AUTOMATIC CONVERSION OF ACTIVITY DIAGRAMS INTO FLEXIBLE SMART HOME APPS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER AND INFORMATION SCIENCES

Supervisors

Dr Roopak Sinha

February 2018

By

Nipuni Perera

School of Engineering, Computer and Mathematical Sciences

Copyright

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of candidate

Acknowledgements

The completion of this thesis would not have been possible without the immense guidance and support given by my supervisor. Dear Roopak Sinha, I am nothing but most humbly grateful to you for guiding me throughout this thesis. Also, I am extremely glad that I chose Software Architecture as one of the papers in the time of the coursework during my first semester at Auckland University of Technology, without which I would not have had the opportunity to be your student.

I would also like to thank my parents, Dear Ammi and Thathi, thank you very much for believing in me and giving me support throughout this year and listening to all my stories whenever I finished a new chapter of the thesis. Also thank you so much for tolerating my stress levels and being there for me during all the emotional breakdowns.

Not to forget my English Literature Teacher from High School, Dear Ms. Lakshila, I am tremendously grateful to you for teaching me to write with a style, if not for your mentoring in writing, this thesis wouldn't have been successfully completed. Thank you, Barry Dowdeswell, for proof reading my work and for the delightful feedback. Thank you to all my friends, family and everyone special who helped me make this journey a success.

Lastly, thank you Thathi for those early morning phone calls despite the time differences, those motivational conversations were one of the main reasons I could get where I am today.

This Thesis is dedicated to my loving Grandmother, Kamala Perera.

Abstract

Despite the availability of a large number of sensor and actuator devices designed to co-perform in a smart home, only a few of these devices are easily integrated into a single smart home unit. However, as devices become more advanced and feature-rich, the need for smart software to orchestrate these devices to offer complex smart home services has risen. The research focus of this thesis is designing and deploying software (or apps) that works with different, and changing, sensor-actuator configurations in smart-homes.

A systematic literature review was used to identify an visual design modelling framework for designing smart home apps. Behavioural models, specifically UML Activity Diagrams, were identified as the most appropriate app design model due to high usability and similarity with flowcharts. The literature review also informed the key qualities of an end-to-end solution to design and deploy these smart home apps.

Subsequently, we design and develop an *automatic translation tool* to address some key usability and deployment challenges. This tool offers a customized and fully-featured UML Activity Diagram Editor that allows non-experts to model any smart home system, such as a smart lighting system. The compiler offered by the Automatic Translation Tool accepts UML Activity Diagrams as input and generates executable Java code which can be deployed into any smart home application. An evaluation using a representative a set of case studies shows that the Automatic Translation Tool features high usability, availability, and performance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Smart homes can provide enhanced comfort and assistance through useful services like smart lighting, smart security, and smart living systems. Smart home technology also promises to support elderly living by themselves immensely, such as by issuing medication reminders or alerts in the case of emergency. These services are possible through the seamless integration of smart devices - sensors and actuators, and software that can control them as required.

Smart-home software, collectively called "apps" (Alaa, Zaidan, Zaidan, Talal & Kiah, 2017), is now getting increasingly complex, thanks to increasing needs for customisation as well as the introduction of innovative and disruptive device and IoT technologies.

This thesis focuses on the problem of easily creating and deploying smart-home apps. Existing tools to create these apps cannot be readily used by non-experts, such as medical experts wanting to deploy apps within their patients' smart homes. Technically, the problem addressed in this thesis is to efficiently and automatically generate correct code for these apps from *behavioural* diagrams. Behavioural diagrams, have been identified in this thesis as more usable for non-experts wanting to design smart home apps. Moreover, existing automatic code-generators only work for *structural* diagrams,

such as UML Class Diagrams (Rumpe, 2016).

The research questions addressed in this thesis are:

- **RQ1**–*Which factors lead to the choice of an app design model to address the challenges such as lack of interoperability when apps need to operate in different sensor-actuator configurations in different smart homes?*: The first research question addresses the factors leading to the choice of an appropriate app design model for non experts. This research question is answered in Chapter 2.

- **RQ2**–*What are the existing solutions and how do they meet the factors identified in **Question 01**?*: This research question identifies the key qualities of a tool for designing and generating code for smart home applications. This research question is also answered in Chapter 2.

- **RQ3**–*What are the characteristics of the architecture of an automatic translation tool, which translates an app design (based on the model developed after answering RQ1) to a customized smart home app?*: This question focusses on the architectural design of the translation tool, focussing on the qualities identified in RQ2. The architecture design of the tool is covered in Chapter 4.

- **RQ4**–*How can the high-level architecture of an automatic translation tool obtained from Q3 lead towards the implementation of a prototype automatic translation tool?*: The final research question is concerned with the implementation of the tool, and is answered in Chapter 5.

The design of this research followed a systematic literature review to refine and answer RQ1 and RQ2, and an adapted Design Science methodology to design and develop a solution (RQ3 and RQ4). Details of the research design are presented in Chapter 3.

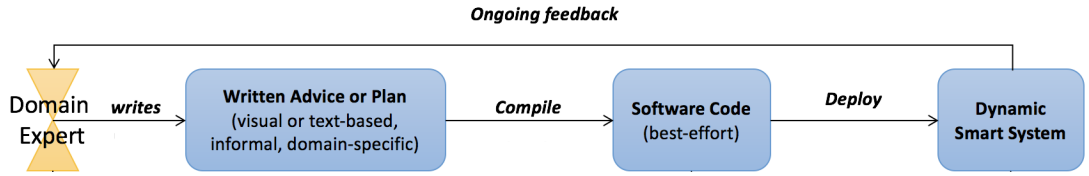

Figure 1.1: Overview of the automatic translation tool

Fig. 1.1 shows an overview of the features of the automatic translation tool proposed in this thesis. The tool automatically generates executable Java code from a UML Activity Diagram. Activity Diagrams were identified as the most appropriate app design model for smart home apps while answering RQ1 (details are presented in Chapter 2). The Eclipse-based automatic translation tool offers a fully featured editor for designing smart home apps. The tool includes a compiler which accepts a visual plan, expressed as a UML activity diagram, and translates the visual plan into a runnable code which can then be deployed to a smart home application. An evaluation of the tool shows that the editor has high usability, and that it can model any UML Activity Diagram. The compiler is also shown to generate error-free and compact code (Chapter 6), which can be deployed easily into any smart-home.

The primary contributions of this thesis are:

1. Choosing UML Activity Diagram as the more appropriate app design model for designing smart-home apps. UML Activity diagrams are easier to use for non-experts due to their similarity with flowcharts, and also because they capture behavioural aspects rather than the structural aspects of a program.

2. The design and development of a customized and fully-featured Activity Diagram model editor. This editor features strong UML activity diagramming features, model element relationships and navigation among the model elements between the model editing space and the customized palette. Moreover, the editor can be used to design any smart home app.

3. The design and development of a compiler that can generate correct executable Java code from UML activity diagrams. This code can be deployed to a smartphone application in any smart home.

The rest of this thesis is organised as follows. Chapter 2 presents a comprehensive systematic literature review carried out to answer RQ1 and RQ2. Chapter 3 presents the research methodology followed throughout the thesis. Chapter 4 defines the architecture of the proposed solution, followed by Chapter 5 which details the implementation of the automatic translation tool. Chapter 6 evaluates how well the automatic translation tool achieves the identified system requirements, especially as compared to other existing solutions. Finally, Chapter 7 summarizes the thesis and provides future research directions.

Chapter 2

Literature Review

This chapter carries out a Systematic Literature Review (SLR) to find the literature evidence about the challenge of lack of interoperability in the context of smart homes. In addition, finding and summarizing the prior research, this chapter provides justifications in order to establish credibility. Prior to the SLR, most commonly used terms are defined, these terms include: smart home, app design model, lack of interoperability and sensor actuator configurations.

The systematic literature review is followed to find valid and accurate findings and evidence. This comprises of various phases such as scoping, planning, identification, screening, and eligibility. During the SLR process, the current state of the art, which identifies and assesses the existing solutions and products which categorizes them into research-based solutions and commercial solutions are addressed in detail.Furthermore, the research questions formulated during the scoping phase are answered with the support of literature evidence in the findings section of this chapter, which also emphasizes the research gaps.The identified existing solutions are evaluated on a specific criterion which in turn supports the design of the proposed tool "***The Automatic Translation Tool***".

2.1 Background

This section identifies and emphasizes on relative keywords that will be used throughout this thesis. The keywords include *smart home, app design models, lack of interoperability and dynamic sensor actuator configurations*.The figure shown below represents a typical smart home with sensors such as light sensor, a dust sensor and motions sensor, actuators such as smart light, smart refrigerator, and smart microwave. Also, data from these sensors and actuators are collected and analyzed by different smart home systems such as temperature control system, health, and wellness systems and elderly care systems. As per Figure 2.1., the data from each device, sensor and smart home system are stored in a cloud-based smart home server and controlled via the wearable sensor.

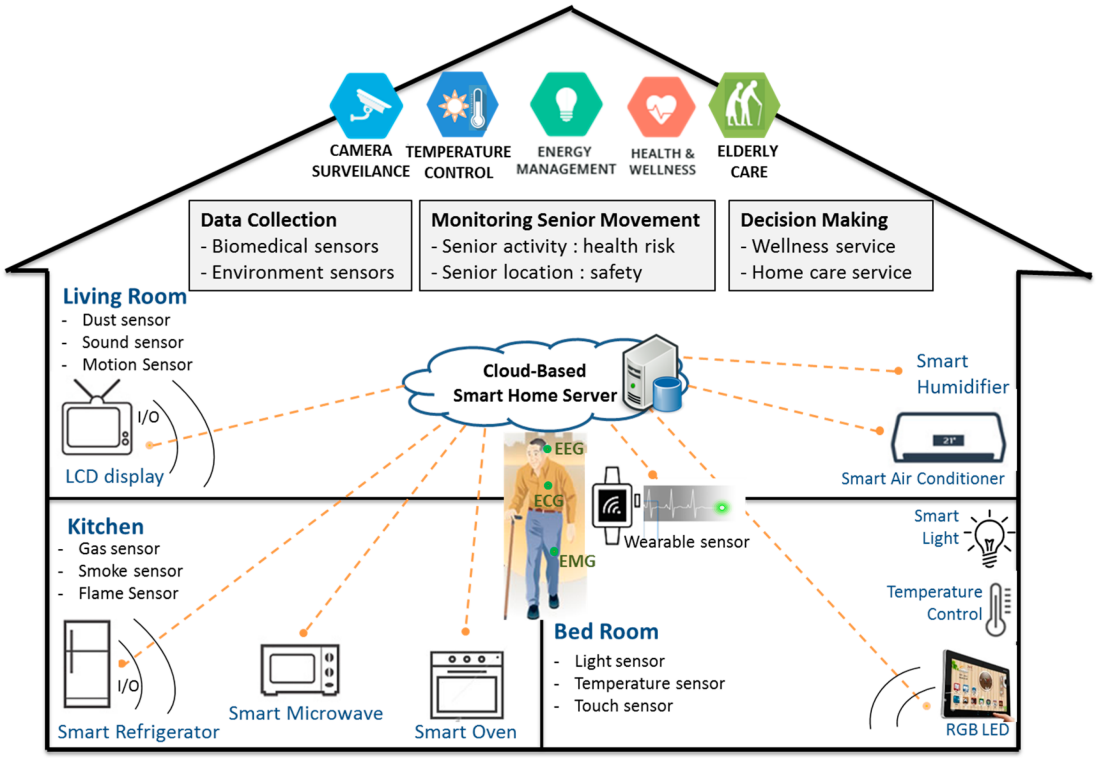

Figure 2.1: Smart Home

2.1.1 Smart Home

As shown in Figure 2.1 (Jung, 2017), a smart home involves centralized and semi-automated control of environmental systems such as heat and light with the use of technology to monitor and control the compatible objects in a smart home environment. A smart home is a complex entity with a set of diverse systems facilitating various functionalities in order to occupy the requirements based on information obtained from smart objects and occupants context (Perumal, Sulaiman, Mustapha, Shahi & Thinaharan, 2014). Smart homes consist of smart objects which are programmed using programming languages and are not accessible by end users. Smart homes may consist of multiple compatible smart objects (Hafidh, Osman, Arteaga-Falconi, Dong & Saddik, 2017).

2.1.2 Visual Plan

Visual plans, also called visual paradigms / patterns, architectural patterns, blue prints. The use of an app design model when designing software, various additional opportunities are offered such as high levels of quality improvements, automated code generation, improved problem solving and design capabilities and overall productivity during the development of the software (c, Miah & McAndrew, 2016).

2.1.3 Interoperability

As per (Miller, 2000), interoperability is the ability of a system to function with other systems without the need for need for special effort.This term is also defined by (Ritter, Zirpins, Schoenherr & Motahari-Nezhad, 2007) as the capability of numerous, autonomous and heterogeneous systems to use each other's functions and services

effectively. Furthermore, it involves meaningful sharing of information which in turn would support the achievement of common goals. In terms of Figure 2.1, interoperability is the ability of the smart light system and the smart refrigerator to be used in the same smart home without any challenge in terms of compatibility.

2.1.4 Lack of interoperability

As defined earlier, interoperability is the ability of a system to function with other systems without special effort from the user's end. In terms of healthcare and smart homes, lack of interoperability emphasizes on low levels of ability of multiple tools and devices to function cooperatively without having the need to obtain functionality supports from users (Capitanelli, Papetti, Peruzzini & Germani, 2014).

2.1.5 Dynamic sensor actuator configurations

Each smart home has its own sensor actuator configuration, which might differ from sensor actuator configurations of other smart homes. As shown in Figure 2.1, a smart home consists of different kind of sensors such as light, temperature and touch sensors which are connected to their respective devices (such as smart lights and temperature controls)also know as actuators via smartphone and tablets.

2.2 Systematic Literature Review (SLR)

There are several ways to carry out a literature review for a research study. Two of the most significant techniques include the traditional literature review and the systematic literature review (also known as SLR). The traditional method in international research

development mostly and exclusively focuses on the results of other studies without taking factors such as the research design and data collection methods into consideration. This technique also is restricted to the literature that is already known by the existing authors which are found by conducting little more than cursory research (Mallett, Hagen-Zanker, Slater & Duvendack, 2012). This, in turn, means the same existing studies are cited frequently which can lead to persistent bias in the literature review.

On the other hand, the systematic literature review emphasizes more strongly on evidence, impact, casualty and the validity of the research study. This technique also helps in eliminating research bias by adopting broad search strategies, predefined search strings and uniform inclusion and exclusion criteria which forces the researchers to search for studies beyond their subject of research (Mallett et al., 2012). Moreover, by extracting the information from the research design, the systematic literature review performs effectively at gauging the robustness of the research evidence. Also, this method encourages the researchers to engage with the research studies with advanced criticality and to have a consistent prioritization of empirical evidence over preconceived knowledge (Mallett et al., 2012).

Therefore, when the above facts are taken into account, the systematic literature review is chosen as the techniques to be applied when carrying out the literature review for this research study. In addition to the above-mentioned benefits of applying SLR over traditional literature review, there are various additional advantages such as improved quality of reviews through transparency, the inclusion of studies with greater breadth and improved objectivity and reduction of implicit research bias.

2.3 Process of Systematic Literature Review

2.3.1 Scoping

This part of the research involves formulating research questions based on various focus areas such as information required in terms of the topic area, who will be your audience; do you have a clear idea of the research findings that will be relevant to addressing of the research question chosen, are the formulated research questions answerable (this would support in achieving successful and informative review).

1. **Formulation of Research Questions**

   In order for the formulation of potential research questions, there is the need to identify the problem definition which lies in the unresolved challenges area, which is identified below:

   Problem definition: ***Each smart home may consist of their own sensor-actuator configuration; therefore, code generation and mobile app development for each smart home may be undesirable***.

   The above-mentioned problem definition leads to the identification and formulation of the below-listed research questions

   - **Research Question 01** - *Which factors lead to the choice of an app design model to address the challenges such as lack of interoperability when apps need to operate in different sensor-actuator configurations in different smart homes?*
   - **Research Question 02** – *What are the existing solutions and how do they meet the factors identified in* ***Question 01***?

- **Research Question 03** - *What are the characteristics of the architecture of an automatic translation tool, which translates an app design (based on the model developed after answering Q1) to a customized smart home app?*
- **Research Question 04** - *How can the high-level architecture of an automatic translation tool obtained from Q3 lead towards the implementation of a prototype automatic translation tool?*

2. **Clarify if Systematic Literature Review has already been carried out on the chosen topic**

   This part of the thesis ensures various factors, such as: if literature review in terms of the area of focus is carried out previously if relative literature review for the area of focus exists and does it require improvements and updates. Furthermore, if the literature review already exists, how long ago were the literature reviews carried out and if they consist of any potential flaws and errors.

   Various readings and literature findings are identified in terms of smart home scenarios, app design models, automatic code generation and lack of interoperability. These readings are analyzed and assessed the later current state of the art section of this chapter.

   Lack of interoperability and low levels of compatibility in terms of the health of elderly people is addressed by Smirek whereby Eclipse Smart Home project is introduced with Universal Remote Control in 2016 which involves the integration of various technologies into a single smart home environment. However, use of this framework may lead to security and privacy issues (Smirek, Zimmermann & Beigl, 2016).

   A translation tool is introduced by (Sinha, Narula & Grundy, 2017) which addresses the interoperability issues when smart homes are involved with dynamic

sensor actuator configurations. This framework accepts parametric state charts as user input which may cause complications in terms of user-friendliness and understandability.

Therefore, although various literature findings in addition to the solutions listed above exist, which answers the above-defined research question, the above-stated facts prove that yet there exists room for improvement, this will be addressed in the new studies that will be carried out later in this chapter.

2.3.2 Planning

This part of the SLR focuses on the breaking down of the chosen research question into individual concepts in order to generate meaningful search terms and formulate preliminary exclusion and inclusion criteria. All the chosen research questions would go through the above-mentioned stages as shown below.

1. ***Break down of the research questions into individual concepts in order to create meaningful terms***

   Research Question focused: ***Which factors lead to the choice of an app design model to address the challenges such as lack of interoperability when apps need to operate in different sensor-actuator configurations in different smart homes?***

   The above-mentioned research question is translated into multiple search terms which in turn will support in answering the research question as a whole. This part of the literature review breaks down the above-mentioned question based on the search terms (search terms are determined by synonyms, broader or narrower terms, classification terms used by databases) identified below.

- App design models, software design models in software development, Model Driven Engineering (MDE), Engineering design, software design, design process, design models and visual models
- Lack of interoperability, inconsistencies, and incompatibilities in smart homes
- Dynamic sensor actuator configurations and smart homes

Therefore as explained by (Siddaway, 2014), the planning phase of the literature review involves breaking down the research questions into meaningful sub-questions taking the keywords into consideration. Hence, the above-mentioned research question is broken down into three sub-questions based on the key factors and keywords identified above. The sub-questions are as shown below:

- Which factors influence the need for an app design model (app design model can also be called the above-mentioned relative search terms)?
- How do challenges such as lack of interoperability incur during the practice of various smart homes?
- How does the introduction or use of a specific app design model ensure a low level of interoperability in dynamic sensor actuator scenarios?

The answers to the above-identified sub-questions are provided in the findings section of this chapter.

2. Formulate preliminary inclusion and exclusion criteria

- **Research Question**: ***Which factors lead to the choice of an app design model to address the challenges such as lack of interoperability when apps need to operate in different sensor-actuator configurations in different smart homes?***

- **Definition or Conceptualization**: App design model, Lack of interoperability, Different sensor-actuator configurations, Smart home, Healthcare and people with disabilities
- **Measures/Key Variable** : What is the need for an app design model?,How does the introduction or use of a specific app design model ensure low level of interoperability in dynamic sensor actuator scenarios?,How does the introduction or use of a specific app design model ensure low level of interoperability in dynamic sensor actuator scenarios?
- **Research Design** : Qualitative study
- **Participants** : N/A (focused participants in the development include old age people, people with disabilities and people with special needs)
- **Time Frame** : No particular time frame for now

When carrying out the inclusion and exclusion criteria, various keywords were excluded from the search criteria, this includes smart cities, wireless sensor networking, blue tooth, and Wi-Fi. Additional words will be added here when further research is carried out.

In addition to answering the formulated Research Questions, during the planning stage of the Systematic Literature Review, an exclusion inclusion criteria are carried out in order to ensure the accuracy and validity of the readings carried out during the research. For this purpose, various techniques are performed, this includes: sorting the search results and readings based on the keywords, for each reading excluded during the exclusion and inclusion criteria, reasons are provided as to why they are being excluded.

As mentioned above, PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) (Haroutiunian, Nikolajsen, Finnerup & Jensen, 2013),

is practiced during the exclusion and inclusion criteria of research and readings, this involves readings been assessed at various stages in terms of a flow chart which consists of various steps.

Step	Funtions	No.of Records
Identification	Records identified through database searching Additional records identified through other sources	31 0
Screening	Records after duplicates are removed Records screened Records excluded	30 30 01
Eligibility	Full text articles assessed for eligibility Full text articles excluded with reasons	30 01
Included	Studies included in qualitative synthesis Studies included in qualitative synthesis (meta-analysis)	30 30

Table 2.1: Inclusion and Exclusion Process

Generally, during the inclusion and exclusion criteria of the readings, in addition to the repeat of readings, various other factors were taken into consideration, these factors include the choice of search engines, keywords used and relative abstracts of each reading chosen.

The above-shown table illustrates how the steps indicated by PRISMA are taken into account when excluding the research studies that are invalid, repeated or not relevant to the area of research focus. When Table 2.1 is taken into consideration, only one research study is excluded due to repetition.

2.3.3 Searching

This section of the Systematic Literature Review involves finding available published and unpublished work which addresses the research questions identified above. This is carried out in various ways. The ways are as described below, which is related to the research area of interest.

Even though various search engines are available, chosen search engines were used to find the readings, these search engines include Google Scholar, IEEE and AUT Library which supported in finding the readings in relation to the research area of interest. When the context of each chosen article is taken into consideration, various aspects of the article were given importance, these aspects include abstract which gives an overview of the article, journal or the conference paper. In addition to the abstract, keywords highlighted in the articles would be taken into consideration in order to ensure the accuracy of the mapping of the focused keywords and keywords provided in the chosen article. As mentioned above, various keywords will be used during the searching process of the readings. This process involves uses of relative keywords such as smart homes, lack of interoperability, app design models, Source code generation, IoT, dynamic sensor actuator configurations, elderly care, dynamic smart homes, flexible IoT apps, and deployment of mobile applications (Moher, Liberati, Tetzlaff, Altman & Group, 2009).

Furthermore, Boolean search operators are taken in to consideration which involves the use of AND and OR between the keywords used during the search process, for instance: lack of interoperability and smart homes and use of "NOT" to ensure unnecessary context is not found, for instance: "smart homes NOT Smart Grids". Additionally, various other search terms were carried out with the support of Boolean search operators during the search process, these search terms include:

- Smart Homes AND Lack of Interoperability
- mHealth Apps AND Aging Population
- UML Diagrams AND Automatic Code Generation
- App design models AND Lack of Interoperability

This part of the systematic literature review also focuses on inspection of search results, which involves examination of search results in order to ensure higher levels of accuracy and quality. This is ensured in various ways, such as: (1) ensure effectiveness and reliability in terms of the inclusion and exclusion criteria carried out (2) identification of new and additional search terms such as architectural designs, automated code generation, code design and testing in relation to the smart home scenarios (Moher et al., 2009).

In addition to the facts mentioned above, additional searches are carried out to ensure that potential readings are published. This is ensured by considering the number of references in the readings chosen and by attempting to find the relevant context in potential book chapters, this includes the Prototyping Methodology proposed by Kenneth E.Lantz and Software Architecture in Practice by Len Bass.

2.3.4 Screening

This part of the systematic literature review involves exporting the search results to a citation manager, in this context, the citation manager is EndNote. Use of EndNote provides various benefits. These benefits include: the search results are saved and backed up which would eliminate data loss issues, use of cite manager reduces the search result duplication which eliminates issues in terms of data redundancy, EndNote provides the ability to share the search results with other users and compile the search result list in to a form of array of reference styles (Moher et al., 2009).

2.4 Current State of Art

This section of the chapter identifies and assesses existing solutions, these existing solutions are categorized in to two subsections, i.e.: research-based solutions such as: Simple Internet of Things Enable (SITE), a proactive architecture for Internet of Things, rule-based intelligence for domotic environments and IoT suite framework and commercial based solutions such as: Eclipse Smart Home project supported by Universal Remote Control, ZigBee Alliance and LonWorks and translation tools to support automatic code generation.

2.4.1 Research Based Solutions

Various studies and developments have been carried out to address various challenges confronted by the users of smart homes. Even though the use of smart homes have provided the solutions for few of the challenges such as fall and disaster predictions and detections, few other challenges remain unsolved, which will be addressed in this subsection of the thesis.

Simple Internet of Things (SITE)

(Hafidh et al., 2017) presents a simple Internet of Things enable (SITE) which is a smart home solution that allows the users to specify and gain control of Internet of Things (IoT) smart objects. This commercial based solution supports end-users by providing a user interface which graphically illustrates the data obtained from smart objects which in turn enhances usability and user-friendliness.

The End User Development (EUD) paradigm proposed by Hafidh allows the

end users to design the smart home applications visually with the use of a pen-based interface. This, as a result, permits the end users to specify the input and output devices. Also, the users are allowed to set up behavioral logic using "if-then-else" statements which describe the preferred conditions and associations by every specific user (Hafidh et al., 2017).

The EUD paradigm is based on a smart home control and monitoring system known as the Simple Internet of Things Enabler (SITE). This solution interacts with two types of entities, i.e.: users and Smart Objects (SOs). This solution defines a user as an individual that creates as an SOs with the use of General Purpose Transducers Network (GPTN), this configures a smart home environment by sending commands to smart home objects (SO) and visualize the information produced by the smart home object by using SITE Central Visualization and Control CVC. In order to configure a smart home environment, the user is allowed to specify the function the SITE CVC should perform in order to respond to the data received from sensors of the smart object (SO) (Hafidh et al., 2017).

Even though this solution aims to achieve higher levels of usability, few of the models remain untested which leaves the attempt to achieve better levels of end-user usability unachieved to a certain extent (Hafidh et al., 2017). In addition to the above, the area of focus including the problem definition and the formulated research questions aims to achieve lower levels of interoperability which is not addressed by this solution, which leaves the existing challenges unaddressed thus leaving room for improvement.

Parametric State Charts

Furthermore, the above-mentioned solution does not address the problem of lack of interoperability when smart homes are dealt with dynamic sensor actuator configurations.

Thereby a translation tool is proposed by (Sinha, Narula et al. 2017) which accepts a parametric state chart with a specific static sensor actuator configuration and converts to a standard state chart which is compatible with the given system configuration. This, in turn, can be translated and compiled to an Android code with the help of a code generator and a custom compiler (Sinha et al., 2017). Although the above-mentioned solution addresses the problem definition largely and provides appropriate and relative solutions for various challenges such as low levels of interoperability and compatibility, use of statecharts in the process may introduce the users to further challenges. These challenges include, as the systems improve in size the statechart may become complicated and become difficult to understand (Zhang, Roy et al. 1999). As this project is mainly focused on elderly people, level of user-friendliness and the need to understand the system stand as important factors.

Furthermore, state charts offer limited potential reuse, although actions associated with transition provide a powerful extension, repetitive use of these actions may move parts of the systems' state information from the states themselves to variables, which in turn may cause difficulties during the analysis stage (Zhang, Roy et al. 1999).

A Proactive Architecture for IoT

A proactive architecture for the Internet of Things supporting management of smart homes is identified by (Perumal et al., 2014). This solution addresses various complications confronted by smart home users, these challenges include: huge diversity of various smart home applications has caused interoperability requirements and current IoT management along with the practice of physical platforms have confronted challenges such as lack of intelligence on decision making.

The proactive architecture consists of various layers to perform decision-making functions for the IoT systems. The core layer consists of the web services that offer service coupling between the system and its services, this layer also represents a set of IoT systems, home gateway, and web service modules. The home gateway functionality enables access to external networks. In addition to the above, as mentioned previously, lack of interoperability and challenges in terms of compatibility occur during the introduction of new devices to a smart home system, this challenge is addressed by the practice of specific modules such as device APIs and device stub in order to ensure new systems' dependencies.

As per the above stated facts in relation to the proactive architecture for IoT management in smart homes, it proves that even though lack of interoperability is addressed to a greater extent with the use of different layer and inclusion of different modules in the architecture, the challenge of usability in terms of end-user context is left unaddressed thus leaving room for improvements.

Rule based intelligence for Domotic Environments

Rule-based intelligence for domotic environments (Bonino & Corno, 2010) is largely supported by home automation systems mainly on human-related environments from homes to hospitals. This specific solution aims to address two main challenges, such as lack of interoperability and insufficient support for advanced user-home interaction (Bonino & Corno, 2010). Bonino's research work defines a solution to address these challenges, which is, Intelligent Domotic Environment (IDE) which allows integration of different automation systems, appliances, and devices into a single powerful environment, which is capable of providing Ambient Intelligence (AmI) functionalities.

As mentioned above, IDE addresses the challenge of lack of interoperability

with the support of a strong modeling component, whereby a specific home is environment is presented through formal definition in the form of an ontology. Modelling allows to make device descriptions independent of technology-specific aspects thus enabling integrated automation scenarios (Bonino & Corno, 2010).

In addition to the practice of IDE, rule-based reasoning is adapted to identify the behaviors of IDE. This methodology supports in defining rules for a home environment with the support of structural rules which allows checking constraints for smart home or smart plant configurations with respect to architectural elements, devices, and appliances (Bonino & Corno, 2010). Additionally, structural rules are able to verify the in-house conditions depending on the current state of the device which in turn provides support for advanced policies. Even though the feasibility studies show higher levels of efficiency in terms of rule-based reasoning, the results also emphasize on lower levels of flexibility during multi-stage reasoning (Bonino & Corno, 2010). In addition to the above, various other challenges still continue to exist even after this solution was adopted, these challenges include the challenge of handling complexities during dynamic and uncertain environments and the challenge of cooperating devices with varied (Bonino & Corno, 2010).

Architecture for Software Defined Smart Homes

(Xu, Wang, Wei, Song & Mao, 2016) proposes an architecture for software-defined smart homes which divides the smart home platform into three layers, these layers include smart hardware layer, a control layer, and external service layer. Each of the above-mentioned layers consists of various devices and offers a variety of functionalities and services. The smart hardware layer consists of all types of smart hardware at a smart home, this includes smart sockets, smart bands, sensors, and cameras. The control

layer provides central management services which in turn supports in physical hardware deployment at a smart home, abstraction of equipment deployed in the cloud which can also be generated through traditional intelligent devices. Furthermore, the control layer is designed to protect the hardware details of the hardware in the smart hardware layer thus perceiving and analyzing the user demands and supporting the automatic and intelligent management of smart homes. The external service layer incorporates existing home service resources thus offering high-quality and highly efficient personalized services (Xu et al., 2016). This architecture offers variety of features, these features include: (1) centralized controlled which offers global information, configuration and optimization thus enhancing on accurate and comprehensive information collection and effective management, (2) open interface which allows seamless integration between applications and networks, this, in turn, may offer additional benefits such as editable dynamic interfaces and easier access and (3) network virtualization which involves decoupling of networks from physical networks and robust fault tolerance, this, in turn, offers other benefits such as independent connections, support during partial network during (Xu et al., 2016).

Even though the above-mentioned architecture is practiced in smart homes with the variety of benefits offered, there still exists relative challenges and drawbacks. They include: security and privacy issues in terms of family: although system design and platform architecture provides accurate user requirements along with personalized customer services, to ensure the information used is not stolen and if stolen can lead to higher levels of security and safety issues for the smart home users which in turn would also lead to huge economic loss (Xu et al., 2016).

Ontology System

(Li et al., 2012) proposes an ontology system which specifies the semantic information about the devices, services, and workflows involve in a particular smart home which also allows the users to compose and recompose services based on their preferences and requirements. Furthermore, this ontology system allows the users to add and remove preferred services and devices into an ontology which is supported by an automation system such as a code generation process. This research-based home automation solution involves the practice of three steps such as abstract workflow design, construction of functions, device recovery and code generation which is supported by various computing platforms such as Java and Open Services Gateway initiative environments.

Thereby, the proposed ontology system supports rapid construction of smart homes using the framework which involves various functionalities such as discovery and composition using the current services and workflows. By proposing the ontology system, (Li et al., 2012) aims to address three challenges confronted by the smart home users, these challenges are constructed around (1) integration of devices in a single smart home in order to create a set of complex intelligent control services addressing the interoperable workflows and other relative services, (2) process of discovering devices in a smart home and (3) carrying out services and recovery process during device and service failure. In order to address these issues, ontology introduced by (Li et al., 2012) consists of four components such as a smart home knowledge base to store the knowledge about the services and devices and household profile involved in a specific smart home.

Moreover, the smart home knowledge base also consists of an ontology system which holds the semantic information and knowledge specifications and reasoning of

the smart home. Secondly, the solution consists of a service modeling and composition to provide the service specifications supported by the modeling platforms including the security policies for the ontology. Thirdly, a code generation process is included in the solution which translates the model created to code using the CodeSmith template. The code generated can also be used by other target systems such as Web services and OSGI (Open Service Gateway Initiative) services. A service deployment process is used to deploy the code generated in an execution engine. Finally, this ontology solution involves a service execution management where services can be uploaded and scheduled to run, which allows the devices to send and receive commands and messages to and from the devices of a specific smart home. During a device failure, the execution engine is designed to change the control software using other services in order to prevent the potential failures (Li et al., 2012).

However, even though the ontology models tend to answer the challenge of lack of interoperability in a smart home unit to a greater level, there still exists gaps. These gaps include the requirement of improved levels of knowledge in terms of engineering skills, lower levels of support provided in terms of time-related seasoning, ontology modeling being computationally expensive and limited ability to deal with the uncertain and dynamic context of objects (Ni, García Hernando & de la Cruz, 2015).

Even though the above-mentioned solution attempt to address the challenges such as lack of interoperability which is the main challenges this thesis aims to address, this specific solution does not address the challenge of low levels of interoperability during the integration of various appliances or devices in to a single smart home environment, this in turn leaves room for further improvement and challenge of interoperability in terms of variety of device integration unaddressed.

2.4.2 Commercial Based Solutions

Eclipse Smart Home (ESH)

Lack of interoperability and low levels of compatibility occur during the addition of a new device to the smart home or integration of various other devices to the smart home. These challenges are addressed by Eclipse Smart Home (ESH) project and Universal Remote Control (UCL) (Smirek et al., 2016). Lukas Smirek proposes a framework to support the integration of multiple technologies into a single smart home system thus ensuring that personalized user interfaces are provided to each different smart home user (Smirek et al., 2016).

Smirek also focuses on the elderly people and their need to use user-friendly interfaces and devices, therefore a separation between the physical devices and its abstraction in relation to the smart home system is carried out through a special user interface. Furthermore, ESH provides a flexible framework, which provides modules for abstraction and translation functions thus enabling use cases and interaction across protocol boundaries. This is supported by IoT platform (Smirek et al., 2016). Even though this integrated system offers open personalized user interfaces based on a resource server, this may yet result in misuse and injection of malware, thus causing security and privacy issues (Smirek et al., 2016).

This, in turn, may cause further challenges in relation to usability, reliability, and user-friendliness of the mobile apps and the smart home procedure as a whole (Bonfè, Fantuzzi & Secchi, 2013). This has caused gaps in terms of compatibility, usability between the existing solutions and the requirements of the aging community and challenges in terms of a level of interoperability when different smart homes are introduced with differed sensor-actuator configurations (Bonfè et al., 2013). The

problem of interoperability between different smart homes may occur due to its use of dynamic sensor actuator configurations (Smirek et al., 2016).

AppleHealth

As mentioned previously, IFTTT, presented by (IFTTT Inc., San Diego, CA, USA) is an action-reaction middleware which allows users to connect to web services with the support of chains of conditional statements. This particular technology enables the users to capture the changes occurring in the state of web services which would be useful when triggering their own application according to the respective changes. Likewise, AppleHealth offers an interaction solution in order to centralize the mobile health applications to support the management of health information (Vega-Barbas, Pau, Martín-Ruiz & Seoane, 2015). However, in order to use the above-mentioned technologies, the users need to have prior knowledge in regard to web environments and gain familiarity with, the web applications in order to take advantage of the services offered by these technologies (Vega-Barbas, Pau, Martín-Ruiz & Seoane, 2015).

SmartThings by Samsung

In addition to the above, SmartThings offered by (SmartThings Inc., Samsung, Washington, DC, USA) allows the end users to control and monitor their smart home from a usable mobile application (Vega-Barbas, Pau, Martín-Ruiz & Seoane, 2015). This platform also permits interconnection of various smart home elements such as furniture, doors and electrical appliances such as sensors and actuators which in turn may support automation of human tasks in a home. Additionally, there exist other relative commercial solutions such as GrandCare system and BeClose which perform tasks similar to SmartThings by Samsung, however, emphasizes mainly on elderly care (Vega-Barbas,

Pau, Martín-Ruiz & Seoane, 2015).

However, according to (Fox-Brewster, 2016), anyone relying on SmartThings critically for home security purposes may confront various vulnerabilities. Although in a smart home environment of SmartThings hub is connected to a homes' own motion sensors which act like traditional alarms and notify people when an absurd activity is detected. This system allows hackers to enter a smart home undetected. Even worse, when connected to a smart lock, (Zillner, 2015) proves that a robber can break into a smart home without the support of any brute force. Even though SmarThings have not revealed this to its users, this can lead to critical security issues.

UbiQ Scenario Control

UbiQ Scenario Control designed by (Advantech, 2009) offers a centralized home automation system which supports lighting control, home security systems, emergency warning system, door entrance intercom and other community services. This solution uses a scenario manager to support the configuration of different multiple devices. Additionally, Nokia has introduced a Home Control Centre (Hayes & Black, 2006) offering an open platform which allows third parties to integrate different smart home services and solutions. This solution consists of a control panel which allows the smart home users to monitor and control the devices. Even though the solutions are statistically defined, they do not offer the users with the functionality of personalizing the services, addition or removal of devices without the assistance of the service providers (Li et al., 2012). This, in turn, leaves gaps in relation to usability and interoperability during the use of multiple devices in a single smart home unit.

MisterHouse

MisterHouse, an open source smart automation program (Mäyrä et al., 2006) is written in Perl and involves execution of actions by time, voice and serial data. This solution offers an execution platform which supports multiple platforms. These platforms include Universal Powerline Bus, Z-Wave, and ZigBee. However, in order to practice this platform, the end users need to be expertise in Perl which as a result may hinder some of the users for developing their own control system for the smart home (Li et al., 2012). This may also pose challenges in relation to usability and user-friendliness of the system overall.

Calaos - An Open Source Automation Smart Home Project

Calaos, an open source automation smart home project involves a software stack written in C++ and consists of a server domain, a touchscreen interface, a web interface and a mobile app supporting both iOS and Android. This also comprises of the require configuration tools and the complete operating system which allows the users to create a smart home automation solution. Calaos aims to produce an entire software suite which allows the users to configure, control and monitor the smart home according to their preferences.

Furthermore, Calaos consists of a custom version of Linux designed especially for IoT which includes a server platform to centralize third-party devices and as mentioned previously, supported by mobile apps such as Android and iOS which allows the users to control the complete smart home system from a single interface thus enhancing the aspect of usability.

In addition to the above, the source code of Calaos is released under the

open source license which in turn would allow anyone to view the code and make amendments which would also help them in designing their own custom version of the software thus enhancing the aspect of personalization and in turn usability of the overall platform.

However, there exist challenges in terms of security and privacy with the code being available as open source platform which would allow anyone to hack into a smart home system. On the other hand, this open source home automation system allows any hardware developers to ensure their hardware is compatible with a particular smart home system thus assuring improved levels of interoperability during the integration of various devices into a single smart home unit.

Criteria	**Definition**
Usability	Degree to which the smart home user understands and adopts to the device. Usability may also depend on the need to have prior knowledge about the way the device needs to be used (Demiris, Oliver, Dickey, Skubic & Rantz, 2008).
Interoperability	refer to Section 2.1.3 of this chapter
Security	The ability of any user (unauthorized users) to gain access to a smart home may cause lead to security issues (Zillner, 2015).
Customizable	The ability to customize the application based on smart users' preference (Groppe & Mueller, 2005).

Table 2.2: Defining Criteria to Compare Existing Solutions

As shown in table 2.3, all the existing solutions (this includes both research-based and commercial based solutions) are illustrated in the form of a table. The existing solutions are evaluated against the most common features addressed by most of the solutions that are also directly related to the problem definition and the proposed solution of the thesis. The crosses define the how the solutions are unable to achieve those factors to a considerable level, the correct mark denotes how the solutions have been successful in achieving the respective factors and N/A implies how the solutions have not addressed those factors when building the framework or the device.

Base	Solution Name	Comparison Criteria			
		Usability	Interoperability	Security	Customizable
Research	Simple Internet of Things (SITE)	×	×	N/A	✓
	Parametric State Charts	×	✓	N/A	✓
	Proactive Arhcitecture	×	✓	N/A	N/A
	Rule Based Intelligence	×	✓	N/A	N/A
	Architecture for SDSM	✓	✓	×	✓
	Ontology System	×	×	N/A	N/A
Commercial	Eclipse Smart Home	×	✓	N/A	✓
	AppleHealth	×	N/A	✓	✓
	SmartThings	×	✓	×	✓
	Ubiq Scenario Control	×	✓	N/A	×
	MisterHouse	×	✓	N/A	N/A
	Calos	×	✓	×	N/A

Table 2.3: Comparison of Existing Solutions

Therefore, when the above table is considered, every solution fails to meet at least one of the stated factors. As the two most important factors being interoperability and usability, even though some of the solutions meet the quality of interoperability, they fail to be effective in terms of usability. Therefore, the proposed solution would mainly focus on being both usable and interoperable in smart home environments thus ensuring the research gap identified in this stage of the systematic literature review is eliminated to a greater level.

2.5 Findings

This section of the literature answers the previously formulated research questions, identifies and elaborates on the current solutions, evaluates the current solutions and identify current and existing gaps. As addressed previously, the research question broken down into three sub-questions which are answered comprehensively as shown below.

2.5.1 Factors leading to the choice of an app design model

1. **Which factors influence the need for an app design model (app design model can also be called the above-mentioned relative search terms)**?

   The use of an app design model when designing software, particularly when UML diagrams are used, various additional opportunities are offered, such as high levels of quality improvements, automated code generation, improved problem solving and design capabilities and overall productivity during the development of the software (c et al., 2016). Shorter lead-time is provided by increasing the level of abstraction, which in turn reduces the gap between the problem definition and the proposed solution with the use of app design models (c et al., 2016).

   As mentioned above, in the search terms, even though Model Driven Engineering was identified as a similar term and claim to manage complexities and provide improved levels productivity and software quality during software development, there are very little literature findings proving its improved levels of effectiveness and usability (Mohagheghi & Dehlen, 2008).

   In addition to the MDE, various another modeling of object-oriented software along with the practice of other methodologies are practiced. These methodologies include: Jackson Structure Design (JSD), Object Modelling Technique (OMT), Booch method and Object-Oriented Software Engineering (OOSE), which support structural design during software development to a greater, extend. However, these techniques may introduce to additional challenges such as; JSD is identified as a poor approach and lightly supports the approach of high-level data analysis and database design, use of pseudocode representation may introduce to further complexities in terms of usability and understandability (Loomis, Shah & Rumbaugh, n.d.). Furthermore, Booch method practices detailed description

levels of the notation, which results in, challenges such difficult to understand and the risk of information fragmentation across various diagrams, this, in turn, would result in challenges in terms of usability (Jungclaus, Wieringa, Hartel, Saake & Hartmann, 1994).

Few of the types of app design models may introduce various challenges as mentioned above, however, adoption of an app design model overall may support in achieving various functionalities during software development. These functionalities include traceability, communication, code design and generation and testing (c et al., 2016). Furthermore, software architecture and architectural designs cater to various other functionalities such as general software architecture design activities, generates generic artifacts, performed tasks and used or recommended techniques (Reyes-Delgado et al., 2016).

In addition to the above-mentioned factors, design modeling enhances on various other functionalities during the software architecture design process, they include backlog control, architectural analysis, architectural synthesis and architectural evaluation (Reyes-Delgado et al., 2016). Furthermore, as the software architecture may lead to complex procedures, an iterative process can be practiced during the design and development stages, which may provide further benefits such as improved software quality (Reyes-Delgado et al., 2016).

In addition to the facts mentioned above, even though UML diagrams are addressed as a whole previously, this part of the systematic literature review would focus on different types of UML diagrams and their relative factors highlighting the need for such app design models. As mentioned previously, UML diagrams support automatic code generation to a greater extent which in turn provides 100 percent complete source code. In addition to the above, UML diagrams used for system structures and modeling may provide high-level design details of the

system which would support the programmer to a greater level during the system implementation. These high-level details include variable declarations, initializations, pre-defined constant values, method definitions and class definitions (Viswanathan & Samuel, 2016).

When different types of UML diagrams are taken into consideration from an individual perspective, state charts are recognized to be the first candidate which is suitable for event-driven applications due to an ability to show the state changes of a particular object during its lifetime (Viswanathan & Samuel, 2016). However, state charts/ state models fail to support behavioral modeling which is provided by activity diagrams. They also support in representing the control flow of the system from one activity to another. Furthermore, activity diagrams have the ability to specify which object is responsible for which activity which is identified as a unique feature when compared with the featured offered by other existing UML and modeling diagrams (Viswanathan & Samuel, 2016). Moreover, this feature supports automatic code generation to a greater extent.

Even though activity diagrams concentrate on control flow generation, method definition is not generated by activity diagrams even if the actions inside the activity may not be specified by the activity diagrams. However, fine-tuning the activity models in order to include these details may enhance the automatic code generation procedure to a greater level (Viswanathan & Samuel, 2016).

As mentioned previously, UML state machines and state charts are most widely used to specify the dynamic behavior of reactive systems. However, generating code from state machine diagrams may lead to further complexities due to its dynamic environment which in turn may cause incompatibilities state machine specification concepts and object-oriented programming languages (Domı, Pérez,

Rubio et al., 2012). On the other hand, state charts will provide various benefits during the software development stage, this includes software maintenance, modularity and re-usability (Domı et al., 2012).

In addition to the above, various other types of UML diagrams such as class diagrams and sequence diagrams are used during the software design phase. Sequence diagrams support in significantly improving the comprehension of software requirements which in turn may help in generating better comprehend functional diagrams. However, during source code comprehension and code generation, sequence diagrams as UML diagrams generated requirement engineering process may abstract the problem definition thus neglecting the design and implementation details during the development stage (Gravino, Scanniello & Tortora, 2015). Moreover, UML class diagrams emphasize on source code comprehension (Gravino et al., 2015).

The above-mentioned factors prove the need for an app design model during the design and development of an application despite few relative challenges, which can vary, with the choice of the type of app design model.

2. **How do challenges such as lack of interoperability incur during the practice of various smart homes?**

   Design and creation of smart homes have introduced the problem of lack of interoperability (Capitanelli et al., 2014). Lack of interoperability or in other words low levels of consistency and compatibility may occur during the incorporation of various devices into a single a smart home. These complexities may occur due to various reasons such as integration of different models of different smart devices, different devices.

   Even though numerous smart homes are designed which involves the incorporation of strongly technology-oriented devices, lack of interoperability during the integration of devices would lead to further challenges such as neglected user satisfaction and benefit analysis (Capitanelli et al., 2014). In addition to the above, during the smart home scenarios, lack of interoperability, higher levels of consistency and compatibility may occur when different devices are incorporated to provide a customized system design to match various users' needs and habits (Capitanelli et al., 2014).

   As mentioned previously, lack of interoperability is regarded as one of the main issues during the design and development of a generic smart home, which has led to further costs during the integration of various smart home devices (Stojkoska & Trivodaliev, 2017). Various products such as Z-Wave and ZigBee are designed, which includes product properties which require for different vendors to build interoperable devices to support home automation and healthcare. The above-mentioned devices are designed to ensure easy integration and higher levels of interoperability (Stojkoska & Trivodaliev, 2017).

   IoT (Internet of Things) architecture with the support of wide range of devices

and software apps allows to carry out multiple functionalities such as home automation, remote monitoring, and healthcare. When the focused area which is mobile health applications are taken into consideration to support run rehabilitation system, various devices are integrated to add to a smart home in order to run various applications preferred by the user or patient. These applications include rehabilitation plans, fall detection plan, weight management plan, diabetes management app and medication reminder app (Sinha et al., 2017). Thereby, due to the users need to use various applications which are provided by various devices may lead to the need for the integration of existing devices or addition of new devices to a single smart home. This, in turn, may lead to lack of interoperability.

As stated previously, various devices are integrated to address different health issues in a smart home when catering to the aging community and healthcare scenarios. Every device consists of its own sensor actuator configuration and when integrated into a single smart home may lead to challenges in terms of interoperability and compatibility.

In addition to the above, various other technical challenges are introduced in terms of interoperability while attempting to integrate health and other types of data and information which is used or required by various devices in a single smart home (Gay & Leijdekkers, 2015).

Therefore, although various solutions such as the use of mobile applications are practiced to achieve low levels of interoperability during smart home scenarios, this challenge is yet confronted as per the above-mentioned facts.

3. **How does the introduction or use of a specific app design model ensure high level of interoperability in dynamic sensor actuator scenarios?**

   Various types of visual designs and app design models are used to achieve high levels of interoperability during varied dynamic sensor actuator configurations in smart home scenarios. This has yet led to further challenges such as low levels of interoperability and the need to write a new piece of code for every new installation or a change of an existing device (Sinha et al., 2017).

   In order to overcome these challenges, app design models and UML designs such as dynamic modeling (statecharts) are introduced to present an appropriate a formal design to design an app which in turn may support in generating the code automatically. This may eliminate the need to write the code repeatedly every time a new device is added or integrated. Furthermore, the chosen app design model would be compatible in other application areas apart from smart home and health-care scenarios where the designed software may achieve higher levels of flexibility in order to interact with dynamic sensor actuator configurations. This, in turn, may support in achieving improved levels of compatibility and interoperability (Sinha et al., 2017).

   In addition to the above facts, a practice of design model apps may cater various other benefits in relation to the health apps and healthcare scenarios. These benefits include: formulation of an app design model may capture the inherent flexibility of a particular software that is able to interact with dynamic sensor actuator configurations in IoT systems, an existing app design model which is built to perform a specific function, for instance app design model designed for fall detection mechanism can also be reused to support other similar functionalities such as alert mechanism provided that dynamic sensor actuator configurations are compatible (Sinha et al., 2017).

When the above-mentioned facts are taken into consideration, the practice of a relevant app design model supports in code reusability and app design reusability thus addressing the previously mentioned challenges, such as lack of flexibility, inconsistency, interoperability, and incompatibility.

During the choice of an app design model, various criteria should be accomplished before the code generation, the criteria include: should be easy to use, higher levels of understandability by any domain expert in terms of the language used. Also, the chosen app design model should be highly formalized in order to support the translation to a mobile code without having the need to obtain to manual effort (Sinha et al., 2017).

Even though the choice of an app design model for IoT systems such as smart homes is critical, this may provide further relative benefits. These benefits include: the modeling language supports to develop and design workflows for multiple systems (Sinha et al., 2017).

2.5.2 Evaluation of Current Solutions

This section of the literature review answers the second research question, which is: **What are the existing solutions and how do they meet the factors identified in Question 01?**

This will be answered in form of a table as shown below. As identified previously, various relative existing solutions exist in terms of the development of IoT deployment devices for smart homes. These identified solutions will be assessed in order to acknowledge the relative factors which were identified in research question 01. The existing solutions and their related factors which lead to the choice of an app design model due to challenges such as lack of interoperability are shown in the table below.

Factors	Definition
Quality Improvements	Quality of app design models are determined with consistency between various app design models where they represent the same system but from different viewpoints. Thereby, change in one app design model should not affect the existing underlying models (Bashir, Lee, Khan, Chang & Farid, 2016).
Automated Code generation	This involves the translation of implemented specific details, which is carried out with the support of structural modeling, a combination of multiple models and other relative algorithms (Viswanathan & Samuel, 2016).
Improved problem solving	Involves generation and analysis of how processes can be redesigned, this may also involve visual representations (Figl & Recker, 2016).
Improved levels of usability	Usability ensure better levels of performance and flexibility during app modeling (Gravino et al., 2015).
Traceability	Manages and maintains the transformation of models and requirement exchange and corresponding changes of the models (Gravino et al., 2015)
Software architecture design activities	Adds quality to software development procedures, provides artifacts to support software development process, provide support in various other areas such as activities, tools and provide an elaboration of software systems (Reyes-Delgado et al., 2016).
Task performance	Aims to achieve an acceptable level of reliability with the use of various app design models (Gu et al., 2012).
Backlog control	Keeps a track of the requirements and informs about the changes required to be made during various stages of the software design and development (Jacobson, Spence & Kerr, 2016).
Architectural analysis, synthesis and evaluation	Aims at identifying the critical components of the architecture, focuses on automation of UML models and evaluation determines a reliability of software at design level (Ayav & Sözer, 2016).
Behavioral modeling	Represents dynamic aspects of software during the development stage (Bashir et al., 2016).
Representation of the control flow of the system	Ability to support both dynamic and static aspects thus enhancing system performance and reducing system complexities ((Xie, Liu, Hu, Yang & Fu, 2015)
Specification of which object is responsible for which activity	Ensures better levels of clarity and under stability of each activity and its effect on other relative activities (Bashir et al., 2016)
Software maintenance, modularity, and reusability	Enhances on modifications and changeability during software design and development (Fernandez-Saez, Genero, Chaudron, Caivano & Ramos, 2015)

Table 2.4: Selection Criteria of App Design Model

Factors	Class	Sequence	Activity	StateChart
Quality Improvements	✓	✓	✓	✓
Automated Code generation	✓	✓	✓	✓
Improved problem solving	✓	✓	✓	✓
Improved levels of usability				✓
Traceability	✓	✓	✓	✓
Software architecture design activities	✓		✓	
Task performance	✓		✓	
Backlog control				
Architectural analysis, synthesis and evaluation				
Behavioral modelling				✓
Representation of the control flow of the system			✓	
Specification of which object is responsible for which activity			✓	
Software maintenance, modularity and re usability				✓

Table 2.5: Evaluation and Selection of App Design Models

Table 2.5 evaluates the UML app design models/diagrams based on the factors identified in Table 2.4. As per this table, the most suitable app design model is the UML activity diagram as it details the flow of activities (the behavioral aspect) of the smart home systems and any other systems, which is later generated to runnable code automatically. Hence, even though various other app design models are discussed in Section 2.5.1, prior to the evaluation shown in Table 2.5, a very comprehensive evaluation was carried taking all the types of app design models in to consideration,out of which the most suitable 5 were chosen in order to choose the app design model which supports both the behavioral and the ability to be generated to executable code aspects.

2.5.3 Current / Existing Gaps

This part of the literature review address the existing gaps in terms of smart homes, app design models and other relative areas of focus mentioned above.

Interoperability is the key to the open markets which supports in achieving competitive solutions in IoT. However, as mentioned previously, lack of interoperability in generic smart home solutions is associated with higher costs due to the integration of different smart home devices (Stojkoska & Trivodaliev, 2017). Even though Z-Wave with the alliance of ZigBee has produced products to be interoperable with various other devices and products by different vendors. These systems include public application profiles catering to various areas such as Home Automation Systems, healthcare, and remote control. However, these products, appliances, and devices still remain to be interoperable across profiles and areas (Stojkoska & Trivodaliev, 2017) thus leaving the challenge of interoperability unaddressed to a certain level.

Lack of security and privacy is regarded as one of the most significant challenges confronted by smart grid development when in relation to cybersecurity. The smart grid is regarded to be a targeted by the cyber-terrorists which in turn makes a security and privacy a critical concern (Stojkoska & Trivodaliev, 2017). IoT is inherently vulnerable to security threats and attacks during wireless networking scenarios. Furthermore, IoT requires an additional security policy to ensure improved levels of security which in turn would lead to further costs (Stojkoska & Trivodaliev, 2017). In addition, during the practice and implementation of security and privacy may require additional need to provide authenticity (to ensure the device is not malicious), integrity (received data is identical to the data transmitted) and confidentiality (ensure the data is not readable by unauthorized readers) (Schneps-Schneppe, Maximenko, Namiot & Malov, 2012).

Furthermore, the practice of IoT involves generation of data in big amounts which are often considered real time in nature may lead to uncertainties in terms of provenance (Stojkoska & Trivodaliev, 2017). The challenge of handling big data is regarded to be critical as the overall performance is directly influenced by properties of

data management services (Dobre & Xhafa, 2014). Even though various applications are generated to enhance data integration, data storage, and data presentation which in turn would support the overall analytical pipeline. However, the challenge of managing big data in the cloud still exists in terms of data integrity, due to its negative impact on the quality of data and challenges in terms of security and privacy of outsourced (Stojkoska & Trivodaliev, 2017).

There exist challenges in terms of networking protocols when implemented in IoT solutions. Smart home networking protocols are expected to adopt the existing Wireless Sensor Networks and Machine-to-Machine communications. However, this installation and implementation procedure involves the inclusion of advanced features which may lead to further challenges such as increased costs, decreased levels of ease of use and design of an appealing protocol to achieve both costs and performance may be a trivial task (Chen, Wan & Li, 2012).

Thereby, when the above facts are taken into consideration, challenges in addition to lack of interoperability in relation to smart home scenarios exist. These existing gaps include security and privacy issues, issues in managing bid data and challenges in relation to networking protocols.

2.6 Conclusion

This chapter of the thesis provides a background which identifies the relative keywords, i.e.: smart homes, app design models, lack of interoperability and dynamic sensor actuator configuration for which specific definitions are provided for each of these keywords. In addition, the process of the systematic literature review is emphasized, whereby every stage of the process (i.e.: scoping, planning, searching, and screening)

is illustrated in terms of the studies and research carried in relation to the smart home scenarios.

This section of the thesis also emphasizes the current state of the art which is sub-sectioned into research-based and commercial solutions. The solutions included in this section covers various areas such as the degree of usability in relation to home automation and monitoring systems, the support provided to smart homes by the proactive architecture of IoT systems and rule-based intelligence enhancing on home automation systems are few of the existing findings and solutions. Furthermore, the research questions formulated and modularized during the systematic literature review are answered here during the findings section this chapter. As mentioned previously, the research questions answered address the existing solutions and relative factors and evaluations are carried out between the existing solutions and the factors identified based on the factors leading to the choice of an app design model.

The following chapter of the thesis presents the research methodology used, this section emphasizes on various areas such as; overview of the literature methodology used, illustrations on architectural designs and prototype methodologies practiced. Furthermore, validation methods practiced to during the validation of both theoretical and expected results are enhanced.

Chapter 3

Research Method

After the validation of various research methodologies, Design Science Research Methodology is chosen as the approach to be followed to design, develop and This chapter provides a plan to solve the research questions formulated in the previous Chapter 2.3.1. Furthermore, this chapter involves the selection of a suitable research methodology with the support of comparison studies carried out between qualitative and quantitative research methods along with the consideration of their respective advantages and disadvantages. Additionally, this chapter also evaluates and examines various other relative research methodologies prior to the selection of a specific research methodology.

After the validation of various research methodologies, Design Science Research Methodology is chosen as the approach to be followed to design, develop and evaluate the Automatic Translation Tool. This methodology consists of six phases, they include problem definition, the definition of objectives and solution, design and development, demonstration, evaluation, and communication, which are addressed in detail. Moreover, methodologies used to evaluate the tool are addressed in detail with success criteria for each methodology. Finally, this chapter concludes with a summary of every section included in this chapter.

3.1 Selection of a suitable Research Methodology

Researchers carried out both quantitative and qualitative research methodologies based on the data they intend to collect during the research. Quantitative studies are carried out in order to examine the relationship among the identified variables, describing the trends, attitudes, and ideas of the population the research is being carried out on (Yates & Leggett, 2016). In addition, the practice of quantitative approach may enhance the degree of precision in terms of research which in turn represents a 2-dimensional view of the findings thus enhancing the comprehensiveness of the data collected through the qualitative approach (Yates & Leggett, 2016).

On the other hand, quantitative research approach explores the how and why of the case study or a specific story the research is based on. Furthermore, this particular research approach focuses on various other factors such as the sample size, data collection, analysis and interpretation methods thus supporting the researcher when planning the future studies(Yates & Leggett, 2016).

However, there exist challenges with the practice of both the research approaches mentioned above, even if research is carried out with both approaches combined. (Jick, 1979) proposes a triangulation methodology which involves the combination of multiple research methodologies which belongs to the same area of the phenomenon. Moreover, this specific approach provides multiple viewpoints and higher levels of accuracy in terms of judgments made during the research and data collection (Jick, 1979). Since the early 1980s, underlying presuppositions of qualitative and quantitative research strategies have led to higher levels of arguments due to one of the research approaches attempting to emerge superior to the other research approach. However, (Newman & Benz, 1998) rejects this debate and believes that these two research methodologies are neither mutually exclusive nor interchangeable due to their

difference in relation to the point of time at which they are invoked by a researcher.

3.1.1 Other Methodologies

Various research methodologies are approaches are proposed and practiced by various authors and researchers during the evaluation and validation procedures of the UML translation tool. The research methodologies and approaches that are not suitable for the validation of the above proposed UML translation tool are eliminated and therefore identified as alternative research methodologies.

1. **Case Study Research** - This approach may include various units of analysis such as the case study of a particular organization. Furthermore, this specific research approach ensures the investigation of a particular contemporary phenomenon based on real-life context and focuses on boundaries between phenomenon and context which are not clearly evident (Lillis, 2008). Additionally, the case study research approach can be positive, imperative and critical based on the underlying philosophical assumptions of the researcher (Lillis, 2008).

2. **Data Driven Research Article Methodology** - This research approach, which is considered as one of the most commonly used methodologies involves the inclusion of discourse analysis, case studies and focus on discussion groups and emphasize on error analysis procedures. (Lillis, 2008).

3. **Ethnography** -This approach is concerned with collection and analysis of empirical data (this includes a range of other various sources such as observations and relative informal conversations) extracted from the real world contexts thus avoiding the experimental conditions provided by the existing researchers (Lillis, 2008) Additionally, during the practice Ethnography as a research methodology,

the analysis of data involves the interpretation of meaning and functions of human behavior and actions carried out during verbal communications and explanations (Lillis, 2008).

4. **Action Research** - Action research approach aims to contribute to both practical concerns of people which are involved with an immediate area of problem and the main goals of the area of research which is also based on the mutually accepted frameworks (Myers, 1997).

5. **Grounded Theory** - This particular research methodology enhances various theories which are grounded in terms of data systematically gathered and analyzed. Martin and Turner (1986) propose a grounded theory which is considered as "inductive" due to its ability to permit the researcher to develop a theoretical account based in general features of a specific topic whereas the general grounded theory involves grounding of the account in relation to the empirical data and observations. However, when in comparison with every other research methodology, the grounded theory proposes the continuous interplay among data collection and data analysis (Myers, 1997).

6. **Narrative and Metaphor Analysis** -Narrative is defined as a story, tale or a recital of facts conveyed by an individual. The narratives are of different types, these types include oral and historical narratives. Metaphor involves the name, descriptive term or a phrase of an application which is not directly related to the application (Myers, 1997).

7. **Hermeneutic Approach** - This approach is treated as both an underlying philosophy and a particular mode of analysis, whereby human understandings are emphasized thus focusing on the philosophical perspective which is also considered to be a research methodology. In addition, as a mode of analysis, this

specific research methodology ensures the way in which the textual data is understood (Myers, 1997).Furthermore, the hermeneutic approach is involved with the meaning of the text or text analog (i.e.: an organization can be regarded as a text analog) which is understood by the researcher with the support of oral or written text (Myers, 1997).

3.1.2 Chosen Research Approach

When the above-indicated research methodologies and approaches which belong to both qualitative and qualitative research areas, even though they support the research procedure in various ways, they may yet introduce to various challenges and weaknesses as identified and discussed in this section of the chapter. Additionally, this section identifies and analyses the selected research methodology that would be practiced during design and evaluation phases of the architecture of UML translation tool.

When quantitative research approach is taken in to account as a whole, various challenges exist, they include: the researchers categories and theories not consisting of the ability to reflect on understanding of the local constituents', higher possibility of the researcher missing out the phenomenon occurring as a result of improved levels of focused provided to theory of hypothesis during the testing phase of the research process. Moreover, the knowledge generated with the support of this specific research methodology may be too general and abstract for a direct application to a certain context, situation or an individual (R. B. Johnson & Onwuegbuzie, 2004).

Furthermore, there exists challenges and weaknesses in terms of qualitative research, they include difficulty in making quantitative predictions, complications during the testing phases of hypothesis and theories. Moreover, data analysis is considered to be time-consuming and the result obtained from the practice of qualitative

research methodology tends to be easily influenced by researcher's personal biases and idiosyncrasies (R. B. Johnson & Onwuegbuzie, 2004).

Design Science Research Methodology (DSRM) introduced by engineering as propose by (Hoffman, Roesler & Moon, 2004) raises the need for concentration on a common design science research methodology (Kao et al., 2016). Furthermore, (Archer, 1984) proposes a type of design science methodology which focuses on building systems instantiations based on the research outcomes which also concentrates on formulating a purposeful solution to the problem definition identified from the design theory as proposed by McPhee (1996) (Kao et al., 2016). When practicing design science in terms of a research methodology in order to introduce a purpose-oriented design for hospital beds and a specific mechanism in order to eliminate fire doors from being propped open, (Archer, 1984) practices a specific design that could be codified thus supporting the creative aspects and industrial engineering research outcomes and providing reflections based on different views in terms of the research methodology practiced (Kao et al., 2016).

As per (Fischer, Greiff & Funke, 2012) and (Von Alan, March, Park, Ram, 2004) design science research methodology tends to be an effective methodology in terms of developing ways to solve complex issues with the support of artifacts which in turn would enhance the identification, design, and development of the solution. Moreover, the above-mentioned authors believe that various additional phases should be taken in taken into consideration, these phases include: identifying, acquiring and utilizing the knowledge in order to find the relative solutions for the problem definition (Von Alan et al., 2004).

In addition to the design science approach as a research methodology, (Genemo, Miah, McAndrew, 2016) proposes a marking assessment methodology which identifies

the assessment process manually thus observing the results obtained from the assessment which would help in gaining a better understanding of the phenomena which is later investigated in terms of the domain knowledge and previous literature and evidence carried out by the expertise. This specific methodology is recommended by the above-mentioned authors when understanding the problem domain whereby multiple relative solutions are presented which can be helpful to a greater level during the design and development of artifact prototypes.

Besides, design science caters to different types artifacts, they include constructs, models, methods and implementations which in turn enhances the performance of goal-directed activities (March & Smith, 1995). In addition to the above-mentioned facts in relation to design science approach, rather than concentrating on basic theoretical knowledge, design science research methodology involves the application of knowledge and tasks in terms of relative tasks and circumstances in order to produce effective and useful artifacts (March & Smith, 1995).

When the above-mentioned research approach, i.e. Design science research methodology, it involves the practice, creation, and evaluation of rigorous IT design artifacts which in turn would support in solving problems identified in relation to the case study. Thereby, this particular research approach is practiced during the design and evaluation phases during modeling of the UML translation tool.

3.1.3 Systematic Literature Review

As mentioned in the previous chapter, SLR was used as the technique for a literature review. This methodology was supportive in various ways, such as formulation and modularization of research questions based on the keywords identified, practice the of inclusion-exclusion criteria with the use of PRISMA which in turn helped in clarifying

and filtering the readings and literature findings collected. Furthermore, use of Boolean operators during the search of existing literature findings and evidence enhanced the search process thus permitting to identify readings with higher levels of relevance. In addition to the above, the systematic literature review identified the current state of as shown in Chapter 02, which were evaluated against the factors identified through the first research question formulated. This chapter overall identifies the relative existing gaps in addition to the illustrations in relation to the studies carried out in terms of existing literature evidence and findings.

3.1.4 Design Methodology

As mentioned previously, Design Science Research Methodology (DSRM) would be practiced as the research approach during design and the evaluation of the architecture for the UML translation tool. This specific research approach involves a rigorous procedure which may support in designing artefacts to solve the focused problem definition which is identified in the refer to Chapter 2 Section 2.1, in order to make a contribution to the research process, to evaluate the designs and ultimately to communicate the results to the intended audience (Peffers, Tuunanen, Rothenberger & Chatterjee, 2007). These artifacts may include the models, methods and instantiations, social innovations, properties of technical, social and informational sources which includes the definitions of any objects designed with involves an embedded solution supporting in understanding the research problem identified (Peffers et al., 2007).

Design Framework

When the design framework is taken in to, the Design Science Research Methodology (DSRM) emphasizes on a set of rules and guidelines which in turn would support

in the modeling and identification of relative tasks and activities thus enhancing the architecture of UML translation tool. This procedure will also be supported by the generation of iterative prototypes 0.thus ensuring improved levels of refinement during the design and evaluation phases of the architecture.

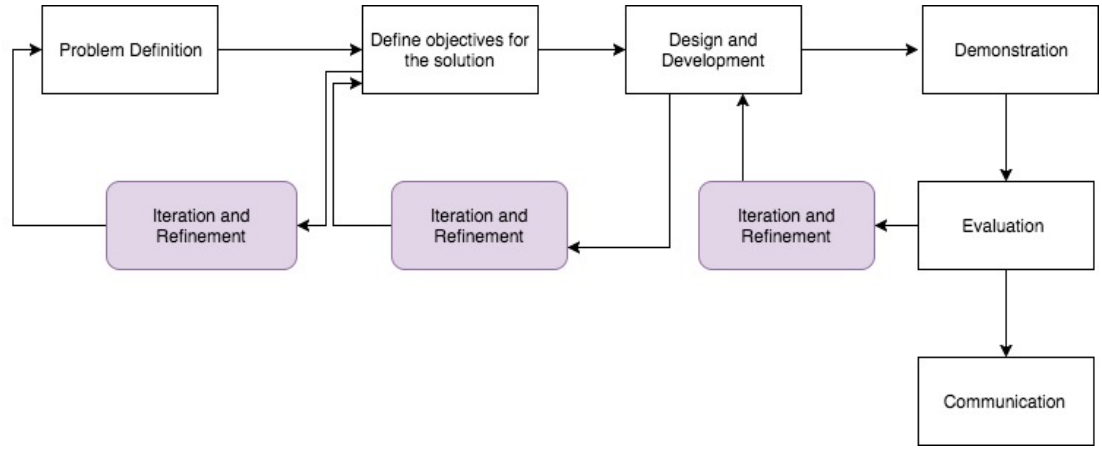

Figure 3.1: Design Science Research Methodology

The diagram shown above indicates the stages, phases or protocols that will be followed during the design science research methodology. The guidelines as identified above will be defined in this section of the chapter which will additionally address in terms of the architecture of the UML translation tool.

1. **Protocol 01: Problem Identification** : (Peffers et al., 2007) suggests the identification of the problem definition during the initial phase of the Design Science Research Methodology (DSRM) due to various reasons such as: identifying and defining the problem area may support in designing and developing relative artefacts which in turn would effectively provide a solution which may be useful during the atomization of the problem conceptually thus providing a solution that can capture relative complexities effectively. This phase may involve the achievement of two additional factors: they include motivation provided to the researcher, audience, and stakeholders in relation to the research. Furthermore,

this phase would also provide sufficient reasons, results, knowledge and required resources which would support the researcher and the audience in understanding the state of the problem and importance of the solution. Therefore, as mentioned previously, the problem definition lies in the area of interoperability in relation to the practice of various smart devices in a specific smart home with varied sensor actuator configurations. Furthermore, the problem definition is extended to lack of user-friendliness, understandability, and usability in relation to the current interfaces introduced and practiced by various individuals, researchers and organizations.

2. **Protocol 02: Definition of Objectives**

   As identified and mentioned in figure 01, the second phase involves the identification and definition of objectives which in turn addresses the problem definition as mentioned in section 5.2.1 of this chapter. Furthermore, this phase of the design science of the research methodology procedure acknowledges the feasible and possible solutions addressing the problem definition (Peffers et al., 2007). In addition to the above-mentioned facts, design science research methodology emphasizes on a cross-disciplinary approach which enhances both design and development activities which are further subdivided into discreet activities thus highlighting on an iterative procedure (Von Alan et al., 2004). When the proposed UML translation tool is taken into account, the proposed tool addresses the problem definition and research questions identified in chapter 02 of this thesis. This specific tool is designed to achieve various objectives which in turn would address the problem definition to a greater extent when compared with the existing literature and findings. The UML translation tool is designed and developed to achieve various objectives as mentioned below.

   - Address the challenge of lack of interoperability during the integration of

multiple smart home devices with varied sensor actuator configurations into a single smart home.

- Achieve improved levels of usability, user-friendliness, and understandability in terms of the user interface of the proposed UML translation tool as the proposed tool is designed and developed to support the elderly people.
- In addition to the above-identified objectives, quality attributes will be addressed by the proposed solution as a part of the objective achievement, these quality attributes will be generated in the following chapter of this thesis.

3. **Protocol 03: Design and Development of Artefacts**

   This protocol involves the design and development of artifacts which as mentioned previous can potentially be constructs, methods, models or instantiations. This can also be "a new property which belongs to technical, social or informational resources" (Järvinen, 2007). Furthermore, the design artefacts can be designed in a way to highlight on relative research contributions carried out whereby the research is embedded to the artifact designed, this specific feature enables the determination of desired functionalities of the artifacts designed and its respective architecture during the design and development of the actual architecture (Järvinen, 2007). Additionally, the resources involved in the design of the artifact assures the concentration on the objectives identified and defined in the previous phase of the design science approach. This, as a result, may provide further knowledge in terms of theory which in turn may support during the generation of the proposed solution. As shown figures 01, an iterative and refine procedure shown between the definition of objectives and design and development of artefacts (protocol 02 and protocol 03) thus ensuring the artefacts designed and developed to support the architecture of the UML translation tool thus mapping back to the

identified objective in the previous phase in order to ensure the solution generated meets the initial objectives derived with the support of existing literature evidence and findings.

4. **Protocol 04: Demonstration**

   This phase involves the practice and demonstration of the artifacts designed during the design and development phase of the design science approach. This particular demonstration protocol may involve various additional methods such as experimentations, simulations, case studies proof and other relative activities. Moreover, various additional resources are required to support the demonstration of the designed functionality, this includes further knowledge in terms of how the artifacts are designed to solve the problems identified and designed during the initial phase of the design science methodology (Peffers et al., 2007).

   Demonstration phase which is also known as a building functionality, in fact, involves the practice of instantiations whereby, (Newell and Simon, 1972) emphasizes on regards, this specific overall procedure as an "empirical discipline". This is mainly due to its ability to provide working artifacts which in turn enhances the further required advancements, iterations and refinements based on the results obtained from the demonstrations carried out.

5. **Protocol 05: Evaluation** This measures how well the designed artifacts support the solution proposed to address the problem definition. Besides, this protocol involves a comparison study between the defined objectives and actual results obtained from the artifacts demonstrated during the demonstration phase of the design science research approach (Peffers et al., 2007) practiced to architect the UML translation tool.

   This process may require additional knowledge in terms of relative metrics and

analysis techniques (Peffers et al., 2007). This phase also relies on the nature of the problem and relative artifacts generated which as a result may cause the evaluation procedure take multiple forms, this includes: comparisons of functionalities produces by the designed artifacts with the objectives identified in previous stages of the followed design science approach to architect the UML translation tool. Towards the final stages of this research methodology, the iteration and refinement procedure is implemented and practiced in order to enhance the artifacts designed in terms of effectiveness and feasibility (Peffers et al., 2007).

Furthermore, in terms of evaluating the artifacts and prototypes generated to support the architecture of the UML translation tool will be supported with the further documentation which will be included in chapter 04 of this thesis. This includes emphasizing on functional and non-functional requirements which also comprises of generation and identification of relative quality attributes. Thereby, during this phase of the design science research methodology, generated prototypes and artifacts will be taken into consideration and iterated back to prior protocols to ensure improved levels of refinement and ensure each object defined is achieved.

6. **Protocol 06: Communication of Research**

This protocol involves the communication of the problem and its relative importance, the artifact and prototypes generated, its relative utility and novelty in terms of the research methodology practiced and rigor of the design carried out (Peffers et al., 2007). In terms of communicating the research carried out during the design and development of the UML translation tool, various fresh ideas and facts will be published and directed in terms of every aspect considered throughout this thesis, these include the problem identification, research methodology practiced,

architectural design and development practices.

3.2 System Design Methodology

Due to the growth in complexity of embedded products and their need to be developed rapidly, the practice of a single system design methodology may no longer be adequate (Sangiovanni-Vincentelli & Martin, 2001).Therefore a hybrid methodology is used to design the architecture and develop the automatic translation tool. As discussed earlier, in addition to the literature review, this thesis consists of other phases such as the architecture creation and design and development of the proposed automatic translation tool. Different methodologies are followed by these phases in order to achieve the system requirements and ultimately solve the problem definition.

(Rumbaugh, Blaha, Premerlani, Eddy & Lorenson, 1991) asserts that the use of Object Orient Modeling (OOM) methodology during the design and the development of a modeling system promotes a better understanding of systems requirements, cleaner designs, and maintainable systems. Even though OOM may support modeling aspects and object-oriented programming to an increased level, this aspect does not emphasize on model transformation perspective which is a significant area in this tool.

Numerous methodologies are designed for architecture creations of systems which are used to generate the system requirements. These methodologies include requirement engineering (RE), Architectural Pattern (AP), System Architecture and System Design. Most of the software architecture methodologies such as RE requires the gathering of requirements from stakeholders (Mazón, Pardillo & Trujillo, 2007). Since this thesis does not involve the participation of external parties (stakeholders), phases such as requirements gathering cannot be successfully completed. As a result,

other methodologies suitable for the architecture creation, design and development of the automatic translation tool are chosen to define and achieve the system requirements and develop the tool methodically.

3.2.1 Architecture Creation

Attribute Driven Design by SEI (Software Engineering Institute)(Pesante, 2003) methodology is used to create the architecture of the automatic translation tool. This approach requires three significant components such as the primary functional requirements, quality attribute requirements, and other architectural drivers. These drivers defined and detailed in Section 4.1.1 in Chapter 4. This driver is used to generate a well-defined architecture for the automatic translation tool with the support of various other aspects such as software quality attribute scenarios (QAS), architectural tactics and patterns (Wojcik et al., 2006) in order to ensure the architecture of the proposed tool meets the architectural drivers which are derived from the research questions.

3.2.2 Tool Design and Development

As discussed earlier, the system requirements are formulated and an architecture is designed to prove how the requirements are achieved. This is carried out using Attribute-Driven Design (refer section 4.1 in chapter 4) and Model Driven Architecture (MDA) (refer section 4.2.3 in chapter 4). A tool is designed to meet these requirements with the support of Model Driven Engineering (MDE) which supports the use of multiple technology spaces (TC) during the development of a system (Favre, 2004).

When the tool is taken into account, the two most outstanding functions

involve the model transformation and code generation from behavioral models. Even-though, MDE supports model transformation to a greater level, when MDE is used for code generation it is regarded to introduce new risks to the generated program (Klein, Levinson & Marchetti, 2015). On the other hand, use of MDE to build tools that generate code automatically can be more efficient and can lead to a shorter development cycle when in comparison with the use of the traditional approach.

Therefore, Model Driven Engineering approach is used to design the automatic translation tool due to support given in terms of both model transformation and code generation procedures. This, in turn, helps the model transformation from activity diagram model to a code model which is then transformed to an executable code thus meeting both model transformation (refer section 5.4 in chapter 5) and automatic code generation aspects of the tool (refer section 5.5 in chapter 5).

3.3 System Evaluation Methodology

Various evaluation techniques are used to evaluate different types of software and tools. However, as per Section 3.2.2, the automatic translation tool is of model-driven engineering nature and may require an evaluation criterion that evaluates tools built based on the MDE.

(Klein et al., 2015) asserts evaluation criteria for MDE tool and this is based on three areas. The areas include (1)product engineering risk area, (2) development environment risk area and (3) program constraints risk area. Product engineering area evaluates the activities that were used to create the system to achieve the system requirements (refer section 4.1.1 in chapter 4).Development environment emphasizes on the risks related to the development and management of the system and finally, the

program constraints address the arising from drivers external to the tool.

Furthermore, (Mohagheghi, 2010) also proposes evaluation criteria for tools created based on MDE, this criterion is known as *Methodology-Practices-Promises -Metrics* and evaluates the tool based on understandability, ease of use, compatibility with other tools. Additionally, this evaluation criteria also evaluate modeling and meta-modeling frameworks and generation of artifacts from models and meta-models.

3.3.1 Tool Validation Methodology

As mentioned earlier, the proposed automatic translation tool designed based on MDE comprises of both model transformation aspects and automatic code generation feature. Evaluation criteria for both of these features need to be defined in order to carry out a complete evaluation process.

3.3.2 Model Validation Methodology

Modeling feature of the tool involves various functionalities such as meta-modeling, modeling, generating model instances from meta-models, generation of artifacts from models and meta-models, model-to-model transformations (refer section 4.2 in chapter 4) and model-tot-text transformations.

In order to evaluate this aspect of the tool, Methodology-Practices-Promises -Metrics evaluation methodology proposed by (Mohagheghi, 2010). This criterion addressed the various perspectives. They are as defined below.

1. Use of modeling frameworks based on meta-models that are generated to define the architecture of different models based on different views. This is supported

by various other criteria such as the ability to be applied to suitable scenarios and check for necessary concepts in relation to modeling and modeling dependencies between different views.

2. Ensuring improved levels of model performance. This is ensured by the ability to integrate the performing models with the testing tools, ability to achieve interoperability via models generated based on relevant case studies and the ability to model the listed case studies in Chapter 2.

3. Ensure greater levels of efficiency during the model transformation process by. This is determined by the time took to write a transformation rule in the case of a comparison with another tool and the time taken to generate the target model from a source model with the support of transformation rules and queries.

3.3.3 Automatic Code Generation Methodology

In order to evaluate the code generation and requirement generation and achievement aspects of the tool, previously stated evaluation method by (Klein et al., 2015) is followed. As per Section 3.3, this methodology consists of three areas. This section emphasizes on how each section addresses the evaluation of the tool in relation to the above-stated aspects.

1. Project Engineering: this involves evaluation of various areas. (1) Evaluation of system requirements by ensuring its completeness, clarity, validity, feasibility, and stability. (2) Evaluation of system design via testing its functionalities, performance, and testability.(3) Code generated needs to feasible, the tool needs to generated code that exposes internal state of the software, the tool should be able to generate code that will execute in current and any other target environment,

the tool designed and the code generated should be compatible with other integrated environments, the code generated needs to be correct and robust. (4) The generated code needs to meet quality attribute requirements such as reliability, maintainability and other quality attribute requirements identified in Section 4.1.1 of Chapter 4.

2. Development Environment: ensures the tool is able to generate code repeatedly and the code generation tool is available for the lifetime of the system.

3. Program Constraints: ensures improved productivity in terms of code generation.

3.4 Methodology Overview

This section addresses methodologies followed by each phase (refer section 3.1 of this chapter). The initial phase, which is the problem definition is detailed in a systematic way in Literature Review (2). The objectives are identified as architectural drivers which are used as the input for the design of the architecture of the proposed tool. This is carried out using the Attribute-Driven Design (refer section 4.1 of chapter 4). The Automatic Translation Tool is then designed and developed with the support of Model Driven Engineering (MDE) where the most important components of the tool are emphasized. Finally, the developed tool is evaluated using two different evaluation criteria, the Methodology-Practices-Promises-Metrics is used to evaluate the modeling component and the Product Risk Taxonomy for the automatic code generation component of the tool. Overview of the methodologies used is expressed in the form of a table below.

Phase in Design Science Research Methodology (Refer to Figure 3.1)	Reference to Chapters	Methodology	Methodology Overview	
Problem Definition	Literature Review Chapter 2	Systematic Literature Review	Scoping	Searching
			Planning	Screening
Define Objectives for Solution	Architecture Selection Chapter 4	Attribute Driven Design (ADD) Model Driven Architecture (MDA)	Input Architectural Drivers	
			Architectural Drivers	
			Represent Architecture Design via 4+1 Views	
Design and Development	Design & Development Chapter 5	Model Driven Engineering (MDE)	Modeling of Apps	
			Generating Artefacts	
			Model Transformations	
			Automatic Code Generation	
Evaluation	System Evaluation Chapter 6	Methodology-Practices-Promises-Metrics (MPPM)	Modeling Frameworks and Meta-Models Model performance Efficient Model Transformation	
		Product Risk Taxonomy	Project Engineering Risk Area	
			Development Environment Risk Area	
			Program Constraint Risk Area	

Figure 3.2: Methodology Overview

3.5 Conclusion

This chapter details the methodology followed to design and build the automatic translation tool. The complete tool is built based on the design science research methodology (refer section 3.1.4 of chapter 3). However, the creation of the proposed Automatic Translation Tool consists of sub-phases such as architecture creation (refer chapter4), design and development (refer chapter 7) and the evaluation of the tool (refer chapter 6). Each of these phases follows a significant methodology as discussed in this chapter. These methodologies will be followed during their respective phase /chapter. This chapter as its name says emphasizes the overall methodology and sub methodologies practiced during each phase to build the tool that meets the system requirements.

Chapter 4

Architecture Creation

This chapter explains the creation of the architecture of Automatic Translation Tool. This chapter also identifies and details the features of the tool.Section 4.1 addresses the architectural drivers such as the primary functional requirements, quality attribute requirements and business and technical constraints. The architectural design template proposed by (Clements et al., 2002) is followed for the rest of the chapter. The practice of this template supports the identification of architectural drivers, definition, and documentation of views and viewpoints of Automatic Translation Tool and how different views used supports the achievement of different architectural drivers identified during the initial stages of this chapter. In addition to the design of the architecture of the tool, this chapter details various other relative terms such as model, modeling, meta-model, meta-modeling and transformations that are used to design the tool. This chapter practices the Attribute-Driven Design (ADD) method to define the software architecture of the tool which uses the previously mentioned architectural drivers as the primary inputs of the architectural design procedure. This methodology is followed to ensure all the required drivers are used and are processed with the support of various design techniques such as the architectural patterns and scenarios. These functions are

performed to deliver a complete architecture of the tool using the 4+1 view method.

4.1 Attribute-Driven Design Methodology (ADD)

Attribute-Driven Design Methodology recommended by the Software Engineering Institute (SEI) is known as a systematic methodology followed step-by-step in order to design the software architecture of a software-intensive system. This approach mainly focuses on defining the software architecture the system based on the architectures' quality attribute requirements (Clements et al., 2002).

This methodology comprises of various steps, they include: (1) identify candidate architectural drivers, (2) choose a design concept that satisfies the architectural drivers (3) ensure to provide tactics to achieve quality requirements and (4) generate a complete software architecture for the tool which addresses all the architectural drivers (Clements et al., 2002).

In addition to the above steps, ADD is a three-step process which involves input, process and output stages and this terminology will be taken in to practice during the design of the software architecture of the proposed Activity Diagram to Java Code Generation Tool.The software architecture design process is as shown below.

4.1.1 Identification of Architectural Drivers

As mentioned previously, the required input to ADD includes the primary functional requirements, quality attribute requirements and other constraints. When expressing these architectural drivers in terms of the Automatic Translation Tool, the functional requirements are expressed with the support of use cases which when considered from

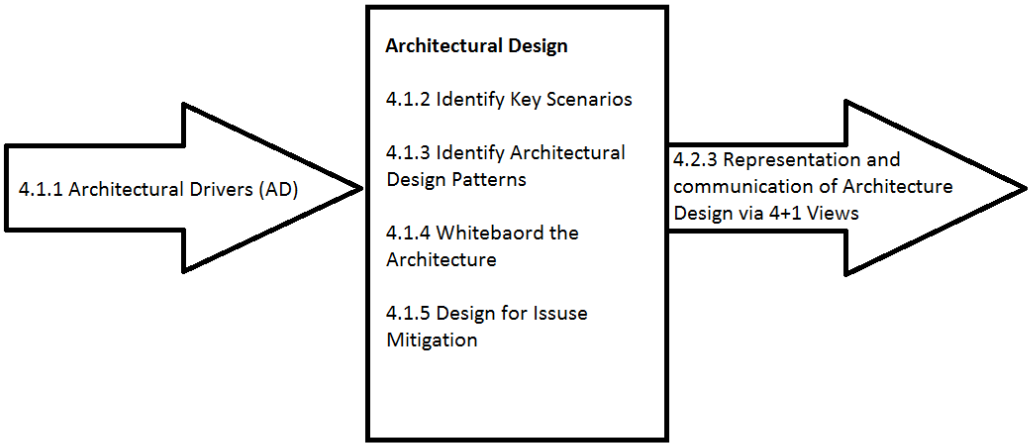

Figure 4.1: Three-Step Process of ADD for Architecture Design of Activity Diagram to Java Code Generation Tool

the perspective of 4+1 architecture (an architectural model designed by (Kruchten, 1995) to describe the architecture of a software-intensive system based on various, concurrent views) belongs to the logical view. The quality attribute prioritization prior to the system being built is supported by the Quality Attribute Workshop (QAW) which is also recommended by the SEI. Finally, business and technical constraints are determined by external factors.

Primary Functional Requirements

As the name of the tool defines, the Automatic Translation Tool generates runnable Java Code from an activity diagram. The tool should allow the software architect to construct an activity diagram in the provided model editor which is afterward transformed to a set of lines of Java code. This is carried out with the support of various another set of developed plug-ins which allow the transformation of an instance of the constructed activity diagram to a Java instance which is later transformed to understandable and runnable Java code.

Therefore, there are a number of functional requirements this tool comprises of, in order to meet the previously stated objectives and to achieve the expected performance of the tool. Furthermore, the functional requirements of the Automatic Translation Tool are based on the problem definition and the aspects taken into consideration in terms of answering the research questions (refer section 2.3.1 of chapter 2). Therefore based on the above-mentioned aspects, the most important functional requirements that need to meet are listed based on their importance to the tool below.

1. **FR1**: Ability to generate code automatically from the modeled app, which in this case is an activity diagram from which Java Code needs to be generated.

2. **FR2**: Ability to manage interoperability during the inclusion of dynamic sensors and actuators, the ability to support improved levels of interoperability may also mean support any kind of case study (e.g.: Smart Lighting System, Smart Door System or the Smart Security System) thus focusing more towards compatibility.

3. **FR3**: Ability to understand the transformation at different levels including the code generation process.

4. **FR4**: The code generated should be a representation of the behavior of the modeled app The app design model will be carried out by the user and therefore in order to achieve higher levels of understandability, the code generated should have the same syntax as the manually developed code and should match with the syntax used in the modeling of the app.

Quality Attribute Requirements (QARs)

Fundamentally, the systems' utility is determined by both functional and non-functional requirements, these characteristics include usability, flexibility, performance, interoperability, and security. Even though there had been an uneven emphasis on functional requirements, the functionality of any system is not considered useful or usable without the necessary non-functional requirements (Chung & do Prado Leite, 2009). In terms of both software architecture and ADD, the non-functional requirements are referred to as ***Quality Attribute Requirements***. Various useful aspects in relation to system requirements are achieved with the address of QARs. One of the main aspects includes the guidance provided during the design and creation of quality attribute scenarios which will be addressed after the identification and prioritization of the QARs of this tool.

The QARs of a system is derived in various ways, however in this case, as stated earlier, Quality Attribute Workshop is used in various phases such as the scenario and tactic identification of the QARs. This method derives the QARs from the mission or business goals. Since the design of the proposed UML to code generation tool does not have any external stakeholders involved in the requirement specification, the derivation of the QARs is entirely based on the ADD and other relative methodologies such as the QAW.

The Automatic Translation Tool also addresses these QARs during the design of its architecture and the tool. In addition to these QARs, some additional QARs are looked at, in order to address the functionality of the tool as a whole. The QARs are interoperability, usability, understandability, scalability, comparability, modifiability, re-usability, availability, testability, flexibility and performance of the system, which are justified later in this section. These quality attribute requirements are prioritized

based on specific criteria as detailed below. Architectural tactics and scenarios for the three most important quality attribute requirements are provided in later sections of this chapter.

Prioritization of Quality Attribute Requirements

Prioritization of system requirements are based on various aspects such as: importance determined by the stakeholders, importance determined by the system performance, penalty introduced if the requirements are not achieved, estimated implementation cost, lead time influenced by other factors such as training and development of support infrastructure, internal and external risks, volatility and other aspects (Berander & Andrews, 2005). Moreover, there are various requirement prioritization scales used by different authors, the two most common scales proposed by (Wiegers, 1999) are **High-Medium-Low** scale and **Essential-Conditional-Optional** scale.

High level focuses on mission-critical requirements that need to be achieved prior to the next release, medium level concentrates on the necessary system operations and low level looks at quality enhancements. Essential level ensures the product is not accepted until all the requirements are achieved, conditional level support product enhancements and ensures the product is completely available and optional level looks at functions that are optional. For the prioritization of system requirements for the Automatic Translation Tool, both High-Medium-Low and Essential-Conditional and Optional criteria are used. The prioritization levels are named as 1, 2 and 3 as detailed below.

1. **Priority Level 1**: This is the highest level of priority which means it is compulsory that the requirements belonging to this level of priority should be achieved to avoid system failure. (Wiegers, 1999). In terms of the Automatic Translation Tool,

the most critical requirements the tool needs to achieve include usability, availability, and interoperability. The tool needs to easy to use and completely available for successful code generation from UML Activity Diagrams. Moreover, the tool should allow the user to model any smart home case study to achieve high levels of interoperability. The Table 4.1 details these quality attribute requirements based on their level of priority.

QAR	**Name**	**Description**
QAR1	Availability	High availability of the Automatic Translation Tool can be achieved by ensuring the presence of all the required plug-ins and the stated version of the required software. They can also be achieved by ensuring all the required plug-ins are up-to-date.
QAR2	Usability and Understandability	Enhanced usability can be achieved by designing a simple model editor which would allow the designer to understand the notations of the activity diagram and how to model the app easily. These requirements can also be achieved by providing the user with an additional manual detailing how to use the tool completely from modeling to code generation.
QAR3	Interoperability	Activity Diagram to Java Code Generation Tool can achieve improved levels of interoperability in a smart home by allowing the designer to model activity diagrams for any case study, for instance, the tool should be able to model activity diagrams for the smart lighting system, smart door system, and smart security system.

Table 4.1: Priority Level 1 Quality Attribute Requirements

2. **Priority Level 2**: This is the medium level of priority whereby achievement of these requirements may only support the implementation of the tool and may not have a major impact on the performance enhancement of the tool. These requirements support necessary system operations (Wiegers, 1999). Code re-usability, modifiability, scalability and testability of the code generated from the UML Activity Diagram are recognized as medium level requirements as the quality of the code generated does not affect the implementation or performance of the Automatic Translation Tool.

QAR	Name	Description
QAR	Re-usability	Ensure improved levels of code re-usability
QAR5	Modifiability	The tool should be able to cope with the modifications or changes made by the designer or the smart home user.
QAR6	Scalability	The chosen modeling language is sufficient to manage massive modeling projects.It should be able scale down to smaller projects without the need for additional work and effort.
QAR7	Testability	The activity diagram model editor should be able to design activity diagrams based on any given case study.Also, the code generation process should generate Java code based on any activity diagram modeled by the model editor.

Table 4.2: Priority Level 2 Quality Attribute Requirements

Other Architectural Drivers

These drivers include the business and **technical constraints**. The constraints are requirements for which the design decisions are pre-specified (Kruchten, 1995). Technical constraints of the tool areas listed below.

Technical Constraints include the computer system, Applications, and plug-ins that are used to model and generate Java code. This tool does not consist of any **business constraints** due to the non-involvement of external stakeholders.

4.1.2 Identification of Quality Attribute Scenarios (QAS)

The concept of QAS was first introduced in 2003 by (Bass, Clements & Kazman, 2013) in order to support the development of software architectures. In other words, the QAS helps the quality attribute requirements to be expressed in terms of the operational form of the system.

Quality Attribute Workshop (QAW) supports addressing the quality attribute requirements in the form of scenarios. There are two different types of QASs such as the general scenario and the concrete scenario. The general scenario emphasizes the system-independent specifications which provide a template for a set of requirements. These scenarios, however, need to be transformed into concrete scenarios ultimately in order to be more system specific (Wu & Kelly, 2004). Since the tool is not yet developed and is in the design stage, the general scenario approach is used to express the QASs of the tool. The general scenario proposed by (Len, Paul & Rick, 2003) consists of six aspects, such as stimulus, a source of stimulus, artifact, and environment, response and response measure. They are defined as shown below.

Source of stimulus	The entity which generates the stimulus(during execution or development).
Stimulus	a phenomenon that needs to be considered when it arrives at the system
Environment	Conditions under which the stimulus occurs
Artifact	the artifact affected by the stimulus
Response	Activities that need to be undertaken after the arrival of the stimulus
Response Measure	the attribute-specific constraint that needs to be satisfied by the response.

Table 4.3: Quality Attribute Requirements Six-Part Scenario Format

The above mentioned QAR six-part scenario format proposed by (Len et al., 2003) will be taken into consideration when generating the QAR scenarios for the tool. Even though the tool addresses 7 QARs in total, the three most important QA

requirements, based on the prioritization (refer section 4.1 in chapter 4) will be taken into account during the QA scenario generation. The QA requirements scenarios for QAR1, QAR2, and QAR3 are defined as shown below.

Availability Scenario

As mentioned earlier, availability requirement of the tool ensures the tool is ready for use. This part of the chapter will address a scenario in which availability of the UML to code generation tool will be achieved. When the user needs to use the tool, he or she would click on the icon of the application which is used to model the UML diagrams. After the user selects the appropriate workspace in which all the files are saved, the application checks for the availability of the required plug-ins to supports the configurations once the application is set to use. Prior to the installation of the missing plug-ins, the application inquires the user if the plug-ins should be installed via a dialog box. To ensure complete availability, the user needs to confirm the plug-in installation.

Source of stimulus	End User (Software Architect, Software Developer or the Smart Home Users)
Stimulus	Software crash and delay in timing
Environment	Starting the application and setting up the workspace
Artefact	App design and transformation application
Response	Notify the user about the unavailability of the required plug-ins
Response Measure	Repair time,time taken to re-install the missing plug-ins and relaunch the application..

Table 4.4: Availability Quality Attribute Requirement Scenario

Usability Scenario

Usability, which also means the level of ease in terms of learn-ability, efficiency, reliability, and satisfaction in using the application to generate code from the UML

activity diagram modeling. The functions carried out during runtime are addressed in this scenario. Usability can be achieved with, how easily the user locates the relevant workspaces, files, and plug-ins, how well the user know how to run and configure the relevant meta-models and Java files, how well the user understand the generation of models and model transformations, how confident the user is during the error recovery and finally how well the user is satisfied with the code generated from model.

Source of stimulus	End User
Stimulus	Run and configure the models and Java files
Environment	Runtime
Artefact	UML modelling and code generation application
Response	Use the tool effectively thus understanding the underlying process
Response Measure	Time to model the app and generate the code

Table 4.5: Usability Quality Attribute Requirement Scenario

Interoperability Scenario

This QAR based on a smart home scenario ensures any smart home case study can be modeled in a single model editor. Therefore this scenario will address two case studies which will be modeled in the UML model editor. The user will model, for instance, a smart light system and a smart security system in the proposed model editor to ensure the achievement of interoperability.

Source of stimulus	End User ,
Stimulus	Creating any UML Activity Diagram
Environment	After the configuration of the UML meta-model
Artifact	Meta-model of the UML, model and the model editor
Response	The tool allows any well-formed UML Activity Diagram to be designed. It rejects any invalid activity diagrams.
Response Measure	Success percentage in modeling case studies using the model editor

Table 4.6: Interoperability Quality Attribute Requirement Scenario

4.1.3 Identification of Architectural Pattern (AP)

Architecture Patterns are common architectural structures which are well understood and documented (Schmidt, Stal, Rohnert & Buschmann, 2013). Every individual pattern details the high-level structure and behavior of a general software system and aims to achieve relative primary functional and quality attribute requirements (Harrison & Avgeriou, 2010). Also, the architectural patterns can be categorized based on their focus area.

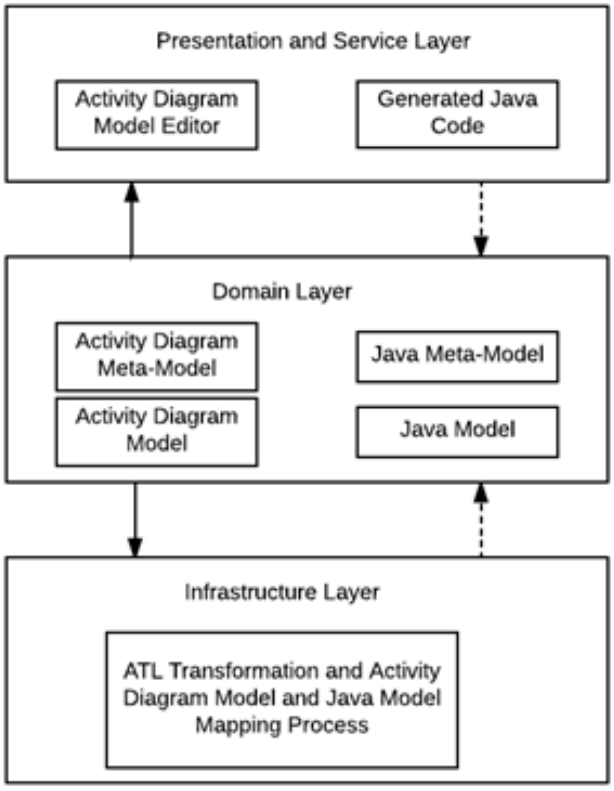

Figure 4.2: Pipe and Filter Architecture Pattern for the Proposed Tool

Therefore from the previously listed architectural styles, the Pipe and Filter architecture will be used to emphasize on the modelling of the domain model (in this

case, the activity diagram models and Java models) and partitioning of the application concerns in order to address the abstract view of the tool as a whole while emphasizing on the individual roles and responsibilities of each layer. This architectural pattern as graphically represented in Figure 4.2 consists of three layers, i.e.: the domain layer, presentation, and service layer and the infrastructure layer. The presentation and the service layer focuses on the front-end and the User Interface (UI) of the application, thereby as per the proposed tool, the end user is able to model an activity diagram in the generated model editor and generate Java code from the modeled activity diagram. This is further emphasized and understood in section 4.1.4 in chapter 4, where the architecture of the proposed tool is expressed in the form of the white-board architecture thus communicating further details of the tool and its architecture.

The domain layer addresses the domain models (representations of an application domain that can be used to achieve a variety of operational goals) of the tool. In the case of the tool, its domain models are the activity diagram meta-model, Java meta-model, generated activity and Java models. The configurations of Activity Diagram meta-model can generate a model-editor which sits in the presentation and service layer thus establishing a relationship between the two layers. The infrastructure layer enables the application to interact with the external systems support receiving, storing and providing data when requested (Garlan, Cheng, Huang, Schmerl & Steenkiste, 2004). Since the proposed tool does not involve with any data storage and exchange, the ATL transformation process is included in this layer since it involves a model mapping and providing the Java model to the domain model when requested to support and complete the code generation process.

4.1.4 Architecture Tactics

A tactic is a design mechanism to achieve the desired level of QAR by manipulating some aspect for an analysis model for the QAR with the support of design decisions. This technique is used to mitigate the design issues during the design of the architecture of the proposed tool. In order to explain how these issues will be addressed and mitigated, the scenarios for each QAR from the priority list 01 is addressed. How each scenario belonging to a QAR will be mitigated or achieved is explained in the table below.

QAR	**Stimulus (4.1.2)**	**Tactics**
QAR1	Model multiple case studies using the UML model editor	This involves locating and managing of interfaces. **Locate** the workspaces, projects, files, and plug-ins. Manage interfaces, **orchestrate** multiple plug-ins into a single Eclipse workspace, integrate multiple applications into a single automated process.
QAR2	Run and configure the models and Java files.	**Support User Initiative**, which allows manipulations within the system such as: undo, cancel, pause, resume and aggregate. **Support System Initiative** which manages the projects and workspaces, manage the model transformations, manage the system and user models.
QAR3	Software crash and delay in timing. This involves fault detection, fault recovery and fault prevention.	**Fault Detection** involves monitoring the application, application configurations, exception detection, monitoring model transformations and monitoring the absence of plug-ins installed.**Fault Recovery** involves software upgrade, exception handling, installation of required plug-ins and reconfigurations.**Fault Prevention** ensures exception prevention, ensure all the plug-ins are installed and ensure the software is upgraded thus ensuring the software is completely available for use.

Table 4.7: Quality Attribute Requirement Tactics

4.2 Architecture of the Tool

Figure 4.3 provides an overview of the architecture of the proposed tool. This figure provides a very brief idea of how the UML diagram is transformed to a runnable software code with the support of two significant techniques, such as model to model

transformation and model to text transformation. For this process, various tools, plug-ins, and techniques are used which will be addressed in detail section 4.2 of this chapter and Chapter 7 where the technical aspects of the tool are emphasized to a greater level.

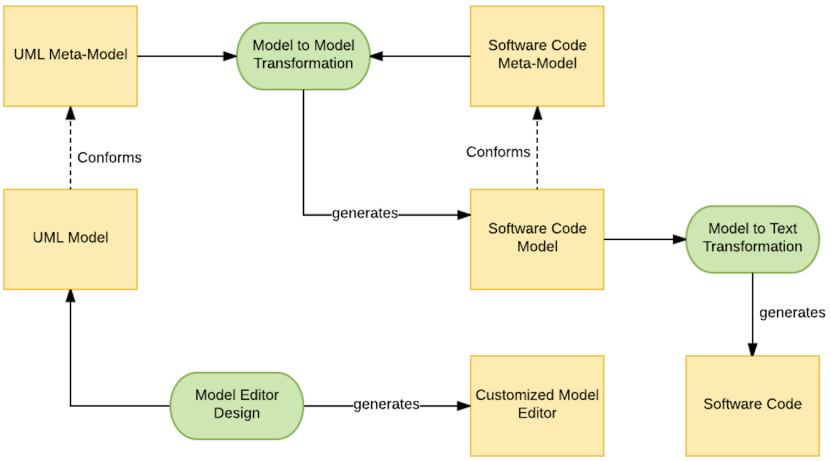

Figure 4.3: Overview of the Architecture

The 4+1 architecture is designed to describe the architecture of a software-intensive system based on the practice of multiple and concurrent views.This architecture addresses different views and viewpoints of a specific system. The view model, which is also another word for a framework is systems and software engineering. The view model consists of a set of views, which are representations of a complete system from the perspective of different users such as end-users and the developers(Choi & Yeom, 2002).

This framework consists of 5 views: development view, logical view, physical view, process view, and scenarios. Out of these views, the process view and scenarios are chosen to express the perspectives of the Activity Diagram to Java Code Generation Tool. Out of these 5 views, 3 views, i.e.: the process view, scenario view and logical

view are used to express the views of the Activity Diagram to Java Code Generation view model from the perspective of the end users and the developers as shown below.

4.2.1 White-Board Architecture of the Tool

In order to provide the better understanding of the proposed tool, how generation and configuration of meta-models support the generation of another model thus eventually helping the achievement of the code generation functionality and architectural drivers as explained later in this chapter with the help of 4+1 architecture. The purpose of whiteboarding the architecture of the tool is to enhance on the understanding ability of the tool in terms different views and to show the relationship and communication between different meta-models and models belonging to different layers as shown in Figure 4.4.

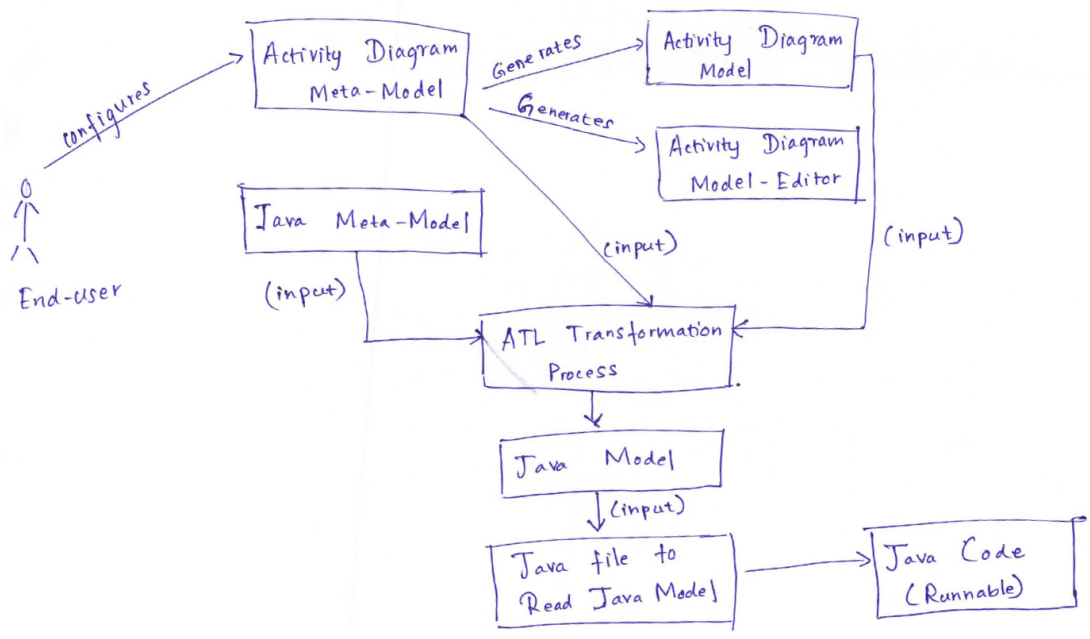

Figure 4.4: Architecture White-Board showing High Level Design of the Tool

Furthermore, even though the whiteboard architecture depicts the high-level design of the proposed tool, it also highlights which tool, language or visual app design

technique is used at different phased on the tool. These components will be addressed in detail in section 4.2 of this Chapter and in Chapter 5.

4.2.2 Logical View - Sequence Diagram

Logical view graphically details the functional requirements of the system. The sequence diagram and the class diagram shown below depict how the functional requirements are achieved during the model-to-model transformation of Activity Diagram Model to Java Model transition.

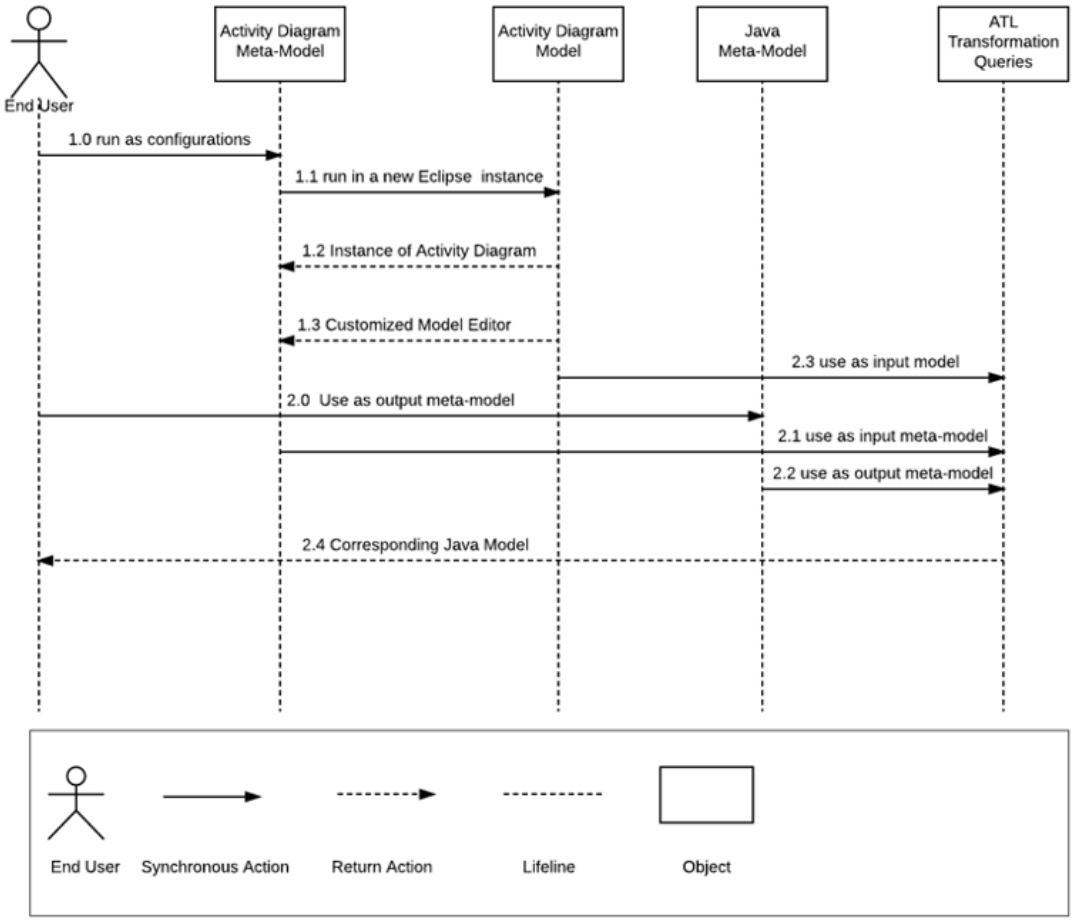

Figure 4.5: UML Sequence Diagram - Activity Diagram to Java Code Generation Tool

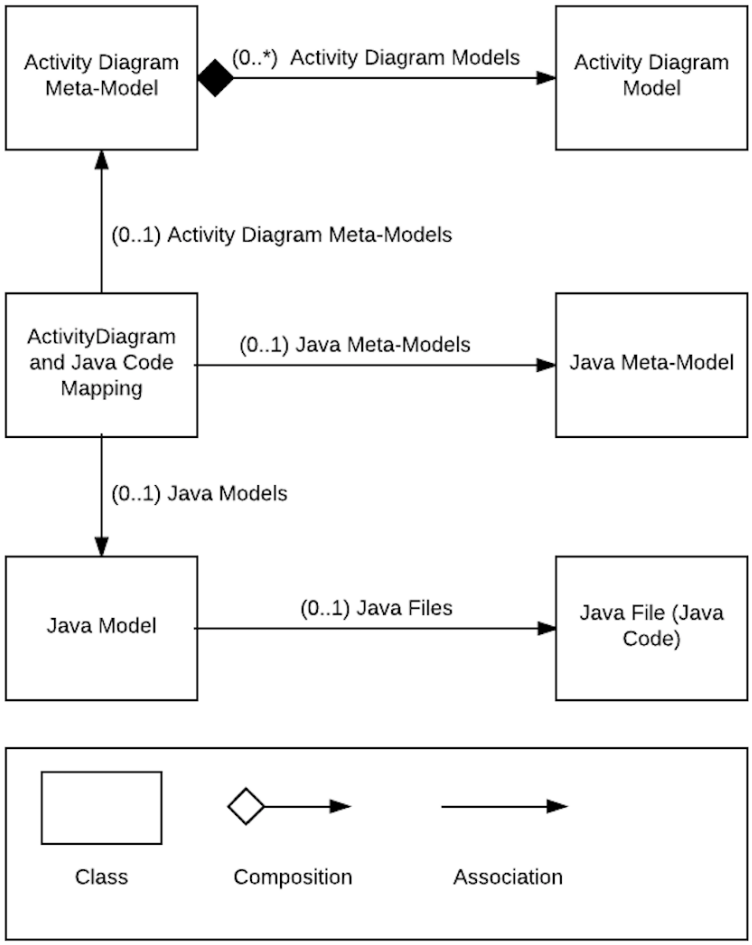

Figure 4.6: UML Class Diagram - Activity Diagram to Java Code Generation Tool

The above-shown class diagram shows the entities involved in the design of the proposed tool. These entities consist of different types of relationships such as associations and compositions. When Figures 4.6 and 4.7 are taken into account, it is very clear that ***FR1, FR2, FR3 and FR4*** are achieved with the support of this view.

4.2.3 Process View - Activity Diagram

This view expresses the dynamic aspects of the system and details the system processes and how the communication between various elements and entities are carried out (Kruchten, 1995). When the activity diagram shown in figure 4.4 is taken into account, it shows how the required plug-ins are installed if unavailable thus achieving ***QAR3***, i.e.: Availability. Also, allowing the user to model any activity diagram using the given model editor supports the achievement of ***QAR1***, i.e. interoperability (refer section 4.1).

Model Driven Architecture

As defined by Object Management Group (OMG), MDA (Model Driven Architecture) is known to be a technique to organize and manage enterprise architectures supported by automated tools and other services, this also supports in defining the models and the transformations between the model types (Brown, 2004). Even though UML (Unified Modeling Language) also introduced by OMG is practiced to describe the object-oriented software artifacts, the internal architecture and the scope of applicability of UML is not yet completely stabilized (Bézivin, 2001). Therefore, in order to support UML and other similar languages, OMG has introduced a framework based on MOF (Meta Object Facility) and UML and some additional modeling stacks. This modeling framework is known as Model Driven Architecture (MDA).

Model

MDA comprises of various fundamental concepts such as models, modeling, views and model transformations. A model provides an abstraction of a physical system

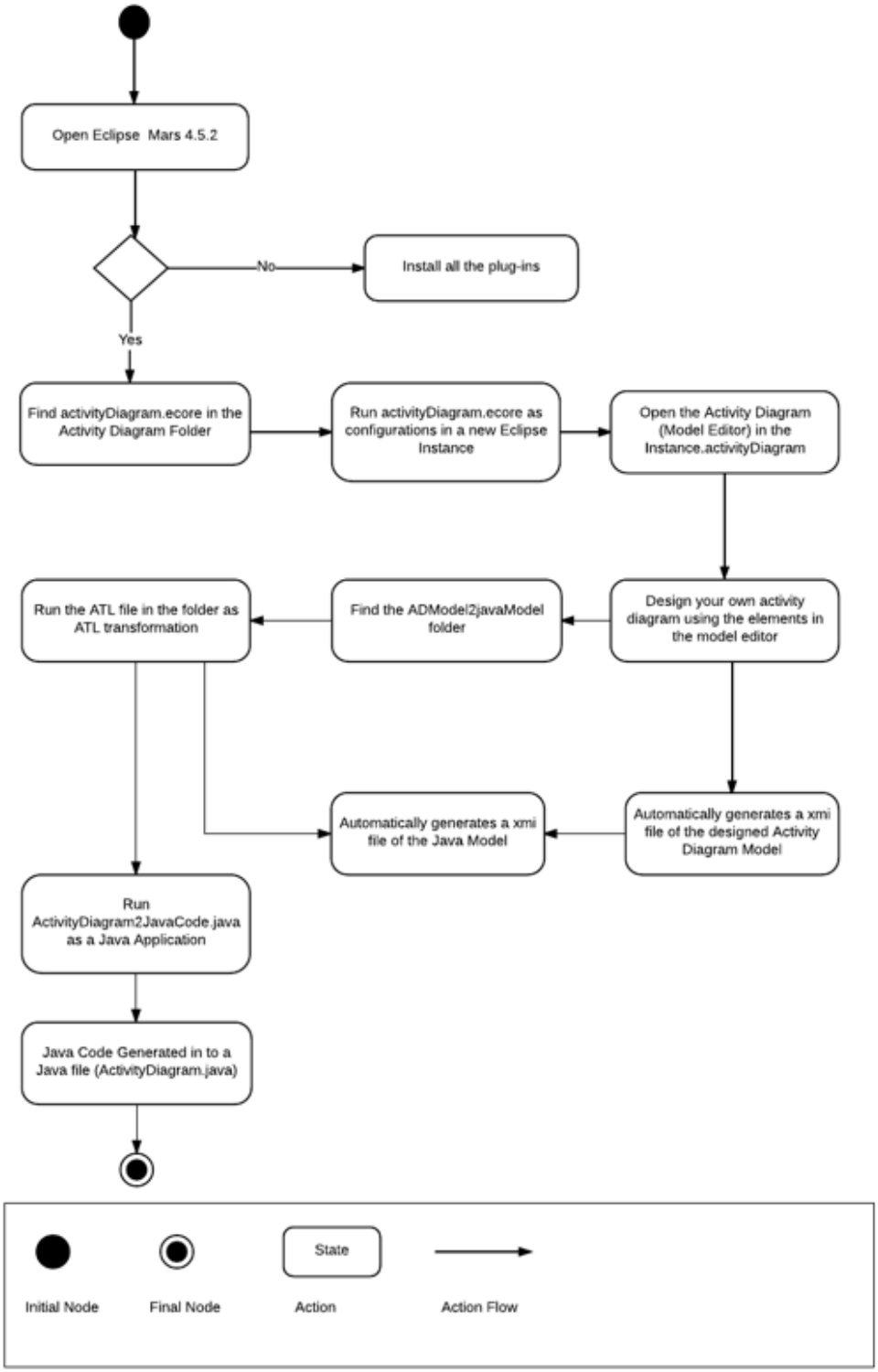

Figure 4.7: UML Activity Diagram - Activity Diagram to Java Code Generation Tool

which provides the designers and developers with valid reasons with as to why selective requirements should be taken into consideration during the system design and construction (Brown, 2004).

Modeling

The process of designing a software application which occurs prior to the coding. Also, modeling is regarded as an essential part of software projects belonging to any size ranging from small, medium to large. Modeling process supports carrying out work at a higher level of abstraction by hiding and masking the details thus emphasizing on the big picture or by focusing on various levels and domains of prototypes (Gessenharter & Rauscher, 2011).

Levels of abstraction for activity diagrams come in two significant levels: design level and conceptual level. At the conceptual level, activity diagrams emphasize on the abstract views than those on the design level. In other words, activity diagrams created in this level defines relative characteristics such as actions and operations. However, this level of abstraction does not provide concrete implementation information. During design level, further details in relation to the implementation are provided. For instance, each action in the activity diagram needs to be mapped to a piece of implementation such as a function or a loop (SAP, 2007).

When the above-mentioned levels of abstractions are taken into consideration, activity diagrams modeled for the smart home will be based on design level due to its need to generate code from activity diagrams modeled. This may require mapping of every node to a certain level in order to support the characteristic of code generation which will be discussed in detail in Chapter 5.

Unified Modeling Language (UML)

The current state of practice in regard to modeling is considered to UML as identified and detailed in section 4.1.4 of this chapter. This is considered to be the primary modeling notation which also allows the capture of the most significant characteristics of a system in relation to its corresponding model (Brown, 2004).

UML is used to model the Activity Diagram to Java Code Generation Tool. UML consists of both structural and behavioral aspects out of which behavioral aspect will be taken into consideration when graphically representing different views of the architecture of the tool. This will be explained in detail in the following subsection of this chapter.

Model Views and Model Transformations

The model transformations are of different kinds such as the model-to-model transformation and model-to-text transformation, which will be addressed comprehensively in later sections of this chapter. With the rapid transition from code-oriented to model oriented software production techniques, much importance is given to the practice of meta-models which defines itself a language for detailing a particular domain of interest (Bézivin, 2001).

The model transformation is also supported by OCL (Object Constraint Language) which defines the behavior of constraints during model transformation. The constraints prior to the model transformation are known as source model and after the model transformation known as a target model. Post-conditions ensure that target model is valid output for the transformation to the respective source model. Preconditions make sure the model transformation is carried out effectively (Wąsowski & Lönn, 2016)

A specific model can follow various modeling concepts and notations in order to highlight on particular views and perspectives of the system. In the design of the Activity Diagram to Java Code Generation Tool, 4+1 architecture is taken into consideration when modeling the tool based on views and perspectives.

Meta-Modeling

The meta-model is basically another model that supports the construction of other models. Even though they are all considered as models, one model is expressed in terms of the other where one model is an instance of the other and one model conforms to the other.

In the context of MDE, a meta-model is regarded as a structural diagram which defines the model elements, for instance, the elements of the activity diagram (such as initial node, final node, actions and decision nodes) and their relations. However, further emphasize is required describe rules and the relationship among the elements. Definition of rules in a meta-model play a very important role especially if the model requires some kind of transformation to text (which can also mean code generation) or to another type of model (Gronback, 2009). When the UML meta-models are taken into consideration, they define the structure of the UML models. Furthermore, (Andrews et al., 2003) regards the activity of meta-modeling as a construction of an object-oriented model of the abstract syntax of a language. Also, when the definition meta-model is put into a wider context, a meta-model is regarded as a model which defines the language completely thus addressing the concrete syntax, abstract syntax, and other semantics (Aßmann, Aksit & Rensink, 2005).

Model-to-Model Transformation

Taking this technique into account, two meta-models are built during the design of the Activity Diagram to Java Code Generation Tool. Initially, a meta-model is designed for the activity diagram modeling thus identifying the nodes of the activity diagram as classes, i.e.: a single class for each node. The nodes include activity node, initial node, final node and the decision node.Furthermore, since the activity diagram is needed to be transformed to Java code, prior to the model-to-text transformation, the activity diagram meta-model is mapped with a Java meta-model. This meta-model comprises of classes such as Java statements of different types such as the assignments and declaration statement, decision statement and initial and end statements.

The activity diagram meta-model is used to generate a model of the activity diagram. A mapping process is carried with the support of ATL transformation language queries where the activity diagram meta-model and the Java meta-model are mapped. During this mapping process, the generated activity diagram is used as the input model. The output model generated from running these ATL queries is the Java model.These meta-models, models, and transformations will be described in details chapter 5 thus emphasizing on each of their functionalities and how they address the research questions and the functional and non-functional requirements.

Model-to-Text Transformation

Model-to-text transformation technique is taken into consideration when transforming the Java model generated during the model-to-model transformation to runnable Java code. A built-in Java file is provided which reads the instances of the Java model and writes and writes them to a new Java file. The filtering is carried out based on the

statement type. This is detailed in chapter 5.

Modeling Activity Diagrams

As per the tool, the tool involves modeling of an activity diagram in a provided customized activity diagram model editor. This functionality is achieved with a design of a meta-model for the activity diagram in Eclipse, version Mars 4.5.2.

Once the user runs Eclipse application within a specific workspace, depending on the projects available in the chosen workspace, the software checks for the availability of the required plug-ins and installs them if unavailable. The user then needs to run the meta-model of the activity diagram, i.e.: activity diagram.ecore as configurations in a new Eclipse Application instance. This opens a new Eclipse instance with a customized activity diagram model editor which provides the user with a modeling space and palette consisting of all the relevant elements for the modeling of an activity diagram. The modeling of the activity diagram generates an XMI file of the activity diagram model stating the corresponding nodes and connectors.

Transformation of Activity Diagram to Java Model

ATL is a known to be a hybrid transformation language which consists of both declarative and imperative constructs (Jouault & Kurtev, 2006).This, in fact, supports the model-to-model transformation process thus enhancing the activity diagram model to Java model transformation.

The end user needs to run the ADModeltoJavaModel.at the file as ATL transformation to generate the corresponding Java model from the activity diagram model.

Transformation of Java Model to Java Code

This involves model to text transformation process where the Java model is transformed to runnable and understandable Java code. As for the Activity Diagram to Java Code Generation Tool, there is a built-in Java file called ActivityDiagram2JavaCode.java which when run as a Java application writes the corresponding Java code to another separate Java file, thus completing the activity diagram to code generation procedure. Finally ensuring all the listed functional and non-functional requirements are achieved as listed.

4.2.4 Scenario View - Use Case Diagram

The scenario view explains the functionalities of the system from the preservative of the outside world. Moreover, this view comprises of diagrams detailing the functions of the system from the view of a black box. This view is graphically presented in terms of a use case diagram. Thereby, a use case diagram is designed to show the scenario view of the Activity Diagram to Java Code Generation Tool.

A use case diagram basically consists of actors involved in the functions of the system, in the case of the Activity Diagram to Java Code Generation Tool, there may be actors for different levels of functionalities. Design and modeling of the activity diagrams can be carried by the software architect or the designer whereas the code generation from the activity diagram can be carried out by the software developer to ensure the code generated is understandable and runnable. In the meanwhile, the client may also learn the complete process from the design and modeling of the activity diagram to the code generation which is the combination of the functions carried out by both software architect and the software developer.

The use case diagram for the tool has one user called the "End User" who carries out all the functions required to generate Java code from the activity diagram. The use case diagram has rectangular shaped boxes distinguishing different layers of functions as shown in Figure 4.8.

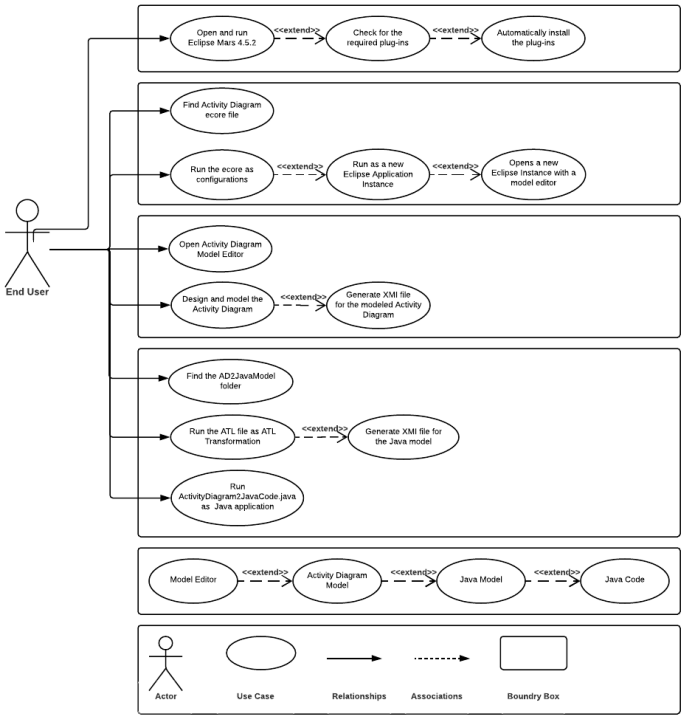

Figure 4.8: UML Use Case Diagram - Activity Diagram to Java Code Generation Tool

Both Figure 4.7 and 4.8 graphically explain the process of Activity Diagram to Java Code Generation Tool in terms of the process view and the scenario view. These diagrams include the use of different applications, plug-ins, file formats, models and model transformations. These will be explained in this part of the thesis.

4.3 Conclusion

When the architecture design of the proposed Activity Diagram to Java Code tool is taken into consideration, as discussed in this chapter, ADD methodology is practiced to design the architecture for this tool. As shown in Figure 4.1, the architectural drivers are used as the inputs to the design and several steps are followed during the design phase, they include identification of architectural scenarios and patterns mitigating the issues during the design with the support of architectural tactics. Finally, the architecture design of the tool is represented and communicated using the 4+1 views approach. In order to ease the understanding of the tool, various terms are defined and explained in this chapter in addition to the design of the architecture. These terms include modeling, meta-modeling, mode transformations and a white-board of the architecture which gives a complete and clear understanding of the proposed tool.

Chapter 5

Design and Development

This chapter details the design and development of the proposed tool. As explained in earlier chapters of this thesis, the tool involves generating runnable Java code from a simple UML activity diagram.In order to provide a better understanding of the tool, a detailed view of the is provided thus illustrating how the proposed tool supports the achievement of the deployment of smart home applications. The tool comprises of a set of components such as design of meta-models for UML activity diagram and Java programming language, design of the activity diagram model editor, model to model transformation and model to text transformation. Each of these phases or the components require a set of technology decisions that need to be made which in turn would enhance the design and the development of the tool. Additionally, the design and the development of each component will be elaborated with the support of snapshots from the tool design to add further understanding to the tool.In the meanwhile, the global case study of this thesis, which is the smart light system will be addressed throughout this chapter thus showing how the smart light system was modeled using the proposed tool to ensure the promised architectural drivers are achieved.

5.1 Detailed View of the Architecture

As said earlier, the figure shown below is a very detailed view of the architecture of the proposed tool. During the literature review, which is Chapter 2 in this thesis, various tools which address the smart home scenarios and challenges relating to the smart homes such as lack of interoperability and other challenges were discussed. In order to eliminate the problem definition, a tool is proposed as shown in Figure 5.1 which is also highlighted in the figure shown below. From the entire framework, the section in the red rectangle will be taken in to further consideration when designing, developing and building the tool.

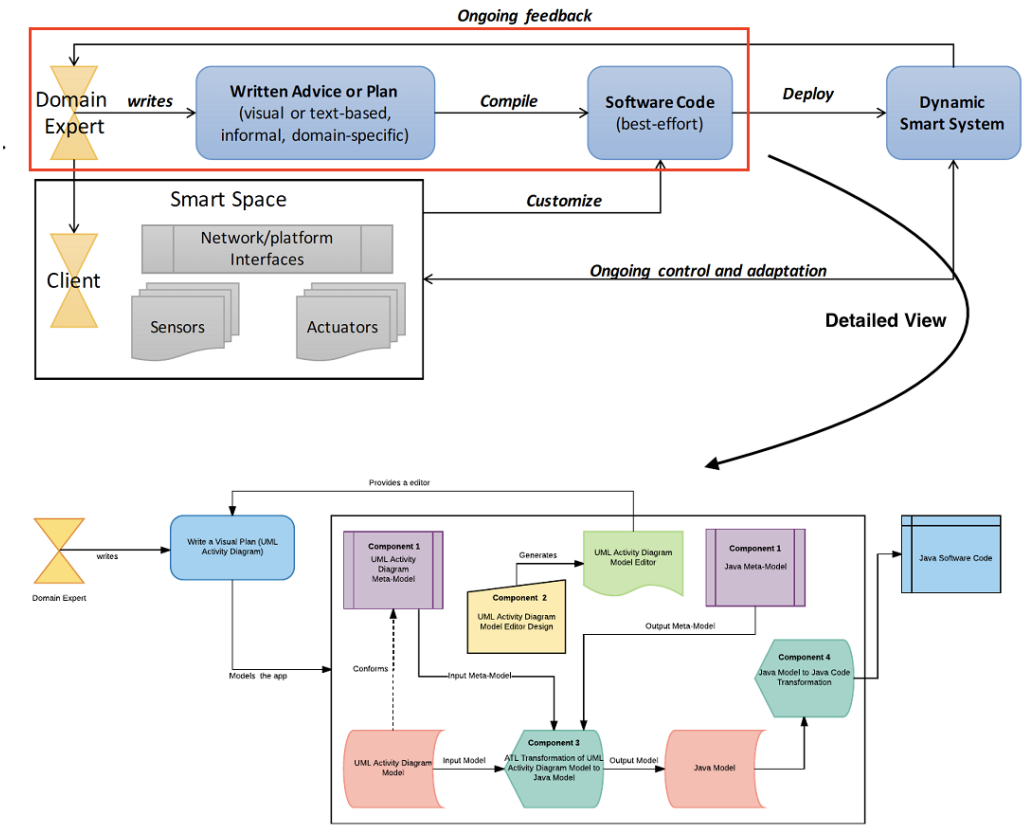

Figure 5.1: Detailed View of the Architecture

The architecture in the second part of the figure describes how the visual domain-specific plan is compiled to a software code with the use of various techniques, tools, and plug-ins in detail. Each of these components, how they are developed and configured and how they eventually achieve the stated architectural drivers(ADs) will be discussed comprehensively in the following section of this chapter. This is supported by a comprehensive class diagram that describes the structural aspect of the proposed solution as a whole.

5.1.1 Class Diagram

The class diagram depicts a detailed structural view of the proposed tool which generates runnable Java code from modeled UML Activity Diagram. A class diagram describes this structure of the system with the support of classes, attributes, operation (methods) and the relationship among these objects. As for the proposed tool, each processor phase during the tool design is considered as a separate class such as Activity Diagram Meta-Model Class for the design of the Activity Diagram Mete-Model. The relationship between the processes is shown in terms of reference and composition association based on their types of relationship with each other. Furthermore, relative attributes and operations of each phase are included in their respective classes as shown below.

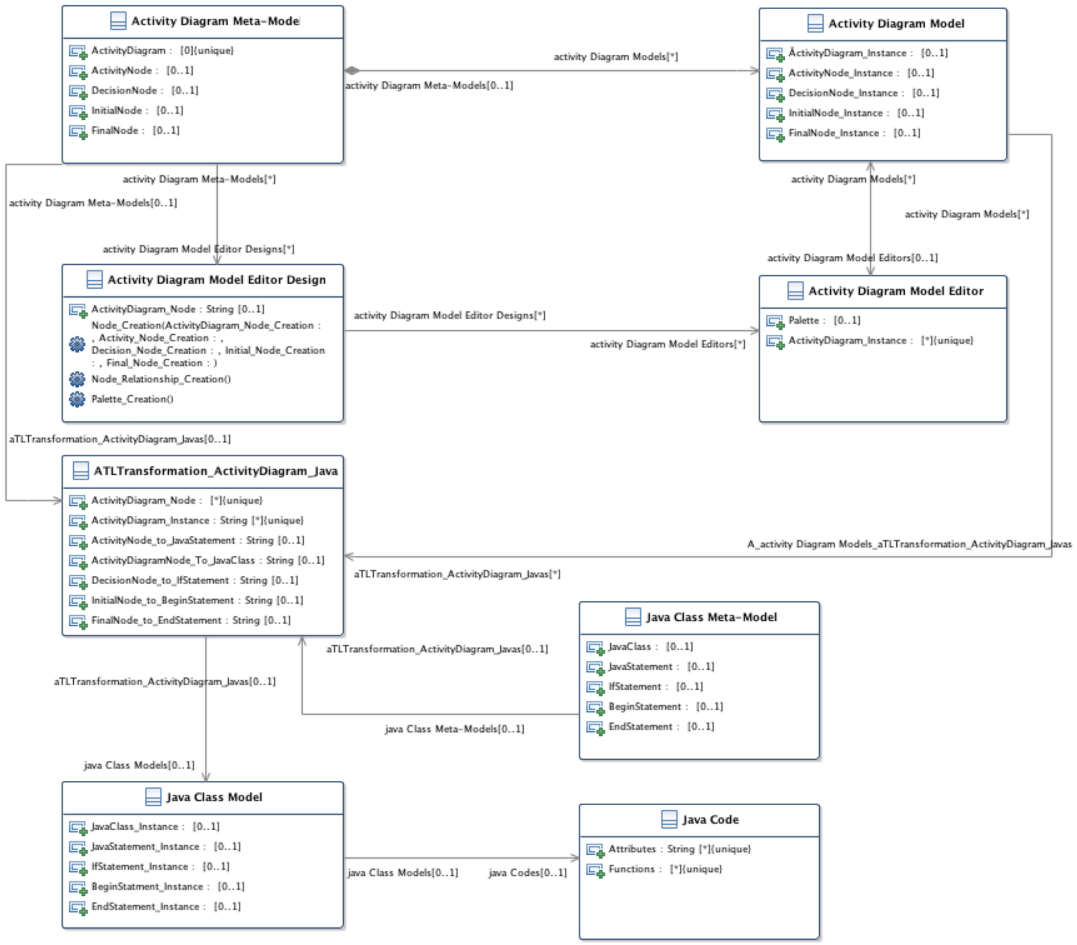

Figure 5.2: Detailed Class Diagram

As discussed earlier, the most significant components of the proposed tool will in addressed in detail, where the technical decisions made in relation to the design and development of the tool and how each component is designed will be discussed individually.As per Figure 5.1, there are 4 components in the proposed architecture.They are addressed in length as follows.

5.1.2 Development Work-flow

The below-shown work-flow will be followed during the development of the proposed tool. The steps in the yellow rectangles need to be followed and developed in order as the output of a specific phase (represented in orange rectangles) is used as the input in one of the following phases.

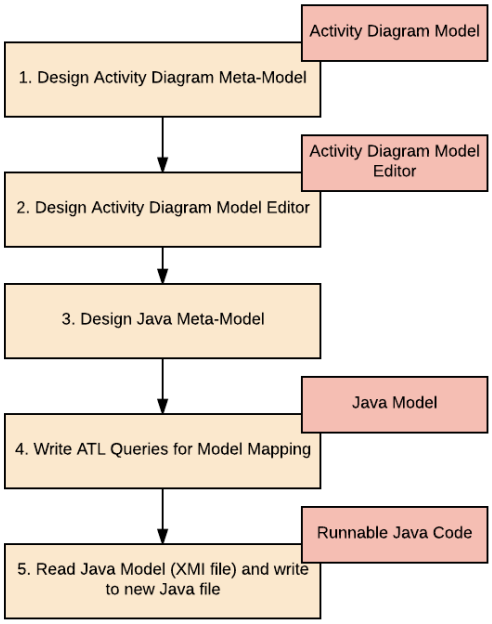

Figure 5.3: Development Work-flow

5.2 Component 1:Meta-Model Design

As per Figure 5.1, component 1 represents the design development of both UML activity diagram meta-model and the Java meta-model. The activity diagram meta-model is designed to support the generation of an activity diagram model which will eventually help in the code generation procedure.This will also help the design of the activity

diagram model editor, especially during the design of the activity diagram nodes.The Java meta-model is designed so that it can be used as the "output meta-model" during the model transformation process. There are various applications supporting the design of meta-models. However, technological decisions are made during the selection of the most suitable application based on defined selection criteria.

5.2.1 Technology Decisions: Selection of an Application for App Modeling

There are various applications which support the modeling of app design models. These tools include Microsoft Visual Studio, Visual Paradigm, Star UML, IBM Rational Enterprise, Enterprise Architect, Papyrus, Eclipse Modeling Framework a plug-in by Eclipse, ArgoUML, Magic Draw, Poisedon and Net Beans UML plug-ins. Each of these tools allows the modeling of a visual app, however only one of these tools are chosen to model the app design of the proposed tool, this is being carried out based on selected criteria as said previously.

As shown in the Table 5.1, the criteria to chose the best application to design app design models include the ability to structure the models, levels of robustness and easiness to model apps, the ability of the application to allow generation of a customized model editor, model accuracy and the ability to generate models instanlty. The above mentioned apps are evaluated agianst this criteria to chose the most suitbale app that support both app designing and code generation aspects. This is depicted in Table 5.2.

Criteria	Definition
Structuring of Models	Ensuring creation of classes, packages, associations, attributes and name spaces specially supporting the building of meta-models (France, Ghosh, Dinh-Trong & Solberg, 2006).
Robustness and ease of use	Improved levels of efficiency when managing the tool, limited involved during the configurations (Ho, Jézéquel, Le Guennec & Pennaneac'h, 1999).
Ability to generate a customized model editor	The ability of the tool to support the generation of customized model editor (Nordstrom, Sztipanovits, Karsai & Ledeczi, 1999).
Model Accuracy	Ability to provide well defined and well understood details of the classes, associations, name-spaces and attributes created which in turn would support the generation of an accurate model instance and a model editor (Rumpe, 2016).
Standard C ompliant	Ability to create complex systems with less communication ambiguity (Inc, 2017)
Model instance generation	Ability to generate an instance of the UML class diagram to enable model-to-model transformation (Ehrig, Küster & Taentzer, 2009)

Table 5.1: Selection Criteria: App Modeling Application

Criteria	Poisedon	Enterprise Architect	Magic Draw	EMF by Eclipse	Papyrus
Structuring of Models	✓	×	✓	✓	✓
Robustness and ease of use	✓	✓	✓	✓	✓
Ability to generate a customized model editor	✓	×	×	✓	×
Model Accuracy	✓	×	✓	✓	✓
Standard Compliant	✓	×	×	✓	×
Available for free	×	×	×	✓	✓
Model instance generation	×	✓	×	✓	×

Table 5.2: Selection of App Modeling Application

Based on the above results, Eclipse Modeling Framework (EMF) seems to be the most suitable application to model the visual app. In addition, Eclipse as a foundation and a corporation facilitates the integration and collaboration of extensible tools and frameworks thus supporting building, deploying and managing the software (Steinberg, Budinsky, Merks & Paternostro, 2008). Therefore, Eclipse (version Mars 4.5.2) is used to build the complete tool where a different plug-in is integrated into to a single working space whereby usability and compatibility are achieved to a very great

extent.

5.2.2 Development of Activity Diagram Meta-Model

Meta-Model and Meta-Modeling as addressed in detail in section 4.2.3 of chapter 4 , describes the structural aspect of a model and supports the generation of models with the inclusion of relative attributes, classes, and associations. Therefore, in order to generate a model of an Activity Diagram, a meta-model of it is designed in Eclipse (version Mars 2.5.7) (refer section 5.1.2 of chapter 2) with the support of Ecore Tools, which is an Eclipse plug-in. This helps the generation of model instances with the design of meta-models. Basically, an activity diagram comprises of various elements such as actions, decision, final and initial nodes. In this tool, each node is identified as a class in the activity diagram meta-model where aspects such as name and other titles are considered as attributes in respective classes. These nodes may have different types of relationships among each other, which are also defined in the meta-model. The design of the meta-model is graphically shown below.

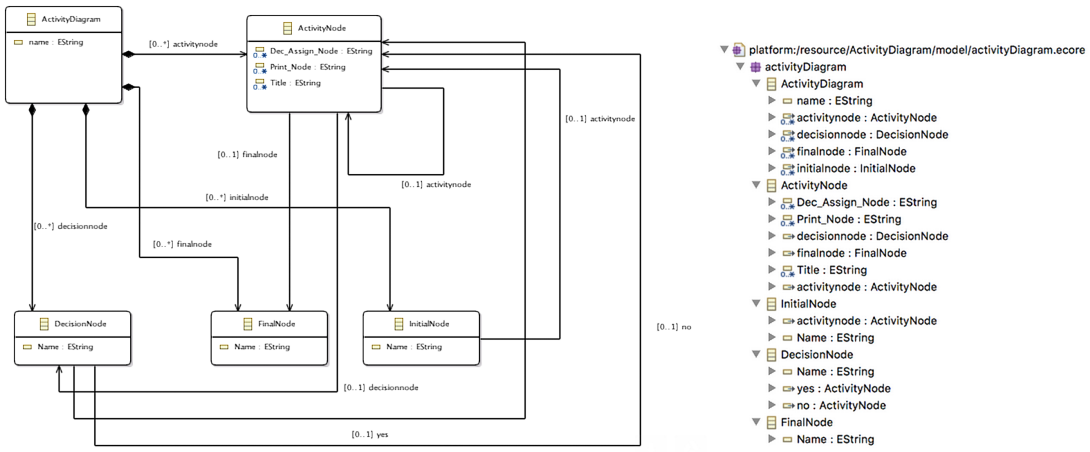

Figure 5.4: Activity Diagram Meta-Model

In addition to the titles of the nodes, the active node consists of two additional

attributes called "*Dec Assign Node*" and "*Print Node*", these attributes are created to match the Java statements which in turn would support the model mapping and later code generation.

As per Figure 5.4, the Left Hand Side (LHS) figure is the class diagram of the activity diagram meta-model as mentioned earlier entails of the classes, associations, and attributes describing the structural perspective of an activity diagram. The Ecore file of the activity diagram on the right-hand side (RHS) depicts a graphical representation of the activity diagram Ecore model via the context menu of an Ecore file.The core file as stated defines the elements of the activity diagram meta-model, these elements include eClass, attribute, eReference, and eDataType. Additionally, Ecore model is basically a root object representation of the complete activity diagram model. As stated earlier and as shown in the activity diagram Ecore file, the model consists of classes with children for every class such as its attributes.

Activity Diagram Node

The activity diagram is regarded as a node in the activity diagram meta-model, due to its need to be generated to a line of runnable code. The activity diagram generally consists of various types of nodes such as the activity, decision, final and initial nodes. Therefore, as per the meta-model, activity diagram node has one-to-many eReference (type of association) relationship with every node since the activity diagram can have many nodes from each type of node mentioned earlier. Moreover, in terms of eAttributes, the class consisting of a single eAttribute called "Name" with the type "String". Even though activity diagram having a name is unusual, this is carried out to support the model-to-model transformation and the ultimate code generation requirements.

Activity Node

Activity node is one of the main elements of the activity diagram which depicts a specific behavior, however, in terms of the proposed tool, the activity node consists of two attributes such as the *Dec-Assign-Node* and *Print-Node* to support the variable assign and declaration and print statement in a Java program. The activity node is designed this way to support the mapping of meta-models which will eventually help the code generation. Furthermore, as shown in the active node (as a class) in the activity diagram meta-model in Figure 5.4, the said node consists of Dec-Assign and print nodes with one-to-many relationships to ensure the user is able to define multiple values via a single variable. This is carried out to avoid repetition of multiple activity nodes in a single activity diagram which may lead to redundancy issues during the modeling of the activity diagram. Furthermore, the node holds a variable type string called "text" holds both values from the dec-assign node and print node. This variable is mainly used during the design of activity diagram model editor, where the user is allowed to store values of both *dec-assign-node* and the *print-node* in the *text-node* variable which is invoked when modeling the activity nodes in the activity diagram. Also, as presented, all the variables in the activity node class are of type ***STRING*** and these values need to key in by the user as shown in the sample below.

- **Dec-Assign-Node**: "int var = 20" or "String var = "action""
- **Print-Node**: "System.out.println(var);"

Initial Node

The initial node is regarded to be the initialization of the flow of activities in an activity diagram. Likewise, the initial node is represented in terms of a class in the activity

diagram meta-model. The shape of the initial node, which is a circle with a dot in it is defined during the design of the activity diagram meta-model (refer to section 5.3). The initial node class consist of an attribute called "***Name***" which is of type "***String***". This is to support the code generation, where the user needs to key in an open parenthesis. This is explained in detail later during the code generation process. In terms of the associations, the initial node can lead a single activity node and not any other type of node. This association is determined in the activity diagram meta-model where the initial node class has a one-to-one association with the activity node class. Furthermore, the activity diagram may have more than one initial node in an activity diagram, this is determined by the one-to-many relationship between the activity diagram class and the initial node class.

Decision Node

The word "decision" adds meaning to this node, in other words, this node basically accepts a value or a set of values of an activity node or 's', makes a specific decision and provides two outgoing nodes, one accepting a scenario and the other rejecting. In simple words, this is an if-statement which if accepted, executes and otherwise proceeds to the else statement. In terms of the associations, the decision node can occur due to one or many activity nodes and therefore may have a one to may relationship with the active node. As every other node, this node also has an attribute called "Name" with type "String" where the user is key in a set of strings such as: "***if (motion-sensor == true)***". This condition checks if the motion sensors are switched on. The execution after the if condition or in other words the decision node may take place in activity node, which is why the decision node may have one-to-many type os associations with the active node.

Final Node

The final node terminates the flow of activities in an activity diagram. The final node in the proposed activity diagram performs the same function, however as every other node class in the activity diagram meta- model, final node class also consists of an eAttribute called "***Name***" by type "***String***", which allows the user to key in a set of strings during the modeling of the activity diagram phase. In terms of the eAssociations of this node, the final node terminates the flow of activities and therefore these two nodes share a one-to-one association between each other.

5.2.3 Development of Java Meta-Model

The Java meta-model addresses features of a Java class, these features include the Java class, Java statements, and if-statements. In order to match the activity diagram meta-model to support the model mapping process, two additional statements are added, they include: begin and end statements. Even though a Java class also comprises of packages, methods other structural aspects, this tool only addresses the flow of statements thus enhancing the dynamic side of the code and the structure. The relationship between the Java statement nodes is corresponding with the activity diagram node associations to support the model mapping process.

Java Meta-Model

As defined previously, meta-model describes the structural aspect of any type of diagram, however, a meta-model for Java programming language is created to support the model mapping process which in turn would support the overall code generation procedure. Furthermore, this meta-model for Java consists of various classes such as the Java class,

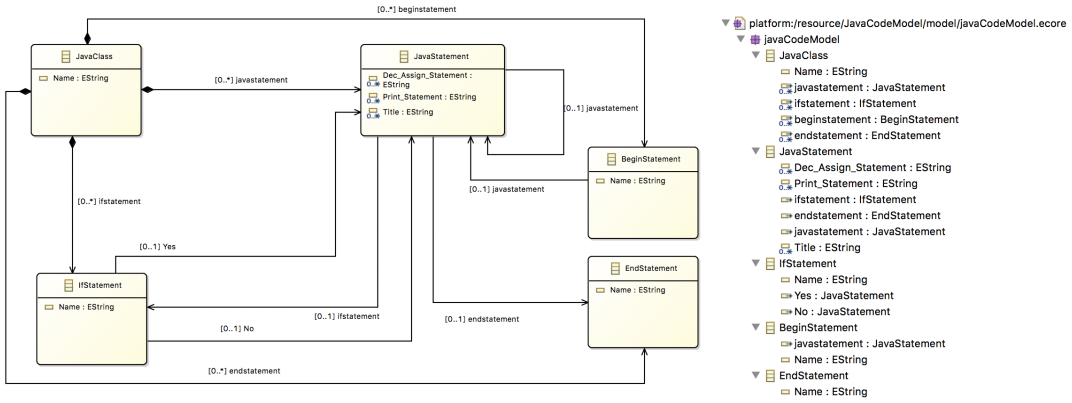

Figure 5.5: Java Class Meta-Model

Java statement, if statement, begin statement and end statement. Even though Java as a programming language when conformed to standard meta-model would consist of Java package, Java class, Java methods, fields, interfaces, and parameters (Favre, 2003), the Java meta-model defined for the proposed tool does not consist of the Java standard meta-model elements. This is due to its need to help the model mapping with the activity diagram meta-model which is based on a very dynamic structure. This section details the elements of the defined Java meta-model.

Java Class Element

As defined by Oracle Java documentation, Java class is a body between the braces where the objects are constructed, initialized and declared, the fields are declared to provide a state for the class and its objects and methods to implement behavior to the class and its objects (Bronner & Olubando, 2009). Although the Java class element defined by the proposed tool does not support the specification of Java objects, Java method is addressed to a very great extent due to its emphasis on the behavioral nature of the Java class overall. As shown in Figure 5.5, Java class element consists of an attribute called "***Name with variable type: String***", this allows the user to give a name to the Java class,

for instance when mapped with Activity Diagram model and generated would generate a Java statement similar to: "*public class **JavaClassName***" which when compared with the standard Java code Java class declaration would look identical.

Java Statement Element

When Java statements are taken into consideration generally, they can object declaration and invoke, variable assignments, variable declarations, method invoking and print statements. However in order to match the Activity Diagram meta-model and the shapes defined during the design of the Activity Diagram model editor, the Java statement therefore as matched consists of three string variables with one-to-many relationships for each variable. A single variable is set for both variable declaration and variable assignment, this variable is called "***Dec-Assign Statement***"which allows the user to both declare and assign values to the variables. Moreover, this element comprises of the ***print statement*** which allows the user to print multiple statements using a single attribute. This element also consists of a title variable which holds both values from Dec-Assign statement and the print statement. This variable is helpful during the code generation where the values from the title variable are called when reading the XMI file.

Begin Statement Element

Even though there is no such element called the begin statement in the standard Java meta-model since there is an element called initial node in a standard activity diagram, the Java meta-model defined for the proposed tool also comprises of a begin statement to map with the initial node of the activity diagram model.A Java class as mentioned previously is surrounded by beginning and ending braces, therefore the beginning statement is regarded to be the beginning brace which in this case is called the "***begin***

statement".

If Statement Element

According to Oracle Java documentation, if-then-else statement is regarded as a control flow statement which allows the execution of a certain statement only if a particular test evaluates to "***true***" and ***otherwise*** the else statement is executed. In the case of the proposed tool, the if statement element consists of a variable called "Name "which allows the user to read the if statement,i.e.: for instance, ***if(light == false)*** and the else statement is supported identified with a new Java statement, this is created with a relationship between the Java statement element and the if statement element (5.5).

End Statement Element

The end statement as stated previously, the Java class is surrounded by the braces, therefore at the end of the Java code in a Java class is ended with a closing brace which in this case is denoted by the "***end statement***".

5.2.4 Meta-Model Configurations

The run configuration functions of Eclipse has supported the configuration of the program when being launched, however, when launching the developed meta-models, they are configured as "***run as an Eclipse Application***" which generates a new instance of Eclipse. This instance allows the launching and debugging of the meta-model by generating a model instance of the meta-model. This function is used to generate a model instance of the activity-diagram model. An example is shown below.

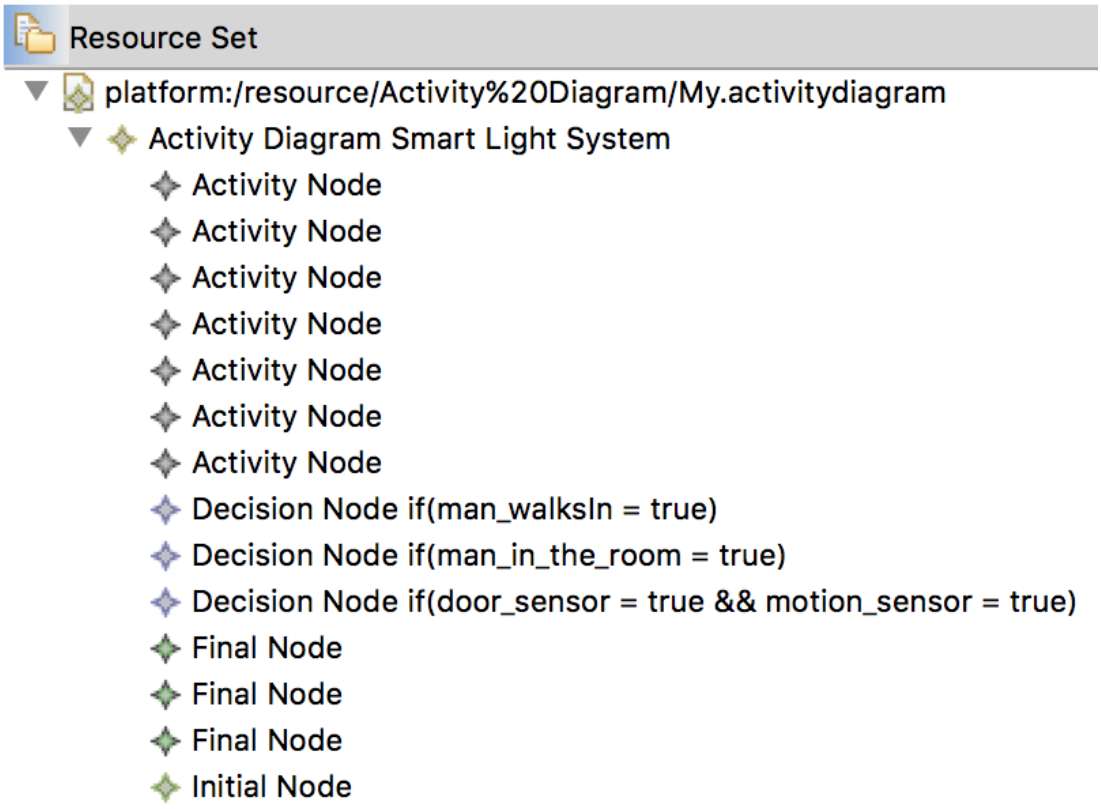

Figure 5.6: Activity Diagram Model Instance

The above-shown activity diagram model instance is used during the modeling of the activity diagram (refer section 5.3) and the model-to-model transformation process where the model instance is used as the input model to the transformation process (refer section 5.4).

Furthermore, the activity diagram model instance shown in Figure 5.6, is a model instance of the smart light system. The activity diagram for the smart light is system is modeled, i.e.: graphically or in other words, diagrammatically represented in Component 5.3 after the build of the activity diagram model editor.

Even though the model instance for the activity diagram meta-model is generated through configuration of the meta-model as a new Eclipse application, the Java model instance is generated with the support of the model-to-model transformation.

This is detailed in Section 5.4.

5.3 Component 2: Model Editor Design

A customized model editor is designed with the selected tools and plug-ins supported by Eclipse (Mars version 4.52).Prior to the design and the development of the stated model editor, the suitable tools are plus-ins are chosen based on defined criteria. This is detailed in section 5.3.1.As mentioned earlier, this section also emphasizes on the design and the development of the model editor. Lastly, the global case study, the smart lighting system is modeled as an activity diagram in the customized model editor.

5.3.1 Selection of Model Editor Design Application

There are various tools, plug-ins, and applications supporting the design and generation of customized app model editors. They include Graphical Modeling Framework (GMF), Sirius, GenMyModel, Enterprise Architect, EcoreTool and Obeo Designer. Out of these, the most compatible and user-friendly tool will be selected for this process. The selection criteria will ensure that the chosen tool supports the achievement of the stated architectural drivers.

When the results from the above table and other factors are taken into consideration, Sirius is an Eclipse project which allows the design and generation of customized model-editor by leveraging Eclipse modeling technologies thus ensuring 100 percent compatibility with Eclipse. For the design of the model editor, both Sirius and Ecore tools are used due to its support in building meta-models and model editors. Even though GMF is an Eclipse plug-in, it has left the Eclipse train three years ago thus when used now may lead to availability, up-to-date and compatibility issues (Pelechano,

Criteria	Definition
Usability	Degree to which the model editing designer supports in achieving the stated architectural drivers,i.e.: in this case the ability of the developer to design and build a customized UML model editor(Shackel, 1991).
Up-to-Date	The chosen plug-in or tool must be up-to-date to achieve improved characteristics.
Availability	The plug-in or the application should be available at all times to support the building and designing of the model editor (Mohilo, 2017).
Compatibility	The plug-in or tool should be compatible with Eclipse (version Mars 4.5.2) top support further development of the tool (Mason & Criswell, 1998)
Customizable	Allows customizations of the model editor (Steinberg et al., 2008).

Table 5.3: Selection Criteria: App Modeling Editor Application

Criteria	GMF	Enterprise Architect	Sirius	GenMyModel	EcoreTools
Usability	✓	✓	✓	✓	✓
Up-to-Date	×	✓	✓	✓	✓
Availability	×	×	✓	×	✓
Compatibility	×	×	✓	×	✓
Customizable	✓	✓	✓	✓	✓

Table 5.4: Selection of App Modeling Editor Application

Albert, Muñoz & Cetina, 2006).

5.3.2 Design of Activity Diagram Model Editor

Even though there are existing model editors supporting the design of UML diagrams, a customized model editor is designed and generated to support the model confirmation and generation of model instances which are required for the model mapping and code generation processes. Therefore, as mentioned earlier, a customized model editor is designed with the support of Sirius (Eclipse plug-in). The model editor consists of a viewpoint (rfefer section 4.2.3 of chapter 4) which supports the connection of the activity diagram meta-model designed in this section and the model editor. Moreover,

viewpoint in Sirius provides a set of representations, in this case, the representation is called the "***activitydiagram***" which is synchronized with the activity diagram meta-model built-in component 1 (refer section 5.3).The activation of this viewpoint allows creating and editing of the corresponding activity diagram in the activity diagram model editor. (This includes the creation of both nodes and the edges),

Furthermore, the activity diagram nodes are given different shapes such as a rectangle for the activity, a diamond for the decision node and dot for the initial and final nodes. Also, the relationships among these nodes are designed with the help of ***relation based edges***. In order to help the user add these nodes to the modeling area, a ***palette (section)*** is designed with the listed nodes and their associations. The design of the model editor is shown below.

From the developed model editor design, shown in figure 5.6, the activity diagram model editor is generated as shown in figure 5.7. As discussed earlier, the nodes from the activity diagram meta-model are synchronized with the model-editor design using the "***activitydiagram***" viewpoint. When the associations are taken into consideration, there are six relation based edges defining the relationship between these nodes.

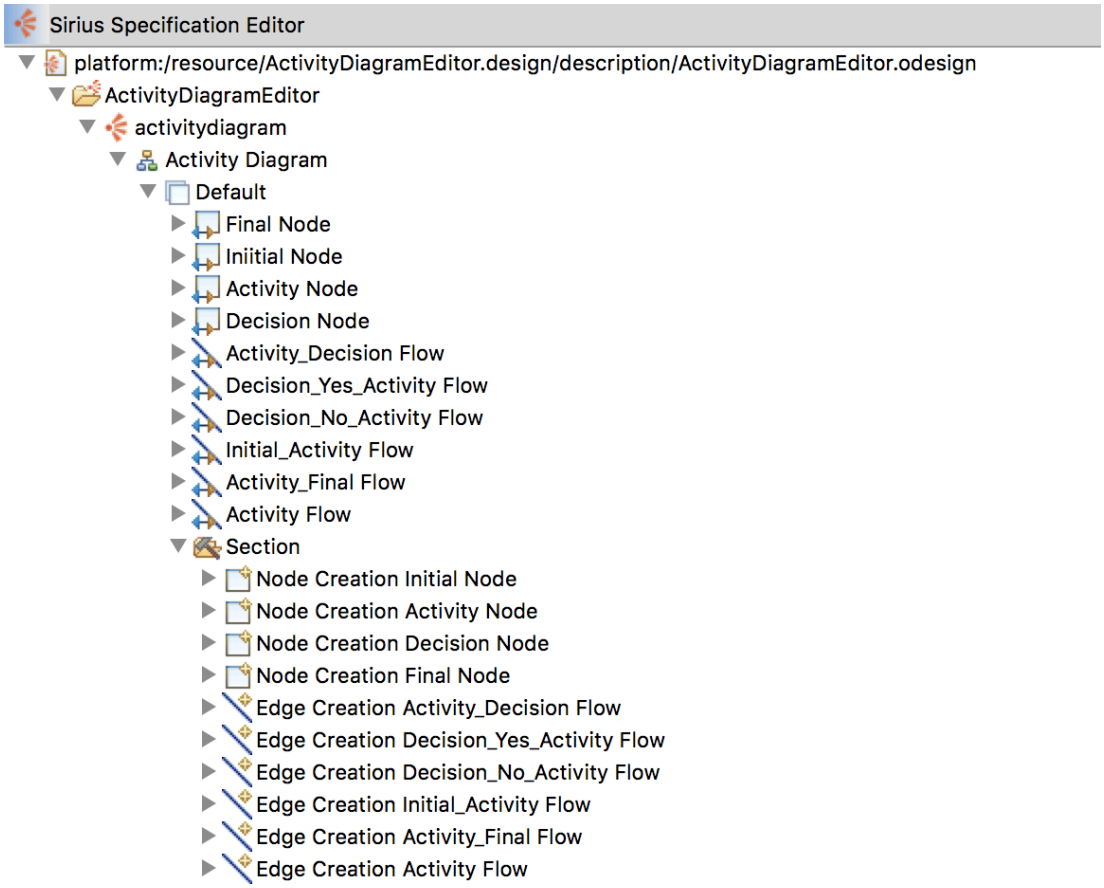

Figure 5.7: Activity Diagram Model Editor Design

The associations shown in figure 5.7 are defined in the table below.

5.4 Component 3: Model-to-Model Transformation

Model transformation plays an important role in Model Driven Engineering (MDE), in another word, MDE promotes the use of models and regards the models to be the primary artifacts that drive the complete software development process (Jouault, Allilaire, BÃ©zivin & Kurtev, 2008). Furthermore, MDE involves a series of model transformations over models, as per (Jouault, Allilaire, BÃ©zivin & Kurtev, 2008), model transformation is defined as a set of refinement steps over models which decrease

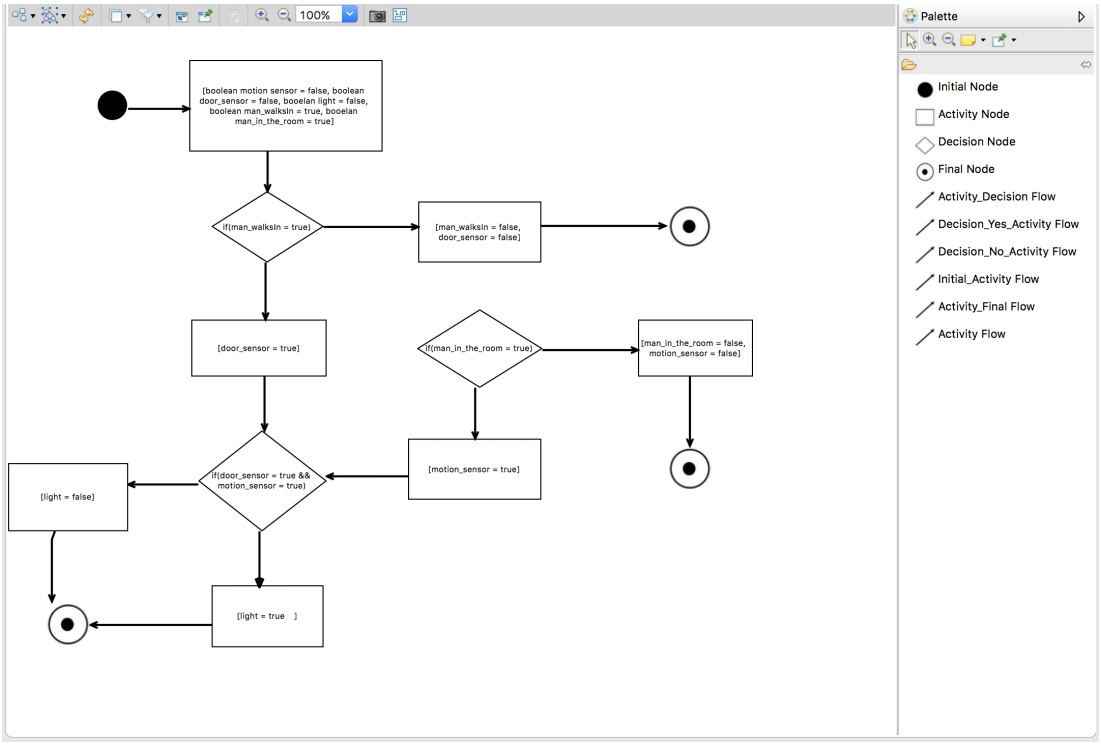

Figure 5.8: Activity Diagram Model Editor

the level of abstraction. The goal of model transformation is to generate a new model that contains sufficient details to support generation of executable code. Model transformation follows a specific pattern called the "***model transformation pattern***", this pattern is applied in the model transformation process when building the proposed tool. This component, therefore, emphasizes the language, patterns and the tools used in the model transformation process in order to generate a model that supports automatic runnable code.

5.4.1 Selection of Model-to-Model Transformation Language

As explained in section 4.2.3 of chapter 4, Model-to-Model transformation technique is used to map the Activity Diagram model to the Java model which will eventually support

Association	Definition
Activity-Decision Flow	Defines the association between activity and decision node, The activity node may have one decision node as shown in the metamodel. The activity node can lead to a single decision node as shown in the activity diagram modeled in the activity diagram model editor.
Decision-Yes-Activity Flow	Decision Node can have two activity nodes, i.e. activity node stating the yes node and an else node as per Java. Therefore, decision node can lead to one of the activity nodes with the decision-yes relationship.
Decision-No-Activity Flow	Also, decision node can lead to another activity node with the decision-no edge which explains the else aspect of a decision of an activity diagram.
Initial-Activity Flow	Initial node in activity diagram can only lead to and have one activity node thus making it have a one-to-one relationship(reference kind of association) with activity node (5.5).
Activity-Final Flow	Activity Node can only have or lead to one final node, this is shown clearly in Figure 5.4.
Activity Flow	This describes the association between the activity nodes in an activity diagram, where one activity node can have a one-to-one relationship with another activity node.

Table 5.5: Association Definitions

the generation of software code. There are various languages which support the model-to-model transformation method, they include ATL, (Atlas Transformation Language), ETL(Epsilon Transformation Language), Kermata, QVT (Query View Transformation) and Graph Rewriting and Transformation (GReAT). The transformation language is chosen based on criteria which address the ability to map models, ability to compile and debug the code, traceability, expressibility, and simplicity.

As per the above table, ATL is chosen to map and transform models since it tends to meet all the criteria. Also, ATL is a plug-in supported by Eclipse which in turn may support the design and development in term of compatibility. Even though ETL is a plug-in supported by Eclipse, this tends to be weak in terms of traceability and expressibility (Samimi-Dehkordi et al., 2014). Furthermore, in terms of simplicity, ATL and ETL are considered to be the two most simple languages during the model transformations. GREaT is an independent tool which allows the generation of EMF

Criteria	Definition
Ability to map models	Ability of the language to create a mapping between the models (Brown, 2004).
Ability to compile and debug	Ability to compile, debug and generate runnable code to support model transformation (Jouault, Allilaire, Bézivin & Kurtev, 2008)
Traceability	Ability to create a link between, the defined models (Samimi-Dehkordi, Khalilian & Zamani, 2014)
Expressibility	Ability to specify transformation requirements of models (Samimi-Dehkordi et al., 2014)
Simplicity	Determined by the number of basic constructs, a small number of of constructs lead to a simple language (Samimi-Dehkordi et al., 2014).

Table 5.6: Selection Criteria: Model-to-Model Transformation Language

Criteria	ATL	ETL	Kermata	QVT	GReAT
Ability to map models	✓	✓	✓	✓	✓
Ability to compile and debug	✓	×	✓	✓	✓
Traceability	✓	×	✓	×	✓
Expressibility	✓	×	✓	×	×
Simplicity	✓	✓	✓	×	✓

Table 5.7: Selection of a Model-to-Model Transformation Language

models, however, this tool cannot be used with Eclipse which can lead compatibility issues when building the proposed tool (Whittle, Clark & Kühne, 2011).

5.4.2 Overview of ATL Transformation Process

This section describes the model transformation pattern used in the transformation process. This pattern consist of two types of models known as the "***source model***" and the "***target model***". The source model is the model that is being transformed and the target model is the model to which the source model is transformed in to. ATL meta-model, in this case, source model and the target mode are confirmation of a model. Additionally, all the meta-model in this process conforms to MOF. In the case of the proposed tool, the source model is the Activity Diagram meta-model (refer figure 5.4)

and the target model is the Java meta-model (erefer figure5.5). The Activity Diagram meta-model, which is the source model is the conformation of the activity diagram model instance (refer figure 5.6). These models together with the support of MOF and ATL generates a new model called the "***Java Model***" which can be easily transformed to executable code automatically. Therefore, this transformation pattern is used to transform the activity diagram model (source model) to a Java model (target model) according to the transformation definition "***''ActivityDiagramModel2JavaModel.atl***" written in ATL language. The pattern of the transformation is shown below.

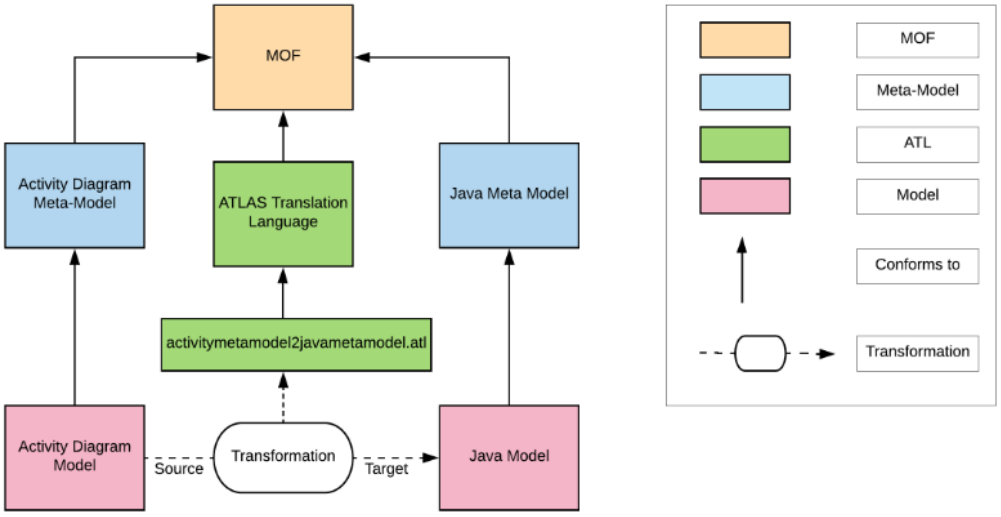

Figure 5.9: Overview of ATL Transformation Process

As per figure 5.9 and (Jouault, Allilaire, BÃ©zivin & Kurtev, 2008), all the meta-models conform to MOF (Meta Object Facility).MOF is an OMG meta-modeling and metadata repository standard. It is an extensible model-driven integration framework used in defining, manipulating and integrating meta-data (Tang, 2009). Likewise, the models depicted in purple rectangles conform to the meta-models shown in light blue rectangles which in turn conform to MOF. The Java model, which is the target

model is generated with the support of ATL rules written in the "ActivityDiagram-Model2JavaModel.atl file. In other words, the activity diagram meta-model (the source model) is used as the input meta-model, see MM in figure 5.9, the Java meta-model is used as the output meta-model (refer to MM1 in Figure 5.9 and the Activity Diagram model is used as the input model (refer to IN in Figure 5.9). Figure 5.10 represents the configuration of the ATL approach used to generate the Java model from activity diagram model instance (refer figure 5.6).

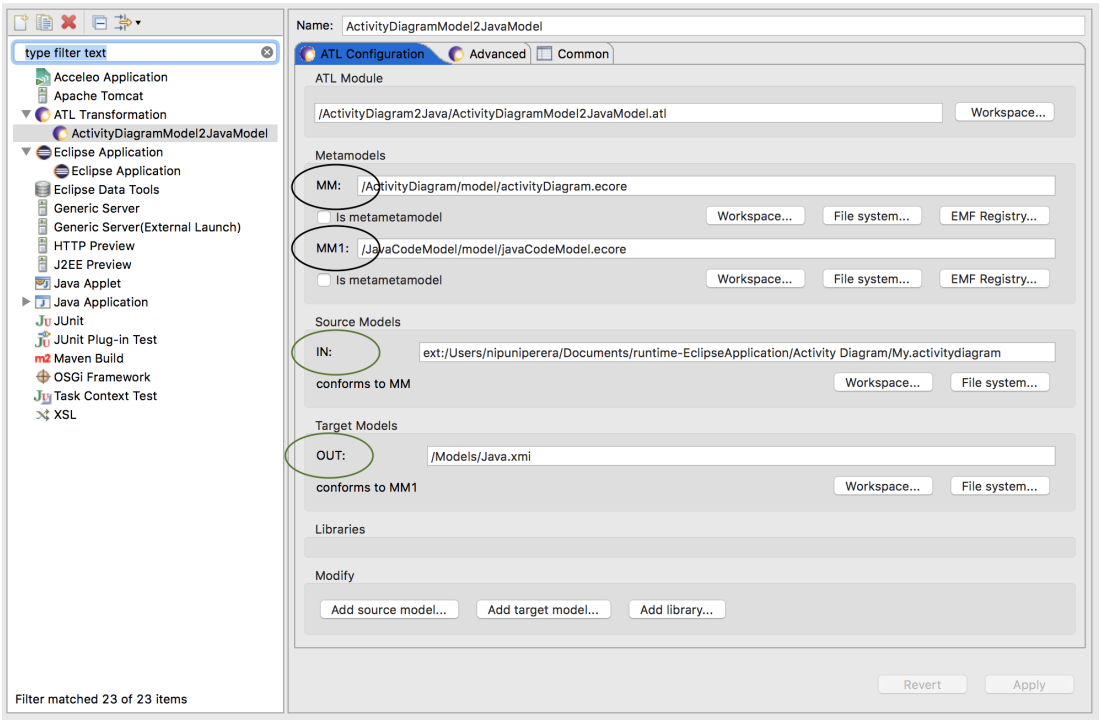

Figure 5.10: ATL Configurations Window

In addition to the use of models and meta-models for the transformation process, a set of ATL rules need to be written to map the elements of Activity Diagram Meta-Model (MM) and Java meta-model (MM1).The classes, attributes and associations of the Activity Diagram Meta-Model (refer figure 5.4) needs to be mapped with the classes, attributes, and associations of the Java Meta-Model (refer figure 5.5). The mapping of activity diagram nodes and Java classes are shown in the figure below.

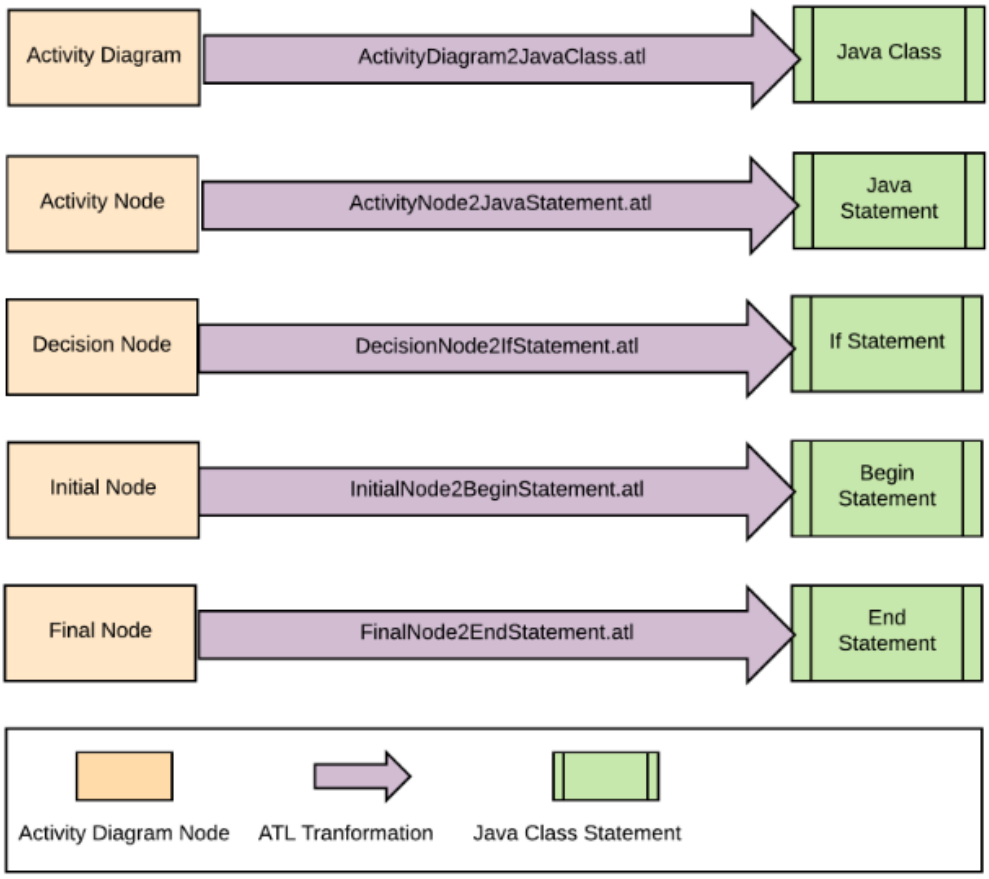

Figure 5.11: ATL Mapping Approach

As shown in Figure 5.11, each activity diagram node is mapped to a corresponding Java statement through an ATL file. Each Activity Diagram node is represented as a class in the Activity Diagram Meta- Model, these classes consist of attributes. These classes, attributes, and associations are mapped with the corresponding Java Classes in the Java class Meta-Model. Each mapping is detailed individually in this section.

5.4.3 Transformation of Activity Diagram Model to Java Class Model

```
-- @path MM=/ActivityDiagram/model/activityDiagram.ecore
-- @path MM1=/JavaCodeModel/model/javaCodeModel.ecore

module ActivityDiagramModel2JavaModel;
create OUT : MM1 from IN : MM;

--Transforms Activity Diagram to Java Class
rule ActivityDiagram2JavaClass {
    from
        input_name : MM!ActivityDiagram
    to
        output_name : MM1!JavaClass (
            Name <- input_name.name
        )
}
--Transforms Activity Node to Java Statement
rule ActivityNode2JavaStatement {
    from
        input_name : MM!ActivityNode
    to
        output_name : MM1!JavaStatement (
            Title <- input_name.Title,
            Dec_Assign_Statement <- input_name.Dec_Assign_Node,
            Print_Statement <- input_name.Print_Node,
            endstatement <- input_name.finalnode,
            ifstatement <- input_name.decisionnode,
            javastatement <- input_name.activitynode
        )
}
```

```
--Transforms Decision Node to If Statememt
rule DecisionNode2IfStatement {
    from
        input_name : MM!DecisionNode
    to
        output_name : MM1!IfStatement (
            Name <- input_name.Name,
            No <- input_name.no,
            Yes <- input_name.yes
        )
}
--Transforms Initial Node 2 Begin Statement
rule InitialNode2BeginStatement {
    from
        input_name : MM!InitialNode
    to
        output_name : MM1!BeginStatement (
            Name <- input_name.Name,
            javastatement <- input_name.activitynode
        )
}
--Transforms Final Node to End Statement
rule FinalNode2EndStatement {
    from
        input_name : MM!FinalNode
    to
        output_name : MM1!EndStatement (
            Name <- input_name.Name
        )
}
```

Figure 5.12: ATL Transformation Rules

As per figure 5.12, ATL rules are written for the transformation of every activity diagram node to Java class statement. ATL as a hybrid transformation language is a blend of imperative and declarative constructs. However, the declarative style is used to write the transformation rules due to: its ability to specify relations between the source and the target models and its ability to encapsulate complex transformation algorithms behind a complex system (Jouault, Allilaire, BÃ©zivin & Kurtev, 2008). The target model, i.e.: Java model is generated from the configuration of the ATL rules shown in figure 5.12. Moreover, the Java model generated, which is an XMI (eXtensible Meta Interchange) format. This file format is mostly used to exchange UML diagram model design information. The activity diagram model instance is also an XMI file which when generated creates a new Java file, which is a corresponding XMI file matching the mapping rules written using ATL. The generated Java model instance (in XMI format) is shown below, simultaneous to the respectively matching activity diagram model instance.

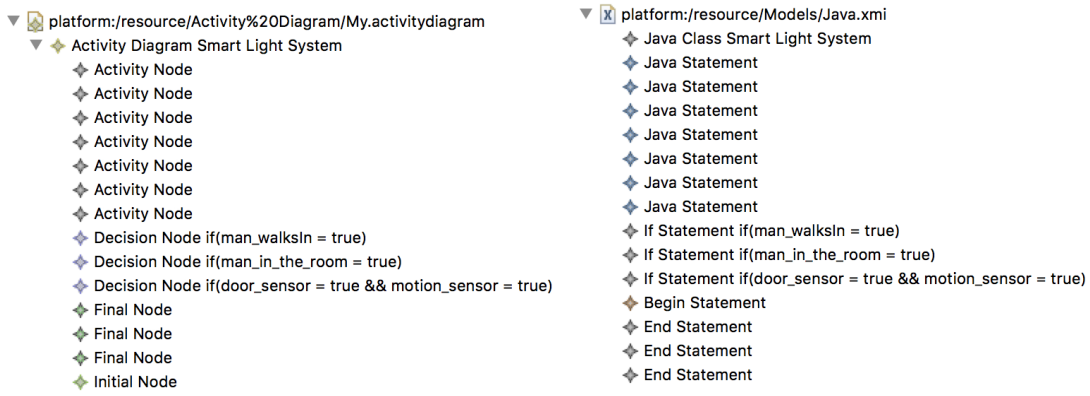

Figure 5.13: Java Model Instance Generation

5.5 Component 4: Model to Text Transformation

As explained in detail in section 4.2.3, the Model-to-Text transformation method is used in creating the tool to support the code generation from model generated during the mode-to-model transformation (refer figure 5.13). Three most significant projects supported by Eclipse to generate text from model to include: Acceleo, Xpand, and Jet. However, these code generating projects support the code generation from meta-models (Schamai, Fritzson, Paredis & Pop, 2009), whereas this proposed tool requires a software code generated from a model instance (this will be explained in detail with the support of snapshots from the tool later in this chapter). The selection criteria are shown below.

However, in order to generate Java Code from the model instance which is in XMI format, the file is read using a Java code, line by line where each line comprises of a tag name which corresponds with the type of the statements in software code model. Every line is written to a Java file which will be in the format to execute and debug immediately. Furthermore, in terms of the programming language to which the model will be translated, the models will be transformed to Java code due to its improved level in terms of ease, learn-ability, compilation and debugging functions.

5.5.1 Selection of Model-to-Text Transformation Technique

Various techniques are used to translate models to text. These techniques are mentioned earlier and in order to evaluate and choose the most suitable method.This is achieved by defining criteria against which each existing techniques are assessed. The existing approaches are evaluated in terms of various areas such as ability to translate the Java models generated in section 4.2.3 of chapter 4,to understandable, executable and runnable code, the ability of the model transformation approach to translate behavioral app design models to code easily, the ability to generate code from any case study modeled in the model editor (refer figure 5.8) thus meeting both interoperability and compatibility features of the tool.

Criteria	**Definition**
Ability to generate Java code that meets the defined criteria	Approach to generate Java code that is executable and understandable (refer section 4.1.1 of chapter 4).
Ability to generate code from behavioral models such as activity diagrams.	Ability to transform activity diagram model instances (refer figure 5.6) to runnable Java code easily (Baker, Loh & Weil, 2005).
Optimization and refactoring of the generated code	Ability to achieve improved performance during the model to text transformation process and the ability to generate the internal structure of the tool in order to achieve other system requirements such as understandability, modifiability, code-red usability,modularity and adaptability without having to change the obvious behavior of the app design model (Mens & Van Gorp, 2006)
Simplification and Normalization	Helps to achieve reduced levels of syntactic complexity by translating Java model instances into a more primitive language construct (Java code) (Mens & Van Gorp, 2006).
Ability to customized the code	In order to have their own smart home environment, the smart home application needs to be customizable which in turn requires customizations in the code generated from the tool (refer section 4.1.1 of chapter 4).

Table 5.8: Selection Criteria: Model-to-Text Transformation Approach

Based on the above-defined criteria, there are Model-to-Text transformation approaches such as Acceleo, ATL, JTL (Janus Transformation Languages), ETL (Epsilon Transformation Language), Kermata Language, Acceleo and Xtend (Erata,

Challenger & Kardas, 2015).

Criteria	ATL	ETL	Acceleo	Xtend	Java
Ability to generate Java code that meets the defined criteria	✓	✓	×	×	✓
Ability to generate code from behavioral models	✓	×	×	✓	✓
Optimization and refactoring of the generated code	✓	×	✓	×	✓
Simplification and Normalization	✓	×	✓	×	✓
Ability to customized the code	×	✓	×	✓	✓

Table 5.9: Selection of a Model-to-Text Transformation Approach

As per the above table, various model-to-text transformation approaches are evaluated in order to choose the most suitable one to be used to generate Java code from the Java Model Instance (refer figure 5.13). Therefore, according to the above results, the most suitable technique seems to be the best choice since it meets every criterion. Most importantly, the transformation technique needs to transform behavioral models such as UML Activity Diagrams to Java code. Most of the transformation techniques mentioned above only support code generation from structural models such as class diagrams (Erata et al., 2015).

Even though ATL has the ability to support translation of behavioral models such as Activity Diagrams to Java code, it still leads to other drawbacks such as the ATL not being very supportive during incremental model transformation procedures, therefore all though a complete source model is used to generate a complete target model (the generated source code), if modifications are made to the target model they may not be preserved (Biehl, 2010), this, in turn, may affect the customizable feature. During the development procedure, ATL was first used to generated Java code from model instance, however due to various reasons such as the tag names in the model instance being different and the lack of time to change the underlying meta-models and relative model artifacts in order to support the code generation using ATL, reading the

model instance using Java was chosen.

Reading the model instance which is in the format of XMI using Java as the programming language to read the file meets every criteria above and therefore is used to transform the Java model instance to executable Java code.

5.5.2 Reading Java Model Instance

The target model generated from the Model-to-Model transformation process, i.e.: Java instance model (refer figure 5.15) which is in XMI format is used as the input to the Java code. This code reads the Java model instance and generates a set of lines of Java code which meets the primary functional requirements in relation to the automatic code generation aspect (refer section 4.1.1 of chapter 4).

```
try{

    DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
    Document doc = docBuilder.parse(new File("/Users/nipuniperera/Documents/ActivityDiagram/ReadJavaModel/JavaModel.xml"));

    //normalize text presentation
    doc.getDocumentElement().normalize();
    System.out.println("Root element of the doc is :" + doc.getDocumentElement().getNodeName() );

    NodeList listofJavaStatements = doc.getElementsByTagName("*");
    int totalJavaStatements = listofJavaStatements.getLength();
    System.out.println("Total Number of Java Statements :"+ totalJavaStatements);

    NodeList statementList = doc.getElementsByTagName("javaCodeModel:JavaStatement");
    int nofStatements = statementList.getLength();
    System.out.println(nofStatements);
```

Figure 5.14: Identifying Java Model Instance and Instance Tags

As per Figure 5.14, the DocumentBuilderFactory API is used to obtain a parser that generates DOM (Document Object Model) object trees from XML documents (i.e.: the Java instance model). Afterwards, the DocumentBuilder API is used to obtain the DOM document instances from the XML document (Java instance model). This class is then used to obtain a Document from XML. The Document API is then used to represent the complete XML document (R. Johnson, 2005). Therefore, the use of previously mentioned APIs is used to identify the Java Instance model which is later

read using Java code.

Furthermore, the tags in the Java model instances are varied due to the use of the different types of statement such as Java Statement, If statement, begin and end statements. Therefore for instance (Java Statement is generated as **<javaCodeModel:JavaStatement>** and if statement is generated as **<javaCodeModel:IfStatement>**) each tag is of a different names. The list of tags in the Java Instance Model is regarded as a NodeList (helps to create an ordered list of nodes) when read by Java and in order to read all the nodes which are of different types, the method **getElementsByTagName()** is used, where the tag name is regarded as "*" which helps to read all the tabs in the Java model instance XMI file.

The node type of <javaCodeModel: JavaStatement> is read individually out of which a NodeList is created. This is due to the Java Statement Node having multiple tags inside its own which need to be read during an if statement. This is emphasized further in figure 5.14.

The Java code illustrated in figure 5.14 goes through all the nodes in the Java Model Instance via a for loop with the size of the for loop being defined earlier, which is equal to the total number of nodes in Java model instance. As shown in figure 5.14, the nodes in the XML file (Java model instance) is cast to Element interface in order to acquire all the attributes stored inside a specific node (in XML terms, tags). Furthermore as illustrated in the code below (refer figure 5.15), as the node is cast as element interface, this interface allows the use of methods such as "***node.getNodeName()***" and allows comparisons with the support of the ***equal("String") method*** which eventually has allowed to complete comparisons during if statements.

Moreover, a node list called ***statementList*** is created to obtain the Java Statement tags. This node list is created to call the Java statements which are assigned in an

if condition. The if execution and else execution statements which are both of the type of "Java Statement" are written as, for instance: ***Yes = "/1" No = "/3"***. In this case, a for loop is executed which takes only the digit in the attribute value into consideration (1 or 3) and the node index is set to that specific digit, from which the node details are obtained with the support of the getTextContent() method.

```
        for(int i = 0; i< totalJavaStatements ; i++){
            Node node = listofJavaStatements.item(i);
            if(node.getNodeType() == Node.ELEMENT_NODE){
                Element eElement = (Element) node;
                if(eElement.getNodeName().equals("javaCodeModel:JavaClass")){
                    System.out.println("public class"+" "+eElement.getAttribute("Name"));
                }else if (eElement.getNodeName().equals("Dec_Assign_Statement")){
                    System.out.println("\t"+ eElement.getTextContent()+ ";");
                }else if(eElement.getNodeName().equals("javaCodeModel:BeginStatement")){
                    System.out.println("{");
                }else if(eElement.getNodeName().equals("javaCodeModel:EndStatement")){
                     System.out.println("}");
                }else if(eElement.getNodeName().equals("javaCodeModel:IfStatement")){
                    System.out.println(eElement.getAttribute("Name")+"{");
                    String temp1 = null;
                    String temp2 = null;
                    //print the yes statements
                    for(int index = 0; index < nofStatements ; index++){
                        char aChar = eElement.getAttribute("Yes").charAt(1);
                        String temp = Character.toString(aChar);
                        int result = Integer.parseInt(temp);
                        for(int a = 0; a< nofStatements; a++){
                            temp1 = statementList.item(result).getTextContent();
                        }
                    }
                    System.out.println(temp1 +";");
                    for (int b = 0; b < nofStatements; b++){
                        char aChar = eElement.getAttribute("No").charAt(1);
                        temp2 = Character.toString(aChar);
                        int result1 =  Integer.parseInt(temp2);
                        for (int c = 0; c < nofStatements; c++){
                            temp2 = statementList.item(c).getTextContent();
                        }
                    }

                    System.out.println("else"+temp2+ ";");
                }
            }

            }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Figure 5.15: Readind Tags in Java Model Instance

5.5.3 Code Generation from Java Instance Model

As stated in the previous section, the Java code corresponding to the Java model instance is generated with the execution of the Java code illustrated in Figure 5.15. The Java code generated for the smart light system is shown below.

```
public class SmartLightSystem {

    @SuppressWarnings("unused")
    public void activity() {
        boolean light = false;
        boolean man_walksIn = true;
        boolean door_sensor = false;
        boolean man_in_the_room;
        {
             if(man_walksIn = true){
                door_sensor = true;
                boolean motion_sensor;

                    if(man_in_the_room = true){
                        motion_sensor = true;
                        if(door_sensor = true && motion_sensor == true){
                            light = true;
                        }else light = false;
                    }else {
                        man_in_the_room = false;
                        motion_sensor = false ;
                    }
            } else {
                man_walksIn = false;
                door_sensor = false;
            }
        }
    }
}
```

Figure 5.16: Java Code Generated from Java Model Instance

Java code generated which is the ultimately expected result from the automatic translation tool is achieved. Even though the most significant and the important goal in relation to the tool is achieved, as mentioned in primary functional requirements in Chapter 4, the generated tool required certain set of evaluations in order to achieve the stated requirements. This is carried out in detail in Chapter 6 where each part of the

automatic translation tool is evaluated based on defined evaluation criteria (refer section 3.3 of chapter 3).

5.6 Conclusion

This section details the development aspect of the tool, the selection of different technologies supporting the design of models, model editors and model mapping and finally code generation. These selections are based on defined selection criteria, and these definitions are also proved with valid literature findings. Furthermore, in addition to these selections, other rationals are provided to give reasons as to why these specific technologies are chosen. Subsequently, each phase during the development of the Automatic Translation Tool is emphasized where each phase is implemented based on the previously defined smart home case study, the smart lighting system. Even though smart lighting system is demonstrated using the Automatic Translation Tool throughout this thesis, other smart home systems such as the security and door lock system will be detailed during the evaluation of the tool. The figure below provides an overview of the complete tool.

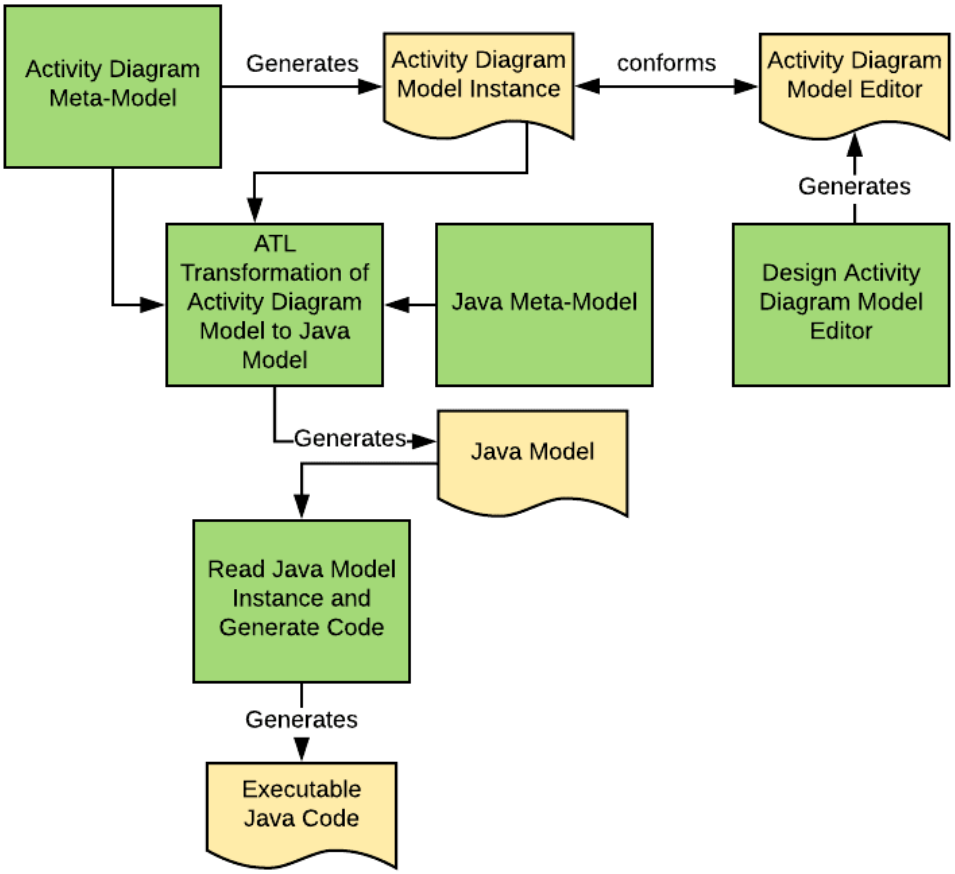

Figure 5.17: Overview of Automatic Code Generation from UML Activity Diagram

Chapter 6

Evaluation

This chapter contains the evaluation of the Automatic Translation Tool. Evaluation criteria are defined based on the quality attribute requirements and primary functional requirements from Chapter 4. Three success criteria are defined to evaluate usability, availability, and performance of the tool. Each success criteria experience a certain number of steps which are followed and used to evaluate the automatic translation tool.

This chapter is divided into few sections which perform very significant functions in order to carry out a well-documented evaluation process for the Automatic Translation Tool. Section 6.1 provides reasons as to why the Smart Lighting System had been used throughout this thesis to demonstrate the Automatic Translation Tool. Section 6.2, details the success criteria based on the modeling and the code generation aspects. Subsequently, the Automatic Translation Tool is evaluated based on the success criteria where findings, results, and drawbacks have been emphasized. This chapter concludes summarizing the evaluation results, data analysis, and discussions of the results.

6.1 Smart Home Case Studies

This part of the thesis identifies different smart home systems in addition to the Smart Lighting System. These Smart Systems include Smart Security, Smart Door Lock, and Smart Weighting Systems. These case studies are modeled using the Automatic Translation Tool.

6.1.1 Smart Lock System

This case study involves activities, decision nodes, initial node (in this case, called the initiator) and final nodes known as the terminators. Initially, when the person walks to the door, the door sensor is activated. The decision node checks if the person present at the door is a resident of the house, this is ensured by the person swiping a card for entrance or attempting to unlock the door with the smartphone. This attempt by the person leads to an authentication of the swipe card or the mobile phone. If authenticated, the control module in the smart lock which has a motor in itself (which also acts as the actuator) is activated which in turn activates the door lock and unlocks the door. If the authentication is denied, the process is terminated automatically. In the meanwhile, if the person is not a resident, the doorbell camera, microphones and the speakers are activated. The person is allowed to enter a customer entry code into the keypad which is checked for authentication. If authenticated, the door is unlocked and otherwise, the visitor is connected to the visitor through a smart phone, doorbell camera, microphone, and speakers. Later access to the smart home is granted based on the residents' preference and the process is terminated.

Figure 6.1 depicts the smart lock system graphically.

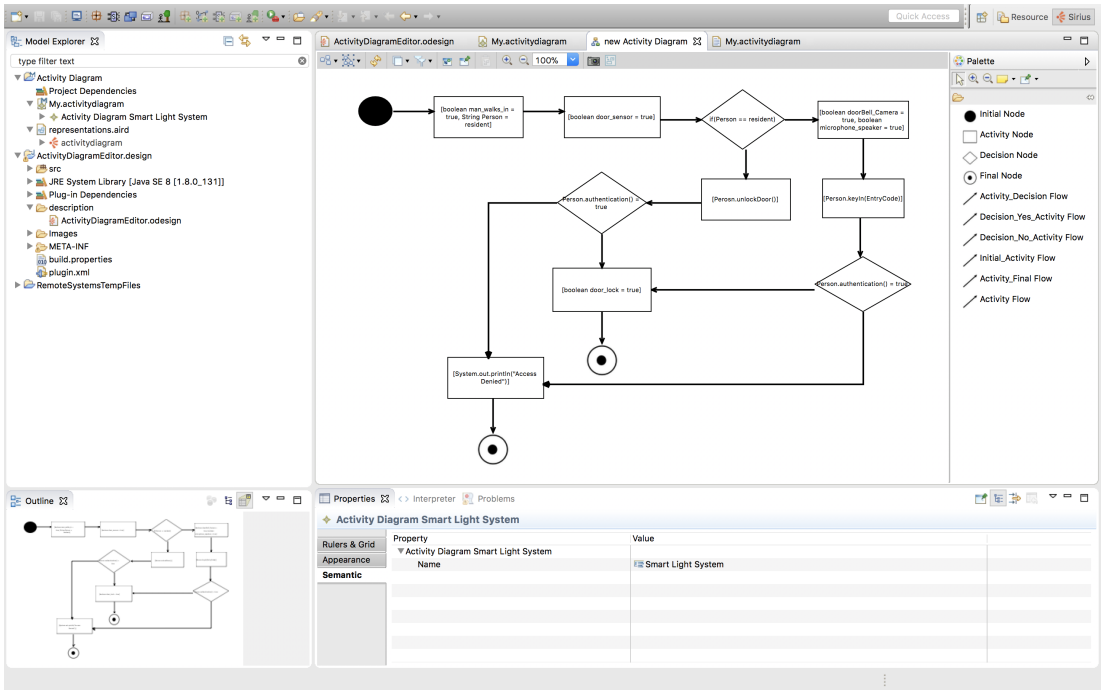

Figure 6.1: Smart Lock System

6.1.2 Smart Security System

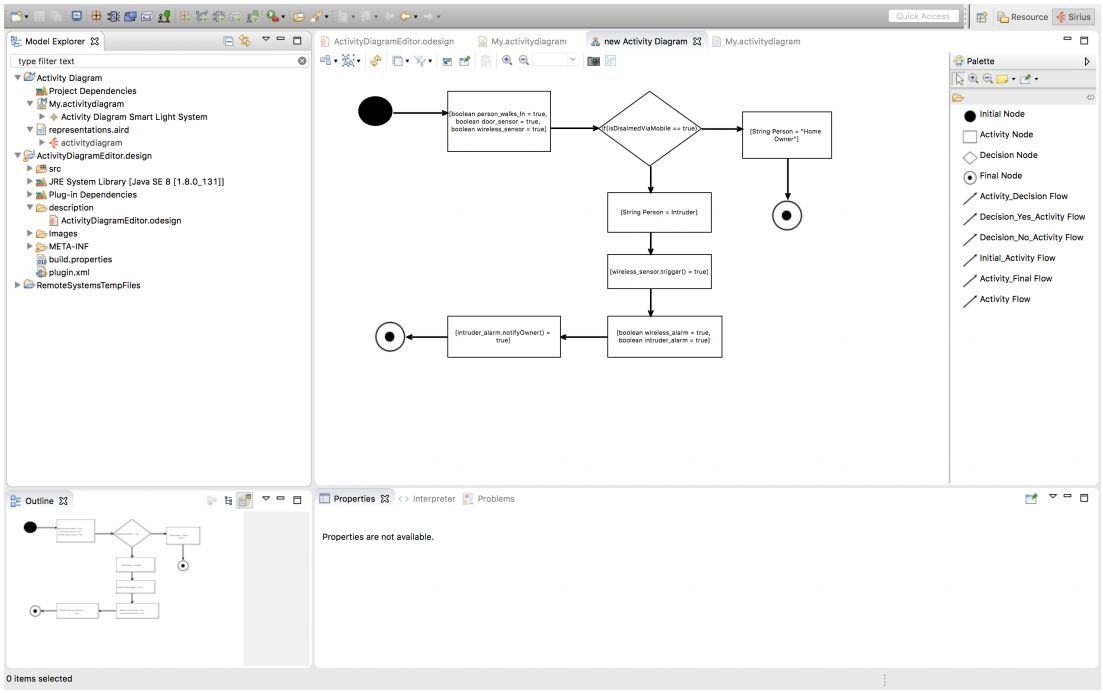

Figure 6.2: Smart Security System

In this scenario, the door sensors and wireless sensors are activated when the person walks to the main door. If the home security system is disarmed via mobile, the person is regarded as the homeowner and therefore the process is terminated. Otherwise, the person is regarded as the intruder. This, in turn, leads to the activation of the wireless sensors and alarms. The intruder alarm then notifies the homeowner and the police via mobile.

6.1.3 Smart Light System

In this case, assuming the person walks to the door, the system checks if the door is open. If he doesn't open the door,the process is terminated. Otherwise, the door sensor is activated. If the person walks into the room, the motion sensor are activated which in turn allows the power outlet to switch on the lights on. Otherwise, the lights and the motion sensor remain deactivated thus terminating the process.

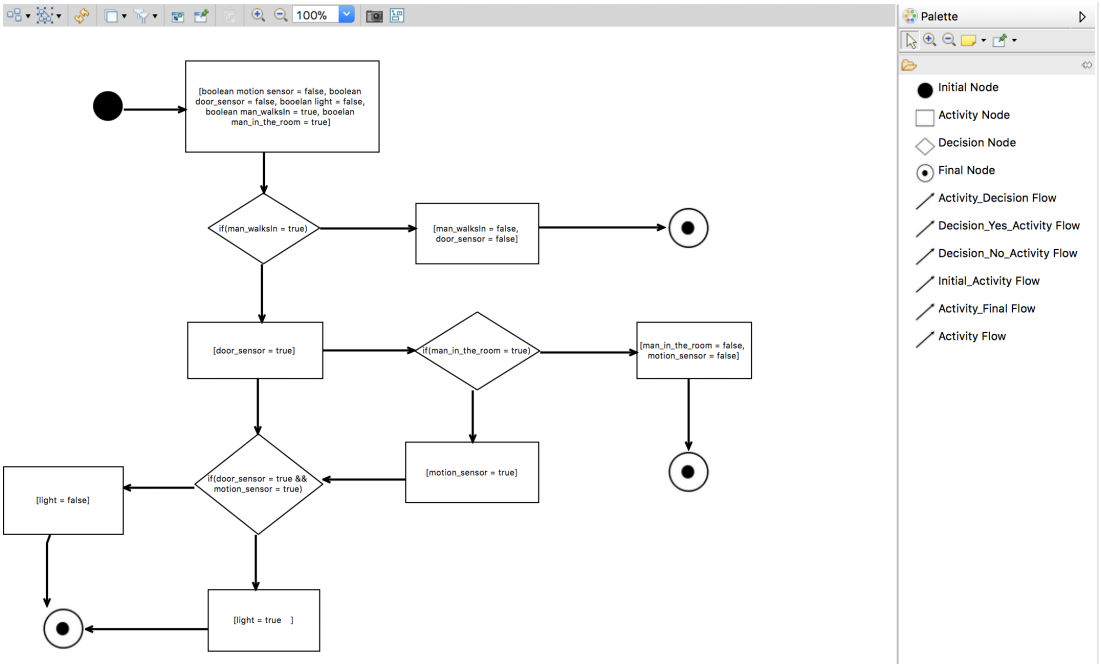

Figure 6.3: Smart Light System

6.1.4 Smart Weighting System

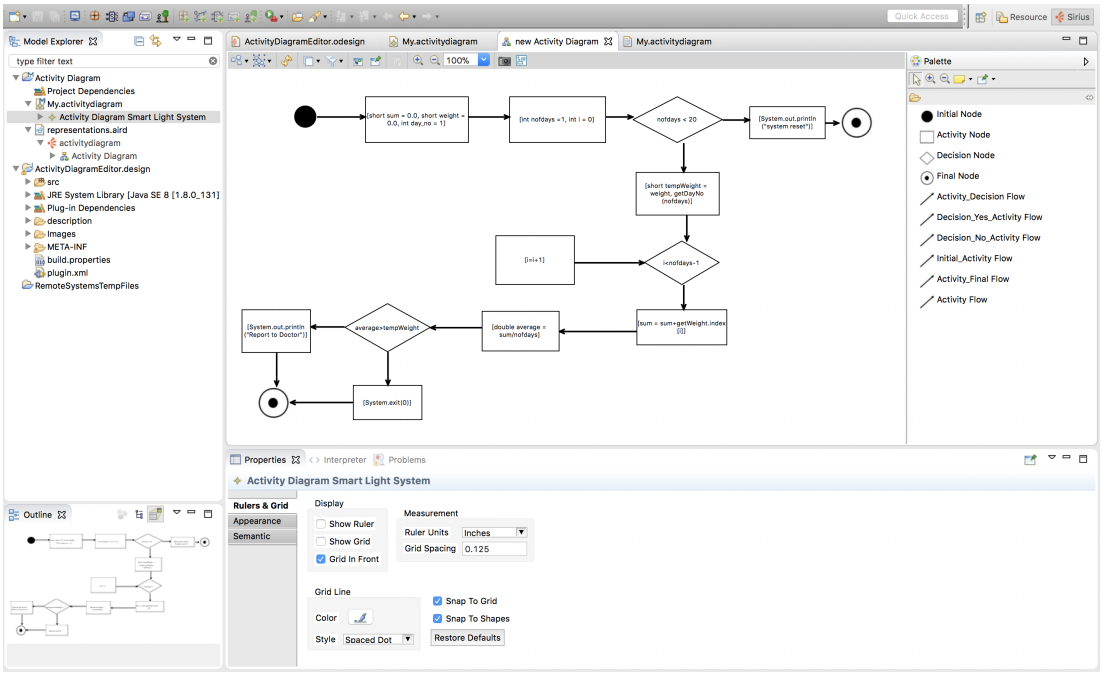

Figure 6.4: Smart Weighting System

This system measures the persons weight every day for fifteen days, then compare the weight fluctuations within these 15 days and if there are any unusual changes in the weight, the system notifies the doctor or otherwise continue tracking the weight.

6.2 Evaluating The Automatic Translation Tool

This section evaluates the Automatic Translation Tool based on success criteria. Even though the tool consists two significant aspects, the modeling and the automatic code generation aspects which as per section 3.3 of chaper 3, which are evaluated based on two different methodologies, in this chapter, a single evaluation criterion is designed taking the model evaluation methodology (Methodology-Practices-Promises-Metrics)

proposed by (Mohagheghi, 2010) and product risk taxonomy (Klein et al., 2015) as the code generation validation methodology in to consideration. In addition, the evaluation criteria also take the architectural drivers (refer section 4.1.1 of chapter 4) the thesis promises to achieve in relation to the Automatic Translation Tool, which is detailed in section 4.1 of chapter 4, into consideration.

6.2.1 Success Criteria

Generally, when evaluating a potential software solution, it is very likely to consider the ability of the software to meet the functional and non-functional requirements (Bandor, 2006).The success criteria evaluate the Automatic Translation Tool based on the architectural drivers in priority level 1 (Table 4.1).They include usability, availability, and interoperability. Evaluation of the tool for these architectural drivers, not only will ensure the achievement of these architectural drivers, but also the methodologies defined to evaluate the modeling and automatic code generation aspects of the tool (refer section 3.3 of chapter 3).

Subject	Success Criteria	Definition
Usability	Number of Nodes and transitions used to model a smart home case study.	Use of the reduced level of activity diagram nodes and transitions to model a smart home case study determine improved levels of usability. This is also due to the ability to model the case studies faster which in other words may lead to high efficiency and reduced complexities.
	Time is taken to model a smart home case study.	Less time taken to model a smart home case study determines high usability, this is also determined by the number of nodes and transitions involved in modeling the case study. Less time spent on modeling may also lead to improved levels of efficiency.
	Ability to manage changes made to a smart home case study.	This is ensured by obtaining the recompilation time during the below-listed circumstances 1. Addition of a new device 2. Removal of an existing device 3. Changes made to an existing device Reduced recompilation time taken is regarded highly usable.

Table 6.1: Success Criteria to Evaluate Usability

However, there exist other architectural drivers, such as primary functional requirements and the second level prioritized quality attribute requirements. In addition to the availability and interoperability qualities (which also included interoperability in their success criteria), the evaluation phase of this thesis aims to satisfy other architectural drivers by establishing an architectural driver called “Performance”. Even though this quality does not address every architectural driver precisely, the success criteria and their respective definitions ensure the efficiency in terms of the performance of the automatic translation tool is accomplished to a potential and a great degree. How each evaluation subject achieves the defined architectural drivers (4.1.1) are summarized in the table below.

Subject	Success Criteria	Definition
Availability	Number of times the tool is completely available and compiled successfully.	Every time a smart home case study is modeled using the automatic translation tool, the tool is launched from the start to ensure the tool is ready to use every time required. This also includes the availability of all the required plug-ins that involve in the automatic code generation from the activity diagrams.

Table 6.2: Success Criteria to Evaluate Availability

Subject	Success Criteria	Definition
Performance	Size of the code generated.	This is determined by the number of lines of code generated by compiling the modeled case study.A large number of lines of code may result from a complex smart home activity diagram. However, improved usability, in this case, is determined by an average number of lines of code which should not seem complex at the same time.
	Time to compile and generate code.	This defines the time taken to generate code from after being modeled in the activity diagram model editor. The time to compile and generate code is not determined by the complexity of the smart home case study or the number of nodes are transitions involved in a case study. The compilation time is standard regardless of the above-mentioned factors. However, there can be variation in time, and less time taken to compile and generate code is regarded highly usable.

Table 6.3: Success Criteria to Evaluate Performance

6.3 Experiment Setup

This thesis does not involve the participation of external users, therefore an experiment design is set up which models activity diagrams from four different smart home scenarios using the activity diagram model editor which are then translated to Java code using the Automatic Translation Tool. Various aspects such as the number of nodes and transitions used in a smart home case study, size of the code generated from the respective case

Evaluation Subject	Respective Architectural Drivers
Usability	FR1, FR2, FR3, FR4, QAR1, QAR2, QAR5
Availability	QAR3
Performance	QAR4, QAR5, QAR6, QAR7

Table 6.4: Mapping of Evaluation Subjects to Respective Architectural Drivers

study and the time taken for compilation, code generation recompilation during sudden changes made to the smart home scenarios are taken in to account and when setting up the evaluation experiments and evaluating the Automatic Translation Tool as a whole.

As mentioned in section 6.2 of this chapter, the evaluation of the Automatic Translation Tool involves three significant experiments which are derived based on the evaluation criteria. These case studies include: measuring usability, availability, and performance.

6.3.1 Measuring Usability

Usability of the Automatic Translation Tool is ensured by the number of nodes and transitions involved in modeling of a smart home case study, time taken to model the case study and the ability of the model editor to manage the sudden changes made to a specific smart home and the recompilation time after the changes are being made. In order to carry out the experiments for these three aspects, the four smart home case studies defined earlier (refer section6.1 of this chapter) are used to experiment each of the above-mentioned criteria.

1. Each smart home case study was modeled using the activity diagram model editor, we then counted the number of nodes and transitions involved when modeling each scenario.

2. Time taken to model each of these smart home scenarios is recorded individually

3. A new device (sensor or an actuator) is added, an existing device is removed and state of an exciting change is modified for each smart home scenario individually and recompilation time is recorded for each case study.

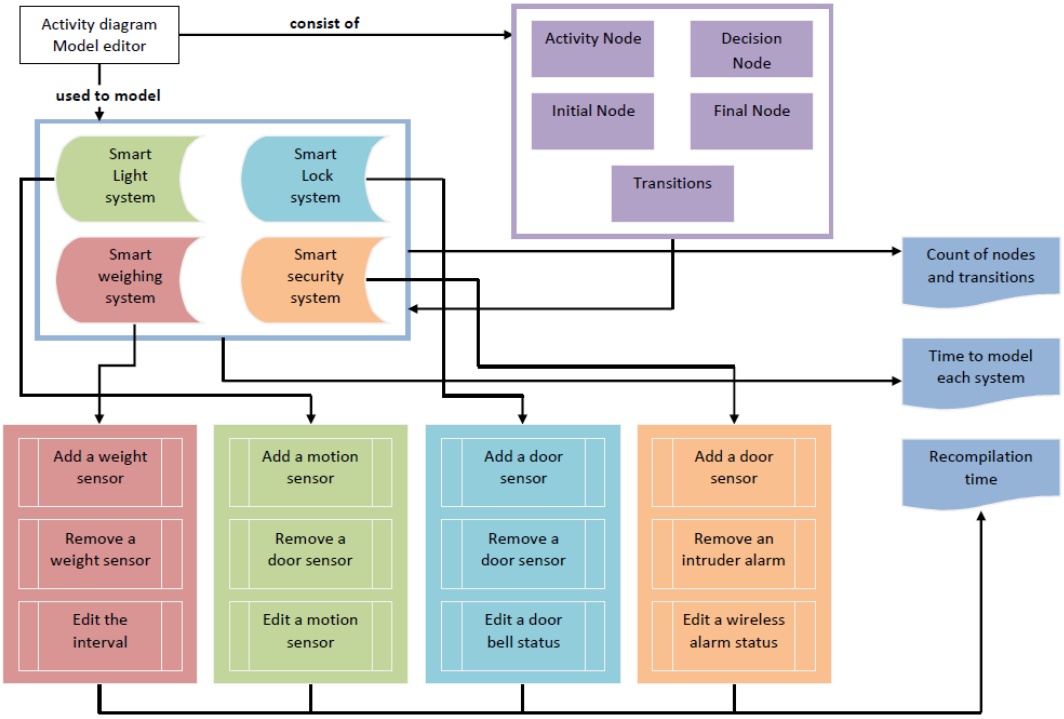

Figure 6.5: Usability Experiment Setup

The complete usability experiment setup is shown in the diagram above. As promised during the evaluation success criteria, usability aspect of the Automatic Translation Tool is achieved by determining the count of nodes and transitions involved in each smart home case study that's modeled using the Activity Diagram Model Editor. Usability is also determined by the time taken to model each of these smart home case studies and the time taken to model and generate code, which in this case is known as the reconfiguration time during an addition of a new sensor or actuator, removal or modification made to an existing sensor or actuator in each smart home system. The

outcomes of usability are depicted with the purple components in figure 6.5. How the outcomes are achieved are shown with the support of the Activity Diagram Model Editor, nodes, and transitions, different smart home systems that are being modeled and which steps are taken to obtain the reconfiguration time in each case.

6.3.2 Measuring Performance

Evaluating the Automatic Translation Tool based on performance as shown in Table 6.7 ensures the achievement of the quality attribute requirements in the second level of the prioritization list. Hence, performance is achieved by experimenting the tool based on the size of the code the smart home scenario generates and the time taken to compile and generate Java code after the modeling of the smart home scenario. However, for the purpose of fine and comprehensive evaluation and experimentation, these experiments are carried out for all the four (previously defined), smart home case studies.

6.3.3 Measuring Availability

Availability, in this thesis, is ensured by the ability of the automatic translation tool to be ready for use and the ability to compile and generate code successfully without any interruptions. In order to experiment these criteria, every time a smart home scenario is modeled for code generation, the tool is launched from the start in order to ensure all the plug-ins that are used in the process are available, the correct workspace is chosen and all the installed plug-ins and up-to-date and usable to support high availability of the tool overall.

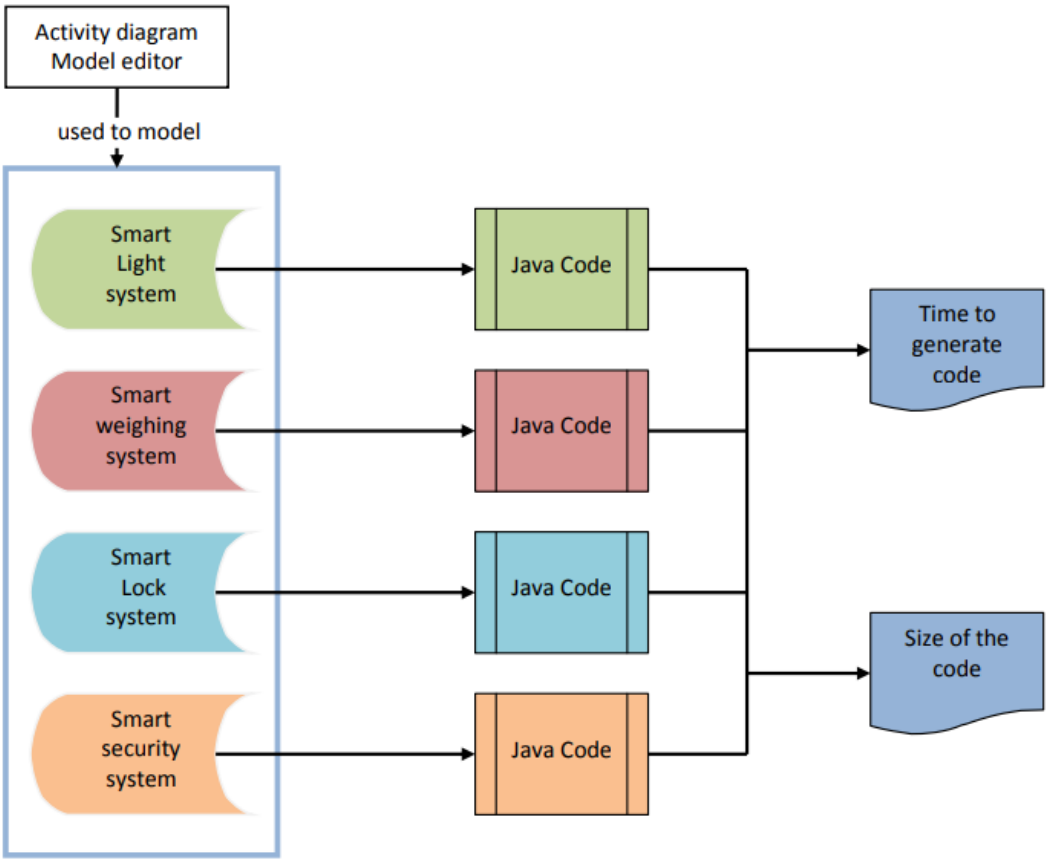

Figure 6.6: Performance Experiment Setup

6.4 Experiment Execution and Data Collection

This part of the chapter carries out the experiments set up in the previous section. As discussed earlier, four smart home systems are modeled using the activity diagram model editor (which is a main component of the automatic translation tool). These case studies are modeled to collect quantitative data to evaluate usability, availability, and performance of the tool. These smart home systems are modeled and the code is generated multiple times to obtain data which are later used to determine the evaluation subjects identified during the definition of the evaluation success criteria for the automatic translation tool.

This section consists of three significant areas which address the collection of data that determine the usability, efficiency, and performance of the automatic translation tool. The data collected are shown in the form of tables for easy comparison and if required, direct graph generation.

6.4.1 Gathering Experimental Data to Determine Usability

Case Study	Number of Nodes				Number of Transitions	Time to Model
	Activity	Decision	Initial	Final		
Light System	7	3	3	1	14	12 minutes 10 seconds
Security System	6	1	1	2	9	8 minutes 32 seconds
Door Lock System	8	4	3	1	16	14 minutes 12 seconds
Weighting System	9	3	1	1	17	13 minutes 45 seconds

Table 6.5: Quantitative Data of Time, Nodes and Transitions to Model Smart Home Systems

Case Study	**Add New Component**	**Time**
Light System	Add a new motion sensor	1 minute 6 seconds
Security System	Add a new door sensor	1 minute 8 seconds
Door Lock System	Add a new door sensor	6 minutes 33 seconds
Weighting System	Add a new weight sensor	1 minute 2 seconds

Table 6.6: Experimental Data of Reconfiguration Time During Addition of a New Device

Case Study	**Delete Component**	**Time**
Light System	Remover a door sensor	1 minute 5 seconds
Security System	Remove an intruder alarm	1 minute 5 seconds
Door Lock System	Remove a door sensor	2 minutes 11 seconds
Weighting System	Remove a weight sensor	43 seconds

Table 6.7: Experimental Data of Reconfiguration Time During Removal of a Device

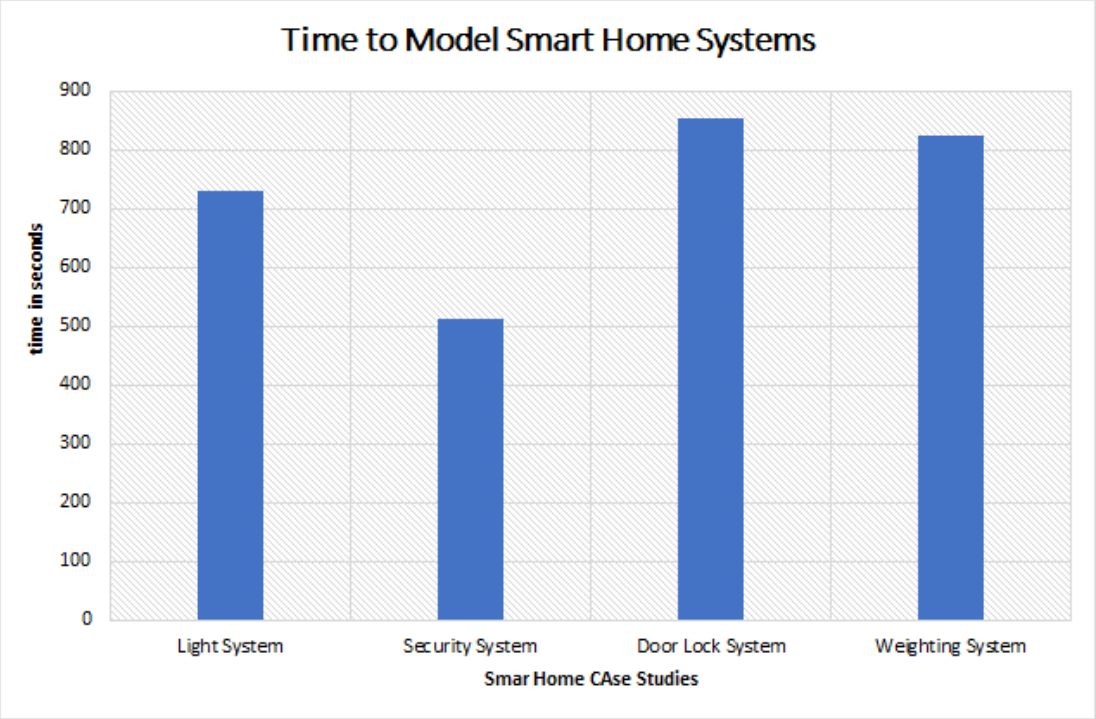

Figure 6.7: Time to Model Smart Home Systems

Case Study	Edit Component	Time
Light System	Modify a motion sensor	1 minute 05 seconds
Security System	Modify wireless alarm status	1 minute 16 seconds
Door Lock System	Modify door bell status	1 minute 27 seconds
Weighting System	Modify the interval	1 minute 18 seconds

Table 6.8: Experimental Data of Reconfiguration Time During Modification of a Device

6.4.2 Gathering Experimental Data to Determine Performance

Case Study	Size of the Code Generated	Time to Compile, Generate Code
Light System	29 lines	13 minutes 40 seconds
Security System	23 lines	9 minutes 18 seconds
Door Lock System	27 lines	18 minutes 50 seconds
Weighting System	30 lines	15 minutes 14 seconds

Table 6.9: Experimental Data of Time and Size of Code Generation

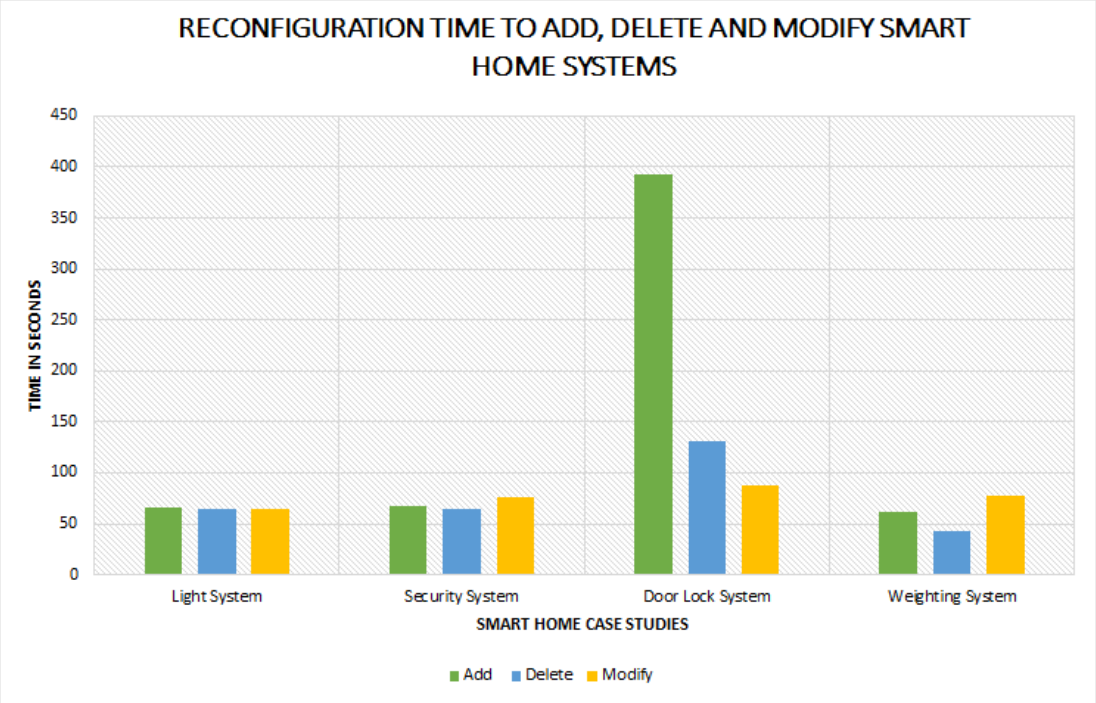

Figure 6.8: Model Reconfiguration Time

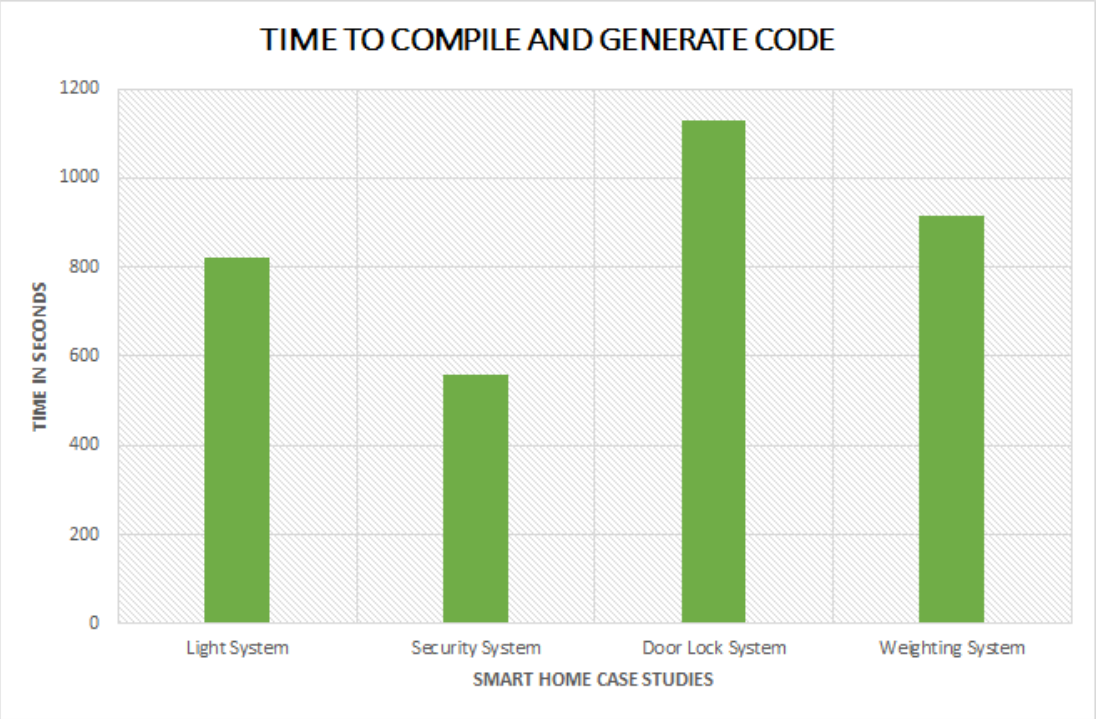

Figure 6.9: Time Generate Code

6.4.3 Gathering Experimental Data to Determine Availability

Case Study	Application Launching	Run and Configure	Availability of Required Plug-ins
Light System	✓	×	×
Security System	✓	✓	✓
Door Lock System	✓	✓	×
Weighting System	✓	✓	✓

Table 6.10: Experimental Data of Availability of Automatic Translation Tool

6.5 Data Analysis

Generally, various statistical analysis methods are recommended and used to analyze the data collected, however due to various reasons such as the sample size of the of data collected being small which may result in statistical assumptions being violated, less accurate results obtained due to small sample size of data may also lead to shortcomings in relation to the variability of the overall statistical test.

Therefore, instead of performing statistical analysis on the data gathered, reasons are provided for significant fluctuations and assumptions are derived from the collected data. This, in turn, will eventually ensure how well the automatic translation tool has achieved the architectural drivers promised earlier. However, if the results do not accomplish expected levels in terms of usability, performance, and availability, they will be regarded as the disadvantages which will later be drawn as future works.

As discussed earlier, the evaluation phase aims to assess the Automatic Translation Tool in terms of usability, performance, and availability (refer section 6.3) which are linked to the architectural drivers (refer section 6.6) the tool promised to achieve earlier during the establishment of the system requirements (refer chapter 4). Therefore, various evaluation tests are carried out using four significant smart

home case studies (refer section 6.1) to meet the evaluation criteria in terms of usability, performance, and availability which will be analyzed individually based on the collected data.

6.5.1 Usability

Usability as defined in chapter 4, aims to achieve increased levels of understanding ability in terms of the Automatic Translation Tool. However, as mentioned earlier, due to the inability to involve external participant to assess the tool, the requirement to evaluate the tool in relation to usability is determined by three important criteria (refer section6.1 of this chapter), they include the number of nodes and transitions involved in modeling a smart home case study, time taken to model a smart home case study and the ability of the Automatic Translation Tool to manage reconfiguration time during a change made to a smart home system.

In order to testify these criteria, four specific smart home systems (case studies in other words) are modeled using the automatic translation tool. Firstly, the number of nodes, they include the activity nodes, decision nodes, initial nodes and final nodes and transitions (refer figure 5.8) used in each case study are recorded as shown in Table 6.12. Simultaneously, the time taken to model each case study is recorded in minutes and seconds which are also represented graphically in figure 6.6.

The number of nodes and transitions used in each smart home case study is based on its size (size of the activity diagram) and complexity. This is directly related to the time taken to model a smart home case study, higher the number of nodes and transitions, longer the time taken to model the activity diagram for a smart home case study. However, when the figures from Table 6.12 are taken into account, most of the smart home systems have taken a relatively equal amount of time to be modeled using

the automatic translation tool. Most of the time was spent on launching Eclipse and ensuring all the required plug-in were available for successful modeling, compilation and code generation. This occurred mainly due to the tool being launched from the start every time a new smart home system was modeled to generate code.

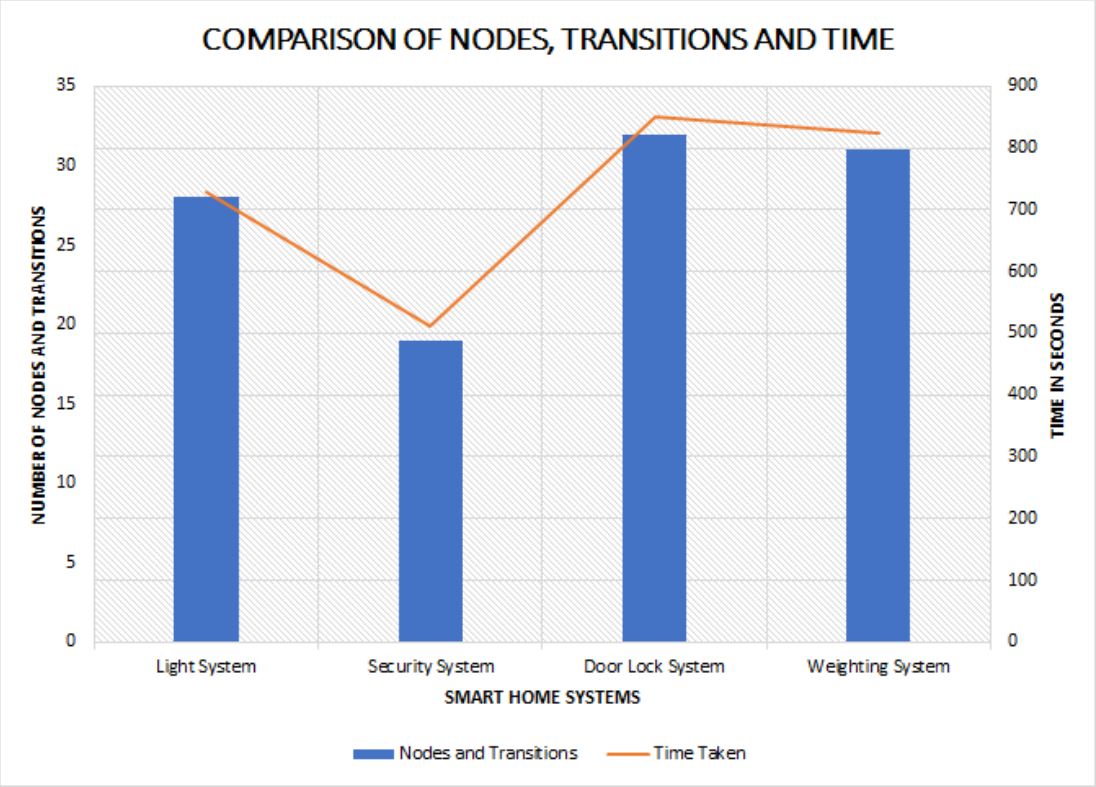

Figure 6.10: Comparison of Nodes, Transitions and Modeling Time

The relationship between the time taken to model the smart home case studies and the number of nodes and transitions used in each smart home system are depicted graphically in Figure 6.10. As the relationship between these components are positive and does not give rise to a significant fluctuation if not based on the number of the nodes and transitions involved with the case study. However, modeling time can be affected by the time taken to launch Eclipse and configure the Automatic Translation Tool, this is discussed in detail when assessing the availability of the tool. Therefore, usability based on the number of nodes, transitions and time taken to model the smart

home systems seem positive when the above facts, data, and graphs are considered.,

The final criteria in terms of usability are, the ability of the automatic translation tool to manage changes made to a smart home case study, these changes include the addition of a new sensor or actuator, removal of an existing sensor or actuator and modifications made to an existing sensor or actuator. Each of these scenarios leads to a specific time, (refer figure 6.7). This time is known as the reconfiguration time which includes the time to model the changes, compile and generate code after the changes made.

Even though, most smart homes scenarios have relatively similar reconfiguration times (lighting system, security system, and weighing system), the door lock system tends to have a significantly high time during the addition of a new door sensor to the system, which can be regarded highly unusable due to the unusually long time is taken to configure. This is addressed in detail later during the discussing of the data analysis.

6.5.2 Performance

Performance, derived from multiple architectural drivers (refer section 6.3 of this chapter), is determined by size of the code generated (number of lines of code) and the time taken to generate code (this includes the time taken to launch the application, install any missing plug-ins, configure the automatic translation tool, model the activity diagram and compile and generate code).

When Figure 6.11 is taken into account, the average time taken to generate code is 14.25 minutes. Most smart home case studies tend to lie around this value beside the security system which has taken considerably less time when compared with

the rest of the smart home case studies to model, reconfigure and generate code. The time is taken to reconfigure and the size of the code is drawn in one graph as shown below to establish the relationship between these two aspects.

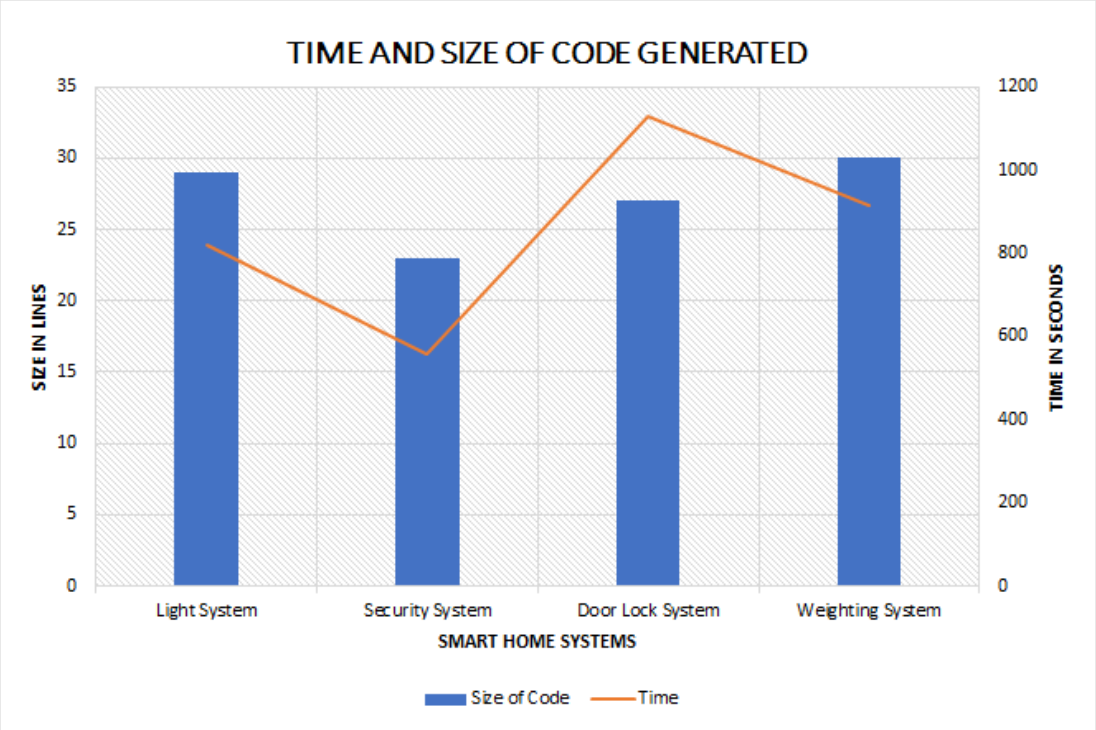

Figure 6.11: Time and Size of the Code Generated

As per the above graph, the size of the code generated does not have a noticeable relationship with the time taken to configure and generate code. Even though in some cases, such as the Smart Security and Door Lock Systems, the relationship between the time and the size of code generation seem positive to a great level, this statement is proved wrong by both smart lighting and weighing systems. This makes us believe that there are other reasons that relate and affect the time taken to configure and generate code. They will be highlighted in detail during the discussion of the data analysis.

6.5.3 Availability

Availability of the Automatic Translation Tool is determined by the ability to launch an application and the tool successfully, run and configure the models and generate code successfully and the complete availability of the required plug-ins. These three factors affect the time taken to model activity diagrams and configure and generate Java code effectively.

However, when statistics form Table 6.10 are considered, the application is launched successfully during the modeling of every smart home case study. During the configuration and generation of code, the automatic translation tool failed to generate code successfully during the compilation of the smart lighting system and produced a "null pointer exception" instead. Nevertheless, the tool generated Java code successfully during compilation of every other case study. In term of availability of required plug-ins, the Automatic Translation Tool is built integrating different Eclipse plug-ins that support Activity Diagram modeling (Ecore tools and Sirius), model transformations (ATL Transformation Language) and code generation (Java).

The availability of all these plug-ins are necessary for the automatic translation tool to execute successfully, however during the modeling of the smart lighting and door lock systems, ATL plug-in was not available which required installation, this could have led to increased configuration and code generation time which could have been avoided with high availability of plug-ins. The code generation, configuration, and modeling of activity diagrams times which are determined by usability and performance are therefore transparently determined by the availability of the automatic translation tool. Relative strengths, weaknesses and potential improvements are addressed in detail in the following section.

6.6 Discussion

This section discusses the data outlined and analyzed in during the data analysis. As mentioned earlier, the evaluation phase assesses the automatic translation tool in terms of usability, availability, and performance. Thereby, firstly, how well the automatic translation tool achieves usability is discussed along with the unexpected situations that lead to low levels of usability during modeling and compilation of some smart home case studies. Performance, which as mentioned previously, determined by the size of the code and the time taken to generate the code, the analyzed data relating to these aspects are considered in order to examine how well performance is achieved and which factors lead the lack of performance at certain levels during the evaluation of the automatic tool. The availability of the tool as said earlier affects both usability and performance, therefore, the factors that lead to lack of performance and usability of the automatic translation tool at certain times during the evaluation process may have caused due to unavailability of the tool. However, all these facts, issues and strengths and weaknesses are addressed in detail in this section. Furthermore, the analyzed data which is discussed is used to answer the research questions.

6.6.1 Usability of The Automatic Translation Tool

Usability is determined by the number of nodes and transitions that are used to model a complete activity diagram for a specific smart home scenario and the time taken to model a smart home case study completely. During the data analysis in terms of usability, it was understood that time taken to model and the number of nodes and transitions taken to model a smart home share a positive relationship (refer figure 6.11). Moreover, the time taken to model all the smart home case studies are relatively similar and does not depict a significant fluctuation. This significant relevance is shown in the

figure below.

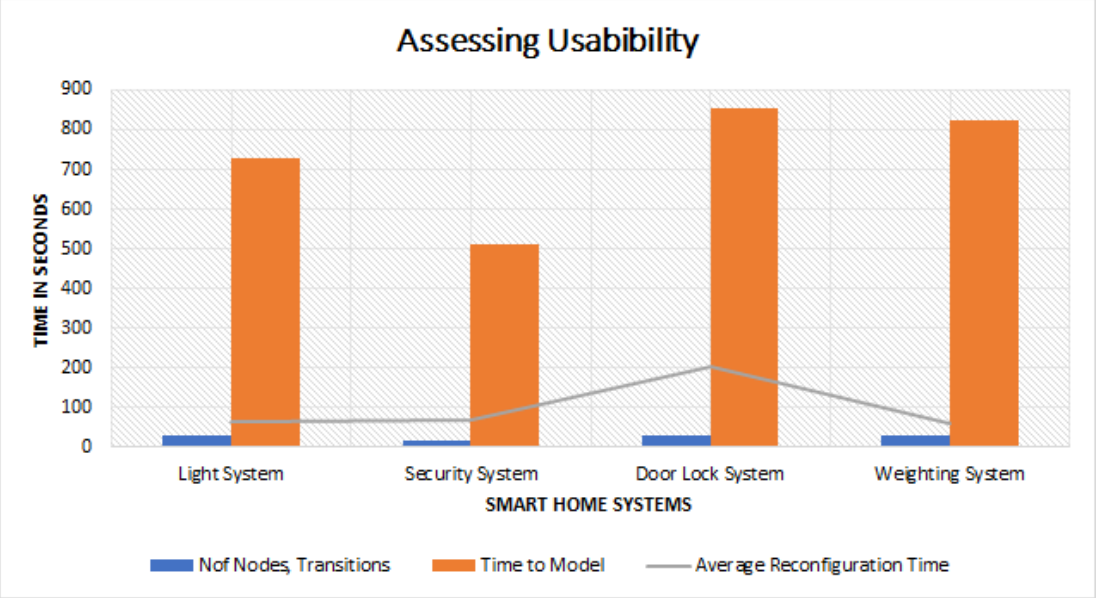

Figure 6.12: Time to Model, Generate Code and Number of Nodes

However, during the reconfiguration of smart home case studies, especially during the addition of a new door sensor to the smart door lock system, it was analyzed that time taken to reconfigure this aspect has taken quite an absurd amount of time to recompile and regenerate code after that changes were made to the smart door lock system. If the Table 6.10 is looked at in detail, the availability of required plug-ins during the modeling, compilation and reconfiguring the Smart Door Lock System is not achieved due to some of the plug-ins such as the ALT transformation plug-in not being installed. Therefore, the missing plug-ins required installation which also involved re-launching of Eclipse. This caused an excessive time spent on modeling, compiling, configuring, the code generating and recompiling the Activity Diagram for the Smart Door Lock System.

Therefore, when the above facts are taken in to account, since the time taken to model the smart home systems are relatively similar, it can be considered that usability is achieved to a certain extent which is however affected by the availability

of the required plug-ins which can also be regarded as a weakness of the automatic translation tool. Hence, the reconfiguration time taken for every other smart home system is remarkably less, besides the smart door lock system. Thereby, it can be said 75 percentage of usability is achieved in terms of usability of the Automatic Translation Tool when evaluated using four significant smart home case studies.

As per the research question this thesis address, RQ3 emphasizes on the characteristics of the automatic translation tool, usability is regarded and one of the most prioritized architectural drivers this tool aims to achieve, which in this case can be interpreted as one of the most significant characteristics of the Automatic Translation Tool. Consequently, as per the above data collection, analysis and discussion in relation as to how the Automatic Translation Tool achieves usability, the evaluation phases overall ensures achievement of high levels of usability despite the occurrence of few challenges. As a result, it can be regarded that the Automatic Translation Tool achieves usability characteristic to an acceptable degree by allowing the users to model Activity Diagrams and generate code in an adequate time.

6.6.2 Performance of The Automatic Translation Tool

During the data analysis in regard to the performance of the automatic translation tool, with the support of figure 6.11, it was determined that size of the generated code does not affect the time taken to compile and generate code. Data analysis of performance concluded that there are other potential reasons that affect the compilation and code generation times, these reasons are discussed in this section.

However, as per the compilation times are shown in figure 6.9, the Smart Security System has taken the shortest time to compile and generate code. This has occurred due to the lowest number of nodes and transitions used to model the activity

diagram, the lowest time is taken to model the activity diagram and reasons such the successful launching of Eclipse to model the smart security system, error-less code generation and availability of all the required plug-ins lead to the transformation of activity diagram to Java code effectively.

On the other hand, the Smart Lighting System, Door Lock System, and Weighing System had a relatively higher number of nodes and transitions during the modeling of Activity Diagrams, as a result, they took longer times to be modeled and to generate code. In the case of the Smart Lighting System and the Door Lock System, all the required plug-ins were not available and required re-installation and relaunching of the application, which led to longer compilation time thus resulting in lack of performance.

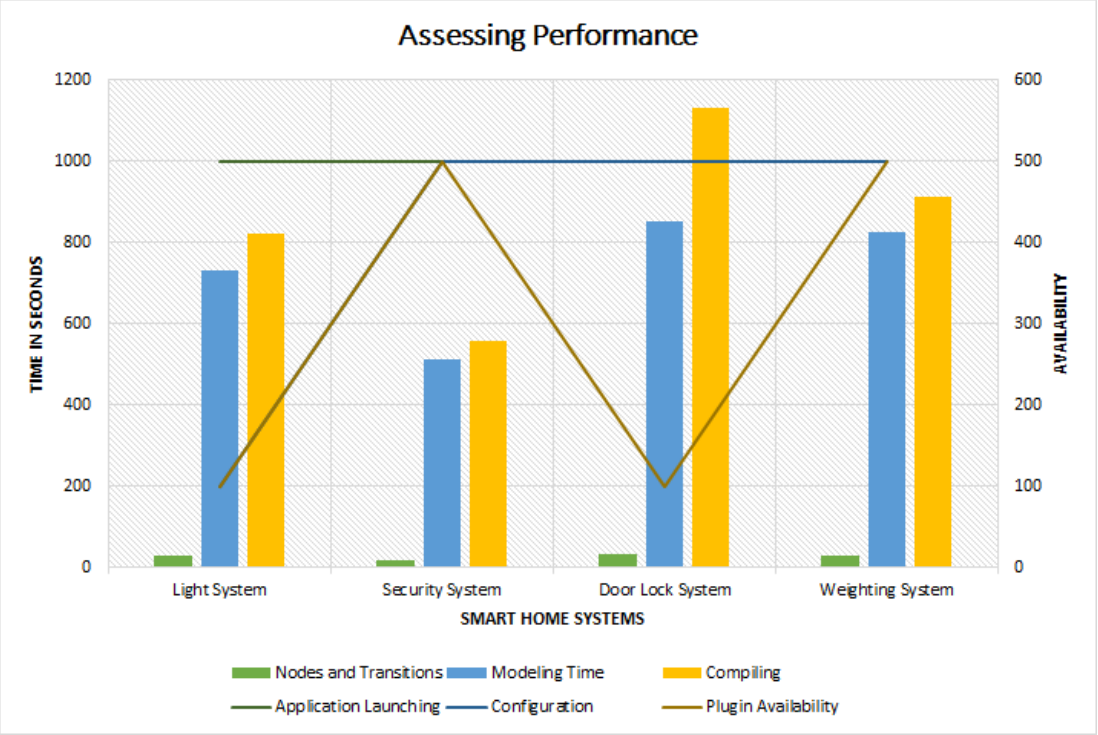

Figure 6.13: Performance of the Automatic Translation Tool

The above figure depicts the compilation and modeling time using the bars and the compiling, application launching and plug-in availability using the lines. This

depicts the relationship between the availability and the performance measurements clearly thus supporting the facts stated earlier about how unavailability of certain aspects can lead to higher compilation times, thus causing challenges in terms of performance. Additionally, even though numerical values were not assigned to the availability aspects (refer section 6.10 of this chapter), for the purpose of discussion of data analysis, a high value was assigned to the availability aspects that were achieved successfully and a low value was assigned to aspects not achieved.

Hence, the performance of the Smart Lighting System is also affected by the inability to generate executable and error-less code. However, due to these reasons, the performance of the Automatic Translation Tool is affected negatively. By ensuring high availability of all the required plug-ins, compiling and generating Java code without errors may help to mitigate these challenges. Moreover, as per RQ3, it can be considered that performance characteristic is achieved by the Automatic Translation Tool to a certain degree despite the challenges discussed above.

6.6.3 Availability of Automatic Translation Tool

Availability of the Automatic Translation Tool is addressed during the discussion of both usability and performance due the performance determine the achievement of both of these characteristics to a great level. As a result, this section will discuss the reasons as to why some of the availability aspects when modeling certain smart home applications were not achieved and how it, in turn, affected both usability and performance levels of the tool.

As shown in Table 6.10, the availability of the Automatic Translation Tool is determined by the ability to launch the application successfully, run, compile and configure successfully and ensure all the required plug-ins are available prior to the

configuration. However, during the modeling of both Smart Lighting System and Door Lock systems, Eclipse failed to ensure all the plug-ins were available, ATL plug-in was un-installed which required re-installation in both situations. This, as a result, caused complexities in terms of the time taken to complete and modeling and compile the activity diagram to Java code, thus causing a negative impact on both usability and performance of the automatic translation tool.

During the compilation of the smart lighting system, Java code was not generated completely and caused a null pointer exception instead thus causing issues in the code generated. However, Java code was generated successfully which resulted in improved levels of usability and performance.

6.6.4 Impact of the Automatic Translation Tool

Customization and Cost Effectiveness

The Automatic Translation Tool provides its users a customized Activity Diagram Model Editor which allows the users to model Activity Diagrams for any given case study. The existing model editors such as Visual Paradigm, StarUML, Papyrus and GenMyModel support modeling of any UML diagram, however, fails to support automatic code generation for behavioral activity diagrams.

Furthermore, the Activity Diagram Model Editor is customized for modeling activity diagrams which in turn will support the users in modeling any type of case studies. The Automatic Translation Tool is developed in Eclipse, which is a freely available software to download. Moreover, the tool is developed integrating multiple Eclipse plug-ins which can be downloaded from the Eclipse marketplace free of cost. The Activity Diagram Model Editor provides an adequate modeling space to model any

activity diagram, also it provides a palette with all the nodes and transitions required to model an activity diagram. Therefore, the Automatic Translation Tool provides a customized Activity Diagram Model Editor which allows easy and cost-effective modeling procedures unlike software such as Enterprise Architect which costs more than USD300 for download.

Automatic Code Generation from Activity Diagrams

As mentioned earlier, the above stated UML modeling editors also allow code generation from the UML models, however that code generation aspect is unfortunately limited to the structural diagrams such as class and sequence diagrams. Moreover, even though Enterprise Architect by (Architect, 2010) offers modeling and code generation from behavioral activity diagrams, it costs as stated earlier more than USD 300 to obtain the ultimate Enterprise Architect package which performs multiple functions such as modeling and generating code.

The Automatic Translation Tool, however, offers automatic code generation from behavioral UML Activity Diagrams within less than 2 minutes approximately solving the problem of inability to generate code automatically from activity diagrams.

6.6.5 Strengths and Weaknesses of Automatic Translation Tool

The advantage of the Automatic Translation Tool is that it allows efficient and automatic translation of UML behavioral Activity Diagrams to executable Java code at free of cost. The tool provides a customized activity diagram which allows modeling of any activity diagram based on any scenario thus providing improved levels of flexibility.

Moreover, as per the evaluation results, the Automatic Translation Tool carries

out the transformation process efficiently with high levels of performance, determined by other factors such as availability and high usability. Furthermore, the Java code generated by compiling a smart home activity diagram is completely executable and therefore can be used to deploy smart home applications when integrated with data from smart spaces.

The weakness of the Automatic Translation Tool is, there can be situations where all the required plug-ins to compile the tool may not be available and may require re-installation and relaunching of the application and the tool. Moreover, the Java code generated may not be completely executable which in turn may require prior knowledge or technical assistance.

6.7 Conclusion

Section 6.1 of this chapter, identifies different smart home systems out of which one specific smart home system is chosen to demonstrate the automatic translation tool throughout the thesis globally. This selection of the smart lighting system is carried with the support of selection criteria. Section 6.2 defines the success criteria used to evaluate the automatic translation tool and the relationship between the evaluation criteria and the architectural drivers defined in Chapter 4. Section 6.3 explains how the data collection is carried out based on the success criteria. Experiment execution, which is section 6.4, performs data collection to meet the success criteria based on the smart home systems defined in Section 6.4. Afterwards, the data collected is analyzed to prove how the automatic translation tool achieves the success criteria and indirectly the architectural drivers (refer chapter 4). Section 6.5 discusses the data analyzed in the previous section thus assuring how achievement of each success criteria answer a research question (refer chapter 2). Furthermore, impacts, strengths, and weaknesses

of the Automatic Translation Tool are discussed based on the evaluation results which concludes this chapter.

Chapter 7

Conclusions

This chapter summarises the thesis while emphasising the extent to which the research questions have been answered. It also discusses the limitations of the proposed solution and reports future research directions.

This chapter concludes the thesis summarizing the results obtained implementing the automatic translation tool for domain and technical experts. Section 7.2 refers back to the research questions formulated during the systematic literature review (refer chapter 2) and discusses how well the thesis answers each research question. However, regardless how well the thesis answers these research questions, there still remains potential gaps and limitations in relation to every research question.

This chapter also discusses these limitations in detail and relate them to future works and suggests potential improvements that can be carried out to achieve improved levels of results. Section 7.3 lists the contributions this thesis has made and how each of these contributions of the automatic translation tool can be used in practical implications. Finally, the chapter and the thesis overall are concluded with a note about the journey of thesis and a take home message for the audience.

7.1 A Chapter-wise Summary

Chapter 1 provided an introduction of the thesis, presenting an overview of all aspects of this research: problem definition, research scope and questions, research design and method, and the proposed solution. Chapter 2 presented the details of a comprehensive systematic literature review carried out to identify an appropriate app design model, as well as the overall qualities and features of the proposed solution. It also introduces key terms used throughout the rest of the thesis. Chapter 3 focussed on the design of this research, which is based on the well known Design Science Research methodology. This design involves the steps involved in all research activities, such as problem definition and refinement, architecture design, system development and evaluation. Chapter 4 reports the architecture design of the Automatic Translation Tool, carried out using the Attribute Driven Design (ADD) method. We show how the proposed architecture addresses primary functional requirements and quality attribute requirements. The 4+1 views and beyond approach was used to document the architecture. Chapter 5 shows how a model-driven engineering approach was used to design and develop the Automatic Translation Tool. This chapter highlights the key components of the tool - the model editor and the compiler built using model-to-model and model-to-text transformation processes. The chapter also shows the corresponding technological decisions made for each phase. In Chapter 6, we first defined key success criteria for the tool, and then carried out a comprehensive evaluation based on qualities like usability, availability and performance. The chapter details experimental set up, data collection, and analysis aspects of this evaluation, which lead to the identification of the tool's strengths and weaknesses.

7.2 Answering the Research Questions

RQ1: *Which factors lead to the choice of an app design model to address the challenges such as lack of interoperability when apps need to operate in different sensor-actuator configurations in different smart homes?*

This question is answered in Chapter 2 with the support of the systematic literature review. In order to solve the challenge of lack of interoperability when a smart home requires integration of various smart home systems with varied sensor actuator configurations, various factors that lead to the choice of an app design model are explored with the help of literature and findings. Various app design models are evaluated based on these factors to find the most suitable visual app design model that can be used to solve the above mentioned challenge. UML activity diagram is chosen based on the evaluation carried out against the factors that lead to the choice of an app design model (refer to table 2.5 and table 2.4). Even though, UML activity diagrams are regarded as user friendly and simple to model, the activity diagram can be too simple where the modelers may attempt to include all the information in one single activity diagram may cause over complexities.

RQ2: *What are the existing solutions and how do they meet the factors identified in Question 01?*

This question is also answered in chapter 2, in terms of the existing solutions that address the problem definition, they are categorized in to commercial based and research based solutions. 14 solutions in total were identified and discussed during the systematic literature review where the research gap of each solution was emphasized. These solutions were then illustrated in the form of a table where each highlighting the gaps individually (refer table 2.3).Some of the factors that are identified in RQ1 were

then used in the said table to show how the existing solutions could not meet the said factors.

Although, the research questions mentions the requirement to evaluate the current solutions against the factors identified in RQ1, new criteria was defined which relates to the factors from RQ1 and are presented in a simpler way.

RQ3: *What are the characteristics of the architecture of an automatic translation tool, which translates an app design (based on the model developed after answering Q1) to a customized smart home app?*

The characteristics of the automatic translation tool are identified in chapter 4 as the architectural drivers, these drivers are composed of the primary functional requirements and quality attribute requirements which the tool aims to achieve towards the end of this thesis. These characteristics are based on both factors derived from RQ1 are translated to primary functional requirements and quality attribute requirements which address the translation of an visual plan (also known as app design model) to executable code which can be deployed to a customized smart home application. Therefore 5 primary functional requirements and 7 quality attribute requirements are generated to, which in this case are known as the characteristics, to address the architecture of the automatic translation tool.

RQ4: *How can the high-level architecture of an automatic translation tool obtained from Q3 lead towards the implementation of a prototype automatic translation tool?*

This question is answered in chapter 5, which is the design and the development of the Automatic Translation Tool. The high-level architecture which is designed in chapter 4 based on the characteristics discussed in RQ3, is transformed to a prototype

called “The Automatic Translation Tool” which can be implemented to translate a visual app design model, which in the case of this thesis is the UML activity diagram, to executable Java code. This can be then deployed to a smart home application. The translation of the prototype to the tool is carried out based on Model Driven Engineering (MDE) which supports the modeling aspect (of UML activity diagrams), model transformation and automatic code generation aspects of the Automatic Translation Tool”. This tool achieves most of the characteristics detailed in RQ3, however fails to achieve them completely (refer chapter 6).

Even though, the Automatic Translation Tool allows modeling of various UML activity diagrams providing a customized activity diagram model editor, Java code generated from the activity diagrams may sometimes lead to errors during compilation. Moreover, although activity diagrams, by most authors, are regarded as very user-friendly and easy to model, in the case of the Automatic Translation Tool, the user is expected to have prior knowledge in Java and programming to model the activity diagrams. This may cause challenges in terms of usability (user-friendliness) since the tool requires prior knowledge and technical assistance during feasible situations.

7.3 Contributions of this Thesis

User-friendly and Customized UML Activity Diagram Editor

As introduced, designed and developed and evaluated, this thesis presents a customized UML activity diagram model editor which can be used to model any kind of activity diagram. Even though, this thesis emphasizes on smart home scenarios during the implementation of the Automatic Translation Tool, the model editor can be used model activity diagrams for any system or scenario. This development is regarded

as one of the most significant contributions of this thesis due to, most of the existing model editing tools cater to all the UML type diagrams including both structural and behavioral, such as Visual Paradigm, Papyrus, GenMyModel, StarUML and Poisedon. However, none of these editors are customized to model one specific type of visual modeling language, which would eventually result in various advantages such as support provided in terms of reuse and maintenance of models and the ability to create your own transformation such as code generation, which in this case in Java. Therefore, design of the customized model editor not only allows the easy modelling of activity diagrams catering to any scenario but also supports automatic code generation via model transformations which as per the case of the smart homes systems, can be used to deploy to a smart application.

Interoperable Smart Homes with Automatic Translation Tool

As per the systematic literature review (refer chapter 2), most of the existing solutions fail to achieve interoperability during the integration of multiple smart systems in to a single smart home application due to varied sensor/actuator configurations. With the introduction of the Automatic Translation Tool, the user is allowed to model activity diagrams for any preferred smart home system, such as smart lighting system, smart security system and smart door lock system (refer section 6.1 in chapter 6), these systems can be compiled using the Automatic Translation Tool to generate Java code for each of these systems regardless their variance in sensor /actuator configurations, they can be deployed in to a single smart home application thus solving the challenge of lack of interoperability to a greater level. This in turn may save the cost of purchasing additional devices that support the interoperation of the multiple smart devices in to a single smart home application.

Effective and Automatic Code Generation from Behavioral UML Activity Diagrams

In addition to the above, the Automatic Translation Tool as said earlier generates Java code from any activity diagram modeled using the customized Activity Diagram Model Editor. Most of the existing model to code generation applications only support code generation from structural UML diagrams such as UML class diagrams, however the Automatic Translation Tool translates the activities in to Java code which may be included in a Java class to define the behavior of the modeled and compiled smart home system. This is regarded as one of the most outstanding contributions this research makes due to the ability of the Automatic Translation Tool to generate Java code within less than 2 minutes (refer table 6.10 in chapter6) and can be considered as a novel deliverable when the literature evidence in terms of automatic code generation tools are researched and studied in depth.

7.4 Future Works and Improvements

This section depicts the future works and improvements that can be made to the Automatic Translation Tool, these are drawn from the limitations of this research and the implementations of this tool which mostly occurred due to time constraints. In terms of the weaknesses of the Automatic Translation Tool, as per section 6.6.5 in Chapter 6, all the plug-ins required to model and compile an activity diagram using Eclipse can be undesirable at times due to unavailability of some plug-ins (ATL plug-in) during the launch of the application (Eclipse). Therefore, in the future, a more concrete terminology (this can be a more concrete plug-in or an application) can used to generate code from activity diagrams, which specifically meets increased levels of availability of the application. In terms of the Java code generation, code is generated for the respective

activities modeled in the Activity Diagram Model Editor, therefore code generated from the Automatic Translation Tool only defines the behavior of an object which will not include the structure of the code such as class and method declarations. This will not support executable code generation unless the code is refined after generated from the Automatic Translation Tool. Therefore, a better code generation aspect can be designed to support both structural and behavioral aspects of Java code which can eventually be executed directly after code generation from the Automatic Translation Tool without having to refine the code prior to execution.

As per the visual modeling language, to ensure high usability, understand ability and learn-ability, UML activity diagram is chosen to model the smart home systems. Even though, they meet the above mentioned criteria and helps to specify the behavior of a system, it does not help the user to specify the structure of the system which would help the code generation of the structure of the system. Furthermore, as per section 6.6 of chapter 6, although UML activity diagrams achieve high levels of usability, it can be undesirable to model complex systems using this modeling language due to the ability to include any kind of information. Therefore a more specific modeling language can be chosen that support both behavioral and structural aspects of a system which in turn would support much effective and effortless automatic code generation which in turn can be used to deploy to a smart home application much easily.

7.5 Final Words

This research study had been nothing but the most adventurous journey I have come so far. This thesis conducts a very comprehensive systematic literature review that helped me answer the first two research questions and achieve a well-organized research study. The Design Science Research Methodology was a great support when designing

this research, other terminologies such as Attribute Driven Design and Model Driven Engineering provided assistance immensely to design and develop the Automatic Translation Tool which has ultimately implemented well, which also helped in answering the remaining research questions thus addressing the problem defining to a greater level thus reducing the research gap in terms of interoperability issue in smart home systems and the inability to generate code automatically from UML activity diagrams.

References

Advantech, C. S. (2009). *Mlc flash technologies and structure.* September.

Alaa, M., Zaidan, A., Zaidan, B., Talal, M. & Kiah, M. (2017). A review of smart home applications based on internet of things. *Journal of Network and Computer Applications*, *97*(Supplement C), 48 - 65. Retrieved from http://www.sciencedirect.com/science/article/pii/S1084804517302801 doi: https://doi.org/10.1016/j.jnca.2017.08.017

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., ... others (2003). *Business process execution language for web services.* version.

Architect, E. (2010). *Sparx systems.* Inc.

Aßmann, U., Aksit, M. & Rensink, A. (2005). *Model driven architecture: European mda workshops: Foundations and applications, mdafa 2003 and mdafa 2004, twente, the netherlands, june 26-27, 2003, and linköping, sweden, june 10-11, 2004, revised selected papers* (Vol. 3599). Springer.

Ayav, T. & Sözer, H. (2016). Identifying critical architectural components with spectral analysis of fault trees [Journal Article]. *Applied Soft Computing*, *49*, 1270-1282. Retrieved from http://www.sciencedirect.com/science/article/pii/S1568494616303222 doi: http://dx.doi.org/10.1016/j.asoc.2016.06.042

Baker, P., Loh, S. & Weil, F. (2005). Model-driven engineering in a large industrial context—motorola case study. *Model Driven Engineering Languages and Systems*, 476–491.

Bandor, M. S. (2006). Quantitative methods for software selection and evaluation.

Bashir, R. S., Lee, S. P., Khan, S. U. R., Chang, V. & Farid, S. (2016). Uml models consistency management: Guidelines for software quality manager [Journal Article]. *International Journal of Information Management*, *36*(Part A), 883-899. Retrieved from http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0268401216303425&site=eds-live doi: 10.1016/j.ijinfomgt.2016.05.024

Bass, L., Clements, P. & Kazman, R. (2013). *Software architecture in practice* [Book]. Upper Saddle River, NJ : Addison-Wesley, [2013] Third edition. Retrieved from http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=

cat05020a&AN=aut.b18728923&site=eds-live

Berander, P. & Andrews, A. (2005). Requirements prioritization. *Engineering and managing software requirements*, *11*(1), 79–101.

Bézivin, J. (2001). From object composition to model transformation with the mda. In *Tools (39)* (pp. 350–354).

Biehl, M. (2010). Literature study on model transformations. *Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK*, 291.

Bonfè, M., Fantuzzi, C. & Secchi, C. (2013). Design patterns for model-based automation software design and implementation. *Control Engineering Practice*, *21*(11), 1608–1619.

Bonino, D. & Corno, F. (2010). Rule-based intelligence for domotic environments [Journal Article]. *Automation in Construction*, *19*(2), 183-196. Retrieved from http://www.sciencedirect.com/science/article/pii/S0926580509001599 doi: http://dx.doi.org/10.1016/j.autcon.2009.10.008

Bronner, L. & Olubando, B. (2009). Development of an integrated sociological modeling framework (ismf) to model social systems. In *Social computing and behavioral modeling* (pp. 1–11). Springer.

Brown, A. (2004). An introduction to model driven architecture-part 1; mda and today's systems. *IBM DeveloperWorks, RationalEdge*.

c, H., Miah, S. J. & McAndrew, A. (2016). A design science research methodology for developing a computer-aided assessment approach using method marking concept [Journal Article]. *Education and Information Technologies*, *21*(6), 1769-1784.

Capitanelli, A., Papetti, A., Peruzzini, M. & Germani, M. (2014). A smart home information management model for device interoperability simulation [Journal Article]. *Procedia CIRP*, *21*, 64-69. Retrieved from http://www.sciencedirect.com/science/article/pii/S2212827114006921 doi: http://dx.doi.org/10.1016/j.procir.2014.03.150

Chen, M., Wan, J. & Li, F. (2012). Machine-to-machine communications: Architectures, standards and applications [Journal Article]. *KSII Transactions on Internet and Information Systems*, *6*(2), 480-497. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84861022395&doi=10.3837%2ftiis.2012.02.002&partnerID=40&md5=3cd197d1ba7f8016eebd988a9875204f doi: 10.3837/tiis.2012.02.002

Choi, H. & Yeom, K. (2002). An approach to software architecture evaluation with the 4+ 1 view model of architecture. In *Software engineering conference, 2002. ninth asia-pacific* (pp. 286–293).

Chung, L. & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. In A. T. Borgida, V. K. Chaudhri, P. Giorgini & E. S. Yu (Eds.), *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos* (pp. 363–379). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-642-02463-4_19 doi: 10.1007/978-3-642-02463-4_19

Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J. & Little, R. (2002).

Documenting software architectures: views and beyond. Pearson Education.

Demiris, G., Oliver, D. P., Dickey, G., Skubic, M. & Rantz, M. (2008). Findings from a participatory evaluation of a smart home application for older adults. *Technology and health care*, *16*(2), 111–118.

Dobre, C. & Xhafa, F. (2014). Intelligent services for big data science [Journal Article]. *Future Generation Computer Systems*, *37*, 267-281. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167739X13001593 doi: http://dx.doi.org/10.1016/j.future.2013.07.014

Domı, E., Pérez, B., Rubio, Á. L. et al. (2012). A systematic review of code generation proposals from state machine specifications. *Information and Software Technology*, *54*(10), 1045–1066.

Ehrig, K., Küster, J. M. & Taentzer, G. (2009). Generating instance models from meta models. *Software and Systems Modeling*, *8*(4), 479–500.

Erata, F., Challenger, M. & Kardas, G. (2015). D3. 1.1 review of model-to-model transformation approaches and technologies. *Text & Model-Synchronized Document Engineering Platform*, 70–85.

Favre, J.-M. (2003). Meta-model and model co-evolution within the 3d software space. In *Elisa: Workshop on evolution of large-scale industrial software applications* (pp. 98–109).

Favre, J.-M. (2004). Towards a basic theory to model model driven engineering. In *3rd workshop in software model engineering, wisme* (pp. 262–271).

Fernandez-Saez, A. M., Genero, M., Chaudron, M. R., Caivano, D. & Ramos, I. (2015). Are forward designed or reverse-engineered uml diagrams more helpful for code maintenance?: A family of experiments. *Information and Software Technology*, *57*, 644–663.

Figl, K. & Recker, J. (2016). Process innovation as creative problem solving: An experimental study of textual descriptions and diagrams. *Information & Management*, *53*(6), 767–786.

Fischer, A., Greiff, S. & Funke, J. (2012). The process of solving complex problems [Journal Article].

Fox-Brewster, T. (2016). *Forbes welcome.* Retrieved from https://www.forbes.com/sites/thomasbrewster/2016/02/17/samsung-smartthings-vulnerabilities/#7e42b55d7d09

France, R. B., Ghosh, S., Dinh-Trong, T. & Solberg, A. (2006). Model-driven development using uml 2.0: promises and pitfalls. *Computer*, *39*(2), 59–66.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B. & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, *37*(10), 46–54.

Gay, V. & Leijdekkers, P. (2015). Bringing health and fitness data together for connected health care: mobile apps as enablers of interoperability [Journal Article]. *Journal of medical Internet research*, *17*(11), e260.

Gessenharter, D. & Rauscher, M. (2011). *Code generation for uml 2*

activity diagrams: Towards a comprehensive model-driven development approach. (Vol. 6698 LNCS). Institute of Software Engineering and Compiler Construction, Ulm University. Retrieved from http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-79959203376&site=eds-live

Gravino, C., Scanniello, G. & Tortora, G. (2015). Source-code comprehension tasks supported by uml design models: results from a controlled experiment and a differentiated replication. *Journal of Visual Languages & Computing*, *28*, 23–38.

Gronback, R. C. (2009). *Eclipse modeling project: a domain-specific language (dsl) toolkit*. Pearson Education.

Groppe, J. & Mueller, W. (2005). Profile management technology for smart customizations in private home applications. In *Database and expert systems applications, 2005. proceedings. sixteenth international workshop on* (pp. 226–230).

Gu, H., Elhanan, G., Perl, Y., Hripcsak, G., Cimino, J. J., Xu, J., ... Paul Morrey, C. (2012). A study of terminology auditors' performance for umls semantic type assignments [Journal Article]. *Journal of Biomedical Informatics*, *45*(6), 1042-1048. Retrieved from http://www.sciencedirect.com/science/article/pii/S1532046412000901 doi: http://dx.doi.org/10.1016/j.jbi.2012.05.006

Hafidh, B., Osman, H. A., Arteaga-Falconi, J. S., Dong, H. & Saddik, A. E. (2017). Site: The simple internet of things enabler for smart homes [Journal Article]. *IEEE Access*, *5*, 2034-2049. doi: 10.1109/ACCESS.2017.2653079

Haroutiunian, S., Nikolajsen, L., Finnerup, N. B. & Jensen, T. S. (2013). The neuropathic component in persistent postsurgical pain: a systematic literature review. *PAIN®*, *154*(1), 95–102.

Harrison, N. B. & Avgeriou, P. (2010). How do architecture patterns and tactics interact? a model and annotation. *Journal of Systems and Software*, *83*(10), 1735–1758.

Hayes, P. H. & Black, J. (2006, December 26). *System and methods for home appliance identification and control in a networked environment.* Google Patents. (US Patent 7,155,305)

Ho, W. M., Jézéquel, J.-M., Le Guennec, A. & Pennaneac'h, F. (1999). Umlaut: an extendible uml transformation framework. In *Automated software engineering, 1999. 14th ieee international conference on.* (pp. 275–278).

Hoffman, R. R., Roesler, A. & Moon, B. M. (2004). What is design in the context of human-centered computing? [Journal Article]. *IEEE Intelligent Systems*, *19*(4), 89-95. doi: 10.1109/MIS.2004.36

Inc, N. (2017). *Choosing the right modeling tool - business process modeling notation (bpmn).* Retrieved from https://www.nomagic.com/getting-started/choosing-the-right-modeling-tool

Jacobson, I., Spence, I. & Kerr, B. (2016). Use-case 2.0. *Communications of the ACM*, *59*(5), 61–69.

Jick, T. D. (1979). Mixing qualitative and quantitative methods: Triangulation in action [Journal Article]. *Administrative science quarterly*, *24*(4), 602-611.

Johnson, R. (2005). J2ee development frameworks. *Computer*, *38*(1), 107–110.

Johnson, R. B. & Onwuegbuzie, A. J. (2004). Mixed methods research: A research paradigm whose time has come [Journal Article]. *Educational researcher*, *33*(7), 14-26.

Jouault, F., Allilaire, F., Bézivin, J. & Kurtev, I. (2008). Atl: A model transformation tool. *Science of computer programming*, *72*(1), 31–39.

Jouault, F., Allilaire, F., BÃ©zivin, J. & Kurtev, I. (2008). Atl: A model transformation tool. *Science of Computer Programming*, *72*(1), 31 - 39. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167642308000439 (Special Issue on Second issue of experimental software and toolkits (EST)) doi: https://doi.org/10.1016/j.scico.2007.08.002

Jouault, F. & Kurtev, I. (2006). Transforming models with atl. In J.-M. Bruel (Ed.), *Satellite events at the models 2005 conference: Models 2005 international workshops doctoral symposium, educators symposium montego bay, jamaica, october 2-7, 2005 revised selected papers* (pp. 128–138). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/11663430_14 doi: 10.1007/11663430_14

Jung, Y. (2017). Hybrid-aware model for senior wellness service in smart home. *Sensors*, *17*(5), 1182.

Jungclaus, R., Wieringa, R. J., Hartel, P., Saake, G. & Hartmann, T. (1994). Combining troll with the object modeling technique [Book Section]. In B. Wolfinger (Ed.), *Innovationen bei rechen- und kommunikationssystemen: Eine herausforderung für die informatik* (p. 35-42). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-51136-3_5 doi: 10.1007/978-3-642-51136-3_5

Kao, H.-Y., Yu, M.-C., Masud, M., Wu, W.-H., Chen, L.-J. & Wu, Y.-C. J. (2016). Design and evaluation of hospital-based business intelligence system (hbis): A foundation for design science research methodology [Journal Article]. *Computers in Human Behavior*, *62*, 495-505. Retrieved from http://www.sciencedirect.com/science/article/pii/S0747563216302965 doi: http://doi.org/10.1016/j.chb.2016.04.021

Klein, J., Levinson, H. & Marchetti, J. (2015). *Model-driven engineering: Automatic code generation and beyond* (Tech. Rep.). Technical report, Software Engineering Institute at Carnegie Mellon University, The address of the publisher, 3 2015.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE software*, *12*(6), 42–50.

Len, B., Paul, C. & Rick, K. (2003). Software architecture in practice. *Boston, Massachusetts Addison*.

Li, W., Lee, Y.-H., Tsai, W.-T., Xu, J., Son, Y.-S., Park, J.-H. & Moon, K.-D. (2012). Service-oriented smart home applications: composition, code generation, deployment, and execution. *Service Oriented Computing and Applications*, *6*(1), 65–79. Retrieved from http://dx.doi.org/10.1007/s11761-011-0086-7 doi: 10.1007/s11761-011-0086-7

Lillis, T. (2008). Ethnography as method, methodology, and "deep theorizing" closing

the gap between text and context in academic writing research [Journal Article]. *Written communication*, *25*(3), 353-388.

Loomis, M. E., Shah, A. V. & Rumbaugh, J. E. (n.d.). An object modeling technique for conceptual design [Conference Proceedings]. In *European conference on object-oriented programming* (p. 192-202). Springer.

Mainetti, L., Mighali, V. & Patrono, L. (2015). An iot-based user-centric ecosystem for heterogeneous smart home environments. In *Communications (icc), 2015 ieee international conference on* (pp. 704–709).

Mallett, R., Hagen-Zanker, J., Slater, R. & Duvendack, M. (2012). The benefits and challenges of using systematic reviews in international development research. *Journal of development effectiveness*, *4*(3), 445–455.

March, S. T. & Smith, G. F. (1995). Design and natural science research on information technology [Journal Article]. *Decision Support Systems*, *15*(4), 251-266. Retrieved from http://www.sciencedirect.com/science/article/pii/0167923694000412 doi: http://dx.doi.org/10.1016/0167-9236(94)00041-2

MarketsAndMarkets. (2017). *Smart home market by product (lighting control, security & access control, hvac, entertainment & other control, home healthcare, smart kitchen, and home appliances), software & service (behavioral, proactive), and geography - global forecast to 2023.*

Mason, D. & Criswell, C. A. (1998, November 10). *Method for negotiating software compatibility.* Google Patents. (US Patent 5,835,735)

Mazón, J.-N., Pardillo, J. & Trujillo, J. (2007). A model-driven goal-oriented requirement engineering approach for data warehouses. *Advances in conceptual modeling–foundations and applications*, 255–264.

Mens, T. & Van Gorp, P. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, *152*, 125–142.

Miller, P. (2000). Interoperability: What is it and why should i want it? *Ariadne*(24).

Mohagheghi, P. (2010). An approach for empirical evaluation of model-driven engineering in multiple dimensions.

Mohagheghi, P. & Dehlen, V. (2008). Where is the proof? - a review of experiences from applying mde in industry [Book Section]. In I. Schieferdecker & A. Hartman (Eds.), *Model driven architecture – foundations and applications: 4th european conference, ecmda-fa 2008, berlin, germany, june 9-13, 2008. proceedings* (p. 432-443). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-69100-6_31 doi: 10.1007/978-3-540-69100-6_31

Moher, D., Liberati, A., Tetzlaff, J., Altman, D. G. & Group, P. (2009). Preferred reporting items for systematic reviews and meta-analyses: the prisma statement [Journal Article]. *PLoS med*, *6*(7), e1000097.

Mohilo, D. (2017). *Eclipse oxygen: A better workflow for editing in sirius - jaxenter.* Retrieved from https://jaxenter.com/eclipse-oxygen-sirius-interview-134128.html

Myers, M. D. (1997). Qualitative research in information systems [Journal Article]. *Management Information Systems Quarterly*, *21*(2), 241-242.

Newman, I. & Benz, C. R. (1998). *Qualitative-quantitative research methodology: Exploring the interactive continuum* [Book]. SIU Press.

Ni, Q., García Hernando, A. B. & de la Cruz, I. P. (2015). The elderly's independent living in smart homes: A characterization of activities and sensing infrastructure survey to facilitate services development. *Sensors*, *15*(5), 11312–11362.

Nordstrom, G., Sztipanovits, J., Karsai, G. & Ledeczi, A. (1999). Metamodeling-rapid design and evolution of domain-specific modeling environments. In *Engineering of computer-based systems, 1999. proceedings. ecbs'99. ieee conference and workshop on* (pp. 68–74).

Peffers, K. E. N., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A design science research methodology for information systems research [Journal Article]. *Journal of Management Information Systems*, *24*(3), 45-77. Retrieved from `http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=28843849&site=eds-live`

Pelechano, V., Albert, M., Muñoz, J. & Cetina, C. (2006). Building tools for model driven development. comparing microsoft dsl tools and eclipse modeling plug-ins. In *Dsdm.*

Perumal, T., Sulaiman, M. N., Mustapha, N., Shahi, A. & Thinaharan, R. (2014). Proactive architecture for internet of things (iots) management in smart homes. In *Consumer electronics (gcce), 2014 ieee 3rd global conference on* (pp. 16–17).

Pesante, L. H. (2003). Software engineering institute (sei).

Reyes-Delgado, P. Y., Mora, M., Duran-Limon, H. A., Rodríguez-Martínez, L. C., O'Connor, R. V. & Mendoza-Gonzalez, R. (2016). The strengths and weaknesses of software architecture design in the rup, msf, mbase and rup-soa methodologies: A conceptual review. *Computer Standards & Interfaces*, *47*, 24–41.

Ritter, D. G. N., Zirpins, B. B. C., Schoenherr, G. F. M. & Motahari-Nezhad, H. R. (2007). Service-oriented computing icsoc 2006.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorenson, W. (1991). Object-oriented modelling and design.

Rumpe, B. (2016). Modeling with uml. *Language, Concepts, Methods. Springer International*, *4*.

Samimi-Dehkordi, L., Khalilian, A. & Zamani, B. (2014). Programming language criteria for model transformation evaluation. In *Computer and knowledge engineering (iccke), 2014 4th international econference on* (pp. 370–375).

Sangiovanni-Vincentelli, A. & Martin, G. (2001). Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, *18*(6), 23–33.

SAP, A. (2007). *Standardized technical architecture modeling: Conceptual and design level. version 1.0* (Tech. Rep.). Retrieved 2010-08-10 from http://www.fmc-modeling. org/download/fmc-and-tam/SAP-TAM_Standard. pdf.

Schamai, W., Fritzson, P., Paredis, C. & Pop, A. (2009). Towards unified system

modeling and simulation with modelicaml: modeling of executable behavior using graphical notations. In *Proceedings of the 7th international modelica conference; como; italy; 20-22 september 2009* (pp. 612–621).

Schmidt, D. C., Stal, M., Rohnert, H. & Buschmann, F. (2013). *Pattern-oriented software architecture, patterns for concurrent and networked objects* (Vol. 2). John Wiley & Sons.

Schneps-Schneppe, M., Maximenko, A., Namiot, D. & Malov, D. (2012). Wired smart home: energy metering, security, and emergency issues. In *Ultra modern telecommunications and control systems and workshops (icumt), 2012 4th international congress on* (pp. 405–410).

Shackel, B. (1991). Usability-context, framework, definition, design and evaluation. *Human factors for informatics usability*, 21–37.

Siddaway, A. (2014). What is a systematic literature review and how do i do one. *University of Stirling*(I), 1.

Sinha, R., Narula, A. & Grundy, J. (2017). Parametric statecharts: designing flexible iot apps: deploying android m-health apps in dynamic smart-homes. In *Proceedings of the australasian computer science week multiconference* (p. 28).

Smirek, L., Zimmermann, G. & Beigl, M. (2016). Just a smart home or your smart home – a framework for personalized user interfaces based on eclipse smart home and universal remote console [Journal Article]. *Procedia Computer Science*, *98*, 107-116. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1877050916321391` doi: http://dx.doi.org/10.1016/j.procs.2016.09.018

Steinberg, D., Budinsky, F., Merks, E. & Paternostro, M. (2008). *Emf: eclipse modeling framework*. Pearson Education.

Stojkoska, B. L. R. & Trivodaliev, K. V. (2017). A review of internet of things for smart home: Challenges and solutions. *Journal of Cleaner Production*, *140*, 1454–1464.

Tang, W. (2009). Meta object facility. In L. LIU & M. T. ÖZSU (Eds.), *Encyclopedia of database systems* (pp. 1722–1723). Boston, MA: Springer US. Retrieved from `https://doi.org/10.1007/978-0-387-39940-9_914` doi: 10.1007/978-0-387-39940-9_914

Vega-Barbas, M., Pau, I., MartÃn-Ruiz, M. L. & Seoane, F. (2015). Adaptive software architecture based on confident hci for the deployment of sensitive services in smart homes. *Sensors*, *15*(4), 7294–7322. Retrieved from `http://www.mdpi.com/1424-8220/15/4/7294` doi: 10.3390/s150407294

Vega-Barbas, M., Pau, I., Martín-Ruiz, M. & Seoane, F. (2015). Adaptive software architecture based on confident hci for the deployment of sensitive services in smart homes [Journal Article]. *Sensors*, *15*(4), 7294. Retrieved from `http://www.mdpi.com/1424-8220/15/4/7294`

Viswanathan, S. E. & Samuel, P. (2016). Automatic code generation using unified modeling language activity and sequence models [Journal Article]. *IET Software*, *10*(6), 164-172. Retrieved from `http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx`

?direct=true&db=bth&AN=120026706&site=eds-live doi: 10.1049/iet-sen.2015.0138

Whittle, J., Clark, T. & Kühne, T. (2011). Model driven engineering languages and systems. In *14th international conference, models* (pp. 16–21).

Wiegers, K. (1999). First things first: prioritizing requirements. *Software Development*, *7*(9), 48–53.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R. & Wood, B. (2006). *Attribute-driven design (add), version 2.0* (Tech. Rep.). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Wu, W. & Kelly, T. (2004). Safety tactics for software architecture design. In *Computer software and applications conference, 2004. compsac 2004. proceedings of the 28th annual international* (pp. 368–375).

Wąsowski, A. & Lönn, H. (2016). *Modelling foundations and applications : 12th european conference, ecmfa 2016, held as part of staf 2016, vienna, austria, july 6-7, 2016, proceedings.* Switzerland : Springer, 2016. Retrieved from http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cat05020a&AN=aut.b1943781x&site=eds-live

Xie, H., Liu, J., Hu, L., Yang, H. & Fu, X. (2015). Design of pilot-assisted load control valve for proportional flow control and fast opening performance based on dynamics modeling [Journal Article]. *Sensors and Actuators A: Physical*, *235*, 95-104. Retrieved from http://www.sciencedirect.com/science/article/pii/S092442471530159X doi: http://dx.doi.org/10.1016/j.sna.2015.09.042

Xu, K., Wang, X., Wei, W., Song, H. & Mao, B. (2016). Toward software defined smart home [Journal Article]. *IEEE Communications Magazine*, *54*(5), 116-122. doi: 10.1109/MCOM.2016.7470945

Yates, J. & Leggett, T. (2016). Qualitative research: An introduction [Journal Article]. *Radiologic Technology*, *88*(2), 225-231. Retrieved from http://ezproxy.aut.ac.nz/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=rzh&AN=119047675&site=eds-live

Zillner, T. (2015). *White paper: Zigbee exploited–the good, the bad and the ugly* (Tech. Rep.). Technical report, Cognosec.

Appendix A

Appendix

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:javaCodeModel="http://
www.example.org/javaCodeModel">
  <javaCodeModel:JavaClass Name="Smart Light System"/>
  <javaCodeModel:JavaStatement ifstatement="/8">
    <Dec_Assign_Statement>boolean man_walks_in = true</Dec_Assign_Statement>
    <Dec_Assign_Statement>boolean motion_sensor = false</Dec_Assign_Statement>
    <Dec_Assign_Statement>boolean door_sensor = false</Dec_Assign_Statement>
    <Dec_Assign_Statement>boolean light = false</Dec_Assign_Statement>
    <Dec_Assign_Statement>boolean man_in_the_room = true</Dec_Assign_Statement>
    <Title>boolean door_sensor = false</Title>
    <Title>boolean light = false</Title>
    <Title>boolean man_walks_in = true</Title>
    <Title>boolean motion_sensor = fasle</Title>
    <Title>boolean man_in_the_room = true</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:JavaStatement ifstatement="/9 /10">
    <Dec_Assign_Statement>door_sensor = true</Dec_Assign_Statement>
    <Title>door_sensor = true</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:JavaStatement endstatement="/12">
    <Dec_Assign_Statement>man_walks_in = false</Dec_Assign_Statement>
    <Dec_Assign_Statement>door_sensor = false</Dec_Assign_Statement>
    <Title>man_walks_in = false</Title>
    <Title>door_sensor = false</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:JavaStatement ifstatement="/9">
    <Dec_Assign_Statement>motion_sensor = true</Dec_Assign_Statement>
    <Title>motion_sensor = true</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:JavaStatement endstatement="/12">
    <Title>man_in_the_room = false</Title>
    <Title>motion_sensor = false</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:JavaStatement endstatement="/13">
    <Title>light = true</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:JavaStatement endstatement="/13">
    <Title>light = false</Title>
  </javaCodeModel:JavaStatement>
  <javaCodeModel:IfStatement Name="if(man_walks_in == true)" Yes="/2" No="/3"/>
  <javaCodeModel:IfStatement Name="if(door_sensor == true &amp;amp;&amp;amp; motion_sensor == true)"
Yes="/6" No="/7"/>
  <javaCodeModel:IfStatement Name="if(man_int_the_room == true)" Yes="/4" No="/5"/>
  <javaCodeModel:BeginStatement javastatement="/1"/>
  <javaCodeModel:EndStatement/>
  <javaCodeModel:EndStatement/>
</xmi:XMI>
```

Figure A.1: Java Model Instance in XMI Format

A.1 Market Need for Smart Homes

In addition to the benefits provided by smart homes as highlighted in section 1.2 of the chapter, presently the smart homes have an increased level of demand. This section identifies the demand for smart homes in different parts of the world due to various additional benefits.

The smart home market is expected to reach USD 121.73 Billion by the year 2022, at a CAGR of 14.07 percent between the years 2016 and 2022. This report aims to identify and highlight the market size and future potential growth of smart home market based on various products and appliances, software, services and geography (MarketsAndMarkets, 2017).

In addition to the above, the smart home market is categorized based on various other factors such as lighting control, security and access control, HVAC control, entertainment and controls, home, healthcare and smart kitchen (MarketsAndMarkets, 2017).

In terms of the geography, the market need for smart homes had been segmented into North America, APAC, and RoW. The growth of this market is expected to rise when the increase in the demand for lighting, entertainment controls, and home health care are taken into consideration. Additionally, the demand for smart homes is expected to increase as result of the benefits offered and real-time energy consumption feedback achieved using proactive solutions. However, due to higher levels of demand expectations for smart homes in the future, the installation rate of smart home systems is expected to increase during the forecast period (MarketsAndMarkets, 2017).

When the US smart home market is taken into account, currently the smart home market is believed to be stuck in “chasm” of the technology adoption curve

whereby difficulties are confronted while attempting to move from early adopter phase to the mass market phase of the adoption. This is occurred due to various existing barriers, the barriers include high device barriers, limited consumer demand and long device replacement cycles and technological fragmentation of smart home ecosystems which requires the consumers to purchase multiple networking devices and the need to use multiple apps to build and run in a single smart home.

However, according to BI intelligence, the demand for smart home systems is expected increase despite the existing barriers and challenges. This is due to analysis carried out which determines the possible areas of growth and methods to overcome the barriers. As a result, the smart home has begun to become more prevalent throughout the US. This is mainly due to the smart home being a stand-alone object which can be connected to the internet and can be controlled from remote locations and has non-computing primary functions. Furthermore, the smart home user is permitted to have multiple smart home devices within a single smart home thus supporting the smart home ecosystems.

In addition to the above, high prices of smart home systems coupled with lack of demand thus preventing the smart home market from moving to the mass market stage from the early adoption stage.

The diagram below describes how the smart home systems are able to move from the early adoption stage to mass market with the support of BI intelligence.

This article describes the smart home market size and relative trends and projections.

"4 billion consumer-facing 'things' are predicted to be connected to the internet worldwide in 2016, up from 3 billion in 2015." — Deloitte

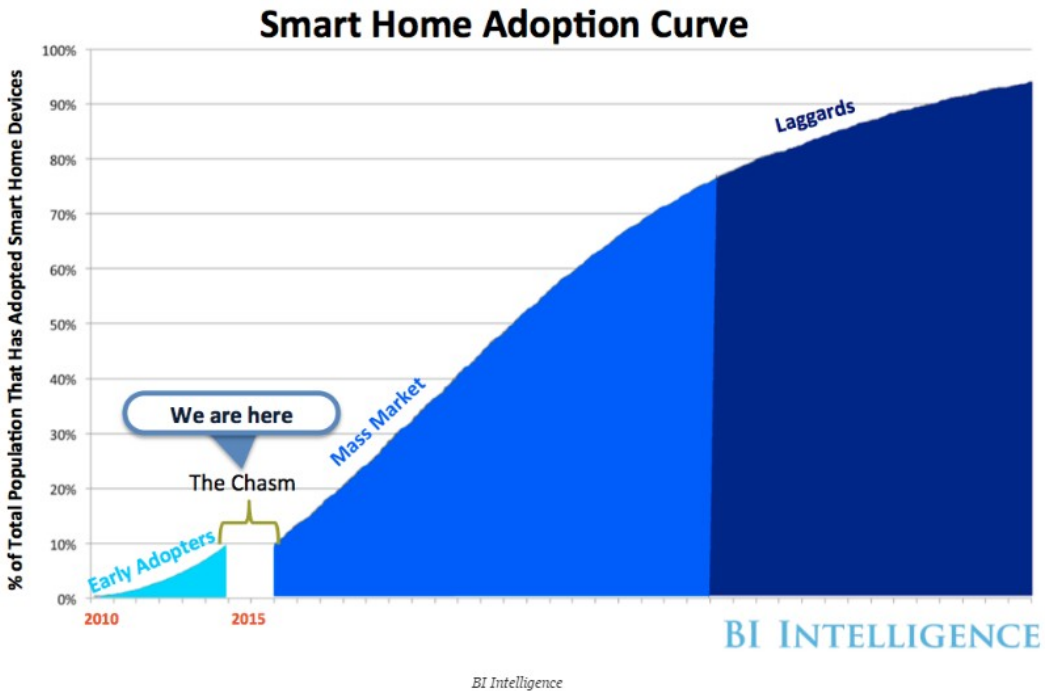

Figure A.2: Smart Home Adoption Curve

The above statement is supported by IoT and Machine-to-Machine which involves the communication between devices without having the need for human interaction. As per the Machina Research predicts, IoT market is forecasted to be worth USD 4.3 million by 2024. As mentioned previously, even though various challenges exist in terms of costs and interoperability during device integration, the percentage of consumers considering to purchase smart homes during the next 12 months are as shown below.

In addition to the above, in the year 2013 smart home global automation market held a value of 4.4 billion US Dollars which is expected to increase to 21 billion US Dollars by 2020. Additionally, revenue of smart home worldwide is expected to grow from 20.38 billion US Dollars to 58.58 billion US Dollars by the year, 2020. According to Berg Insights, North America will make up a majority of this market as shown in the graph.

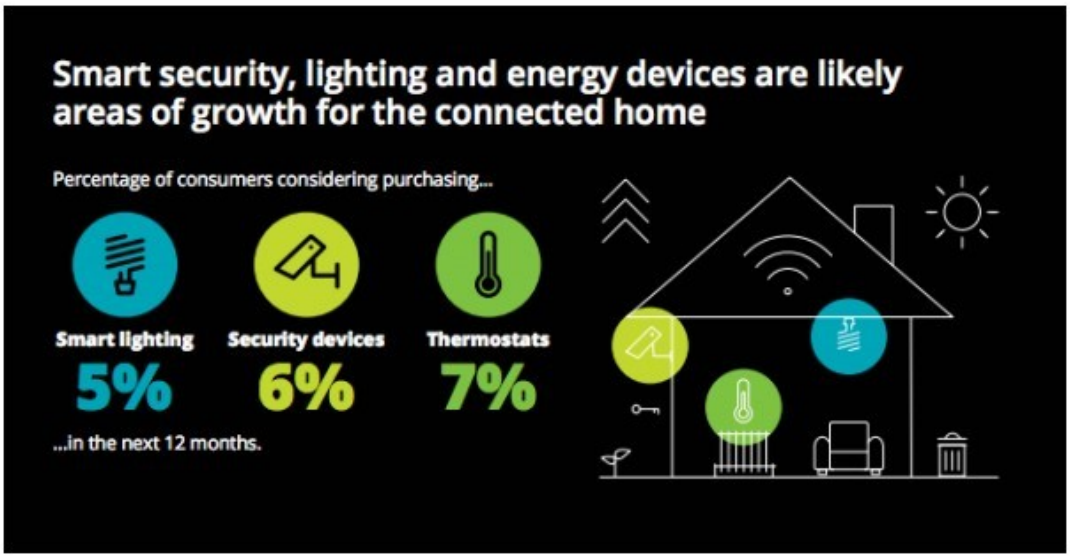

Figure A.3: Growth of Demand for Smart Devices

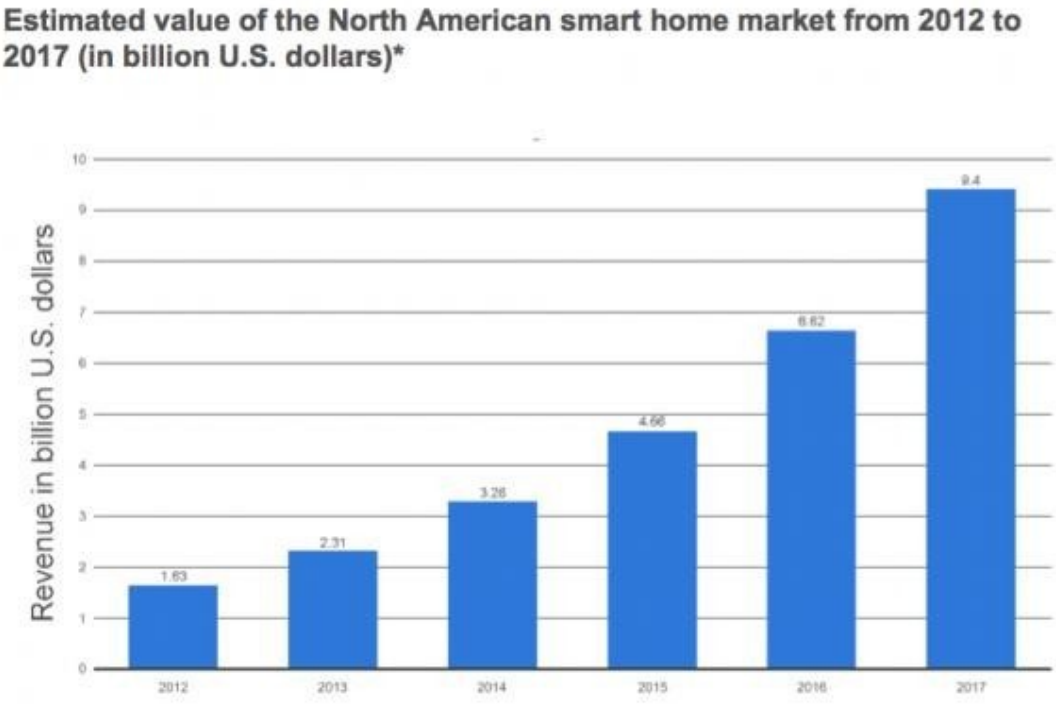

Figure A.4: Value of North American Smart Home Market 2012 - 2017

As per Fung Global Retail Technology, smart home systems offer various functionalities in terms of different aspects, these aspects include security, lighting, temperature, climate control, energy management, kitchen and other home equipment

and home entertainment. In addition, while the internet connectivity supports the users in connectivity aspect, the smart devices are also able to sync with each other which in turn may support in improving consumer experience which will eventually result in mass-market home automation.

As per the industry insights, in 2013 the global smart home industry was estimated at US Dollars 13.07 billion in 2013 which was majorly supported the increase in consumer interest towards energy efficiency and optimum resource utilization. Additionally, growing importance towards security issues played a major role in increasing the demand for smart and connected homes over the anticipated period.

As a result of forecasted demand increase for smart home automation system, real estate industry is forecasted to experience growth in sales. These facts are also supported by the practice of better M2M communication systems, improved IoT and increase in geriatric population base.

In addition to the above, the predicted demand for smart home systems in Asia Pacific smart home markets by the application is shown below. The applications include Security, entertainment, HVAC, lighting, energy management and other.

In terms of healthcare and clinical treatments, smart homes are able to provide the support these scenarios with the help of Information and Communication Technology is able to effectively facilitate novel solutions. However, elderly users in smart homes may find the practice of ICT challenging, as a result, integration of ICT with homes would, in turn, support the communication and automation capabilities thus emphasizing on customization for every particular smart home based on preferences of various users. Additionally, integration of technologies in a single smart home environment would facilitate deployment of useful services and in turn maximize user acceptance (Vega-Barbas, Pau, MartÃn-Ruiz & Seoane, 2015).

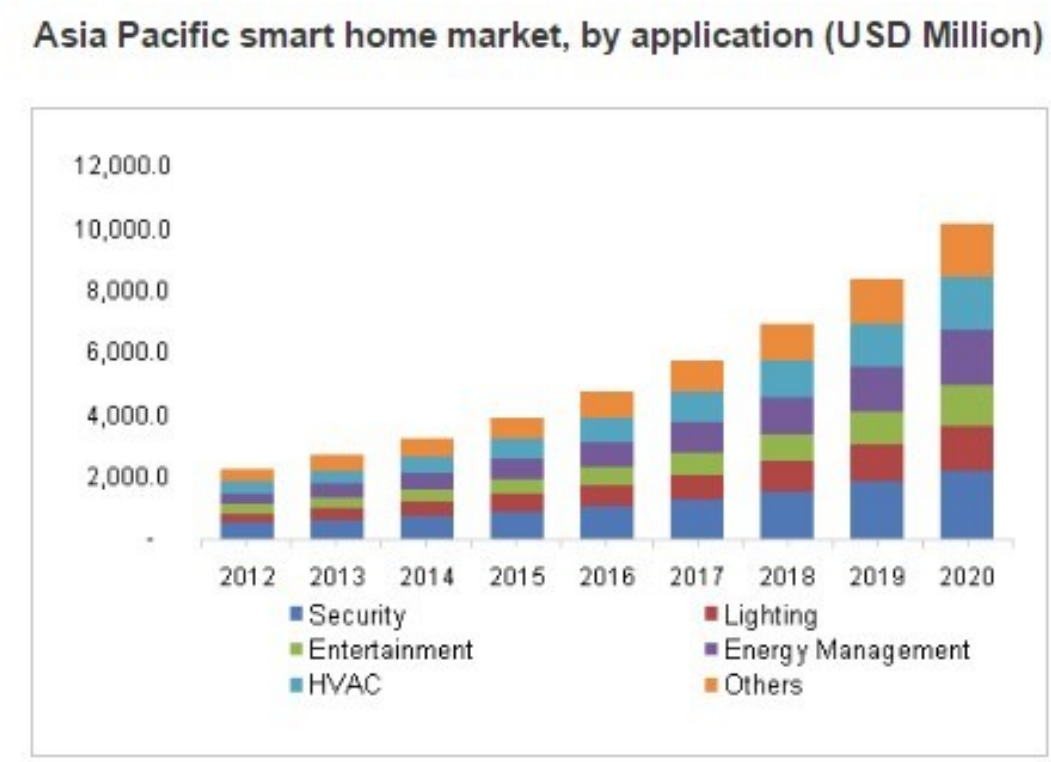

Figure A.5: Smart Home Market in Asia Pacific

ICT and Internet of Things have produced various benefits to smart home systems, these include low power and low-cost devices which can be used in a smart home to create a network of interconnected smart objects. However, this is affected by the heterogeneity of technologies which would prevent the smart objects from interoperating in order to adapt to a single smart home environment. This conflict is addressed by adopting a software ecosystem which is able to interact with the smart devices directly which would also allow the users to define customized interfaces for mobile devices. Furthermore, a multi-protocol can be used which would allow both services and mobile applications to access the physical networks thus hiding the heterogeneities (Mainetti, Mighali & Patrono, 2015).

Domoticz

Domotics, a home automation system is designed to allow its users to use and configure various devices such as lighting systems, switches, various other sensors such as temperature, wind, gas, and water. The notifications and alerts received from these sensors are directed to any mobile device. Moreover, this system is designed to function in various operating systems. The user interface of Domoticz is a scalable HTML5 web front end which is designed to adapt to both desktop and mobile devices. This automation system includes various other additional functions such as extended logging, iPhone and Android push notifications, auto-learning sensors and switches, share and use external devices and achieve improved levels of in terms of simplicity in regard to the overall design and use.

A.2 Overview of the Proposed Solution

The new studies will emphasize on the new, proposed solution that will address the above-defined research question thus ensuring the problem definition is solved to a greater level. This study will allow the user to create a very simple visual design thus ensuring higher levels of understanding the ability and user-friendliness, which the system may accept as the user input with the relative sensor actuator configurations. The new system introduces a compiler which allows the domain expert to write a visual domain-specific plan which is later compiled to software code. This code is then deployed to a dynamic smart home system with the help of customized smart space. The smart space consists of the sensors, actuators and other network platforms.

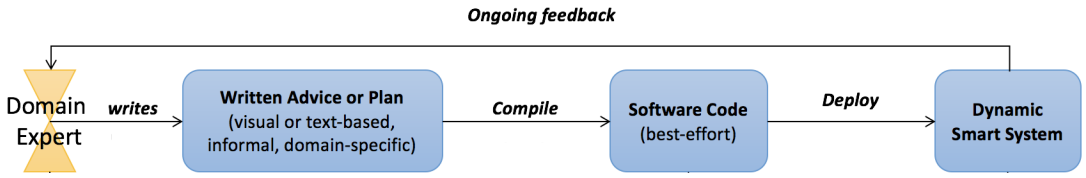

Figure A.6: Overview of the Proposed Solution