	Criterion 4
N1	Score 1	Score 2	Score 3	Score 4
N2				
N3				
N4				

Figure 8.1: Test setup

8.1.2 Abstract Model

We will introduce the building described by building DSL, the control logic specified by Template DSL, object mapping specified by Control DSL, and events described by scenario DSL.

Things and the Building

In this experiment, a “thing” is either a sensor or a luminaire, and there are two types of sensor that are occupancy and button. There are ten Things which are four occupancy sensor, two buttons and four luminaires. The overview of the building is visualised in the Figure8.2.

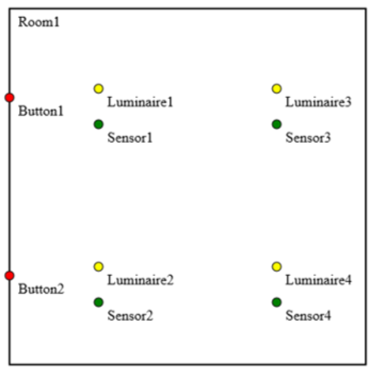


Figure 8.2: Things in the Building

Control Function

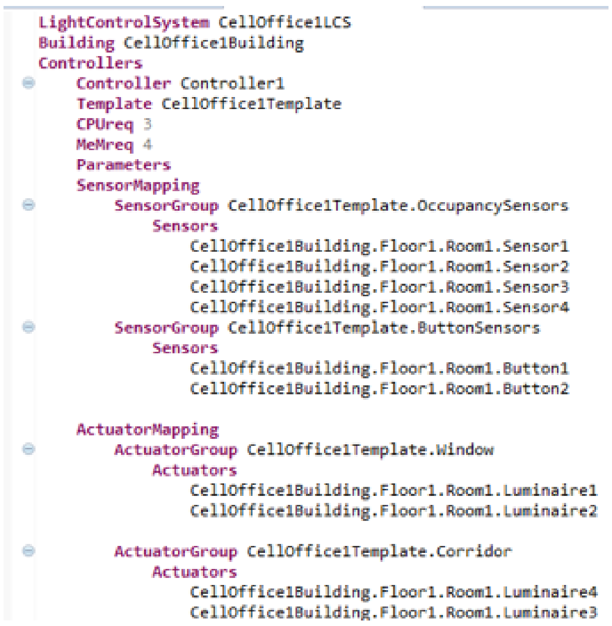
We use one control function in this experiment that is specified by a template DSL. Its control logic (state machine) is attached to the appendix. It has such property

- It requires two sensor groups Occupancy sensors and Button sensors.
- It requires two actuator groups Window luminaires and corridor luminaires.
- five states that “StateOff”, “StateOn”, “StateHold”, “StateManualVacant” and “StateManualOccupied”.

The idle state is “StateOff”, it will turns “StateOn” when occupancy is detected or “StateManualOccupied” when switching on the button. It will become “StateHold” when vacant is detected or “StateManualVacant” when switching off the button. The details of the state machine can be found in the appendix.

Object Mapping

The control DSL maps Things in the building to logical groups required by the control function. Shown in the Figure8.3, there are four groups, where one is a fat group containing four sensors.

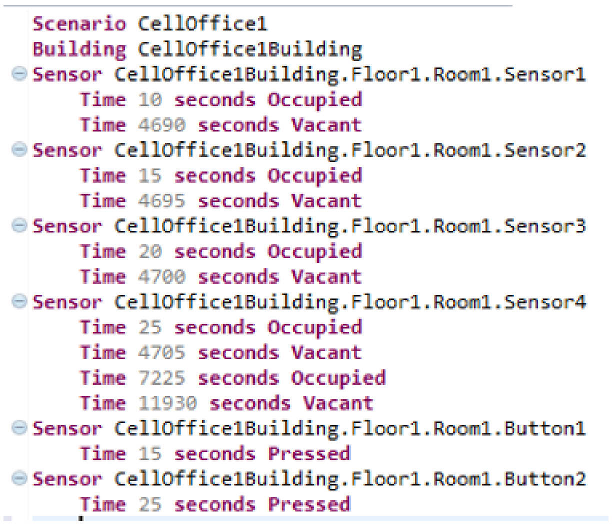


```
LightControlSystem CellOffice1LCS
Building CellOffice1Building
Controllers
    Controller Controller1
    Template CellOffice1Template
    CPUreq 3
    MeMreq 4
    Parameters
    SensorMapping
        SensorGroup CellOffice1Template.OccupancySensors
            Sensors
                CellOffice1Building.Floor1.Room1.Sensor1
                CellOffice1Building.Floor1.Room1.Sensor2
                CellOffice1Building.Floor1.Room1.Sensor3
                CellOffice1Building.Floor1.Room1.Sensor4
        SensorGroup CellOffice1Template.ButtonSensors
            Sensors
                CellOffice1Building.Floor1.Room1.Button1
                CellOffice1Building.Floor1.Room1.Button2

    ActuatorMapping
        ActuatorGroup CellOffice1Template.Window
            Actuators
                CellOffice1Building.Floor1.Room1.Luminaire1
                CellOffice1Building.Floor1.Room1.Luminaire2

        ActuatorGroup CellOffice1Template.Corridor
            Actuators
                CellOffice1Building.Floor1.Room1.Luminaire4
                CellOffice1Building.Floor1.Room1.Luminaire3
```

Figure 8.3: Object Mapping

Events Schedule

We use one scenario file denoting the event happened in the building on every sensor. We assume that the scenario file describes a typical use case in the room in the building. It should be noticed that the number of events happened on sensor four is greater than it on any other sensors, which would yield a certain message ratio between devices.



```
Scenario CellOffice1
Building CellOffice1Building
Sensor CellOffice1Building.Floor1.Room1.Sensor1
    Time 10 seconds Occupied
    Time 4690 seconds Vacant
Sensor CellOffice1Building.Floor1.Room1.Sensor2
    Time 15 seconds Occupied
    Time 4695 seconds Vacant
Sensor CellOffice1Building.Floor1.Room1.Sensor3
    Time 20 seconds Occupied
    Time 4700 seconds Vacant
Sensor CellOffice1Building.Floor1.Room1.Sensor4
    Time 25 seconds Occupied
    Time 4705 seconds Vacant
    Time 7225 seconds Occupied
    Time 11930 seconds Vacant
Sensor CellOffice1Building.Floor1.Room1.Button1
    Time 15 seconds Pressed
Sensor CellOffice1Building.Floor1.Room1.Button2
    Time 25 seconds Pressed
```

Figure 8.4: Events Schedule

8.1.3 Network Model

In this experiment, we assume that every network nodes are "things" in the building DSL, which means each node is a sensor or actuator, and there are ten nodes in a network. We test star networks and cluster networks, their topology are shown in Figure8.5 and Figure8.6.

By assigning weights, which represent link qualities, on each link, we can generate a communication LUT storing all qualities for every end-to-end path.

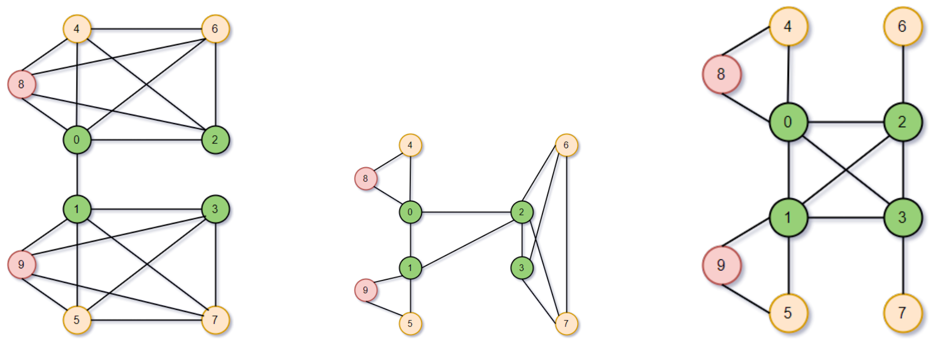

(a) 2 clusters, 2 gateway (b) 3 clusters, 3 gateway (c) 4 clusters, 4 gateway

Figure 8.5: Cluster Networks

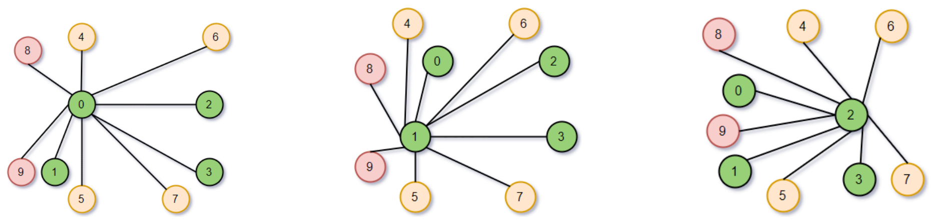

(a) hub on node 0 (b) hub on node 1 (c) hub on node 2

Figure 8.6: Star Networks

8.1.4 Parameter Selection

Referring to OpenAIS2.7[1], the communication cost for the transmission of a packet depends on the packet length, duty cycling, bandwidth, efficiency of encryption and decryption, and so on. The document gives some examples on calculating the end-to-end communication cost considering all those factors. In one of those examples, the communication cost of a packet for one hop distance approximates 9 milliseconds, which is used as the parameter in our experiment.

OpenAIS4.2[2] offers a table showing the occupancy ranking of failures and corresponding failure probability. We assign parameters representing the failure probability from the table in Figure 8.7.

8.1.5 Benchmark

Current deployment decisions at Signify are based on guideline and heuristics accumulated over time that includes factors like a number of node per gateway, layout of the network, distances between nodes and gateways. Over time a multitude of guidelines for the various systems has been developed to support the deployment. Yet personal and regional practice may still differ

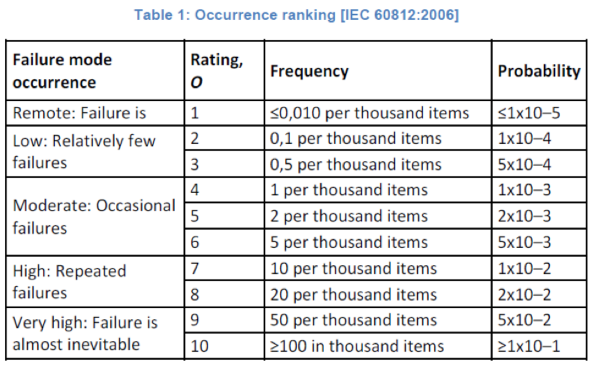


Table 1: Occurrence ranking [IEC 60812:2006]

Failure mode occurrence	Rating, *O*	Frequency	Probability
Remote: Failure is	1	≤0,010 per thousand items	≤1x10–5
Low: Relatively few failures	2	0,1 per thousand items	1x10–4
	3	0,5 per thousand items	5x10–4
Moderate: Occasional failures	4	1 per thousand items	1x10–3
	5	2 per thousand items	2x10–3
	6	5 per thousand items	5x10–3
High: Repeated failures	7	10 per thousand items	1x10–2
	8	20 per thousand items	2x10–2
Very high: Failure is almost inevitable	9	50 per thousand items	5x10–2
	10	≥100 in thousand items	≥1x10–1

Figure 8.7: Failure probability values defined in OpenAIS[2]

somewhat.

Examples of deployment used are fully distributed with each control function having a controlled scope of a single actuator, fully centralized with a single control function deployed to a clearly identified gateway, which connects several, and a combination with a central control deployment per area, in which the control is being deployed to one of the luminaires (quite randomly selected).

8.1.6 Experiment

We have two trials for the experiment. In the first trial, we assign identical weights including communication cost and failure probability on every links and node. In the second trial, we assign failure probability representing “low”, “media”, “high” failure modes in Figure 8.7 randomly on links, and repeat the experiment 8 times, then compare the deviation of deployment decision with results from the first trial.

First Trial – Identical Link Quality and Node Quality

For every star network in Figure 8.6, deploying the controller on the hub (center) yield minimized L_{rsp} , L_{syn} , $F_{degrade}$, $F_{disrupt}$.

Deployment decisions on cluster networks are shown in Figure 8.1. The analysis of the result is as following:

- Though the 2 clusters are symmetric to each other in Figure 8.5a, decisions on node one yield less L_{rsp}, as node 1 is closer to node 3, which is sensor 4 transmitting more messages due to the scenario in Figure 8.4
- In 3-clusters network, node 2 is closer to node 3, so that the decision on node 2 would yield a minimum L_{rsp}.
- In the 4-clusters network, though there is no communication cost between the sensor 4 and controller as they are allocated on the same Thing. The critical path is from a sensor in another cluster through this controller to an actuator in another cluster.

Second Trial – Random Link Quality and Identical Node Quality

In the second trial, the communication cost of links keeps identical while the failure probability on links is randomly assigned. The hub in a star network is the optimal decision which yields minimum failure probability.

	Cluster Networks		
	2 clusters	3 clusters	4 cluster
Min L_{rsp}	Node 1.	Node 2.	Node 0, 1, 2, 3.
Min L_{syn}	Node 0, 1.	Node 0, 1, 2.	Node 0, 1, 2, 3.
Min $F_{degrade}$	Node 0, 1.	Node 2.	Node 0, 1.
Min $F_{disrupt}$	Node 0, 1.	Node 0, 1.	Node 0, 1.
gateway ?	Yes	Yes	Yes

Table 8.1: decisions on cluster networks with identical weights

There are some deviations in the results for the cluster network. In a case of the 3 cluster network, where the links sub-cluster network has higher failure probability, decisions on node two would not yield an optimal failure probability.

	Cluster Networks		
	2 clusters	3 clusters	4 cluster
Min $F_{degrade}$	Node 0, 1.	Node 0.	Node 0, 1.
Min $F_{disrupt}$	Node 0, 1.	Node 1.	Node 1.
gateway?	Yes	Yes	Yes

Table 8.2: decisions on cluster networks with random link quality and identical node quality

8.1.7 Performance of Different Decisions

In this section, we study a case from the second trial of the experiment, where links have random failure probability and same communication cost. The communication cost for one packet on one hop is nine milliseconds where the failure probabilities are shown in Figure 8.8 Assuming nine nodes all satisfy the hardware requirement, there are 9 decisions in the network. We compare the nodes representing gateways with the other two nodes. The QoS calculation result are shown in the table 8.3. Decisions on gateways, such as node 0, 1, 2, 3, yield equal L_{rsp} and L_{syn} that are smaller than those yield by decisions on node 4 and node 5. The decision on node 0 yields the minimum $F_{degrade}$ and the decision on node 1 yields the minimum $F_{disrupt}$. Node 0 and node 1 are gateways connecting more nodes compared with node 2 and 3.

Decision	node 0	node 1	node 2	node 3	node 4	node 5
$F_{degrade}$	0.1207	0.1287	0.1740	0.1277	0.2397	0.1479
$F_{disrupt}$	0.006024	0.006008	0.007211	0.006122	0.007528	0.006256
L_{rsp} (ms)	180.0	180.0	180.0	180.0	288.0	288.0
L_{syn} (ms)	144.0	144.0	144.0	144.0	216.0	216.0

Table 8.3: Decisions in the 4-clusters network and their QoS

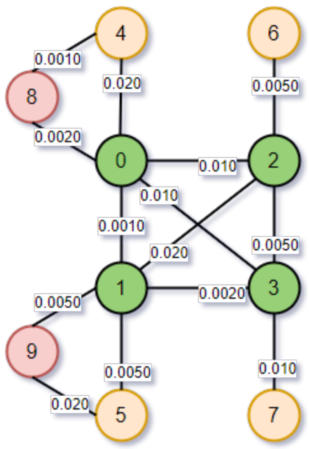


Figure 8.8: 4 Cluster random failure probability on links

8.2 Experiment Two – Distribution and Resilience

This experiment aims at observing the relationship between duplication level and resilience of the lighting control behaviour This experiment aims at observing the influence of distributed deployment (duplication in this context) on the resilience of the system that is the two types of failure probability of the lighting control behaviour.

8.2.1 Assumption for the Experiments

To exclude the influence of deployment, which is the allocation of the control function, we assume that reliability of each link in the network is ideal ($\forall$links, F_{link}=0). With this assumption, connects from a controller to every device are ideal. So that, referring to the FTA, the failure condition of each logical objects depends only on its device. Hence, in this experiment, the two types of failure probability depend on the failure probability of devices and their combination.

8.2.2 Setup

This experiment is done in Matlab. There are 12 things in this experiment, where four Things are sensors and eight Things are luminaires. The distribution is shown in Figure8.9. In a centralised deployment, every sensor groups have two sensors, and every actuator groups have two actuators. In a distributed deployment with two controllers, every actuator groups have fewer actuators compared with those in the centralised deployment.

In this experiment, we calculate the $F_{degrade}$ and $F_{disrupt}$ based on CDF when there one, two, three, four controllers in the deployment.

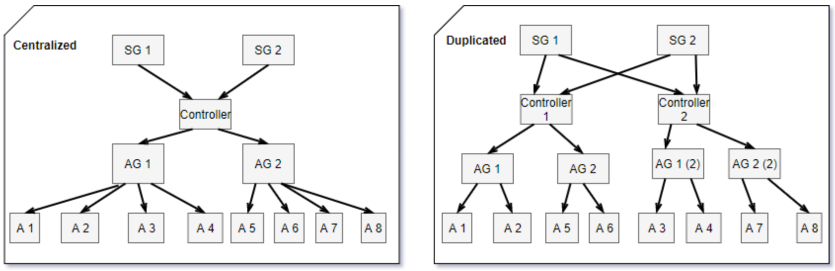


Figure 8.9: Distribution Test

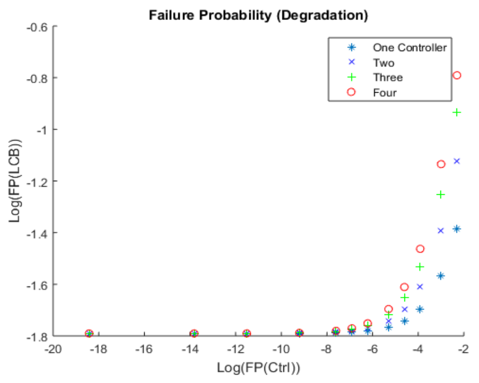
8.2.3 Deployment Network

Firstly Assuming the failure probability of each link is 0, the failure probability of any end-to-end connect (any path) is 0. Thus, referring to the FTA, the failure condition of each logical object equals the condition of the device. Secondly Assuming the failure probability of the controller is constant, and each controller has equal failure probability. The two assumptions help establish that the deployment of the controller will not yield different resilience performance. Also, the influence on the resilience is caused by the number of duplication and distribution of luminaires.

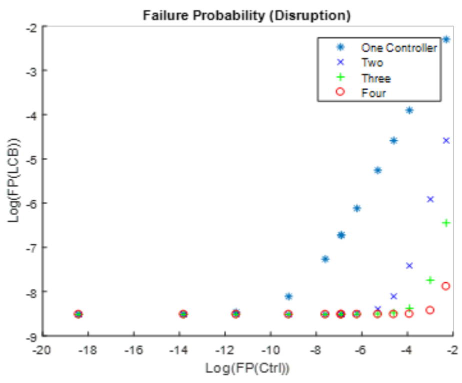
8.2.4 Visualization

With a constant failure probability of each sensor and luminaire, we increase the failure probability of the controller and observe the failure probability of LCB. Figure8.10 shows the growth of $F_{degrade}$ and $F_{disrupt}$ with a increasing F_{ctrl}. The $F_{degrade}$ with distributed deployment is higher than it with centralized deployment. Furthermore, the more duplicated controllers, the faster the $F_{degrade}$ grows when F_{ctrl} increases. Contrarily, the $F_{disrupt}$ with centralized deployment is the highest. And the more duplicated controllers, the slower the $F_{disrupt}$ grows.

With a constant F_{ctrl}, we increase the failure probability of Things and observe the F_{LCB}. Similarly to Figure8.10, Figure8.11 shows that the duplication would make $F_{degrade}$ higher and make $F_{disrupt}$ lower. However, these $F_{degrade}$ yield by different number of controllers approach when F_T increase, so do these $F_{disrupt}$.



(a) $F_{degrade}$ with increasing F_{ctrl}



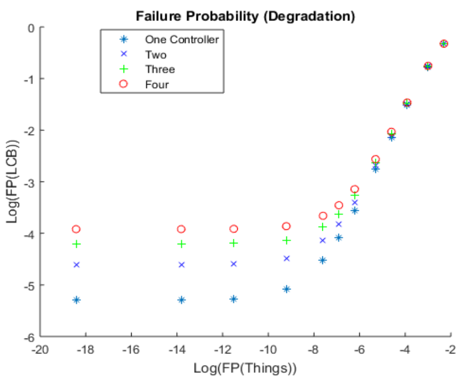
(b) $F_{degrade}$ with increasing F_{ctrl}

Figure 8.10: F_{LCB} and F_{ctrl} with duplicated controllers

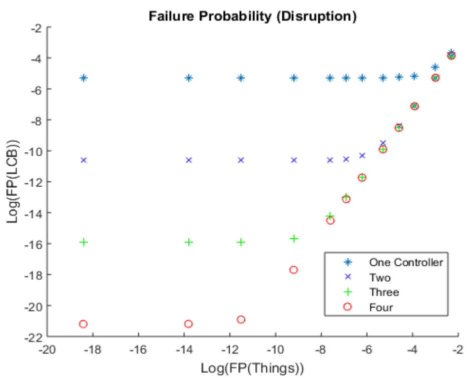
8.2.5 Analysis

The four scatters in Figure 8.10 and Figure 8.11 show that duplication technique have cons and pros influence on resilience. And the effect varies when the failure probability of components change. The observation and analysis are listed as follows:

- More controllers deployed in the lighting system would increase the $F_{degrade}$, which means a fault is more likely to appear when there are more non-ideal components connected to the system. On the other hand, duplication technique help to lower the $F_{disrupt}$ as the influence of failures of a controller and its connected actuator is limited to local control scope.
- Comparing Figure 8.10b By comparing Figure 8.10a with Figure 8.10b and by comparing Figure8.11a with Figure 8.11b , we observe that F_{LCB} with different number of duplications approach, which means the effect of duplication becomes little, when the controller turns reliable or things (i.e. sensors and luminaires) becomes unreliable.



(a) $F_{degrade}$ with increasing F_T



(b) $F_{disrupt}$ with increasing F_T

Figure 8.11: F_{LCB} and F_T with duplicated controllers

Based on the observation, we conclude that the time interval between the moment of start degrading and the moment to start disrupting becomes longer on a resilience curve when the distribution technique is applied. The duplication technique help delay the moment of start disrupting significantly when the controller has greater failure probability than other components, which is considered as one point failure in the lighting control behaviour.

Chapter 9

Conclusion and Future Work

In this chapter, we conclude the thesis work based on the results obtained. Also, possible extensions of this work are discussed which can be used in future.

9.1 Conclusion

The main contributions of this work are

1. Proposed a methodology to find the optimal deployment decision in an IoT system.
2. Proposed a methodology to evaluate resilience in a system considering the network configuration and application properties.
3. Designed a tool that assists users in making the deployment decision. Also, the tool can potentially support deployment design automation.

We reviewed existing work on deployment, considering the features of the lighting system, we identified and formalised the deployment problem and relate to software property and network configuration. Secondly, we identified suitable sub-criteria and metrics for the resilience and proposed evaluation methods. The method is aware of the network configuration (topology, link quality, node quality) and the system (application components dependency).

We also defined the process flow and interface between inputs and evaluation components as well as between evaluation components and the MOOP. With the defined interface, evaluation components can be further developed to improve the efficiency, accuracy. Evaluation of other QoS parameters can follow this pattern. We formalised a deployment decision problem related to the IoT-based lighting systems domain. Deployment is a problem faced by different domains including Internet of Things and virtualised network. Generalization of the solution is a concern of domain experts at Signify which was proposed at the very beginning of this project. To be able to describe different networks and applications specified by DSL, we defined general concepts in chapter 4.

The tool developed allows users to represent the network configuration in a look-up table formed model and the control behaviour via the Domain Specific Language. Also, the tool validates some critical aspects of the deployment and predicts the QoS, so that users can pay more attention to aspects which cannot be easily verified and validated. Furthermore, the tool offers the possibility to support comparison of QoS with distributed controllers and automatic generation of QoS-aware deployment. After giving the network configuration information and DSL description, a user only needs to provide the user preference, non-functional requirements. This help to eliminate human intervention in the current design process.

Based on the result of the experiments we found that:

- For QoS-aware centralised deployment, some QoS obtained by a decision can be a conflict with each other with a network configuration.
- For distribution, duplication as a manner of redundancy can improve an aspect of resilience that is $MTTF_{disrupt}$, especially when the controller is unreliable that is a single point of failure. However, the technique provides less effect when the Things in the control scope are also unreliable because homogeneous actuator in the same group is redundant to each other, distribution of them decrease the effect of redundancy of actuators.

9.2 Future Work

There can be some improvement on current work and research questions for future study.

9.2.1 Improvement on Current Work

- Different importance on functions and users preference on criteria can influence the optimal results. The relationship can be further studied.
- Current implementation on Cosim work does not support multiple controllers. It can be further developed
- Boolean is used to describe the validation of node capacity. we can use other data types, for instance integer, to describe the controllers' consumption of resource on Things.

9.2.2 Future Research on Resilience

- We assumed that controllers have same failure probability and actuators have same failure probability. It is interesting to observe what the optimal distribution plan will be if those failure probabilities are not identical
- we considered that the $MTTF_{degrade}$ and $MTTF_{disrupt}$ are on the resilience curve. Also, there are other criteria can be used to evaluate the resilience. For example, further research can study the quantification of the degradation interval.
- In the fault hypothesis used in this work, failures of components are permanent and independent. What evaluation methods are suitable when the components are recoverable, and the failure events are dependent?

9.2.3 Future Research on Latency

- In this project, we assumed that there is no synchronicity need among multiple controllers and we did not consider the influence of distribution on latency. So it is interesting to study the evaluation of latency considering distributed deployment.

Bibliography

[1] Final reference architecture of openais systems. pages 102–102, 2014. viivii, 7, 10, 11, 15, 16, 27, 45

[2] Fmea and hazard analysis report for openais systems. page 18, 2015. viivii, 12, 45, 46

[3] Jamal N Al-Karaki and Ahmed E Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE wireless communications*, 11(6):6–28, 2004. 6

[4] Jozef Hooman Arjan Mooij. Creating a domain specific language (dsl) with xtext. 2015. viivii, 36

[5] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, et al. *Fundamental concepts of dependability.* 9, 10

[6] Salvatore J Bavuso, JoAnne Bechta Dugan, Kishor S Trivedi, Elizabeth M Rothmann, and W Earl Smith. Analysis of typical fault-tolerant architectures using harp. *IEEE Transactions on Reliability*, 36(2):176–185, 1987. 11

[7] Zoran Bojkovic and Bojan Bakmaz. A survey on wireless sensor networks deployment. *WSEAS Transactions on Communications*, 7(12):1172–1181, 2008. 15

[8] Anna Bozza, Domenico Asprone, and Francesco Fabbrocino. Urban resilience: A civil engineering perspective. *Sustainability*, 9(1):103, 2017. 12

[9] Antonio Brogi and Stefano Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017. 6, 20

[10] V. Cherkassky. A measure of graceful degradation in parallel-computer systems. *IEEE Transactions on Reliability*, 38(1):76–81, April 1989. 11

[11] NM Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE, 2009. 6

[12] Qingzhi Liu Tanir Ozcelebi Emi Mathews, Salih Serdar Guclu and Johan J. Lukkien. Review: The Internet of Lights: An Open Reference Architecture and Implementation for Intelligent Solid State Lighting System, 2017. 14

[13] D Evans. The internet of things: How the next evolution of the internet is changing everything. 1:1–11, 01 2011. 1

[14] Crawford S Holling. Resilience and stability of ecological systems. *Annual review of ecology and systematics*, 4(1):1–23, 1973. 16

[15] Philip Koopman. Elements of the self-healing system problem space. 2003. 10, 11

[16] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88. ACM, 2009. 6

[17] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, Apr 2004. 8

[18] Victor A Netes and Boris P Filin. Consideration of node failures in network-reliability calculation. *IEEE Transactions on Reliability*, 45(1):127–128, 1996. 12

[19] G.W.E. Nieuwhof. An introduction to fault tree analysis with emphasis on failure rate evaluation. *Microelectronics Reliability*, 14(2):105 – 119, 1975. 11

[20] Leila Fatmasari Rahman, Tanir Ozcelebi, and Johan Lukkien. Understanding iot systems: a life cycle approach. *Procedia computer science*, 130:1057–1062, 2018. viivii, 1, 2

[21] Marvin Rausand and HÃ Arnljot. *System reliability theory: models, statistical methods, and applications*, volume 396. John Wiley & Sons, 2004. 11

[22] H. Narayanan Rupesh S. Shelar, Madhav P. Desai. Decomposition of Finite State Machines for Area, Delay Minimization, 1999. 17

[23] Ivanovitch Silva, Luiz Affonso Guedes, Paulo Portugal, and Francisco Vasques. Reliability and availability evaluation of wireless sensor networks for industrial applications. *Sensors*, 12(1):806–838, 2012. 9, 12, 28

[24] James PG Sterbenz, David Hutchison, Egemen K Çetinkaya, Abdul Jabbar, Justin P Rohrer, Marcus Schöller, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010. 8, 9

[25] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007. 11

[26] Petra Vizarreta, Massimo Condoluci, Carmen Mas Machuca, Toktam Mahmoodi, and Wolfgang Kellerer. Qos-driven function placement reducing expenditures in nfv deployments. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–7. IEEE, 2017. 7

[27] D. Wei and K. Ji. Resilient industrial control system (rics): Concepts, formulation, metrics, and insights. In *2010 3rd International Symposium on Resilient Control Systems*, pages 15–22, Aug 2010. 12

[28] Wikipedia. OSI model , 2017. 20

[29] Nita Yodo and Pingfeng Wang. Engineering resilience quantification and system design implications: a literature survey. *Journal of Mechanical Design*, 138(11):111408, 2016. 12

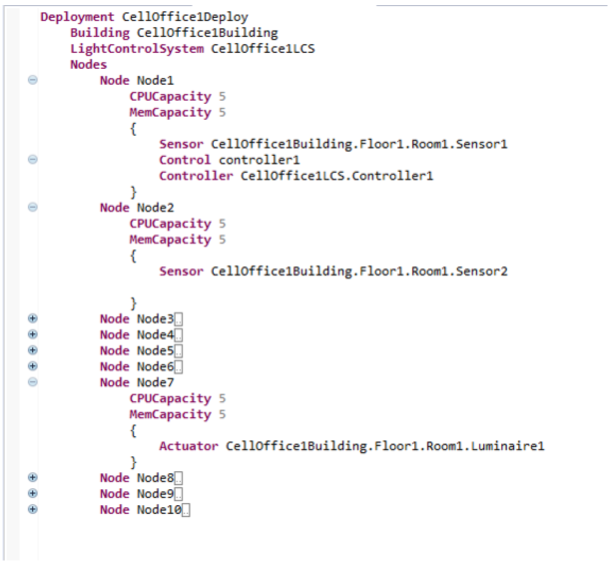
[30] Shih-Yuan Yu, Chi-Sheng Shih, Jane Yung-Jen Hsu, Zhenqiu Huang, and Kwei-Jay Lin. Qos oriented sensor selection in iot system. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*, pages 201–206. IEEE, 2014. 6

[31] Lingling Zhang. Message Obfuscation for Networked Lighting Systems. Master's thesis, Eindhoven University of Technology, 2017. 8

Appendix A

Deployment DSL

The deployment.DSL. The deployment DSL is able to validate if the hardware capacity satisfy the requirement of the controller, and to check if all Things are bound to nodes as well as if a Thing is bound to multiple nodes.



```
Deployment CellOffice1Deploy
    Building CellOffice1Building
    LightControlSystem CellOffice1LCS
    Nodes
        Node Node1
            CPUCapacity 5
            MemCapacity 5
            {
                Sensor CellOffice1Building.Floor1.Room1.Sensor1
                Control controller1
                Controller CellOffice1LCS.Controller1
            }
        Node Node2
            CPUCapacity 5
            MemCapacity 5
            {
                Sensor CellOffice1Building.Floor1.Room1.Sensor2

            }
        Node Node3
        Node Node4
        Node Node5
        Node Node6
        Node Node7
            CPUCapacity 5
            MemCapacity 5
            {
                Actuator CellOffice1Building.Floor1.Room1.Luminaire1
            }
        Node Node8
        Node Node9
        Node Node10
```

Figure A.1: Deplyment DSL, 10 Things, 10 nodes, 1 controller

Appendix B

The Visualization of Template DSL

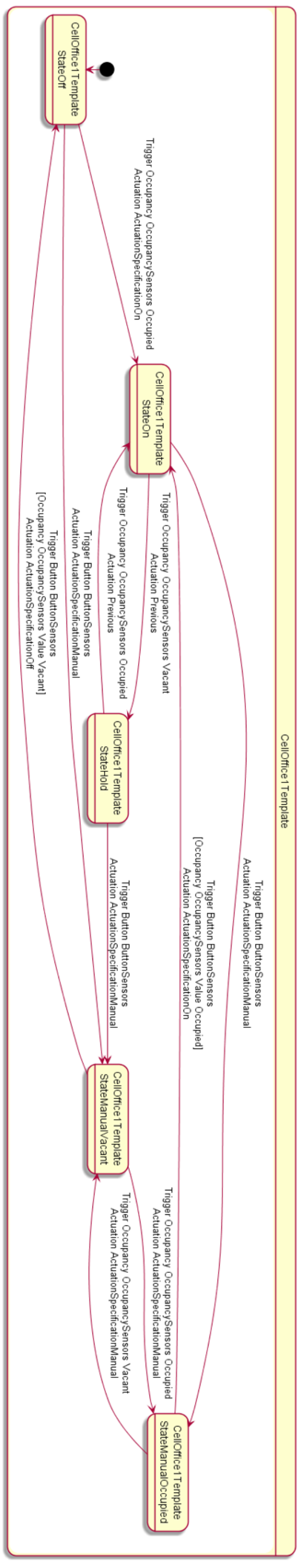

CellOffice1Template
CellOffice1Template StateOff
Trigger Occupancy OccupancySensors Occupied Actuation ActuationSpecificationOn
CellOffice1Template StateOn
Trigger Occupancy OccupancySensors Vacant Actuation Previous
Trigger Occupancy OccupancySensors Occupied Actuation Previous
Trigger Button ButtonSensors Actuation ActuationSpecificationManual
Trigger Button ButtonSensors [Occupancy OccupancySensors Value Vacant] Actuation ActuationSpecificationOff
CellOffice1Template StateHold
Trigger Button ButtonSensors Actuation ActuationSpecificationManual
Trigger Button ButtonSensors [Occupancy OccupancySensors Value Occupied] Actuation ActuationSpecificationOn
Trigger Button ButtonSensors Actuation ActuationSpecificationManual
CellOffice1Template StateManualVacant
Trigger Occupancy OccupancySensors Occupied Actuation ActuationSpecificationManual
Trigger Occupancy OccupancySensors Vacant Actuation ActuationSpecificationManual
CellOffice1Template StateManualOccupied

Figure B.1: The control logic used in experiment