2022

Model-Based Testing of Smart Home Systems Using EFSM, CEFSM, and FSMApp

Afnan Mohammed Albahli
University of Denver

Model-Based Testing of Smart Home Systems Using EFSM, CEFSM, and FSMApp

Abstract

Smart Home Systems (SHS) are some of the most popular Internet of Things (IoT) applications. In 2021, there were 52.22 million smart homes in the United States and they are expected to grow to 77.1 million in 2025 [71]. According to MediaPost [74], 69 percent of American households have at least one smart home device. The number of smart home systems poses a challenge for software testers to find the right approach to test these systems. This dissertation employs Extended Finite State Machines (EFSMs) [6, 24, 105], Communicating Extended Finite State Machines (EFSMs) [68] and FSMApp [10] to generate reusable test-ready models of smart home systems. We present an approach to create reusable test-ready models of smart home systems using EFSMs to model device components (Sensor, Controller and Actuator), EFSMs to model single devices in the SHS and the interaction between the devices. We adopted Al Haddad's [10] FSMApp approach to model and test the mobile application that controls the SHS. These reusable test-ready models were used to generate tests. This dissertation also addresses evolution in smart home systems. Evolution is classified into three categories: adding a new device, updating an excising device or removing one. A method for selective black-box model-based regression testing for these changes was proposed.

Document Type

Dissertation

Degree Name

Ph.D.

Department

Computer Science

First Advisor

Anneliese Amschler Andrews

Second Advisor

Chip Reichardt

Third Advisor

Scott Leutenegger

Keywords

Smart home systems, Applications, Software

Subject Categories

Computer Sciences | Other Computer Sciences | Software Engineering

Model-Based Testing of Smart Home Systems Using EFSM, CEFSM, and FSMApp

a dissertation

Presented to

the Faculty of the Daniel Felix Ritchie School of Engineering and Computer Science

University of Denver

in Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Afnan Mohammed Albahli

August 2022

Advisor: Prof. Anneliese Amschler Andrews

Author: Afnan Mohammed Albahli
Title: Model-Based Testing of Smart Home Systems Using EFSM, CEFSM, and FSMApp
Advisor: Prof. Anneliese Amschler Andrews
Degree Date: August 2022

Abstract

Smart Home Systems (SHS) are some of the most popular Internet of Things (IoT) applications. In 2021, there were 52.22 million smart homes in the United States and they are expected to grow to 77.1 million in 2025 [71]. According to MediaPost [74], 69 percent of American households have at least one smart home device. The number of smart home systems poses a challenge for software testers to find the right approach to test these systems. This dissertation employs Extended Finite State Machines (EFSMs) [6, 24, 105], Communicating Extended Finite State Machines (EFSMs) [68] and FSMApp [10] to generate reusable test-ready models of smart home systems. We present an approach to create reusable test-ready models of smart home systems using EFSMs to model device components (Sensor, Controller and Actuator), EFSMs to model single devices in the SHS and the interaction between the devices. We adopted Al Haddad's [10] FSMApp approach to model and test the mobile application that controls the SHS. These reusable test-ready models were used to generate tests. This dissertation also addresses evolution in smart home systems. Evolution is classified into three categories: adding a new device, updating an excising device or removing one. A method for selective black-box model-based regression testing for these changes was proposed.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Professor Anneliese Andrews for the unlimited help and guidance she provided during my PhD study and research journeys.I truly cannot imagine a better advisor.

I would also like to thank Dr. Chip Reichardt, Dr. Scott Leutenegger and Dr. Chris GauthierDickey for serving in my oral defence committee.

I am also grateful to the Computer Science Department faculty and staff for their support. Special thanks to lab mates and colleagues, Lamees Alhazzaa, Zeinab Abdalla, and Jide Williams for their continuous support.

I am grateful for my wonderful family who made achievement possible and rewarding. My darling husband Turki Aljebreen deserves a particular mention for motivating me when I needed it the most and encouraging me to pursue my goals. I am thankful to my parents, Mohammed and Latefa, for their constant love and assistance. For their prayers and words of wisdom. I also express my thanks to my loving sisters and brother for their never-ending support. To my adorable kids Khalid, Lama and Mohammed, who have served as my inspiration, drive, and motivation. Finally, I want to express my gratitude to Princess Nourah Bint Abdulrahman University for their scholarship and to my country for funding my education.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The Internet of Things (IoT) refers to a combination of devices connected and data shared between those devices via platforms [63]. The term IoT was coined in 1999 and has become widely used over the past decade [73]. The number of devices connected to the Internet has increased rapidly and is expected to continue to grow. In 2020, the IoT analytics estimated that by 2025 the number of connected devices will be more than 30 billion worldwide [71], while the International Data Corporation (IDC) predicts that the number of connected devices will reach 55.9 billion worldwide, generating 79.4 zettabytes (ZB) of data by 2025 [60].

A Smart Home System (SHS) is one of many IoT applications. According to Medium, the number of people who search for smart home systems increase monthly by about 60,000 people [50]. That puts smart home systems among the most common IoT applications, along with other IoT applications such as: smart city, wearable devices, smart grids, connected health, connected cars, and so on. However, smart home systems are the most popular of these applications for personal use.

According to Statista [37], the number of smart home systems in the United States is expected to be 77,1 million in 2025. Fig 1.1 shows the annual increase over the years. The massive number of Smart Home Systems today and the expectation for

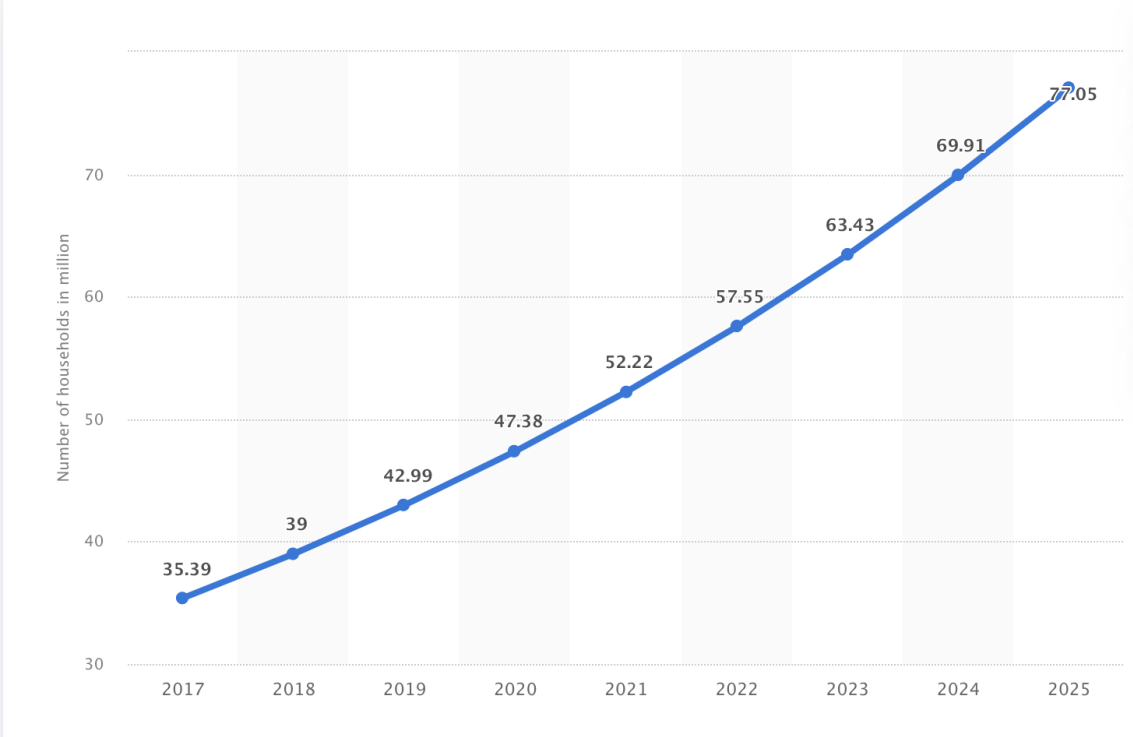

Figure (1.1) Number of Smart Homes in the United States 2017-2025 [37]

it to increase in the future makes it important for systematic testing to also increase.

The Internet of things (IoT) is a collection of devices connected to the Internet. These devices are objects, and those objects can be classified based on their intelligence as smart objects or non-smart objects. Examples of smart objects include smartphones, controllers, or robots, while non-smart objects include sensors and/or actuators [45]. However, new sensors in the market today are often supported with artificial intelligence which allows them to process data, if needed, before sending it to the actuators without the need to send it to a controller smart object, and thus also may be classified as smart objects [89]. Sensors can be defined as the parts

in the smart home system that collect data from the environment (room) or smart objects (smart bulb). This data can be user data such as sign-on time, period of usage, and location and so on. Instead, it could be device data such as room temperature, water level, and brightness and others. Actuators are devices that take actions based on the sensed data which is how the actuator controls the physical environment. An example of data used by an actuator are the temperature control values that are used in smart homes. Mainly, a smart home system consists of a collection of devices most of which fit into the non-smart objects category.

Devices in an IoT environment have the ability and need to interact with each other, as well as interacting with the controller. In most cases this is a mobile application or software on a tablet or desktop. These devices also tend to change over time. For example, a device can be updated and a newer version of that device with new features that can be introduced. Therefore, the dynamic nature of devices must be addressed and considered when any IoT system is studied. Once the IoT system is installed, more devices can be connected, and others might be removed. Some of these devices can work in-parallel. Others can work as stand-alone devices.

With the fast growth in the number of devices connected to the Internet, it becomes very important to guarantee the reliability of such devices as a failure could have quite negative impacts. The sheer number and diversity of devices connected to the Internet presents a challenge for software testers to find a suitable approach to test these systems. Software related failures become increasingly hard to detect but they can have disastrous repercussions. For example, a person might have a smart smoke detector installed that alerts the owner when it detects smoke via a phone app. If the smoke detector is compatible with other smart devices, it might be able to shut down the HVAC system during a fire, turn on cameras to record the emergency and unlock the front doors. If this is not tested well and in the case of a

false alarm, it could lead to shutting down the HVAC, pipes could freeze because of a faulty device or controller software. Opening your front door could lead to theft. To assure robustness and safety, as well as to reduce the risk of failures, it becomes increasingly important that IoT systems be tested systematically and thoroughly before being placed into operation.

However, there is little research on systematic and thorough testing of IoT systems. In general most of the IoT system testing is performed ad hoc. Ad hoc testing can successfully find problems; however, it heavily depends on the testers' experience and skills. The quality of the software systems cannot be assured using the ad hoc approach. A systematic approach is needed to assure that relevant behaviors, including emergent behaviors, are addressed in a rigorous and disciplined manner.

It is a challenge to model smart home systems because of the large number of devices and the dynamic nature of these devices. Therefore, offering reusable models and tests will help in saving time and effort. Testing IoT can be divided into three levels: device testing, device interaction and system level testing. According to [52, 104] most testing is done at the device level because it is easier. However, it is important to test all levels to assure the quality of IoT systems. Device interaction testing is important because a lot of interactions between devices are possible and must work properly. Many characteristics of IoT systems make them challenging to test. Heterogeneity in IoT, standardization problems, security and privacy challenges, complexity with interoperability and test environment issues are challenges facing IoT testers [49]. Many researchers agree that there is not enough work on IoT testing [31, 92]. However, the quality of such systems is critical and this is pushed to the sideline due to the fast innovation in the IoT market [31].

This dissertation aims to tackle the open challenge in testing IoT systems to support design and development of effective and systematic software testing method-

ologies for smart home systems. It employs Model Based Testing (MBT) to create a set of reusable models for testing smart home systems at the device, device interaction and system levels. It uses Extended Finite State Machines (EFSM) to model and test device components and Communicating Extended Finite State Machines (CEFSM) to model and test single device and interaction between devices. For system level testing, FSMApp [10] is adopted to test the controller of the entire system, which is a smart phone application. We proposed test generation approaches suitable for these models. This dissertation will offer reusable models and tests where a user of such system can select from a list of ready models and their tests, or can at least perform necessary changes to the excising models and the generated tests as defined in the proposed approach. To show the reusability of these models and test artifacts, we demonstrate that they are easily modified when adding new devices or upgrading existing devices. In effect, this is accomplished by providing a customized regression testing approach.

1.2 Research Scope and Questions

In this section a number of research questions are developed in order to identify our scope. These research questions will be further expanded and answered.

- RQ1: Is it possible to develop Black-Box Models for modeling device behavior, device interaction and the controller?
  - RQ1.1: What research exists for testing smart home systems that use IoT?
  - RQ1.2: What are the existing research gaps related to testing smart home systems using IoT?

  - RQ1.3: Is it possible to develop Black-Box Models for modeling device behavior?
  - RQ1.4: Is it possible to develop Black-Box Models for modeling device interaction?
  - RQ1.5: Is it possible to develop Black-Box Models for modeling the controller?

- RQ2: Is it possible to develop a MBT technique for these models that includes testing criteria and can be used for device testing, interaction testing and system testing?

- RQ3: Is it possible to develop a special set of reusable test models and other test artifacts so software testers do not have to develop models from scratch for every new smart home system?

- RQ4: How do we effectively and efficiently modify models and test artifacts when technology changes (devices, device interactions, and controllers)?

1.3 Research Agenda

The following work is conducted to answer the research questions:

- For RQ1: A collection of reusable test-ready models are developed for devices, controllers and their interaction. (Chapter 3)

- For RQ1.1 and RQ1.2: A systematic mapping study of the existing testing techniques for smart home systems that used IoT is conducted. (Chapter 2)

- For RQ1.3, RQ1.4 and RQ1.5: Black-Box models are provided for modeling device behavior, device interactions and the controller. (Chapter 3)

- RQ2: A MBT technique is provided for these models that includes testing criteria and can be used for device testing, device interaction testing and system testing. Tests are reusable or customizable. (Chapter 4)

- RQ3: A set of reusable test models and other test artifacts is provided so software testers do not have to develop models from scratch for every new smart home system. (Chapter 3)

- RQ4: How to adjust and modify models and test artifacts when technology changes (devices, device interactions, and controllers) is explained. (Chapter 5)

This document is organized as follows: Chapter 2 covers existing work in MBT using FSM, EFSM and CEFSM, as well as a systematic mapping study of the existing testing techniques for smart home systems that used IoT. A set of reusable test-ready models of smart home systems for the three different testing levels is presented in Chapter 3. Chapter 4 uses these models in the form of an MBT testing approach for smart home systems to model and test single devices, to test device integration and for system level testing. How to efficiently and easily adopt models and test artifacts when technology changes is introduced in Chapter 5. Future work is presented in Chapter 6. A conclusion of this dissertation is drown in Chapter 7. Fig 1.2 shows the structure of the dissertation.

Table 1.1 shows the list of current publications and their location in the document.

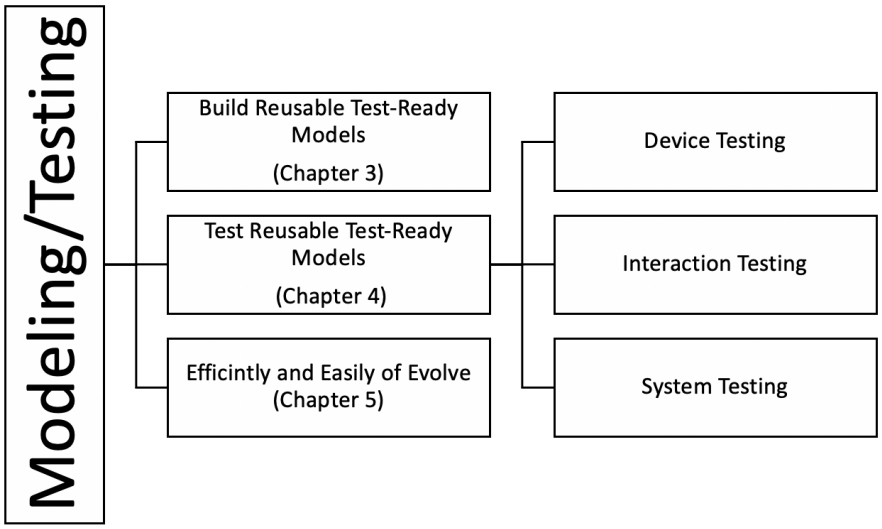

Figure (1.2) Proposed Modeling and Testing

Publication	**Venue**	**Status**	**Chapter**
Model-Based Testing of Smart Home Systems. [8]	ICOMP'21	Published	Ch. 4
Model-Based Testing of Smart Home Systems Using EFSM and CEFSM. [9]	CSCI'21	Published	Ch. 4
Build Test-ready and Reusable Models for Systematic Testing of Smart Home Systems. [7]	CSCE'22	Published	Ch. 3

Table (1.1) Current Publications List

Chapter 2

Background

The main goal of software testing is failure detection when a tester observes a difference between the expected behavior of the system under test (SUT) and the implemented system.

2.1 Model-Based Testing (MBT)

According to Utting et al. [103], MBT is a black box testing technique that generates test cases from a model of the SUT. MBT is an approach to generate test cases using a model of the system under test [80]. 219 MBT strategies are described by Dias-Neto et al. [39]. They described how to choose MBT approaches for software projects, as well as risk considerations that may impact their adoption in industry and how to mitigate them. One advantage of MBT is that test cases can be generated automatically from a model instead of manually [109]. Another advantage of MBT is that this technique allows to test the SUT earlier in the software development life cycle. This reduces cost and time; and improves testing quality [102].

In general, the MBT process starts with building the test model of the SUT based on the system requirements. Then, test selection criteria are chosen. After that, the test selection criteria are used to develop test case specifications. The next step uses this test model and the test case specifications to generate test cases. Finally, the test cases are executed and results are validated. Utting et al. [103] provide a taxonomy of MBT approaches: model scope, characteristics, paradigm, test selection criteria, test generation technology and test execution. They also provide a categorization of MBT notations as State Based, History Based, Functional, Operational, Stochastic, and Transition based.

The focus of transition based notations is on presenting the transitions between the states in the system. Several graph-based models exist such as Finite State Machines (FSMs), Extended Finite State Machines (EFSMs), and Communicating Extended Finite State Machines (CEFSMs). Other examples of transition based notations are UML behavioral models (such as activity diagrams, sequence and interaction diagrams), UML state charts, and Simulink Stateflow charts [103]. In the coming subsections FSMs, EFSMs and CEFSMs will be defined and examples of existing approaches using each of them will be given.

2.2 Finite State Machines (FSMs)

Finite State Machines (FSM) has a long and illustrious history. Since 1970, it has been used as a tool to generate test [35, 56, 58, 86]. FSMs model state-based software behavior [23]. FSMs model the functionality of an application or system by representing the states of the system usually by nodes and the transitions between states by edges. Huang et al. [58] suggest that each edge in an FSM be covered, and Howden [56] suggest that a complete roundtrip be covered without looping.

Pimont and Rault [86] suggest pairs of edges be covered. FSMs have been widely used to model and test different types of software systems. They have been used to test object oriented programs [44, 67]. Offutt et al. [82, 83] use FSM to develop models of formal software requirements. They then adapted control flow-based test criteria to define several testing criteria on the transitions. Hierons [55] uses FSM to model distributed systems. Chow [35] uses FSMs to model control structure. To validate these models, Chow introduces a testing strategy where test sequences are guaranteed to detect errors in the control structure. He uses a specification FSM and and an implementation FSM and compare them. To verify his proposed approach he applies it to three case studies in three different fields: computer graphics, real-time process control, and telephone switching. Andrews et al. [23], introduce FSMWeb to test web applications. In addition, Andrews et al. [18], use FSMWeb to propose an approach for regression testing of web applications. Alhaddad [10], develops FSMApp as an extension of FSMWeb to test mobile applications. FSMWeb and FSMApp do not consider parallelism. Therefore we cannot adapt it to test behaviour of IoT systems that have devices working in parallel. However, FSMApp could be used to test the smart home controller as an App on the Smartphone.

2.3 Extended Finite State Machines (EFSMs)

An EFSM (M) is defined as a 6-tuple (S, s0, PI , PO, V, T) where [6, 24, 105]:

- S is a finite set of states,
- s0 $\in$ S is an initial state,
- PI is a finite set of inputs parameters,

- PO is a finite set of outputs parameters,
- V is a finite set of variables and
- T is a finite set of transitions

Each transition, t $\in$ T is defined as 6-tuple (ss, st, ie, ef, uf, od), where:

- ss $\in$ S is the source state
- st $\in$ S is the target state
- ie is input event and defined in the format event (PI); it represents the interaction of the input event with a list of input parameters, PI .
- ef is the logical expression called guard or enable function and expressed by variables from V ,PI, constants and comparison operators.
- uf is the assignment statement called update function that updates variables v $\in$ V and parameters p $\in$ PO.
- od is the output statement defined in the format disp(), which refers to display the output with a list of parameters either PI ,V ,PO, or constants.

Cheng and Krishnakumar [32], state that EFSMs is an extension of the traditional FSM introduced to overcome major shortcomings of the traditional FSM. This is achieved by adding trigger conditions that need to be satisfied in order for transitions between states to occur. In addition, EFSMs introduced output actions, which are sets of actions to be executed once a transition occurs. It also introduces variables. EFSM extends FSM by modeling a system with control and data parts (unlike FSM which only can model the control parts of a system) [62]. It was adopted to solve the problem of generating test cases manually. When systems grow

huge and fast it became inefficient and error prone to do that manually [110]. Kalaji et al. [62] mention that path feasibility and path test data generation as a challenge during test generation using EFSM. When a conflict occurs between two or more transitions within a test it renders the test path semantically infeasible. Hierons et al. [55], address path feasibility issues in EFSM models by converting EFSM back to FSM. However, due to the huge number of states in FSM models this approach may cause state space explosion. Guglielmo et al. [38] generate test paths from extended finite state machine (EFSM) models. Fowler [40] proposes an automatic test pattern generation approach using EFSMs.

2.4 Communicating Extended Finite State Machines (CEFSMs)

Communicating Extended Finite State Machines (CEFSM) added communication channels between EFSMs to allow for communication. This structure gives us the capability of modeling smart home systems that consists of multiple devices modelled as EFSMs and there is a high possibility to interaction between devices and there is an interaction between devices and the Mobile Apps as controllers. CEFSMs can be defined as a finite set of consistent and completely specified EFSMs [32] that are composed via communication channels that carry input and output messages [68]: A CEFSM = (S, s0, E, P, T, A, M, V, C), such that:

- S is a finite set of states,
- s0 is the initial state,
- E is a set of events,

- P is a set of predicates P = (event, [conditions], actions, messages),
- T is a set of transition functions T: S×P×E→S×A×M,
- A is a set of actions,
- M is a set of communicating messages,
- V is a set of variables, and
- C is the set of input/output communication channels.

CEFSMs are utilized in testing by combining the system under test's behavior as a single machine, generating test paths from the CEFSM model, doing reachability analysis to exclude infeasible test paths, and finally generating test cases for these test paths. However, the states explosion problem is still possible due to the presence of variables and conditional statements in CEFSMs.

CEFSM has been used in modeling and testing distributed systems and network protocols with multiple components [106]. One of CEFSM's strength is that it can model orthogonal states of a system in a flat manner and without the need to compose the entire system in one state [22]. CEFSM has been also used to model robotic systems. Andrews et al. [14] propose a dynamic world testing technique that models the operating environment for autonomous systems. CEFSM was used to depict an active world model that describes environmental actors' behavior. They then used graph coverage criteria to build test paths to cover the active world mode. They also use input-space partitioning to generate test data, which is then used to turn the generated test paths into executable test cases. They use an indoor tour-guide robot as a case study for their methodology. For testing autonomous robotic systems, Andrews et al. [15] offer an MBT technique. They employ CEFSM

and Petri Nets to model the interaction between robots and environment objects. They applied their MBT technique to autonomous robotic applications such as an autonomous ground vehicle (AGV) [14], a tour-guide robot [16], and a search and rescue robot [17]. The findings demonstrate that MBT is a cost-effective method for testing autonomous robotic systems. The results also show that formal models such as CEFSM and Petri Nets can be used to generate test paths. Abdelgawad et al. [3] present a systematic Model-based Testing strategy that specifies what, where, and how to test autonomous system worlds, uncover various forms of errors, enhance scalability, and avoid the state space explosion problem. Instead of flattening all behaviors of environment actors into a single behavioral model, they utilize a hierarchical modeling technique. To develop more scalable models, they also divide the environment landscape into Snippets. Each fragment is linked to a cast of characters. The snippets are connected and modeled using CEFSM. Abdelgawad et al. [4] model a Real-time Adaptive Motion Planning Systems (RAMP) by using CEFSMs. They used different coverage criteria to generate test paths. Edge coverage was used to generate tests for a single component in the system. Edge-Pair coverage was used to generate tests for the interaction between each pair of components in the system. Several studies use CEFSM to test safety critical systems. In [22], a MBT approach to test safety critical systems was proposed by using CEFSM to model both functional behavior and mitigation requirements. CEFSMs were used for testing proper failure mitigation in safety-critical systems [21, 42]. Gannous et al. [43], demonstrate how a testing framework built for safety-critical systems may be used to test a portable insulin pump system, as well as looking at the framework's applicability and contribution to the certification process in the medical domain. The testing framework was created and modeled using EFSMs and CEFSMs. The main goal was to provide enough evidence for an efficient safety certification process; as a re-

sult, it incorporates modeling, safety analysis, and combinatorial testing techniques to provide expanded testing activity outputs that generate a variety of evidence types. A fail-safe testing technique was proposed by Gario et al. [46]. They provide behavioral models using Communicating EFSMs (CEFSM) and a fault model with fault trees for each probable failure. A compatibility procedure transforms the fault trees into CEFSM representation, which is subsequently integrated into the behavioral model. They build test paths for failures using various graph coverage criteria. Li and Wong [70] introduce an automatic test generation methodology from system specifications modeled with CEFSMs.

One issue with CEFSMs is reachability, that is whether a path is semantically feasible. Hessel and Pettersson [54] propose a global algorithm to generate all feasible test paths from a CEFSMs. They implemented and used this algorithm in different experiments and a case study. Bourhfir et al. [30, 29] apply reachability analysis to produce test cases from systems modeled in CEFSMs. They proposed an incremental technique to generate tests and execute them for communication protocols modeled by CEFSMs. Henniger et al. [53] describe the behavior of a system of asynchronous CEFSMs for testing purposes. Mutation testing is used by Kovas et al. [65] to automate test selection in an CEFSM model. To generate test cases in CEFSM models, Boroday et al. [28] combine specification and fault coverage.

2.5 A Systematic Mapping Study of Testing Techniques for Smart Homes that Use IoT

With the fast growth of the number of connected devices to the Internet, it becomes very important to guarantee the reliability of these devices as a failure

could have a very negative impact, even endanger people's lives. Our main objective of this systematic mapping study is to identify and analyze the published work up to October, 2021 in the field of software testing techniques for smart homes that use IoT. This goal has been achieved by summarizing the current articles and discussing the gaps and challenges in the field.

We begin this mapping study by describing the research method in Section 2.5.1, followed by the study classification scheme in Section 2.5.2. We answer the research question and analyze the results in Section 2.5.3. Section 2.5.4 discusses threats to validity. We discuss our findings in Section 2.5.5. Conclusive remarks are in Section 2.5.6.

2.5.1 Research Method

This systematic mapping study follows the guidelines in Petersen et al. (2015) [85] and Charters and Kitchenham (2007) [64]. Systematic mapping proceeds in four main steps (see Fig 2.1), similar to the work by Asadollah et al. (2017) [78]. The following subsection will explain each step in more detail.

Definition of research questions

The overall objective of this study is to investigate and analyze the existing publications (up to October 2021), in the field of software testing techniques for smart home systems that use the Internet of things (IoT). The research questions focus on categorizing the available solutions based on what testing techniques, tools and environments are used in testing of smart home systems that use the IoT. Where are these publications published and when. We also aim to discover research gaps to understand what areas of solutions have not been researched thoroughly.

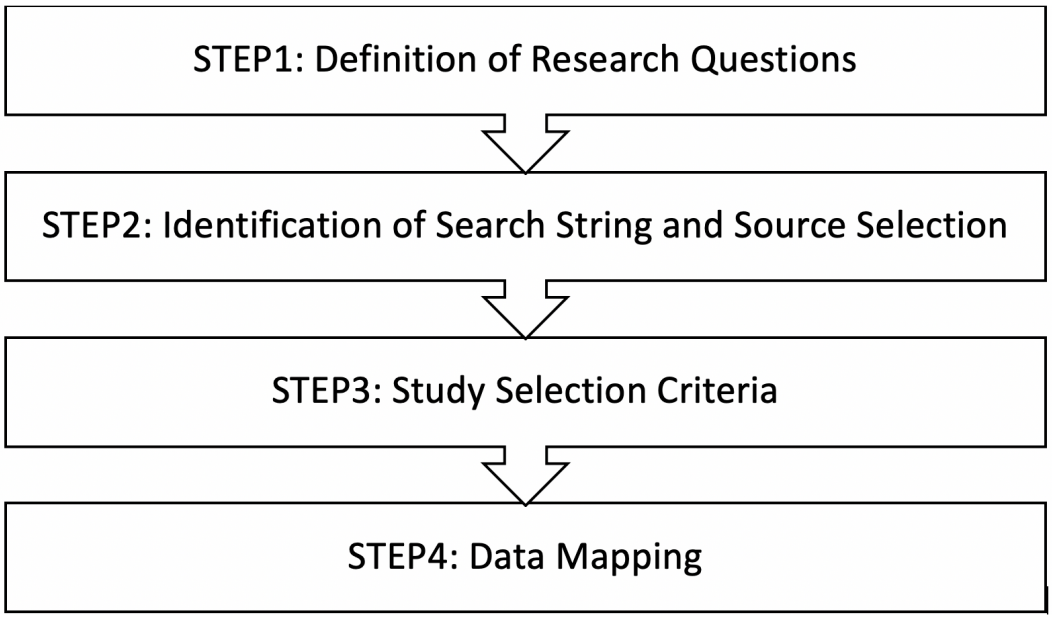

Figure (2.1) Research workflow

Based on that objective, this study attempts to answer the following research questions:

- RQ1: What are the publication trends for testing smart home systems?
  - RQ1.1: What is the annual number of publications in the field?
  - RQ1.2: Which publications are the main publishers of works on testing smart homes?
  - RQ1.3: Which articles are more influential in terms of the number of citations?
  - RQ1.4: Which research groups have published articles?
- RQ2: What are the existing research gaps related to testing smart home systems?

- RQ2.1: What testing approaches have been used?
- RQ2.2: What testing types have been used?
- RQ2.3: What testing levels are addressed in functional testing?
- RQ2.4: What testing environments have been used?

Identification of search string and source selection

To find reliable peer-reviewed papers, we searched the following electronic databases:

- ACM Digital Library,
- IEEE Xplore,
- ScienceDirect – Elsevier,
- SpringerLink,
- Wiley Inter Science Journal Finder.

We developed a search string based on the requirements of each database. Some databases have restrictions on the number of Boolean operators that can be used, while others do not. Table 2.1 shows the search strings used for each database, and this search string has been applied to all fields.

The number of search results per database can be found in Table 2.2. This study considered all search results for the first three quarters of 2021 and before because it was conducted in October 2021.

Study selection criteria

In this step, we selected the most relevant studies from the search results that were collected in the previous step. We excluded articles based on titles and ab-

Database	**Search String**
IEEE, SpringerLink, and Wiley	(("Internet of things") AND (cloud) AND (software testing OR software validation) AND ("black box" OR "model based testing" OR "system testing" OR "functional testing") AND ("smart home" OR "smart homes" OR "smart house" OR "smart houses" OR "smart building" OR "smart buildings" OR "home automation") AND (technique OR approach OR method OR tool OR framework))
ACM	("Internet of things") AND (cloud) AND (software testing OR software validation) AND ("smart home" OR "smart homes" OR "smart house" OR "smart houses" OR "smart building" OR "smart buildings" OR "home automation")
ScienceDirect - Elsevier	("Internet of things") AND (cloud) AND (software testing OR software validation) AND ("black box" OR "model based") AND ("smart home" OR "smart homes" OR "smart house" OR "smart houses" OR "smart building" OR "smart buildings" OR "home automation")

Table (2.1) Search strings in databases

Database	**Search Results**
IEEE	102
ScienceDirect - Elsevier	380
SpringerLink	69
Wiley	27
ACM	363

Table (2.2) Search results

stracts, then skimmed the paper, and if appropriate, read the full text. The following are the inclusion criteria that were applied to titles and abstracts:

- Include papers that are peer reviewed.
- Include papers that are written in English.
- Include papers that can be fully accessed.
- Include papers that have smart homes as an IoT application.
- Include papers that test or validate the smart home systems using IoT.

The following exclusion criteria were applied:

- Studies presenting reviews or surveys.
- Studies that are not peer reviewed.
- Studies that written in languages other than English.
- Books.
- Studies that are not fully accessible.
- Studies of hardware systems.
- Studies that focused on security or network testing techniques.

Fig 2.2 shows the process of applying the inclusion/exclusion criteria and the number of studies retained in each step. The explanation of the process is illustrated below:

1. We conducted the search using the selected database with the search strings (mentioned in Section 2.5.1). We did not bound our search by dates, but because the study was conducted in October 2021, we included all studies up until October 2021. The total search results were 941.

2. Because the search was conducted using the electronic databases IEEE Explore, Wiley, ScienceDirect – Elsevier, SpringerLink, and ACM Digital Library, we guaranteed that all references were peer-reviewed and in English and full access was available online. In this step, we applied our first filter by selecting conference or journal papers only (not books or magazine articles) that gave us a total of 438 studies.

3. We read the title and the abstract of each study to consider whether the paper is related to smart home systems using IoT, or the smart home as an IoT application and whether it presents a testing or evaluation technique. In this stage, we exclude all IoT papers with different applications than smart homes (e.g., smart city, agriculture, health, etc.). The total number of selected studies was 150.

4. As stated earlier, hardware systems were excluded at this stage. In addition, if the main focus of the paper was on security features or network and infrastructure, the paper was excluded. The total number of selected papers after this filter was applied was 125.

5. The remaining papers were skimmed. In this step, our concern was testing and validation and how it was done in the study. At the end of this step, we were left with 31 relevant references.

6. The forward and backward snowball approach was conducted to ensure that we did not miss any relevant studies, as was recommended by Jalali and Wohlin (2012)[61]. The forward snowball check assessed the references lists of each selected paper and applied the inclusion and exclusion criteria, whereas the backward snowball check assessed who cited all the selected papers and applied

the inclusion and exclusion criteria. We added five studies and we did a forward and backward snowball check on them as well. In the end, we had a total of 36 studies.

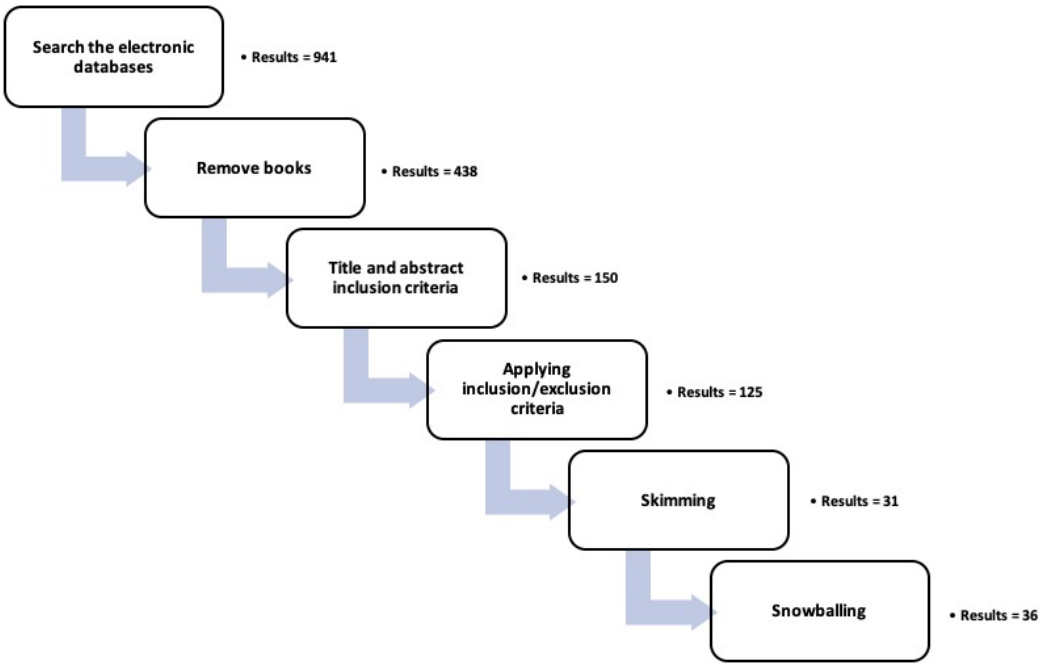

Figure (2.2) Number of included papers during the study selection criteria

Data mapping

At this point, we had collected 36 studies covering all published work up to and including October 2021. Using those papers, data mapping has been performed with the following categorizations: ad-hoc and systematic testing, functional and nonfunctional testing, method of evaluation, used test environment, test method, and test level for functional testing only. Each chosen paper was then read in detail and reviewed by two reviewers. Each of them categorized the paper. If they agreed on the assigned category for a paper, they assigned that category to the paper. If not, they reviewed the paper carefully. After this review, 7 papers were excluded because they were not relevant. By the end of this stage, we had 29 papers.

2.5.2 Study classification scheme

In this section, we classified the selected papers based on several factors. Fig 2.3 shows the first way to categorize the selected papers based on testing approaches, types, level, and methods. From the 29 selected research papers we found 22 papers did only ad-hoc testing and 7 papers did systematic testing. From the 22 papers slightly more than half of the papers did functional testing while the remaining did non functional testing. The functional testing papers 83% of the studies focus on system level testing and less than 20 % of the study did integration testing. 11 papers in both testing levels were black box testing and only one paper was white box. From the 7 systematic testing papers we found 4 papers did functional testing and 3 papers did non functional testing. 75% of the functional testing has been conducted as system level testing while 25% was integration level testing. All the systematic functional testing papers were black box testing.

- Testing approaches:
  - Ad-hoc: when the testing is done randomly and does not follow a method, or plan. Usually this type of testing does not have documentation or test design methodologies to construct test cases.
  - Systematic testing: when the testing has a fully detailed method, plan and documentation.
- Testing types:
  - Functional testing: when the functional requirements of parts of the system are tested.
  - Nonfunctional testing: when the nonfunctional requirements of parts of the system are tested.

- Testing levels:
  - Unit testing: when individual components of the system are tested separately.
  - Integration testing: when individual components of the system are combined and tested.
  - System testing: when a complete system is tested.
- Testing methods:
  - Black box: when the internal aspects of the item being tested are hidden from the tester.
  - White box: when the internal aspects of the item being tested are known to the tester and used in developing tests.

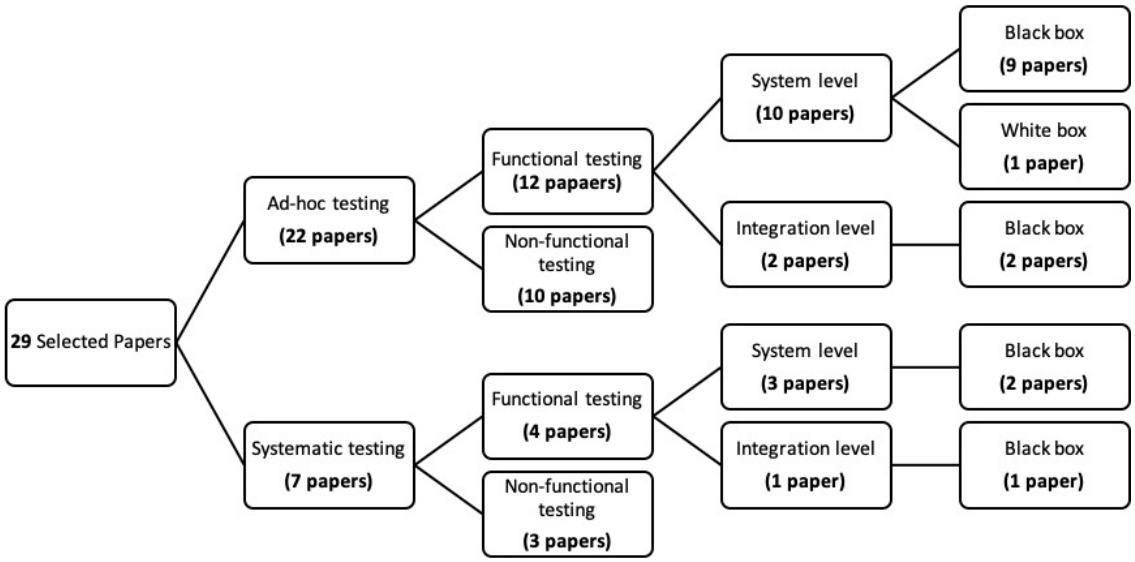

Figure (2.3) First classification method based on testing approaches, types, level, and methods

Another way to classify the selected papers was based on the research gaps and challenges. According to Gaikwad et al. (2015) [41] below are the most common challenges in smart homes that used IoT:

- Test Automation: How can researchers automate the testing process?
- Security: Testing security features (authentication, secure communication, privacy of data, etc.) is more critical for IoT systems because individuals do not want an intruder to control the devices in their home.
- User interface: Usability is a challenge for any software development, and it more complicated in IoT for smart homes because the data comes from different devices, and these data from multiple devices in a smart home must be visualized as one user-friendly visualization.
- Network availability: Connectivity is a major feature in IoT systems, and it needs to be available at any time and in any location. The smart home system's operation in an IoT context should be done in real time, therefore, network availability is another IoT challenge.
- Communication protocols: Choosing the proper connectivity protocol is another issue. 3G services are used for Internet communication. However, it is possible that there is a signal problem, in which case it will not connect every time.
- Data centers: As the number of IoT applications grows, so does the volume of data collected. The is a problem in storing the massive amounts of data collected from these applications. This issue can be solved with a large database. To extract meaningful data from it, artificial intelligence algorithms must be used.

Test environments are another way to categorize the selected papers. After building a system or proposing an approach or an architecture, researchers test their systems or techniques on different environments. We classified the articles based on test environments as follow:

- Real devices: The paper used the actual smart home for testing.
- Virtual devices: The paper used a virtual smart home for testing.
- Simulator: The paper used an environment model of the original system to study and analyze the behavior of the real system.
- Emulator: The paper replicated the original system (hardware and software), which means it can perform all the functionalities of the real system.
- Testbed: A real system (hardware, software, operating system, network configuration, etc.) was used, that can be used to execute the tests and obtain results.

Another classification was based on the following evaluation methods:

- Experiments: This quantitative research method is usually used to test and prove a hypothesis.
- Case study: Part of the system or a special case of it is studied in depth.
- Use cases: A case study can have multiple use cases; however, use cases can be a method to evaluate a system for different cases.
- Comparison: This is comparing state-of-the-art techniques to the proposed one or comparing the results of testing two different algorithms.

One more categorization was based on the contribution in the selected paper, which is what was proposed in each paper.

- New system: Some papers presented a new smart home system based on IoT, then they tested that system.
- Architecture: Some papers proposed a new architecture that supports IoT for a smart home, and others presented a current architecture and updated it to fit with IoT for smart home requirements and then tested that architecture. Under this category, we also included papers that presented a framework. Architecture is an abstract design concept for an application, and the framework is a pre-built architecture that is designed to build on or to extend.
- Approach: Some papers proposed an innovation approach to perform a specific activity in the IoT for a smart home.
- Component: These papers that added a new component (for example, a device) to an existing system, then evaluated the new system after adding that component.
- Tool: The papers that presented a testing tool and evaluated that tool.
- Testing technique: The papers that proposed a testing technique.

The last classification was regarding the bibliometrics of the selected papers by counting number of publications per year, per country, per publisher, and per number of citations.

2.5.3 Analysis of the results

The main objective of collecting the published studies in the previous steps is to answer the research questions. In the following sections, we will present, analyze, and discuss the results obtained from the selected papers and the possible answers to the research questions.

Publication trends up to October 2021

Fig 2.4 shows the count of articles per year, and from that figure, we can confirm that testing smart homes based on IoT is a recent subject. In fact, there are few papers before 2016 (Ciabattoni et al. 2014 [34]; Sobeih et al. 2015 [97]; Yuan 2015 [107]). After that we clearly notice the rapid growth in the number of publications.

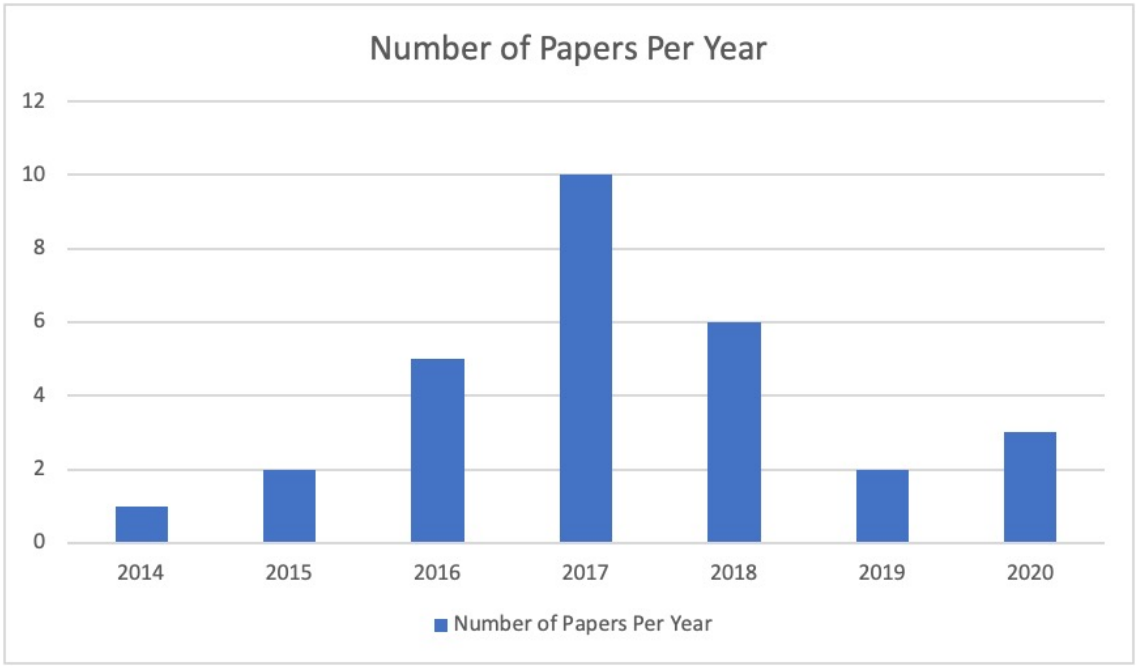

Figure (2.4) Distribution of the selected papers per year

Publication Venue	**Name of Publication**	**Number of Papers**
Journal	Procedia Computer Science	2
Journal	Future Generation Computer Science	2
Journal	Future Generation Computer Systems	2
Journal	The Journal of Systems and Software	1
Journal	Digital Investigation	1
Journal	Computer and Electrical Engineering	1
Journal	IEEE Internet of Things Journal	1
Journal	IEEE Access	1
Journal	ACM Transactions on Internet of Things	1
Total		12
Workshop	ACM International Workshop on Testing Embedded and Cyber-Physical Systems	1
Total		1
Conference	ISoLA: International Symposium on Leveraging Applications of Formal Methods	2
Conference	ICA3PP: International Conference on Algorithms and Architectures for Parallel Processing	1
Conference	International Conference on Internet of Things and Cloud Computing	1

Publication Venue	**Name of Publication**	**Number of Papers**
Conference	2nd IET International Conference on Technologies for Active and Assisted Living (TechAAL 2016)	1
Conference	2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)	1
Conference	22nd Mediterranean Conference on Control and Automation	1
Conference	2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)	1
Conference	2017 International Conference on Mathematics and Information Technology (ICMIT)	1
Conference	2015 International Conference on Computational Intelligence and Communication Networks (CICN)	1
Conference	2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)	1
Conference	2017 6th ICT International Student Project Conference (ICT-ISPC)	1

Publication Venue	**Name of Publication**	**Number of Papers**
Conference	2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing	1
Conference	2018 2nd International Conference on Data Science and Business Analytics (ICDSBA)	1
Conference	2017 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS)	1
Conference	Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016	1
Conference	The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops	1
Conference	2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT)	1

Publication Venue	**Name of Publication**	**Number of Papers**
Total		18

Table (2.3) List of publication venues

Table 2.3 lists all publication types (Journal, Workshop or Conference) and the venue. Two papers were published in each of the following publisher venues: journals (Procedia Computer Science, Future Generation Computer Science and Future Generation Computer Systems) and a conference (ISoLA: International Symposium on Leveraging Applications of Formal Methods). All other journals, conferences, and workshops published only one article. In total, the selected papers were published in 27 venues including journals, conferences, and workshops. Of the publications, 12 were journal papers, 18 were conference papers, and one was a workshop paper. However, two studies were published in both a journal and a conference. Fig 2.5 presents the publishers of these publication venues. It shows that slightly less than half of the publications were published in IEEE venues, while about third of the publications were published in Elsevier and about 10% of the publications were published in each Springer and ACM. Finally, Wiley had no percentage at all.

To address the most influential articles, we examined the number of times the papers had been cited. We used Google Scholar to find the number of citations for each selected article. Fig2.6 shows the number of citations for the articles that have been cited 10 times and more. It is interesting that most of the highly cited papers were published with Elsevier and between 2016 and 2017. The most cited work was by Chung et al. (2017) [33], published in the Digital Investigation journal (Elsevier), having 93 citations up to October 2021. The ecosystem of the Amazon virtual as-

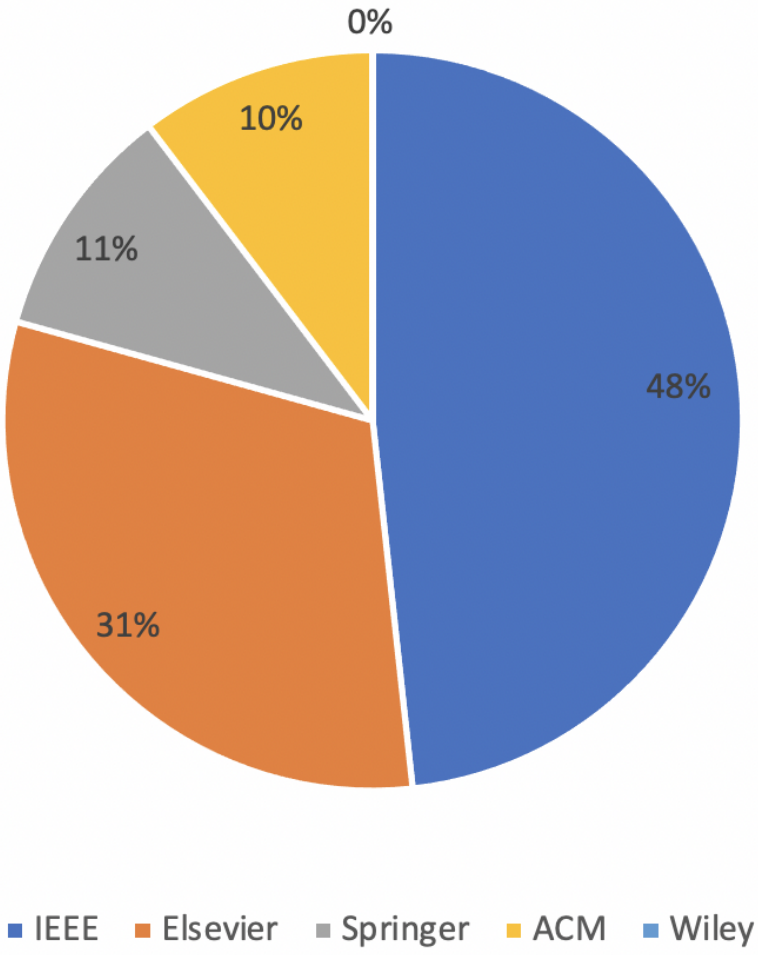

Figure (2.5) Distribution of publications in different publishers

sistance (Alexa) is covered in this paper's discussion of digital forensics techniques. This paper's main contribution is a new, effective method for supporting real-world digital investigations by fusing client-side (forensics for companion devices) and cloud-native (forensics for primary devices) forensics. They use proof-of-concept tool, Cloud-based IoT Forensic Toolkit (CIFT),which is based on a thorough understanding of the targeted ecosystem and facilitates the identification, acquisition, and analysis of both native artifacts from the cloud and client-centric artifacts from local devices (mobile applications and web browsers). Tao et al. (2017) [99] was

cited 53 times. This paper proposed a plan to manage and use of ontology-based data semantics to reduce the impact of continuously rising amounts of heterogeneous data in the smart home data arena. A general domain ontology model is created by defining the relevant concepts based on a smart home system model abstracted from the viewpoint of performing users' household operations, and a logical data semantic fusion model is created in accordance. A relational-database-based ontology data decomposition storage method is then developed by thoroughly examining existing storage modes, and the performance is demonstrated using a group of elaborated ontology data query and update operations. This is done in order to achieve high-efficiency ontology data query and update in the implementation of the data semantic fusion model. Next, Khan et al. [63] (2016) was cited 52 times. Building Smart Home Systems is a big challenge, due to the high level of use of technologies and tools within the smart homes. Consequently, the concept of a context-aware, low-power, intelligent SmartHome (CLPiSmartHome) is presented in this study. We suggest a communication paradigm for CLPiSmartHome that offers a standard means of communication, such as a common language. Additionally, a suggested architecture enables all electronic devices to communicate with each other through a single platform service. Moreover, the viability and effectiveness of the suggested solution are tested using a Hadoop single-node setup on a 3.2 GHz coreTMi5 machine running Ubuntu 14.04 LTS. Finally, Meana-Llorian et al. (2017) [76] was sited 45 times. In order to save energy and provide a more comfortable environment for their users, this paper suggest a novel method of temperature regulation using the Internet of Things along with its platforms and fuzzy logic addressing not only the inside temperature but also the external temperature and humidity. Finally, they drew the conclusion that the fuzzy technique enables us to save money and energy by about 40%. Since the most cited papers did not have something in common, we

believe that they are mostly cited due to popularity of Elsevier and it is the second most publisher in term of number of publication in the area.

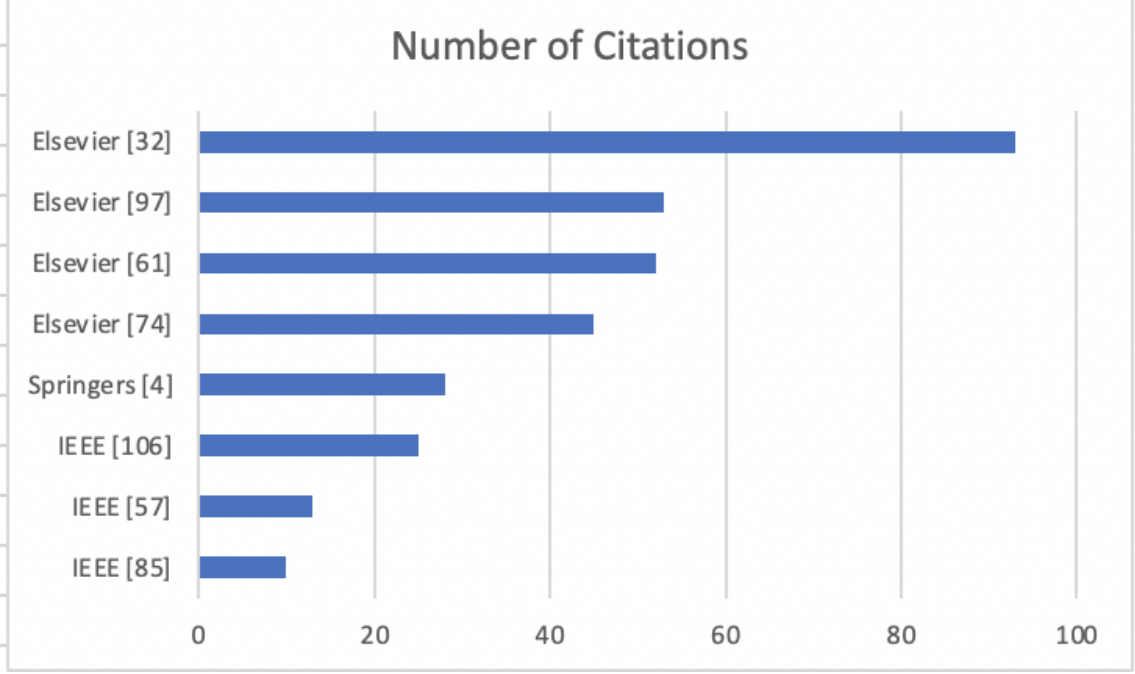

Figure (2.6) The most cited papers

The selected papers have been written by authors from around the world. This shows how this subject is of interest worldwide. We found contributions from all continents. The highest contribution were from China (5), France (3), Spain (4), Algeria (2), the UK (3), and the USA (2), while there was one from each of the following countries: Australia, Belgium, Brazil, Croatia, Czech, Ghana, Italy, Japan, Jordan, Korea, Malaysia, the Netherlands, New Zealand, Pakistan, Singapore, Slovakia, South Korea, Ukraine, and Vietnam. We also noticed that only seven articles have been written by authors from two or three different countries.

Focus and potential gaps

Ad-hoc testing is software testing performed without a specific method, plan or documentation. The majority of the selected papers were validated and tested with-

out a clear plan or documentation that led us to classify about 75 % of the papers as having an ad-hoc testing approach. Only two papers (Yusri et al. 2017 [108]; Sowah et al. 2018 [98]) presented a table of test cases and their results. We considered them ad-hoc testing because there were insufficient details on what testing method was used. Only seven papers used a systematic approach to test smart homes that used IoT. Ahmad et al. (2016) [5] presented a model-based testing approach to test services provided by IoT platforms. Martinez et al. (2018) [75] used a systematic testing approach to present the results of automated testing of IoT using the TESTAR tool. Asensio et al. (2019) [26] validated the interoperability between the real installation and the emulation of a smart home by checking the connection between both the virtual and physical devices. Tulenkov et al. (2019) [101] investigated the functional features of IoT services, as well as the criteria for their evaluation and selection. A practical example of IoT services and cloud platforms usage for Smart House and IoT was illustrated. Gesvindr et al. (2020) [48] evaluated an architecture design of Prototype as a Service cloud application using generated prototypes generated by the PaaSArch Cloud Prototyper tool. PaaSArch is a tool supporting PaaS (Platform as a Service) application architects. The PaasArch tool aids software architects in assessing the quality of PaaS cloud applications by using created prototypes. The program allows for completely functional cloud apps to be automatically built, cloud-deployed, and benchmarked [47]. The whole approach was demonstrated and evaluated on a case study of a smart home as an IoT application. Luo et al. (2020) [72] proposed Gait Recognition as a Service (GRaaS) model and used a case study to evaluate this model in detail. GRaaS is an embodiment of the traditional Sensing as a Service (SaaS) model. It is designed to recognize human gait in smart spaces. They propose a RFID-based gait recognition service following the GRaaS model. Rocha Filho et al. (2020) [90] proposed an intelligent decision system

based on fog computing for an efficient way to control home appliances. The term "fog computing" refers to a decentralized computing architecture where computer resources are located between the data source and the cloud or other data center [88]. By proposing ImPeRIum a smart strategy to managing home surroundings. ImPeRIum monitors and acts on the residential environment through the intercommunication of smart devices, taking into account the profile of residents. ImPeRIum can detect and respond to unexpected events in order to retain the environment in a predetermined state. An detailed analysis of many scenarios revealed ImPeRIum's practicality and efficiency in smart environments with limited resources. The proposed solution is evaluated both in simulated and real environments and the results of these evaluations were provided. The outcomes of this research were: A high success rate with minimal decision-making latency, efficient information transfer with minimal infrastructure overhead, and processing robustness with minimal energy usage.

Out of 29 papers, 16 papers used functional testing, where the functional requirements of the system have been tested (Ahmad et al. 2016 [5]; Alipour 2017 [11]; Asensio et al. 2017 [25]; Asensio et al. (2019) [26]; Balikhina et al. 2017 [27]; Chung et al. 2017 [33]; Ciabattoni et al. 2014 [34]; Guebli et al. 2016 [51]; Huang et al. 2018 [59]; Martinez et al. 2018 [75]; Meana-Llorian et al. 2017 [76]; Sobeih et al. 2015 [97]; Tulenkov et al. 2019 [101]; Yusri et al. 2017 [108]; Sowah et al. 2018 [98]; Zouai et al. 2017 [111]). From the remaining papers, 13 covered non-functional testing: different non-functional requirements were tested. Tao et al. (2018) [100] tested performance by measuring the response time for two different scenarios, whereas Yuan (2015) [107] validated performance by measuring the processing time and by comparing an ordinary smart home system and a cloud-computing smart home. De Buyser et al. (2016) [36] evaluated the feasibility and

the performance via calculating the time range for each series of instructions given to the testers. Skocir et al. (2016) [96] evaluated the performance of two proposed algorithms by testing them to check whether they efficiently detect the activities in the smart home. Khan et al. (2016) [63] tested efficiency by analyzing given datasets to check throughput and processing time, while accuracy was checked via a specific algorithm. Hu et al. (2018) [57], tested performance by comparing cost savings in the two proposed systems against six other traditional systems. Tao et al. (2017) [99] tested accuracy and performance via user behavior reasoning. Nguyen et al. (2018) [79] evaluated the abilities of gathering and monitoring data from multiple IoT sources and modeling the collected date in a central format. Rocher et al. (2017) [91], tested quality and performance of the proposed framework to estimate the gap between the observed and estimated behaviors of smart systems. Pitt et al. (2017) [87] tested the power consumption in a smart house. Gesvindr et al. (2020) [48], tested performance by measuring throughput, scalability and latency of the generated prototypes, then evaluating the impact of the designed architecture of a PaaS cloud application. Rocha Filho et al. (2020) [90] evaluated the performance in terms of processing time of applications. This time considers the time spent between sending the data and the response of the actuation. Luo et al. (2020) [72] tested the performance by comparing attention-based Long Short-term Memory (At-LSTM) model and traditional classification algorithms such as Sparse Representation Classification (SRC). An RFID-based gait recognition service were designed for user identification in smart spaces utilizing commercial off-the-shelf RFID sensors, following the GRaaS (Gate Recognation as a Service) architecture. To accomplish accurate recognition at the edge layer, an attention-based LSTM model is utilized for training and classification of the input data. The service is registered with the service provider layer, which gives clients with a cloud platform via which they can

access the server from any location. Authentication in smart spaces, for example, can then be built in the application layer. The results were compared to recognition accuracy of the gait recognition by using different classification algorithm Sparse Representation Classification (SRC). As one of the state-of-the-art face recognition algorithm [95]. Table 2.4 shows the 29 references that have been selected in the study and classifies them based on the testing type.

Testing Types	**Selected Papers**
Functional Testing	[5], [11], [25], [26], [27], [33], [34], [51], [59], [75], [76], [97], [98], [101], [108], [111]
Nonfunctional Testing	[36], [48], [57], [63], [72], [79], [87], [90], [91], [96], [99], [100], [107]

Table (2.4) List of retrieved papers grouped by testing type

Most of the functional testing papers (13 out of 16) considered system-level testing. Only three papers presented integration testing approaches. The work by Ciabattoni et al. (2014) [34], Asensio et al. (2017) [25] and Asensio et al. (2019) [26] focused on testing the interaction between components. Since most of the papers used an ad-hoc testing approach, as we mentioned in the answer for RQ 2.1, some papers mentioned that unit testing and integration testing had been done, but we could not consider that classification due to a lack of detail.

In most of the papers (17 out of 29), testing used real devices, while two of the papers only tested virtual devices. Four executed tests on simulators. Some papers used more than one test environment (e.g., Hu et al. (2018) [57] started the experiment with a simulation and conducted a case study using a real device). Moreover, three papers (Alipour 2017 [11]; Ciabattoni et al. 2014 [34]; Nguyen et al. 2018 [79]) used a combination of simulators and emulators in their tests. While Asensio et al. (2019) [26] used virtual and physical devices and tested the integration

between the real installation and the emulation. Finally, Rocha Filho et al. (2020) [90] tested some features in a real environment and others by simulation. Table 2.5 classifies the 29 papers by their testing environment.

	Single	**Multiple**
Real devices	[5], [27], [33], [36], [59], [75], [87], [91], [96], [97], [108], [98], [99], [100], [48], [101], [72]	(real and virtual) [90], [57]
Virtual devices	[25], [63]	
Simulator	[51], [76], [107], [111]	(simulator and emulator) [11], [34],[79]

Table (2.5) List of retrieved papers classified by testing environment

2.5.4 Threats to Validity

In our systematic mapping study, we followed a systematic method to find, analyze and classify the papers published in our field of interest. However, our study has some threats to validity. One major threat is due to the inclusion and exclusion criteria that were explained in Section 2.5.1. We might miss some important papers that are written in languages other than English, or not available online, or not in one of the selected libraries. In addition, we excluded papers related to IoT but not to smart homes. This might lead to exclude some testing techniques that could be proposed to test smart homes. The search string and the keywords were developed systematically. However, we might have missed some keywords or synonyms that could have resulted in additional relevant papers. Finally, some papers might be excluded at early stages without fully skimming due to a misleading or unclear title or abstract. Making decisions using multiple reviewers tried to mitigate this.

2.5.5 Discussion

Based on our analysis, we discovered that most of the papers did not propose a systematic testing technique, and the majority proposed ad-hoc testing only. Those papers need to include more details in how they tested their systems, which techniques are used, which tools, as well as more documentation on the test cases and results. Only one paper (Ahmad et al. 2016 [5]) proposed a Model-Based Testing as a Service technique for IoT platforms. Two papers presented two different testing tools. Martinez et al. 2018 [75] applied automated testing and presented the result using the TESTAR tool, and Alipour (2017) [11] proposed an event-based fault-injection tool. This is a new topic, as the concept of IoT was suggested first in 1999 [94]. Publications on Smart Homes using IoT started in 2014 according to our finding. We expect more research in this field to be conducted in the future since this area is new and the research it recent. However, our examination shows that cost is one of the challenges facing testing smart homes that use IoT with devices. A lack of simulation environments for smart homes might be another issue. That could be one reason that few papers were published in this area.

Another finding is that the bibliometrics show that many countries and authors are interested in this area of research. This topic is of interest to both engineers and computer scientists. That makes the findings split between the two areas. Because this is a new topic in computing we found much more work published in conferences than journals. It appears that researchers in the area of smart homes were more interested in building these systems, rather than presenting systematic testing techniques for them, hence the prevalence of ad-hoc testing. While testing is expensive and this new field is growing rapidly, testing is very important and necessary. Therefore, researchers need to spend more effort to guarantee that sensors

in the smart home function well, and the data collected from them are handled well, that device interactions work properly and the systems as a whole is also systematically tested.

2.5.6 Conclusion

This systematic mapping study was done to provide an overview of current research on software testing techniques for smart homes that use IoT. We analyzed and summarized 29 articles published up to October 2021. The analysis showed that there are very few systematic testing techniques. Only one study proposed a model-based testing technique to test IoT platforms for smart homes. Another paper was presenting the test automation results using the TESTAR tool in IoT. On the other hand, the majority of the papers were proposing a new smart home technology and they did not document the testing process in detail. In this study, eight research questions have been answered. The main results are:

1. This is a new topic, no paper was published before 2014, and 2017 shows the largest number of publications.

2. Most of the articles were published in conference proceedings (about 60%) and a lot fewer in journals (about40%).

3. Publications were published by 27 different publishers.

4. Authors from 25 different countries have been working on these articles.

5. One paper proposed a model-based testing as a service technique to test IoT platforms [5].

6. Two papers presented testing tools, one used the TESTAR tool [75], the other paper used an event-based fault-injection tool with Selenium [11].

7. Three papers proposed various new components to be added to the smart home, including Google glasses [97], smart grid [87], and headset [36].

8. Most of the selected papers in this study (28 out of 29) used black-box testing.

While there is quite an amount of published work on this field, most of the testing was ad-hoc and the amount of information of how testing was done is sparse. Considering the high demand of systematic testing this is a wide open area for fruitful research.

Chapter 3

Reusable Test-Ready Models of Smart Home Systems

3.1 Problem Statement

In this Chapter we propose reusable test-ready models of smart home systems (SHS) using Extended Finite State Machines (EFSMs) [6, 24, 105] to model device components (Sensor, Controller and Actuator), Communicating Extended Finite State Machines (EFSMs) [68] to model single devices in the SHS and the interaction between the devices. We adopted Al Haddad's [10] FSMApp approach to model and test the mobile application that controls the SHS. We noticed the difficulty to use model-based testing in testing SHS due to the heterogeneity and dynamic nature of devices. Therefore, we propose a set of reusable test-ready models which can be used as-is or with minor modifications (See Chapter 5). This Chapter addresses the following research question:

- Is it possible to develop a reusable test-ready models of a SHS that can be used to test SHS at the device, device interaction and system level?

This Chapter is organized as follows: The used terminologies throughout this proposal is presented in Section 3.2. Section 3.3 will give an outline of the proposed approach. Section 3.4.1 provides the guidelines to model device components and devices. Modeling the Mobile App is explained in Section 3.4.2. In Section 3.4.3 we will show how to model interaction between devices and between devices and the Mobile App.

3.2 Terminology

We used the following terminology:

SHS: Smart Home System.

IoT: Internet of Things.

Device: Any physical device within the SHS.

Device Component: Sensor, Controller, or Actuator.

Sensor: A device sensor is a part which detects or measures a physical property and records, indicates, or otherwise responds to it.

Controller: A device controller is a system that handles the incoming and outgoing signals of the sensor.

Actuator: A device actuator is the part that causes a device to operate based on the controller decision.

System Controller: Mobile Application/s.

3.3 Proposed Approach

Our goal is to develop a turn-key black-box testing approach for Smart Homes. Fig 3.1 shows the process of Phase 1. The entire approach proceeds in three phases: Phase 1 builds a set of reusable test-ready models, Phase 2 generates tests from these models, and Phase 3 show how these models are easily modifiable for new device types. Phase 1 is discussed in this Chapter.

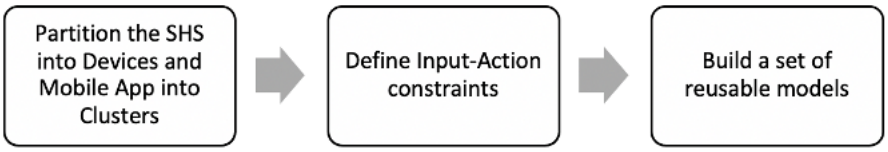

Figure (3.1) Phase 1: Build Test-Ready Models for Devices and System Controller

3.4 Reusable Test-Ready Models

We provide three types of reusable models:

- For the System Controller we use Al Haddad's FSMApp [10] approach.
- Device Components (Sensors, Controllers, Actuators) are modelled as EFSMs.
- Device itself and Device Interactions (with each other and with the System Controller) use CEFSM models.

Communication channels between models are added to the EFSM to exchange messages and actions/commands, as illustrated in Fig 3.2.

Reusable Test-Ready Models = $\{\text{FSM}_{App}, \text{CEFSM}_{Dev}\}$

$\text{CEFSM}_{Dev} = \{\text{EFSM}_{Sensor}, \text{EFSM}_{Controller}, \text{EFSM}_{Actuator}\}$

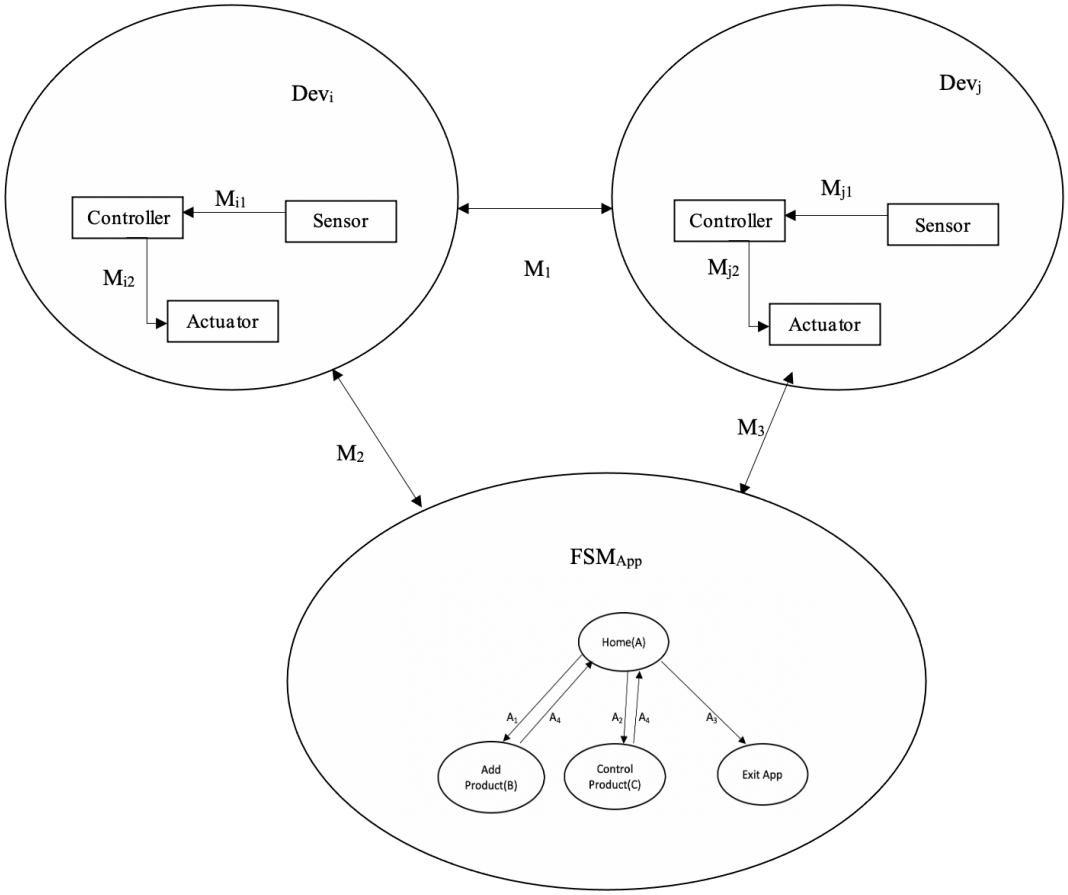

Figure (3.2) Overview of Reusable Test-Ready Models

3.4.1 Model Device Component and Devices:

To model the device components and devices in the SHS, we used the following approach:

- Partition the SHS into devices. These are physical devices like a security camera.
- Define device component and Input-Action constraints for each component. Components are sensors, actuators, and controllers.
- Build CEFSMs for devices as a multi-level hierarchy including an Aggregate FSM (AFSM).

Partition the SHS into Devices

To build a CEFSM$_{Dev}$, we need to divide the SHS into devices. In this step we construct a behavioral model for each device component (Sensor, Controller and Actuator) and a behavioral model for the entire device. These models are built based on the system requirements and specifications. To build these models we used EFSM to model device components individually. CEFSM is then used to model the entire device. The input for this step is system requirements and specifications. The output are EFSM models for device components and CEFSM models for single devices.

Define Input-Action constraints for each Device.

Input for devices is different than inputs for Mobile apps or Web applications. To illustrate the possible inputs for devices we look at the top-ten smart devices on the market in 2022. According to Tom's Guide [66], Table 3.1 shows the top ten smart devices and lists all possible user inputs (Uinput) and environment inputs (Einput) for each device. A device in the SHS can have different type of inputs: user inputs when the user of the system interacts with the device directly; for example, Amazon Echo Dot accepts voice commands directly from the user. Another type of input is an environment input where the environment interacts with the device; for instance, Amazon Echo Dot detects environmental sounds such if it hears broken glass, fire alarms, or other suspicious sounds the device will act based on these sounds and send an alert to the SHS owner. Finally, all devices get commands from the Mobile application that controls the SHS. If there are no user or environmental inputs, commands come only from the mobile app. Table 3.1 shows the user and environment inputs for the top ten smart devices. Table 3.2 shows devices input

constraints R means required and O means optional. The order of input can be either sequential or any order.

Device Name	**Uinput**	**Einput**
Amazon Echo Dot	Voice commands (VC) Click buttons (CB) (+/-button) (mic on/off button)	Environment sounds: (ES) broken glass, or fire alarms
Nest Cam (battery)	None	Motion (M) Sound (S)
Ecobee Smart Thermostat	Voice commands (VC) Touch screen (TS) Temperature Threshold (TThreshold) Threshold Level (ThresholdLevel)	Temperature (Temp) Occupied (Occupied)
Philips Hue White A19 Starter Kit	Time Threshold (TiThreshold)	Time (Time)
August Wi-Fi Smart Lock	Open/Close the door (Open) Lock/Unlock (Lock) Time Threshold (TiThreshold)	Time (Time) DoorSense (DS)
Nest Doorbell	Click button (Dbutton)	Video recording (VR)

Device Name	**Uinput**	**Einput**
		Motion (M) Sound (S)
Wemo WiFi Smart Plug	Voice commands (VC) Click button (Pbutton)	Time (Time)
Nest Protect Smoke detector	None	Smoke and Carbon monoxide sensor (CO) Time (Time) Temperature (Temp), humidity (H) and occupied (Occupied)
Samsung SmartThings Home hub	None	None
Chamberlain MyQ garage opener	None	None

Table (3.1) Top Ten Smart Devices

Input Choice	**Order**
Required (R) Optional (O)	Sequence (S) Any (A)

Table (3.2) Devices Input Constraints

To illustrate what components look like we analyze common components of the top ten devices and what CEFSM$_{Dev}$ input constraints would look like:

1. **Voice command (VC)** indicates what action will occur when the user speaks. The action of the voice command is talk. The input constraint is R().

2. **Click Button (CB)** indicates what action will occur when the user click it. The action of the button is a click. The input constraint is R().

3. **Touch screen (TS)** indicates what action will occur when the user touches the device's screen. The action of the touch screen is a touch. The input constraint is R().

4. **Close/open door (Open)** indicates what action will occur when the user opens or closes the door. The action of the close/open door is a door closed or opened. The input constraint is R().

5. **Lock/Unlock (Lock)** indicates what action will occur when the user lock or unlock the smart lock. The action of the lock/unlock door is a door lock or unlock. The input constraint is R().

6. **Environmental sounds (ES)** allow the device to indicate indicates suspicious sounds. The action of the environmental sound is listen to a fire alarm, a broken glass or any suspicious sound accrue. When these sounds is heard an alert will be sent to the user. The input constraint is R().

7. **Motion (M)** indicates what action will occur when motion is detected. The action of the motion is a person walk, a pet walk or any movement within defined area. When a motion is detected within environment device should act based on the that. For example, lights turn on when a motion is detected. The input constraint is R().

8. **Sound (S)** indicates what action will occur when a sound is heard. The action of the sound is a person talk, a baby cries or any sound accrue. When sound is heard within environment device should act based on the sound. For example, camera start recording when a sound is detected. The input constraint is R().

9. **Time (Time)** indicates what action will occur when a predefined time is reached. The action of the time is a specific time reached. Once a predefined time is reached a device should act. For example, light turn on automatically at specific time. The input constraint is R().

10. **Temperature (Temp)** indicates what action will occur when a predefined temp is reached. The action of the temperature is a specific temp reached. Once a predefined temp is reached a device should act. For example, thermostat temp is increased or decreased automatically when a specific temp is reached. The input constraint is R().

11. **Occupied (Occupied)** indicates what action will occur when a room is occupied. The action of the occupied to check if there is people in a room. Once there is people in a room a device should act. For example, thermostat temperature is decreased automatically when a room is occupied. The input constraint is R().

12. **DoorSense (DS)** indicates what action will occur when someone open or close the door. The action of the DoorSense is detection of someone open or close the door. When DoorSense detect that someone open or close the door it should notify the smart lock with the door status. The input constraint is R().

13. **Smoke and Carbon Monoxide Sensor (CO)** indicates what action will occur when Carbon Monoxide is detected. The action of the Carbon Monoxide is detection of Carbon Monoxide. When Carbon Monoxide is detected an alert will be sent to the user. The input constraint is R().

14. **Humidity (H)** indicates what action will occur when a predefined humidity level is detected. The action of the humidity sensor is detection of humidity. When humidity predefined level is detected the device should act. The input constraint is R().

15. **Final Decision (FD)** indicates what action will occur when a final decision is reached. The action of the final decision is based on the decision. When final decision is reached the device should act based on that decision. The input constraint is R().

16. **Increase Temperature (IncreaseT)** indicates what action will occur when increase temperature is received. The action of increase temperature is increase the temperature of the thermostat. When increase temperature is received the device should rise the temperature of the thermostat. The input constraint is R().

17. **Decrease Temperature (DecreaseT)** indicates what action will occur when decrease temperature is received. The action of decrease temperature is decrease the temperature of the thermostat. When decrease temperature is received the device should lower the temperature of the thermostat. The input constraint is R().

18. **Increase Volume (IncreaseV)** indicates what action will occur when increase volume is received. The action of increase volume is increase the vol-

ume of the device. When increase volume is received the device should rise the volume of the speaker. The input constraint is R().

19. **Decrease Volume (DecreaseV)** indicates what action will occur when decrease volume is received. The action of decrease volume is decrease the volume of the device. When decrease volume is received the device should lower the volume of the speaker. The input constraint is R().

20. **Mute Microphone (Mute)** indicates what action will occur when mute microphone is received. The action of mute microphone is mute the microphone of the device. When mute microphone is received the device should mute the microphone. The input constraint is R().

21. **Unmute Microphone (Unmute)** indicates what action will occur when unmute microphone is received. The action of unmute microphone is unmute the microphone of the device. When unmute microphone is received the device should unmute the microphone. The input constraint is R().

22. **Display Live Video (Video)** indicates what action will occur when display video is received. The action of video is start live video streaming. When display video is received the device should view live video. The input constraint is R().

23. **Idle (Idle)** indicates what action will occur when idle is received. The action of idle is stop live video streaming. When idle is received the device should stop live video. The input constraint is R().

24. **Lock Door(Lock)** indicates what action will occur when lock is received. The action of lock is lock the door. When lock is received the smart lock should lock the door. The input constraint is R().

25. **Unlock Door(Unlock)** indicates what action will occur when unlock is received. The action of unlock is unlock the door. When unlock microphone is received the smart lock should unlock the door. The input constraint is R().

The transitions are between the nodes (the states of the system). We will annotate the transitions between the nodes with input condition, user, and environmental inputs to indicate what inputs, messages and actions lead to a change in state.

Device Models

Any device consists of three components (Sensor, Controller and Actuator). Fig 3.3 shows the analog sensor's behavioral model where S_1 is the initial state where the sensor is idle (SIdle) and S_2 is the final state where sensor is ready to collect data (SActive). Transitions on the edge will be different based on the type of the sensor.

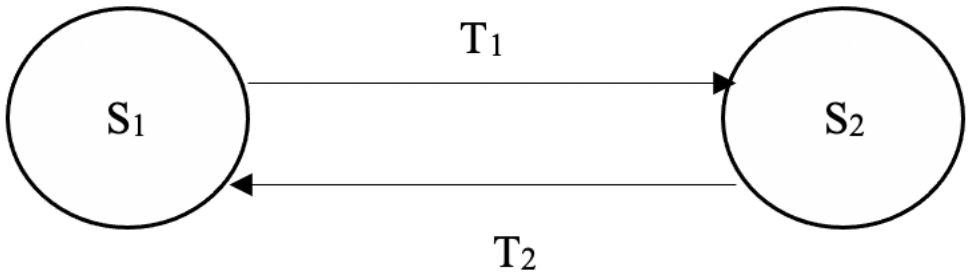

Figure (3.3) Analog Sensor Behavioral Model

A controller is modeled in Fig 3.4 using EFSM where S_5 is the initial state, showing that the controller is ready (CReady). S_9 is the final state, when data is transmitted to the actuator the controller becomes idle (CIdle). S_6 is the check state

(Check), in this state the controller compares the received data from the sensor with the predefined threshold in the device. Based on the result of the comparison, the controller can transit to S_7 or S_8. For example, in the smart thermostat S_6 will check if the received temperature from the sensor is less than or greater than a predefined temperature. If the temperature is less than temperature threshold then state S_7 (increase temperature) is reached. Otherwise, S_8 (decrease temperature) is reached. The corresponding states are specified in Table 3.3 the first column shows the state id, the second column shows the state name, and the last column describe the state. S_1, S_2 represents the state in the analog sensor, S_3, S_4 represents the state in the binary sensor. S_1 will be SIdle and S_2 will be SActive, while an binary sensor will have S_3 as off and S_4 as on. S_5, S_6 and S_9 in the controller are similar for all type of devices. S_7 and S_8 in the controller change based on the device type. For example, if we have a light device the states will be lights should go on, and lights should go off. On the other hand, in a thermostat device, the states will be temperature should increase or decrease based on the check process in S_4 (Check). Finally, the actuators in all devices have the same states S_{10} (Device off), S_{11} (Device on).

An actuator is modeled in Fig 3.5 using EFSM where S_{10} is the initial state where the device is off and S_{11} is the final state where the device is on.

Fig 3.6 shows the device behavioral model using CEFSM. During the integration of the three EFSM models, messages from sensor to controller and controller to actuator are added. Instances of interactions and examples of their behavioral messages are illustrated in Table 3.4. Message Channel M_{11} passes m_{11} the sensed data (SData) from the sensor to the actuator. On the other hand, M_{12} sends m_{12} the decision taken by the controller (FD) to the actuator.

We model Ecobee Smart Thermostat, Amazon Echo Dot, Nest Cam (battery), August Wi-Fi Smart Lock and Nest Protect Smoke detector devices in the following

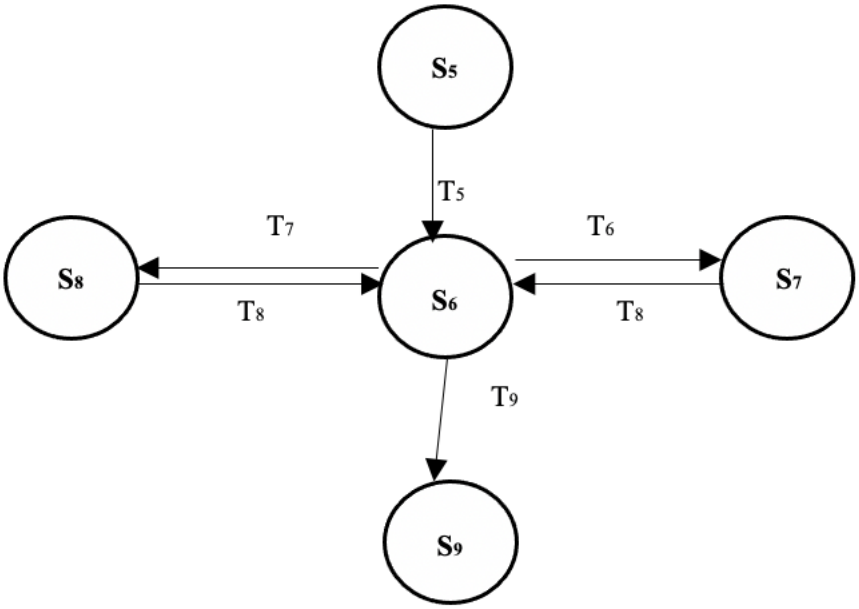

Figure (3.4) Controller Behavioral Model

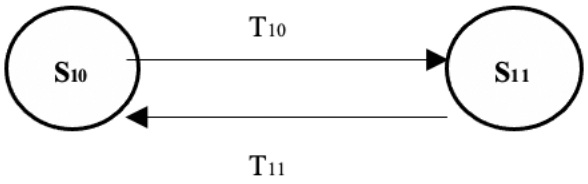

Figure (3.5) Actuator Behavioral Model

subsections, to show how to use the generic device model to build customized models. Note that these are simply the reusable models for Sensors, Actuators, Controllers and devices (Fig **??** - 3.6) what is device specific are the transitions.

Model Ecobee Smart Thermostat

Tom's Guide [66] nominated the Ecobee Smart Thermostat as one of the ten best smart devices in 2022. As any other smart thermostat the temperature will be adjusted according to the sensed temperature (Temp) and occupied (Occupied) in the

State	State Name	State
S_1	SIdle	Analog Sensor is idle
S_2	SActive	Analog Sensor is active
S_3	CReady	Controller is ready
S_4	Check	Check and compare
S_5	On	Device should go on
S_6	Off	Device should go off
S_7	CIdel	Controller idle
S_8	Off	Device off
S_9	On	Device on

Table (3.3) Device Components States

ID	Between	Message Example
M_{11}	Sensor - Controller	Send m_{11} "Sensed Data (SData)"
M_{12}	Controller - Actuator	Send m_{12} "Final Decision (FD)"

Table (3.4) Device's CEFSM Message Example

room. Fig 3.7 shows the smart thermostat device CEFSM model. The different inputs and possible values are listed in Table 3.5. The Ecobee Smart Thermostat takes voice commands (VC), touch screen (TS) and temperature threshold (TThreshold) as user inputs. It takes two environmental inputs (temperature (Temp) and occupied (Occupied)).

Uinput	Input Domain	Einput	Input Domain
VC	(Increase, Decrease)	Temp	[>68, <77]°F
TS	(↻, ↺)	Occupied	(True, False)
TThreshold	[>60, <80]°F [1]		
ThresholdLevel	(Minimum, Basic, Balanced, Super, Maximum) [2]		

Table (3.5) Smart Thermostat Input Values

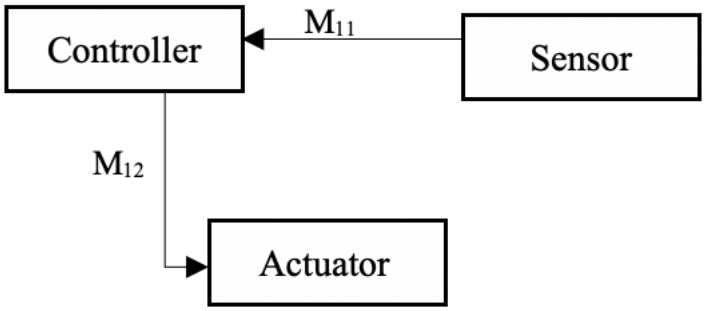

Figure (3.6) Device Behavioral Model

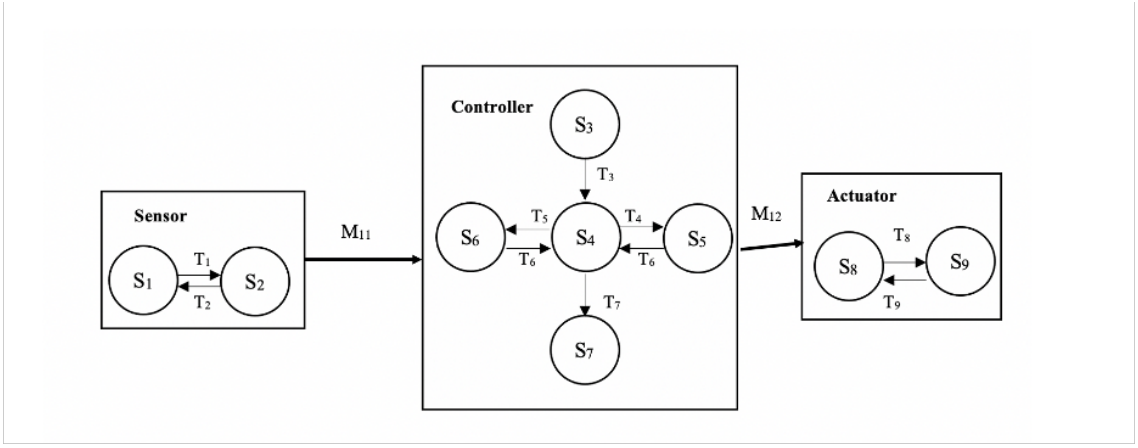

Figure (3.7) Smart Thermostat CEFSM Model

Table 3.6 lists the CEFSM transitions definitions for the Smart Thermostat. A unique id for each transition is given in column 1 and the input action constraints are shown in column 2. Table 3.6 presents 9 transitions (T_1 - T_9) and two message channels (M_{11}, M_{12}) for the Smart Thermostat. The Ecobee Smart Thermostat consists of two analog sensors: Temperature Sensor and Occupied Sensor, a controller and an actuator. Both sensors are working as they are sensing the temperature and

[1]The manufacture of this device did not specified the possible range of temperature, so we consider the average range is anywhere from 68 to 76 degrees Fahrenheit. We extend this range to consider the wormiest at 80 and the coolest at 60.

[2]Minimum (0.5F) - Basic (1F) - Balanced (1.5F) - Super (2F) - Maximum (2.5F)

occupancy of the room they are installed in and they are pushing the sensed data regularly to the controller. Every 15 seconds, the temperature sensor will update its reading. Although such frequent updates are required for temperature, it is not necessarily for occupancy. As a result, the update time for the occupancy is longer, at 5 minutes [77].

ID	**Transition**
T_1	(SIdle, [Einput (Temp, Occupied)] OR [Uinput (VC= increase or decrease, TS= ↻ or ↺)], -)/(SActive, Send m_{11} "SData"))
T_2	(SActive, [After sending the collected data], -)/(SIdle, -)
T_3	(CReady, [After receiving the sensed data], -)/(Check, -)
T_4	(Check, [Temp < = TThreshold and/or Occupied = False and/or VC= increase and/or TS= ↻], -)/(Increase, Send m_{12} FD="Increase the temperature"))
T_5	(Check, [Temp > TThreshold and/or Occupied = True and/or VC= decrease and/or TS= ↺], -)/(Decrease, Send m_{12} FD="Decrease the temperature")
T_6	(Increase or Decrease, [After sending m_{12}], -)/(Check, -)
T_7	(Check, -)/(CIdle, -)
T_8	(Decrease, [IncreaseT], -)/(Increase, -)
T_9	(Increase, [DecreaseT], -)/(Decrease, -)
M_{11}	Send m_{11} "Sensed Data (SData)"
M_{12}	Send m_{12} "Final Decision (FD)"

Table (3.6) Smart Thermostat Transitions 3.7

- **T_1:** This transition goes from state S_1 (SIdle) to state S_2 (SActive). It uses the following inputs: (VC, TS, Temp, Occupied). Message on this transition is send m_{11} "SData".

- **T_2:** This transition goes from state S_2 (SActive) to state S_1 (SIdle). No inputs or actions are needed for this transition. m_{11} must have been sent previously.

- **T_3:** This transition goes from state S_3 (CReady) to state S_4 (Check). No inputs or actions are needed for this transition. m_{11} should have been received.

- **T_4:** This transition goes from state S_4 (Check) to state S_5 (Increase). It uses the following inputs: (VC, TS, Temp, Occupied, TThreshold, ThresholdLevel). Message on this transition is send m_{12}.

- **T_5:** This transition goes from state S_4 (Check) to state S_6 (Decrease). It uses the following inputs: (VC, TS, Temp, Occupied, TThreshold, ThresholdLevel). Message on this transition is send m_{12}.

- **T_6:** This transition goes from state S_5 (Increase) or S_6 (Decrease) to state S_4 (Check). No inputs or actions needed for this transition. After sending m_{12}.

- **T_7:** This transition goes from state S_4 (Check) to state S_7 (CIdle). No inputs or actions needed for this transition.

- **T_8:** This transition goes from state S_8 (Decrease) to state S_9 (Increase) in the actuator. The input of this transition is IncreaseT.

- **T_9:** This transition goes from state S_9 (Increase) to state S_8 (Decrease) in the actuator. The input of this transition is DecreaseT.

The first transition started with a S_1 (SIdle) state which means the sensor is idle before it becomes ready to collect data. It ends with S_2 (SActive) state, i.e. the sensor is active and ready to collect data. The condition for this transition to occur is determined by user or environment inputs. Environmental inputs for the smart thermostat are the sensed room temperature (Temp) and/or the sensed number of people within the room (Occupied). Other possible inputs are from the user by voice command (VC) to increase or decrease the temperature and/or by touching the screen (TS) on the device clockwise to increase or counterclockwise to decrease the temperature. The sensed data (SData) from the environment or the user or both are

sent to the controller via communication channel M_{11}. After sending the collected data the sensor goes back to S_1 (SIdle) (no condition on this transition). T_3 is a transition in the controller model between S_3 (CReady) and S_4 (Check) nodes. m_{11} must be received in order for this transition to occur. State S_4 (Check) is responsible to check and compare the received data with the predefined variables TThreshold (Temperature Threshold) or to check and compare the received user inputs VC (Voice command) or TS (Touch Screen). If the sensed temperature is less than or equal to the defined TThreshold then T_4 moves the controller to state S_5 (Increase) and m_{12} (FD="Increase the temperature") is sent to the actuator to increase the thermostat temp. If the sensed temperature is greater than the defined TThreshold then T_5 moves the controller to state S_6 (Decrease) and m_{12} (FD="Decrease the temperature") is sent to the actuator to decrease the thermostat temperature. T_6 is the transition that moves the controller from the states S_5 or S_6 (Increase or Decrease) back to state S_4 (Check) and this transition has one condition, after sending m_{12}. T_7 has no condition and it only changes the controller's state from S_4 (Check) to S_7 (CIdle).

Model Amazon Echo Dot

Tom's Guide [66] nominated the Amazon Echo Dot as the overall best smart devices in 2022. Amazon Echo Dot is a sphere shape speaker that has a few buttons on the top two of them control volume (+ to increase and - to decrease), mute the speaker, and unmute it. It uses an artificially intelligent assistant called Alexa which is capable to do multiple tasks. Alexa can play music, answer some questions by searching the Internet, set alarms or timer, control other smart devices within the SHS, play games, create a shopping list and many other tasks. Amazon Echo Dot can get user input as voice commands (VC) or click on buttons (+/- , mic

on/off). Any voice command should start with *Alexa* then ask her your question, for example, *Alexa, what time is it?*. Volume can be controlled by clicking on +/- buttons. At any time you can mute the speaker by clicking on microphone mute button or enable it by clicking on unmute button. This device can also get environmental inputs when you choose to set it up on guard mode. When you are away from home you may simply say *Alexa, I am leaving.* Alexa now will detect environmental sounds (ES) such as smoke alarm and glass break sounds and will notify the user through the smart phone application. Many other features can be controlled via the smart phone application. Fig 3.8 shows the Amazon Echo Dot device CEFSM model. The different inputs and possible values are listed in Table 3.7. The Amazon Echo Dot takes voice commands (VC) and click on buttons as user inputs. It takes environmental sounds (ES) as environmental input.

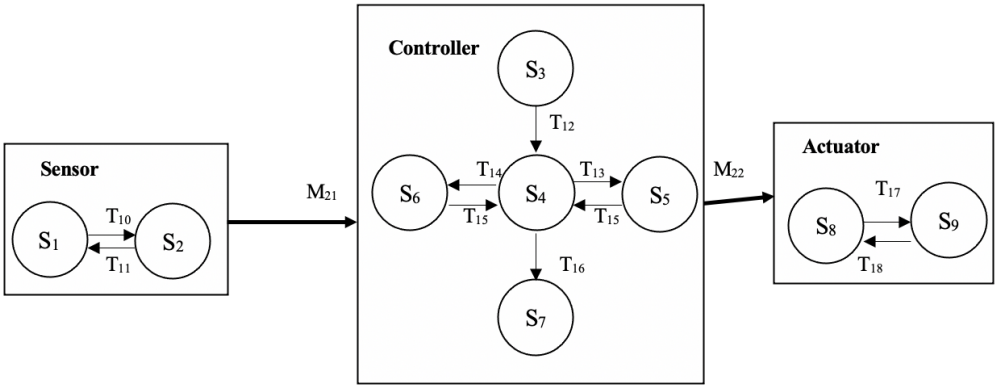

Figure (3.8) Amazon Echo Dot CEFSM Model

Table 3.8 lists the CEFSM transitions definitions for the Amazon Echo Dot. A unique id for each transition is given in column 1 and the input action constraints are shown in column 2. Table 3.8 presents 9 transitions (T_{10} - T_{18}) and two message channels (M_{21}, M_{22}) for the Amazon Echo Dot. The device consists of an analog

Uinput	Input Domain	Einput	Input Domain
VC	("Alexa, any command")	ES	(Broken Glass sound, Fire Alarm sound)
CB	(+, -, mic on, mic off)		

Table (3.7) Amazon Echo Dot Input Values

sensor (Sound Sensor), a controller and an actuator. When a sound detected the sound sensor push the sensed data to the controller. The controller should search the information in the Internet and send the answer to the actuator which push the answer via the speaker.

ID	Transition
T_{10}	(SIdle, [Einput (ES= Glass broken, Fire alarm)] OR [Uinput (VC= "Alexa, ...", CB= + or - or mic on or mic off)], -)/(SActive, Send m_{21} "SData"))
T_{11}	(SActive, [After sending the collected data], -)/(SIdle, -)
T_{12}	(CReady, [After receiving the sensed data], -)/(Check, -)
T_{13}	(Check, [CB = + and/or VC= "Alexa, increase the volume"], -)/(Increase volume, Send m_{22} FD="Increase the volume"))
T_{14}	(Check, [CB = - and/or VC= "Alexa, decrease the volume"], -)/(Decrease volume, Send m_{22} FD="Decrease the volume")
T_{15}	(Increase volume or Decrease volume, [After sending m_{22}], -)/(Check, -)
T_{16}	(Check, -)/(CIdle, -)
T_{17}	(Decrease volume, [IncreaseV], -)/(Increase volume, -)
T_{18}	(Increase volume, [DecreaseV], -)/(Decrease volume, -)
M_{21}	Send m_{21} "Sensed Data (SData)"
M_{22}	Send m_{22} "Final Decision (FD)"

Table (3.8) Amazon Echo Dot Transitions 3.8

- **T_{10}:** This transition goes from state S_1 (SIdle) to state S_2 (SActive). It uses the following inputs: (VC, CB, ES). Message on this transition is send m_{21} "SData".

- **T_{11}:** This transition goes from state S_2 (SActive) to state S_1 (SIdle). No inputs or actions are needed for this transition. m_{21} must have been sent previously.

- **T_{12}:** This transition goes from state S_3 (CReady) to state S_4 (Check). No inputs or actions are needed for this transition. m_{21} should have been received.

- **T_{13}:** This transition goes from state S_4 (Check) to state S_5 (Increase volume). It uses the following inputs: (VC, CB, ES). Message on this transition is send m_{22}.

- **T_{14}:** This transition goes from state S_4 (Check) to state S_6 (Decrease volume). It uses the following inputs: (VC, CB, ES). Message on this transition is send m_{22}.

- **T_{15}:** This transition goes from state S_5 (Increase volume) or S_6 (Decrease volume) to state S_4 (Check). No inputs or actions needed for this transition. After sending m_{22}.

- **T_{16}:** This transition goes from state S_4 (Check) to state S_7 (CIdle). No inputs or actions needed for this transition.

- **T_{17}:** This transition goes from state S_8 (Decrease volume) to state S_9 (Increase volume) in the actuator. The input of this transition is IncreaseV.

- **T_{18}:** This transition goes from state S_9 (Increase volume) to state S_8 (Decrease volume) in the actuator. The input of this transition is DecreaseV.

Nest Cam (battery)

Nest camera is a smart device that allows you to vision indoor or outdoor activity within your smart home. You can install it easily and watch live view from your

phone. You can also talk from your phone to that camera by click on a microphone button in the application and people in the room or on the door if the camera installed outdoor can response to you directly. This device does not have any user inputs, and it has two environmental inputs (Motion (M) and Sound (S)). Fig 3.9 shows the Nest cam device CEFSM model. The different inputs and possible values are listed in Table 3.9. The Nest cam takes two environmental inputs Sound (S) and Motion (M).

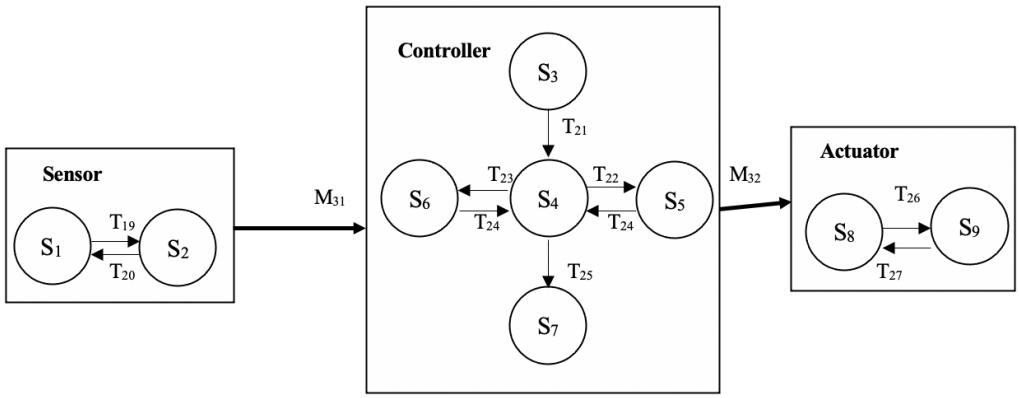

Figure (3.9) Nest Cam CEFSM Model

Uinput	Input Domain	Einput	Input Domain
None	-	Sound (S) Motion (M)	(True, False) (True, False)

Table (3.9) Nest Cam Input Values

Table 3.10 lists the CEFSM transitions definitions for the Nest Cam. A unique id for each transition is given in column 1 and the input action constraints are shown in column 2. Table 3.10 presents 9 transitions (T_{19} - T_{27}) and two message channels (M_{31}, M_{32}) for the Nest Cam. The device consists of two analog sensors (Sound

Sensor and Motion Sensor), a controller and an actuator. When a sound or motion detected the sound and motion sensor push the sensed data to the controller. The received data should be presented on the user phone as a live view of the camera.

ID	Transition
T_{19}	(SIdle, [Einput (S= True or False, M= True or False)], -)/(SActive, Send m_{31} "SData"))
T_{20}	(SActive, [After sending the collected data], -)/(SIdle, -)
T_{21}	(CReady, [After receiving the sensed data], -)/(Check, -)
T_{22}	(Check, [S= True and/or M= True], -)/(Video, Send m_{32} FD="Video"))
T_{23}	(Check, [S= False and/or M= False], -)/(Idle, Send m_{32} FD="Stop")
T_{24}	(Video or Idle, [After sending m_{32}], -)/(Check, -)
T_{25}	(Check, -)/(CIdle, -)
T_{26}	(Idle, [Video], -)/(Video, -)
T_{27}	(Video, [Stop], -)/(Idle, -)
M_{31}	Send m_{31} "Sensed Data (SData)"
M_{32}	Send m_{32} "Final Decision (FD)"

Table (3.10) Nest Cam Transitions 3.9

- **T_{19}:** This transition goes from state S_1 (SIdle) to state S_2 (SActive). It uses the following inputs: (S, M). Message on this transition is send m_{31} "SData".
- **T_{20}:** This transition goes from state S_2 (SActive) to state S_1 (SIdle). No inputs or actions are needed for this transition. m_{31} must have been sent previously.
- **T_{21}:** This transition goes from state S_3 (CReady) to state S_4 (Check). No inputs or actions are needed for this transition. m_{31} should have been received.
- **T_{22}:** This transition goes from state S_4 (Check) to state S_5 (Video). It uses the following inputs: (S, M). Message on this transition is send m_{32}.
- **T_{23}:** This transition goes from state S_4 (Check) to state S_6 (Idle). It uses the following inputs: (S, M). Message on this transition is send m_{32}.

- **T_{24}:** This transition goes from state S_5 (Video) or S_6 (Idle) to state S_4 (Check). No inputs or actions needed for this transition. After sending m_{32}.

- **T_{25}:** This transition goes from state S_4 (Check) to state S_7 (CIdle). No inputs or actions needed for this transition.

- **T_{26}:** This transition goes from state S_8 (Idle) to state S_9 (Video) in the actuator. The input of this transition is Video.

- **T_{27}:** This transition goes from state S_9 (Video) to state S_8 (Idle) in the actuator. The input of this transition is Idle.

August Wi-Fi Smart Locks

August Wi-Fi Smart Locks have DoorSense, a sensor that gives the smart lock the current status of your door if it is open or close. You can check if your door is closed and locked or not from one click on your mobile phone. No more trips back to check that or worrying if you closed it before you left or not. A notification come to you on your phone if anything changes with the status of your door. You could also set up an auto-lock option via the mobile app for any time between 30 seconds and 30 minutes. August knows that you have returned home and unlocks the door for you via geo-fencing feature. With a click on your phone, you can remotely let people to your home or even lock the door if you forgot to do. You can create temporary, or recurring "keys" for friends, family, or others to unlock the door via the August app. These keys can be set to only work during certain hours, or only work for a limited period. You can get a record with the exact time of who get in and out of the house from activity feed in the application. This device can take user inputs which is physically lock or unlock the smart lock, open or close the door, and time to auto-lock.

Fig 3.10 shows the August Wi-Fi Smart Lock device CEFSM model. The different inputs and possible values are listed in Table 3.11. The August Wi-Fi Smart Lock takes three user inputs (Open/Close the door (Open), Lock/Unlock (Lock) and Time Threshold (TiThreshold)) and two environmental inputs (DoorSense (DS) and Time (Time)).

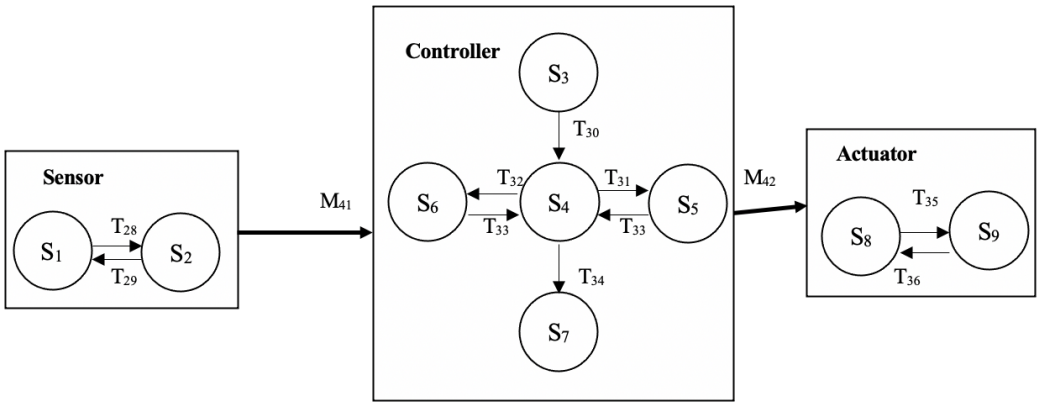

Figure (3.10) August Wi-Fi Smart Lock CEFSM Model

Uinput	**Input Domain**	**Einput**	**Input Domain**
Open/Close (Open)	(True, False)	DoorSense (DS)	(True, False)
Lock/Unlock (Lock)	(True, False)	Time (Time)	[00:00:30 - 00:30:00]
Time Threshold (TiThreshold)	[30 sec - 30 min]		

Table (3.11) August Wi-Fi Smart Lock Input Values

Table 3.12 lists the CEFSM transitions definitions for the August Wi-Fi Smart Lock. A unique id for each transition is given in column 1 and the input action constraints are shown in column 2. Table 3.12 presents 9 transitions (T_{28} - T_{36}) and two message channels (M_{41}, M_{42}) for the August Wi-Fi Smart Lock. The device consists of two analog sensors (Door Sensor and Time Sensor), a controller and an actuator.

ID	Transition
T_{28}	(SIdle, [Einput (Time= Time, DS= True or False)] OR [Uinput (Open= True or False, Lock= True or False and TiThreshold= [30 sec - 30 min])], -)/(SActive, Send m_{41} "SData"))
T_{29}	(SActive, [After sending the collected data], -)/(SIdle, -)
T_{30}	(CReady, [After receiving the sensed data], -)/(Check, -)
T_{31}	(Check, [Open= False and Lock= False and/or Time== TiThreshold], -)/(Lock, Send m_{42} FD="Lock"))
T_{32}	(Check, [Open= False and Lock= True], -)/(Unlock, Send m_{42} FD="Unlock")
T_{33}	(Lock or Unlock, [After sending m_{42}], -)/(Check, -)
T_{34}	(Check, -)/(CIdle, -)
T_{35}	(Unlock, [Lock], -)/(Lock, -)
T_{36}	(Lock, [Unlock], -)/(Unlock, -)
M_{41}	Send m_{41} "Sensed Data (SData)"
M_{42}	Send m_{42} "Final Decision (FD)"

Table (3.12) August Wi-Fi Smart Lock Transitions 3.10

- **T_{28}:** This transition goes from state S_1 (SIdle) to state S_2 (SActive). It uses the following inputs: (Time, DoorSense, Open, Lock, TiThreshold). Message on this transition is send m_{41} "SData".

- **T_{29}:** This transition goes from state S_2 (SActive) to state S_1 (SIdle). No inputs or actions are needed for this transition. m_{41} must have been sent previously.

- **T_{30}:** This transition goes from state S_3 (CReady) to state S_4 (Check). No inputs or actions are needed for this transition. m_{41} should have been received.

- **T_{31}:** This transition goes from state S_4 (Check) to state S_5 (Lock). It uses the following inputs: (Time, DoorSense, Open, Lock, TiThreshold). Message on this transition is send m_{42}.

- **T_{32}:** This transition goes from state S_4 (Check) to state S_6 (Unlock). It uses the following inputs: (Time, DoorSense, Open, Lock, TiThreshold). Message on this transition is send m_{42}.

- **T_{33}:** This transition goes from state S_5 (Lock) or S_6 (Unlock) to state S_4 (Check). No inputs or actions needed for this transition. After sending m_{42}.

- **T_{34}:** This transition goes from state S_4 (Check) to state S_7 (CIdle). No inputs or actions needed for this transition.

- **T_{35}:** This transition goes from state S_8 (Unlock) to state S_9 (Lock) in the actuator. The input of this transition is Lock.

- **T_{36}:** This transition goes from state S_9 (Lock) to state S_8 (Unlock) in the actuator. The input of this transition is Unlock.

Nest Protect Smoke detector

Nest protect smoke detector is a smart detector that help you protect your home from fire, by detecting smoke and carbon monoxide in a residential environment. This device does not have any user inputs, and it has five environmental inputs (Smoke and Carbon monoxide sensor (CO), Temperature (Temp), Humidity (H) Occupied (Occupied) and Time (Time)). Fig 3.11 shows the Nest protect smoke detector device CEFSM model. The different inputs and possible values are listed in Table 3.13. All the input values range for different environmental inputs are from google nest help center [88].

Table 3.14 lists the CEFSM transitions definitions for the Nest protect smoke detector. A unique id for each transition is given in column 1 and the input action constraints are shown in column 2. Table 3.14 presents 9 transitions (T_{37} - T_{45})

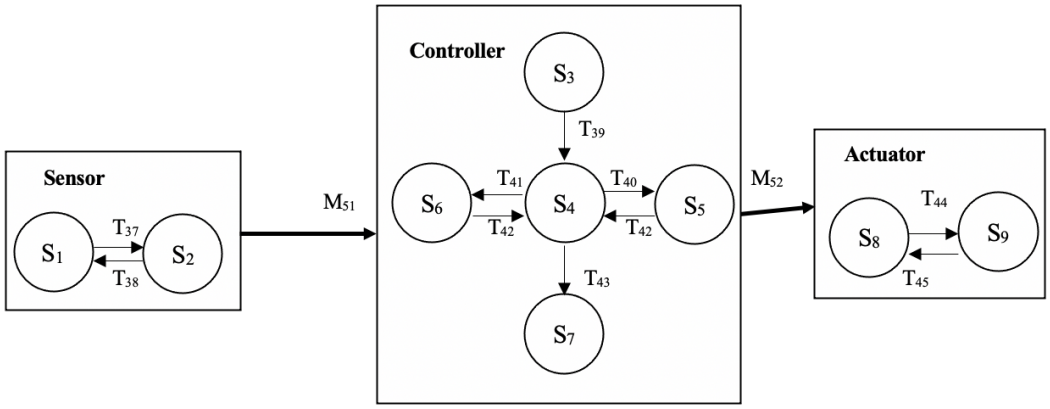

Figure (3.11) Nest protect smoke detector CEFSM Model

Einput	**Input Domain**
Smoke and Carbon monoxide sensor (CO)	[0-400] parts per million (ppm)
Temperature (Temp)	[40°F - 100°F]
Humidity (H)	[Up to 90% RH]
Occupied (Occupied)	(True, False)
Time (Time)	[0 - 300] minuets

Table (3.13) Nest protect smoke detector Input Values

and two message channels (M_{51}, M_{52}) for the Nest protect smoke detector. The device consists of an analog sensor (Smoke and Carbon monoxide sensor (CO)), a controller and an actuator. When certain level of smoke and carbon monoxide detected the sensor push the sensed data to the controller. Based on the level of parts per million of smoke and carbon monoxide in a room the alert will be sent within specific time. Nest protect smoke detector user's guide [1] states that the American National Standard ANSI/UL 2034's standards for alarm reaction time are met by the Nest Protect CO sensor. The following are the standard alarm times:

- At 70 PPM, the unit must alarm within 60-240 minutes.
- At 150 PPM, the unit must alarm within 10-50 minutes.
- At 400 PPM, the unit must alarm within 4 to 15 minutes.

ID	Transition
T_{37}	(SIdle, [Einput (CO= [0-400], Temp= [40-100], H= Up to 90% RH, Occupied= True or False, Time= [0-300])], -)/(SActive, Send m_{51} "SData"))
T_{38}	(SActive, [After sending the collected data], -)/(SIdle, -)
T_{39}	(CReady, [After receiving the sensed data], -)/(Check, -)
T_{40}	(Check, [CO$\geq$ 70], -)/(Alert, Send m_{52} FD="Alert"))
T_{41}	(Check, [CO$<$70], -)/(NoAlert, Send m_{52} FD="NoAlert")
T_{42}	(Alert or NoAlert, [After sending m_{52}], -)/(Check, -)
T_{43}	(Check, -)/(CIdle, -)
T_{44}	(NoAlert, [Alert], -)/(Alert, -)
T_{45}	(Alert, [NoAlert], -)/(NoAlert, -)
M_{51}	Send m_{51} "Sensed Data (SData)"
M_{52}	Send m_{52} "Final Decision (FD)"

Table (3.14) Nest protect smoke detector Transitions 3.11

- **T_{37}:** This transition goes from state S_1 (SIdle) to state S_2 (SActive). It uses the following inputs: (CO, Temp, Time, Occupied, H). Message on this transition is send m_{51} "SData".
- **T_{38}:** This transition goes from state S_2 (SActive) to state S_1 (SIdle). No inputs or actions are needed for this transition. m_{51} must have been sent previously.
- **T_{39}:** This transition goes from state S_3 (CReady) to state S_4 (Check). No inputs or actions are needed for this transition. m_{51} should have been received.
- **T_{40}:** This transition goes from state S_4 (Check) to state S_5 (Alert). It uses the following inputs: (CO, Temp, Time, Occupied, H). Message on this transition is send m_{52}.

- **T_{41}:** This transition goes from state S_4 (Check) to state S_6 (NoAlert). It uses the following inputs: (CO, Temp, Time, Occupied, H). Message on this transition is send m_{52}.

- **T_{42}:** This transition goes from state S_5 (Alert) or S_6 (NoAlert) to state S_4 (Check). No inputs or actions needed for this transition. After sending m_{52}.

- **T_{43}:** This transition goes from state S_4 (Check) to state S_7 (CIdle). No inputs or actions needed for this transition.

- **T_{44}:** This transition goes from state S_8 (NoAlert) to state S_9 (Alert) in the actuator. The input of this transition is Alert.

- **T_{45}:** This transition goes from state S_9 (Alert) to state S_8 (NoAlert) in the actuator. The input of this transition is NoAlert.

3.4.2 Model the Mobile Application (FSMApp Approach)

To model the Mobile Application that controls the SHS, we used the approach described in [10]:

- Phase 1: Build a hierarchical model HFSM:
  - Partition the mobile app into clusters (Cs).
  - Define Logical App Pages (LAPs) and Input-Action constraints for each page.
  - Build FSMs for clusters as a multi-level hierarchy including an Aggregate FSM (AFSM) to represent the top level of the application.

Partition the mobile app into clusters (Cs)

The term *cluster* refers to a group of software modules/application pages that work together to perform a logical or user-level task. The mobile application is divided into clusters in the first step. Clusters represent functions that can be identified by users at the highest degree of abstraction. Hence, an HFSM = $\{\text{FSM}_i\}^n_{i=0}$ with a top level FSM_0 = AFSM. Logical App Pages (LAPs) or clusters are represented by nodes in each FSM. An FSM's edges can be internal or external. Cluster boundaries are crossed by external nodes. These nodes become internal at the next higher FSM level. External edges can enter or exit a cluster FSM. Clusters can be an individual device or commands that the app has to control specific device. A page (screen) that is displayed to the user is an activity in a mobile application. The activity includes a set of layouts for organizing the page's items. The site navigation layout, coupling relationships among the components, and design information can all be used to identify clusters.

Define Logical App Pages (LAPs) and Input-Action constraints for each page

According to Al Haddad [10], in Mobile apps, which consist of several screens, we will consider screens as input components, or Logical Application Pages (LAPs). XML forms are the major component of most of Mobile App screens. These forms can be connected to a different back-end software module. We model theses app screens as multiple LAPs to facilitate testing of these modules. A LAP is either a physical app screen, physical app component, or the portion of an app activity that accepts data from the user through a XML form, and then sends the data to a specific software module. LAPs are abstracted from the presentation defined by

the XML and are described in terms of their sets of inputs and actions. All inputs in a LAP are considered atomic: data entered to a text field is considered as one user input symbol, regardless number of characters entered in the field. Input rules are either be required, or optional and users can enter inputs in any order, or in a specific order. Table 3.15 shows the input constraints, and the order of the inputs. R means input must be entered. R(parm) means at least one value must be entered. O means that an input is optional. C1 means that one option must be selected from a list of choices, and Cn means that more than one option can be selected from a list of choices.

Input Choice	**Order**
Required (R)	Sequence (S)
Required Value (R(parm))	Any (A)
Optional (O)	
Single choice (C1)	
Multiple choice (Cn)	

Table (3.15) Mobile Application Input Constraints

To illustrate what components look like we explain common components of Android applications and what FSMApp's input constraints would look like:

1. **A bottom sheet** is a sheet of material that slides up from the bottom edge of the screen. The action of the bottom sheet is a click. The input constraint is R().

2. **A button** indicates what action will occur when the user touches. The action of the button is click. The input constraints are R(), or select the button then click C1(Select Button, Click).

3. **A card** is a sheet of material with unique related data that serves as an entry point to more detailed information. The actions of the card are click, swipe,

scroll, and pick-up-and-move. The input constraint is C1(, , ,) and the list of choices is provided to select one of them.

4. **Chips** represent complex entities in small blocks, such as a contact. The action of the chip is a click. The input constraint is R(S(,)).

5. **Data tables** are used to represent raw data sets, and usually appear in desktop enterprise products. The actions of the data tables are row hover, row selection, column sorting, column hover, and test editing. The input constraint is C1(, , ,).

6. **Dialogs** inform users about critical information, require users to make decisions, or encapsulate multiple tasks within a discrete process. The action of the dialog is click. The input constraint is R().

7. **Dividers** group and separate content within lists and page layouts. The action of the divider is click. The input constraint is R().

8. **Grid** lists are an alternative to standard list views. The actions of the grid list are vertical scrolling or filtering. The input constraint is C1(,).

9. **Lists** present multiple line items in a vertical arrangement as a single continuous element. It has a checkbox, a switch and a reader. The action of the list is sort. The input constraint is R().

10. **Menus** allow users to take an action by selecting from a list of choices revealed upon opening a temporary, new sheet of material. The actions of the menu are scroll and click. The input constraint is C1(,).

11. **Pickers** provide a simple way to select a single value from a pre-determined set. For example, time and date pickers. The actions of the pickers are drop-down and click. The input constraint is C1(,).

12. **Progress & activity** indicators are visual indications of an app loading content. The action of Progress & Activity is loading. The input constraint is R().

13. **Selection Controls** allow the user to select options. The action of selection controls is click. The input constraint is R().

14. **Sliders** let the user select a value from a continuous or discrete range of values by moving the slider thumb. The action of slider change is scrolling. The input constraint is R().

15. **Snackbars & toasts** provide lightweight feedback about an operation by showing a brief message at the bottom of the screen. The action of snackbars & toasts is click. The input constraint is R().

16. **Subheaders** are special list tiles that delineate distinct sections of a list or grid list and are typically related to the current filtering or sorting criteria. The action of subheader is click. The input constraint is R().

17. **Steppers** convey progress through numbered steps. They may also be used for navigation. The action of the steppers is to show the next steps. The input constraint is R().

18. **Tabs** make an app easy to explore and switch between different views or functional aspects of an app or to browse categorized data sets. The action of tab is scroll. The input constraint is R().

19. **Toolbars** appear above the view affected by their actions. The action of toolbars is scroll. The input constraint is R().

20. **Tooltips** are labels that appear on hover and focus when the user hovers over an element with the cursor, focuses on an element using a keyboard (usually through the tab key), or upon touch (without releasing) in a touch UI. The actions of tooltips are click and hover. The input constraint is C1(,).

21. **Text fields** allow the user to input text, select text (cut, copy, paste), and lookup data via auto-completion. The actions of text fields are lookup table, select text, and write text. The input constraint is C1(, ,)

The transitions are between the nodes and the clusters. They are annotated with input constraints to indicate what inputs and actions lead to the next node or cluster.

Example: Nest Application

We illustrate our approach using the Nest application as an example. To build FSM_{App} models, we need to divide the Mobile App into clusters and logical app pages (LAPs). We will consider only testing the functionality of the SHS devices, i.e. we will assume that all the devices are installed and running. Therefore, dividing the Mobile App will be based on that assumption.

We divided the Nest application into clusters. Fig 3.12 shows the top level of the Nest App, while Fig 3.13 to 3.19 show the other clusters of the Nest App with one device installed (Thermostat). The Nest app has three main screens, the main screen of the application is screen (A). It has two major functions: (1) Add a new Nest product to the smart home as screen (not covered). (2) Control a Nest product as screen (B) in Fig 3.13. We decomposed the main screen (A) into one cluster (B)

as shown in Fig 3.12. Cluster (B) has five subclusters: "Mode" in Fig 3.14, "Eco" in Fig 3.15, "Fan" in Fig 3.16, "History" in Fig 3.17 and "Schedule" in Fig 3.18. Finally, Fig 3.19 represent the set up schedule LAPs. Fig 3.12 shows the top level of the Nest App. There is one main cluster: control a Nest product and exiting the App. Figs 3.13 to 3.19 show the other clusters of the Nest App with a smart Nest thermostat installed. The App screens are classified as: Screen (A) shows the main screen of the application. It has two functions: (1) Add a new Nest device, (2) Control an installed device (Screen (B)). We classify the main Screen (A) into one cluster (B). The Exit node is a LAP. Fig 3.12 shows this as the AFSM for this example. Cluster (B) has five subclusters: "Mode" in Fig 3.14, "Eco" in Fig 3.15, "Fan" in Fig 3.16, "History" in Fig 3.17 and "Schedule" in Fig 3.18. The Main Screen has one LAPs with a button to cancel and exit the application. In the Mode cluster as Fig 3.14 shows, we have two LAPs with two buttons, cancel and select mode. In Fig 3.15, we have two LAPs with two buttons, cancel and select eco mode. While in fan cluster (illustrated in Fig 3.16), the user can select the fan speed from a list of choices.There is a LAP node with a cancel button. In the History cluster (Fig 3.17), we have two LAPs with two buttons, cancel and view history. Schedule cluster in Fig 3.17 has a subcluster Set up a schedule and one LAP with a cancel button. In the lowest level FSM, we have two LAPs with cancel and update buttons as shown in Fig 3.19.

Next, we determine input-action constraints for these models. For example, Fig 3.20 shows the Fan input-action constraints. In the Fan cluster, there are two states: either you select the fan speed (FanSpeed) from multiple choices (High, Medium, Low), or you cancel (with or without changing the fan speed). Incoming and outgoing edges for this cluster connect to the parent cluster. They don't require any user actions.

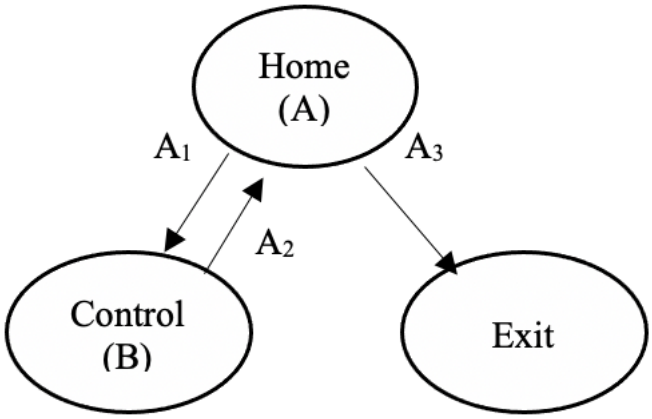

Figure (3.12) Top Level FSM (Main)

Tables 3.16 to 3.23 show the transitions, explanation, and input action constraints. A unique id for each transition is given in column 1. An explanation of the transition is given in column 2 and the input action constraints are shown in column 3. In the caption of these tables the corresponding graphs are mentioned. Table 3.16 shows 3 transitions for the main page cluster (AFSM). The transitions connect the main page with one cluster and one LAP (Exit App). Table 3.17 shows 8 transitions for Control Thermostat cluster to connect with 5 clusters and one LAP node (cancel). Table 3.18 shows 4 transitions for the Mode cluster to connect with two LAP nodes (Select mode and cancel). Table 3.19 shows 4 transitions for the Eco Mode cluster to connect with two LAP nodes (Select eco mode and cancel). Table 3.20 shows 4 transitions for the Fan cluster to connect two LAP nodes (Select fan speed and cancel). Table 3.21 shows 4 transitions for the History cluster to connect with view history and the cancel LAP. Table 3.22 shows 4 transitions for the Schedule cluster to connect with the set up schedule cluster and the cancel LAP. Table 3.23 shows 2 transitions for the set up schedule cluster.

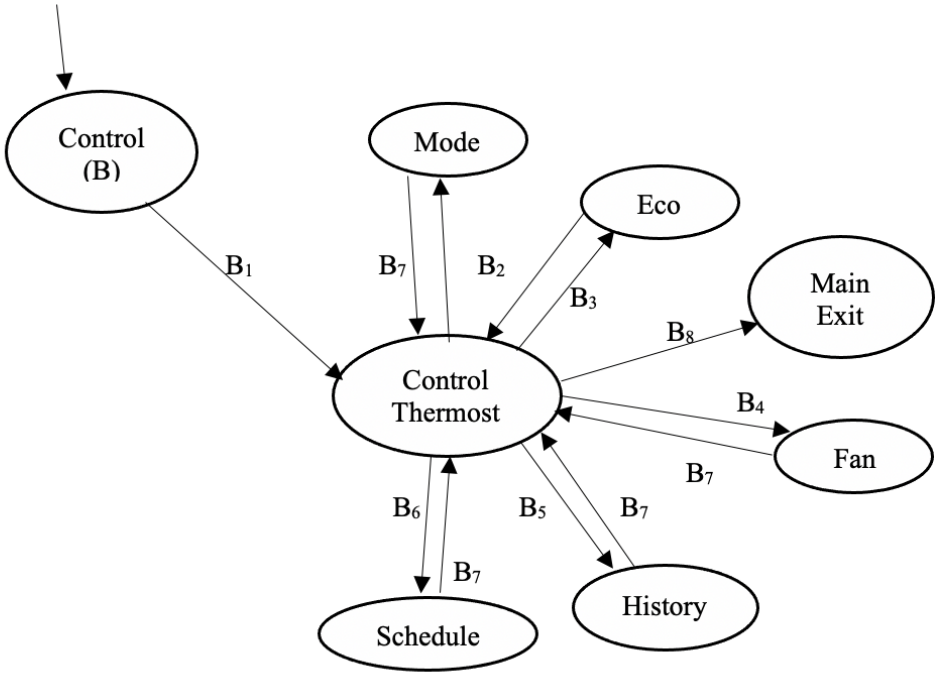

Figure (3.13) Lower Level FSM (Control)

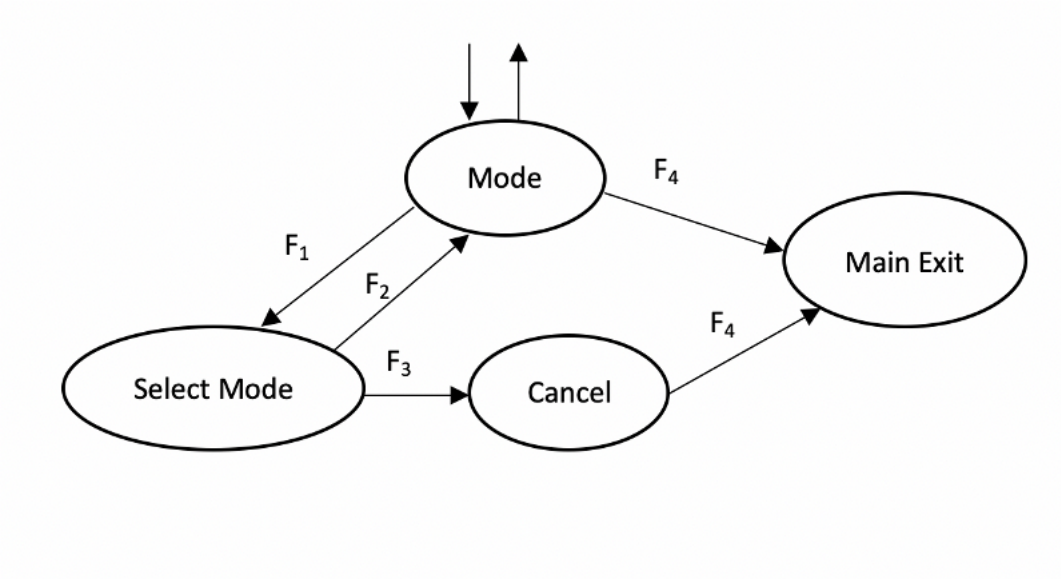

Figure (3.14) Lowest-level FSM (Control (Mode))

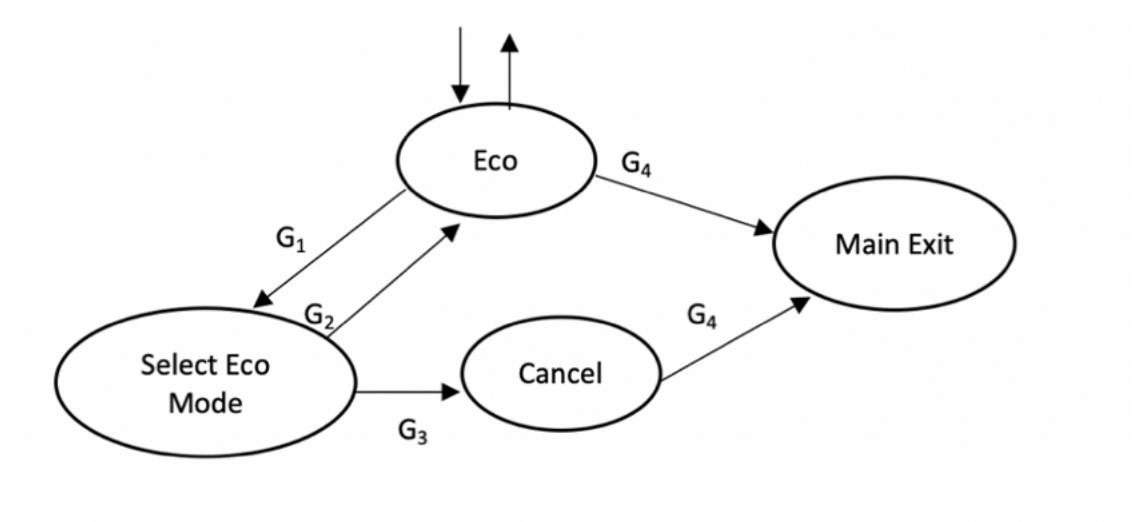

Figure (3.15) Lowest-level FSM (Control (Eco))

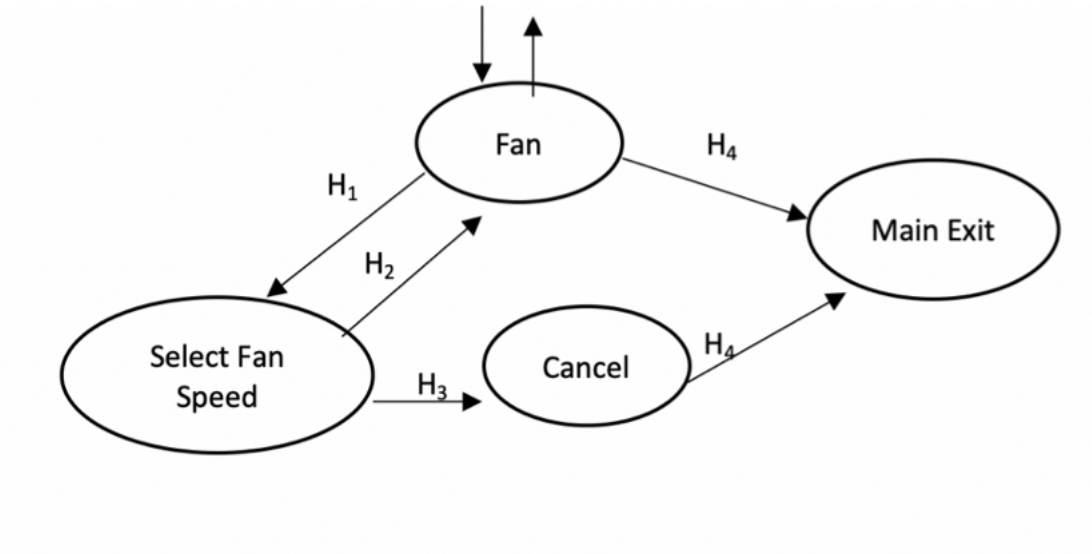

Figure (3.16) Lowest-level FSM (Control (Fan))

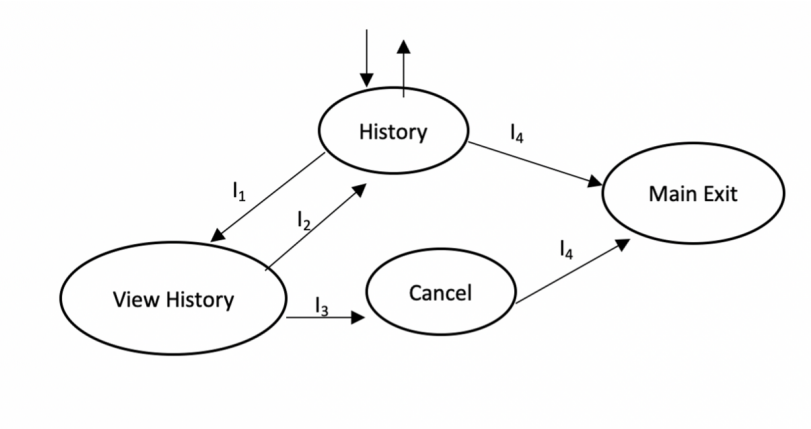

Figure (3.17) Lowest-level FSM (Control (History))

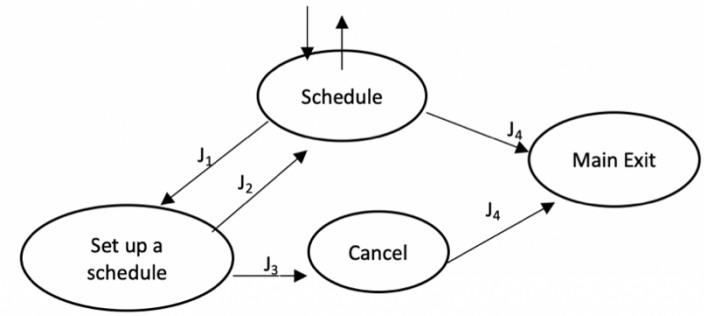

Figure (3.18) Lowest-level FSM (Control (Schedule))

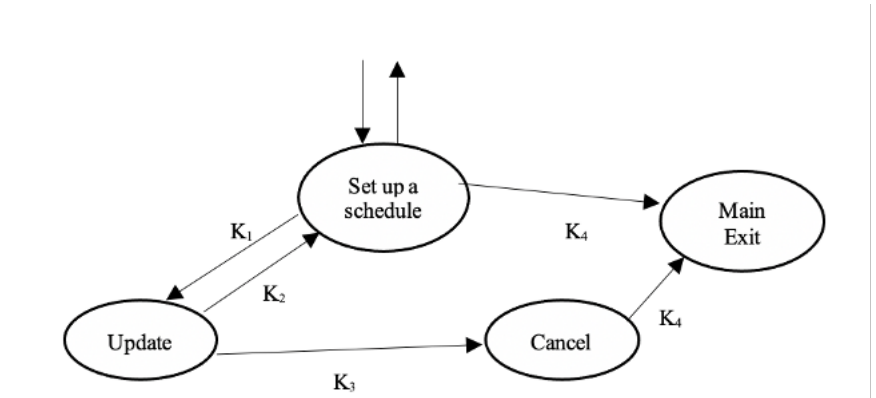

Figure (3.19) Lowest-level FSM (Control (Schedule Set Up))

Transition ID	Explanation	Input-Action Constraints
A_1	Access control product page	R (ManagePbutton)
A_2	Back to home page	R (Backbutton)
A_3	Exit the application	none

Table (3.16) Transitions for Top-level FSM (Home Page) Fig 3.12

Transition ID	Explanation	Input-Action Constraints
B_1	Access control Thermostat page	R (ManageTbutton)
B_2	Access Mode page	R (Modebutton)
B_3	Access Eco page	R (Ecobutton)
B_4	Access Fan page	R (Fanbutton)
B_5	Access History page	R (Historybutton)
B_6	Access Schedule page	R (Schedulebutton)
B_7	Back to previous page	none
B_8	Back to home page	R (Backbutton)

Table (3.17) Transitions for Lower-level FSM (Control Thermostat) Fig 3.13

Transition ID	Explanation	Input-Action Constraints
F_1	Select Mode	C1 (Cool, Heat)
F_2	Back to Mode	none
F_3	Cancel to control thermostat	S ((O (C1 (Cool, Heat))), R (CCT-button))
F_4	Back to home page	R (Backbutton)

Table (3.18) Transitions for Lowest-level FSM (Control (Mode)) Fig 3.14

Transition ID	Explanation	Input-Action Constraints
G_1	Select Eco Mode	R (EMbutton)
G_2	Back to Eco Mode	none
G_3	Cancel to control thermostat	S (O (EMbutton), R (CCTbutton))
G_4	Back to home page	R (Backbutton)

Table (3.19) Transitions for Lowest-level FSM (Control (Eco)) Fig 3.15

Transition ID	Explanation	Input-Action Constraints
H_1	Select Fan Speed	C1 (High, Medium, Low)
H_2	Back to Fan	none
H_3	Cancel to control thermostat	S ((O(C1(High ,Medium, Low))), R (Cancelbutton))
H_4	Back to home page	R (Backbutton)

Table (3.20) Transitions for Lowest-level FSM (Control (Fan)) Fig 3.16

Transition ID	Explanation	Input-Action Constraints
I_1	Select View History	R (VHbutton)
I_2	Back to History	none
I_3	Cancel to control thermostat	S (O (VHbutton), R (CCTbutton))
I_4	Back to home page	R (Backbutton)

Table (3.21) Transitions for Lowest-level FSM (Control (History)) Fig 3.17

Transition ID	Explanation	Input-Action Constraints
J_1	Select Set up a schedule	R (SSbutton)
J_2	Back to Schedule	none
J_3	Cancel to control thermostat	S (O (SSbutton), R (CCTbutton))
J_4	Back to home page	R (Backbutton)

Table (3.22) Transitions for Lowest-level FSM (Control (Schedule)) Fig 3.18

Transition ID	Explanation	Input-Action Constraints
K_1	Set up a schedule	R (Updatebutton)
K_2	Back to set up a schedule	none
K_3	Cancel to control thermostat	R (CCTbutton)
K_4	Back to home page	R (Backbutton)

Table (3.23) Transitions for Lowest-level FSM (Control (Set Up Schedule)) Fig 3.19

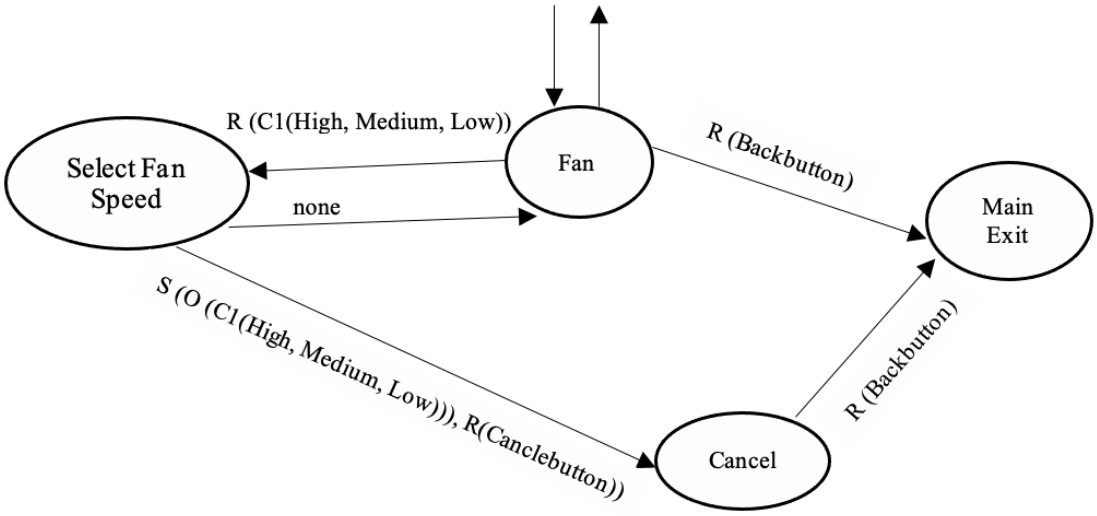

Figure (3.20) Annotated FSM for Fan Cluster of Table 3.20

3.4.3 Model the Interactions

Interactions in Smart Home Systems can be between any two devices or between the device and the Mobile App that controls that device. These interactions will be modeled using CEFSM. The next two sections explain how to model the two different possible interactions within a SHS.

Interactions between devices

Fig 3.21 models the interactions between devices. Device_1 and Device_2 are modeled based on the approach presented in Section 3.4.1. The messages on the transition between the devices send the status of each device. The device must act according to the received message from the other device. Table 3.24 shows the

transitions and the messages sent between the devices. The first column has the transition Id and the second explains the transition. The last column gives the input-action constraints for that transition.

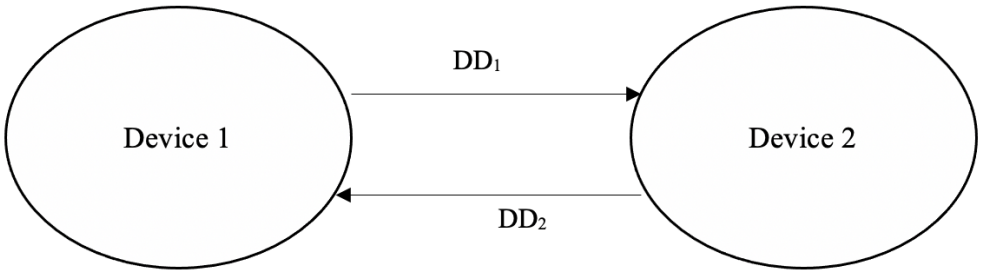

Figure (3.21) Devices Interaction

Tran ID	**Explanation**	**Input-Action Constraints**
DD_1	Send $Device_1$ status to $Device_2$	R($Device_1$ status, Send M_2)
DD_2	Send $Device_2$ status to $Device_1$	R($Device_2$ status, Send M_3)

Table (3.24) Transitions explanation for interactions between devices

Interactions between devices and Mobile Apps

Fig 3.22 models the interactions between a device and its controller (Mobile App). $Device_1$ and App_1 are modeled based on the approach presented in Section 3.4.1 and Section 3.4.2 respectively. The messages on the transition between the device and Mobile App are either a command from a user through a Mobile App or the status of the device to update that in the application. Table 3.25 shows the transitions and the messages sent between the device and the App.

For example, assume we have a smart lock installed on the main door and the Mobile App controlling this smart lock has two commands. The user can send a

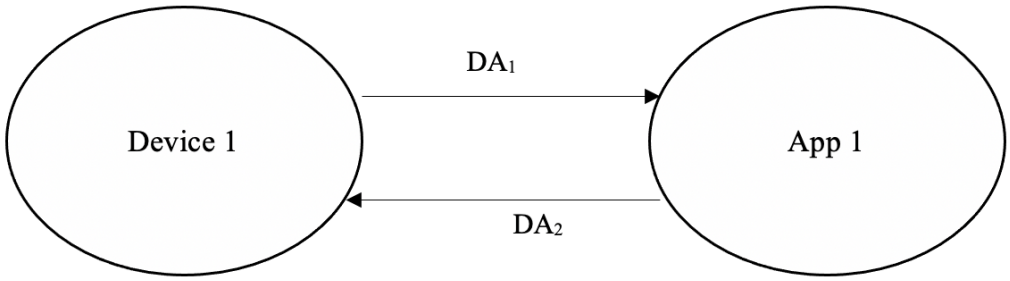

Figure (3.22) Device and Mobile App Interaction

Tran ID	**Explanation**	**Input-Action Constraints**
DA_1 DA_2	Send $Device_1$ status to App_1 User send a command to $Device_1$ by clicking on X button, where X can be for example, Unlock the door button.	R($Device_1$ status, Send M_1) R(Xbutton)

Table (3.25) Transitions explanation for interactions between device and Mobile App

command to lock or unlock the lock. The user can also set up an auto lock feature after a certain amount of time (between 0 minute to 5 minute) from the app. The user can check the status of the lock from their Mobile App. Therefore, the device status must be sent to the application regularly. Fig 3.23 shows the interaction between the smart lock device and the application controlling that device. We can see three transitions between the app and the device. These transitions are defined in Table 3.26. The first column shows the transaction Id and the second column defines the transition. Finally, the last column shows the input-action constraint of that transition. As we can see in Fig 3.23, there are three transitions between the Smart Lock and the Mobile App that is controlling it. DA_1 is the first transition

where the user sends a command to lock or unlock the Smart Lock device from the Mobile App. Required input-action constraint for this transition is clicking on a button from the App to send a command to the device. In this case, the lock/unlock button (L/Ubutton) must be clicked. DA_2 is the second transition and the status of the Smart Lock is sent via M_1 to update the device states on the App. DA_3 is the last transition where the user sends a command from the Mobile App to set up an auto lock/unlock feature. The required input-action constraint for this transition is clicking on an auto lock/unlock button (AL/Ubutton) from the App to send a command to the device then select the time period from multiple available choices (0, 1, 2, 3, 4, 5). The order of inputs in this transition is sequential.

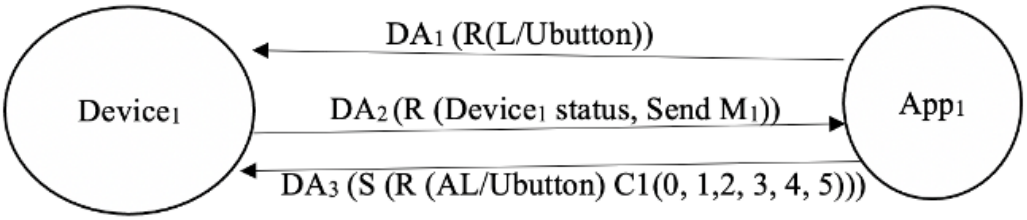

Figure (3.23) Smart Lock and Mobile App Interaction

TID	Explanation	Input-Action Constraints
DA_1	User command to Lock/Unlock door from the app	R(L/Ubutton)
DA_2	Device send the status upon change	R(Device$_1$ status, Send M_1)
DA_3	User command to Lock/Unlock door automatically after certain amount of time between 0 min to 5 min from the app	S(R (AL/Ubutton) C1(0, 1,2, 3, 4, 5))

Table (3.26) Transitions explanation for interactions between smart lock and Mobile App

Chapter 4

Generate Tests from Reusable Test-Ready Models

4.1 Problem Statement

After proposing reusable test-ready models of smart home systems (SHS) in the previous Chapter, it is time to show how these models be used with existing test generation approaches to create tests. We have models for device components, devices and FSMApp, as well as the interaction between devices and the interaction between devices and the system controller. This Chapter addresses the following research question:

- Can we develop a MBT technique for these non heterogeneous reusable test-ready models that includes testing criteria that can be used for device testing, device interaction testing and system testing?

This Chapter is organized as follows: The proposed approach is explained in Section 4.2. Section 4.2.1 provides the guidelines to test device components using

the device models. Testing the Mobile App is addressed in Section 4.2.2. Section 4.2.3 shows how to test interactions between devices and between devices and the Mobile App.

4.2 Proposed Approach

Our objective is to provide an MBT approach using our reusable test-ready models. Testing will be done on multiple levels: the Device level, the Interaction level and the System level. For testing we use the reusable test-ready models as follows: (1) we generate test paths from each model, (2) we aggregate the generated test paths to form abstract test cases, (3) we apply input domains partitioning for the inputs along these paths to convert the abstract test cases to concrete test cases, (4) we make the concrete test cases executable, (5) we execute and validate these test cases and report the findings and the results. Fig 4.1 shows the process of testing the reusable test-ready models.

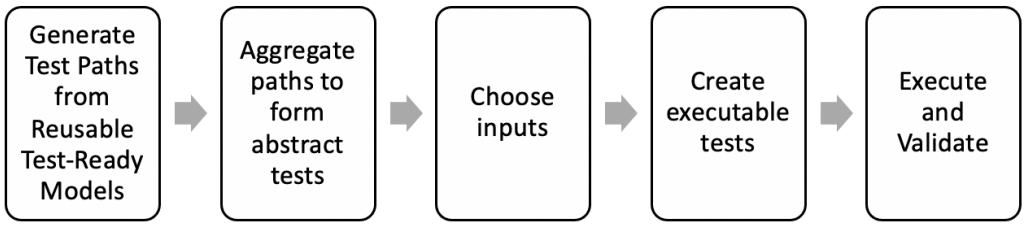

Figure (4.1) Phase 2: Testing Reusable Test-Ready Models

The reusable test-ready models make generating test process easy we will have the same set of tests generated from the same models the only different are the inputs we select along the transitions. These inputs are vary based on the devices.

4.2.1 Device Testing

Generate Test Paths

This contribution is published in the 22nd International Conference on Internet Computing & IoT, (Albahli et al. [8]). A test path is a sequence of transitions through a model. According to Offutt et. al [13] a path p, possibly of length zero, that starts at some node in N_0 and ends at some node in N_f. At device level we will generate test paths from CEFSMs model and through each lower-level EFSMs. We can generate test paths using any graph coverage criteria such as node coverage, edge coverage, edge pair coverage, prime path coverage, simple round trip coverage, complete round trip coverage, complete path coverage or specified path coverage [13]. We will first generate test paths through each EFSM model based edge coverage. We generate the test sequences for entire device that satisfy Each Path coverage. Tables 4.1 to 4.4 show test paths for each sensor, controller, and actuator as sequences of nodes. Device test paths are represented in Table 4.5. In this phase the inputs are the reusable test-ready models for devices constructed in phase 1 (Chapter 3). The output is a set of test paths for each component and a set of test paths of the entire device.

Test Path ID	**Test Path**	**Length**
$ASTP_1$	$[S_1 \xrightarrow{T_1} S_2 \xrightarrow{T_2} S_1]$	3

Table (4.1) Analog Sensor Test Paths of Fig **??**

Aggregate Test Paths

After we generate the test paths we should aggregate them. All combination criteria [13] will be used to aggregate the test paths from different components.

Test Path ID	**Test Path**	**Length**
$BSTP_1$	[S_3 – S_4 – S_3]	3

Table (4.2) Binary Sensor Test Paths of Fig 3.3

Test Path ID	**Test Path**	**Length**
CTP_1	[$S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_4} S_5 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7$]	5
CTP_2	[$S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_5} S_6 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7$]	5

Table (4.3) Controller Test Paths of Fig 3.4

Test Path ID	**Test Path**	**Length**
ATP_1	[$S_8 \xrightarrow{T_8} S_9 \xrightarrow{T_9} S_8$]	3

Table (4.4) Actuator Test Paths of Fig 3.5

Test Path ID	**Test Path**	**Length**
1	[Sensor $\xrightarrow{M_{11}}$ Controller]	2
2	[Controller $\xrightarrow{M_{12}}$ Actuator]	2

Table (4.5) Device Test Paths of Fig 3.6

ATP ID	**Aggregated Test Path**
ATP_1	$[S_1 \xrightarrow{T_1} S_2 \xrightarrow{T_2} S_1] \xrightarrow{M_{11}} [S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_4} S_5 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7]$
ATP_2	$[S_1 \xrightarrow{T_1} S_2 \xrightarrow{T_2} S_1] \xrightarrow{M_{11}} [S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_5} S_6 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7]$
ATP_3	$[S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_4} S_5 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7] \xrightarrow{M_{12}} [S_8 \xrightarrow{T_8} S_9 \xrightarrow{T_9} S_8]$
ATP_4	$[S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_5} S_6 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7] \xrightarrow{M_{12}} [S_8 \xrightarrow{T_8} S_9 \xrightarrow{T_9} S_8]$

Table (4.6) Aggregated Test paths generated from single device CEFSM model

The aggregated test paths are abstract test paths (ATP). We need to generate test data to transform these ATPs into executable test cases. We can use any input-space partitioning such as all combination coverage, each choice coverage, pair wise coverage, base choice coverage etc. [13]. We use the Each Choice Coverage (ECC) coverage criteria where one value from each block for each characteristic must be used in at least one test case [13]. We aggregate different component test paths from Tables 4.1 to 4.4 based on the level of interaction between components. First, we aggregate test paths between binary or analog sensor and controller resulting in 2 test paths. Then, these test paths will be aggregated with the actuator test path and that will result in 2 test paths. Finally, we have total of 4 aggregated test paths of all the device components. Table 4.6 shows all the aggregated test paths for the CEFSM model for a single device.

Choose Input Domains

After generating a set of general test paths from a reusable test-ready model of the devices, we will select specific inputs for a specific device to replace the transitions definitions in the test sequence constructed in the previous step. The test values are selected by the test designer. For example, the Ecobee Smart Thermostat device modeled in Fig 3.7 consist of sensor, controller and actuator.

The test paths generated form that model are presented in Tables 4.1 4.3 and 4.4 and the aggregated test paths are showed in Table 4.6. To illustrate the input selection process we will choice ATP_1 from Table 4.6. Table 4.7 shows an example of a thermostat aggregate test path with the input values. The first column shows the transition id, the second column shows the input constraints, the third column shows the possible values and finally the last column describe the transition. For example, T_1 in smart thermostat is the transition between the two states in the sensor (SIdle (S_1) to SActive (S_2)). The input constraints to this transition are R(VC) or R(TS) or R(Temp) or R(Occupied), which mean that it is required to receive one or more of these data to switch from SIdle to SActive. The user command the Thermostat to increase the temperature verbally by saying "Increase Temperature" or by touch the screen of the thermostat clockwise or by the receiving the sensed temperature or occupied. Possible inputs value for these are: verbal command "Increase Temperature" or "Decrease Temperature" to VC. Touch screen clockwise or counterclockwise to TS. For temperature range could vary between 60° to 80°. Finally, occupied as true or false.

Create Executable Tests

The resulting sequences of test inputs can be made executable by transforming them using MATTER, a tool for generating end-to-end IoT test scripts [84]. The MATTER tool has been experimentally evaluated with two different IoT systems. To verify the effectiveness of the generated test suites in discovering defects, the MATTER tool has been used an ad-hoc validation framework based on Mutation testing [81]. Moreover, the time it takes to run the created test suites was also taken into account in the evaluation.

T ID	Constraints	Values	Explain
T_1	R(VC) OR R(TS) OR R(Temp) OR R(Occupied)	VC= "Increase Temperature" OR TS= ↻ OR Temp= 70° OR Occupied= False	The user command the Thermostat to increase the temperature verbally OR by touch the screen clockwise to increase the temperature OR by the sensed temperature reached below the TThreshold OR the occupancy sensor sense that room is not occupied.
T_2	-	-	This is an extra transition
M_{11}	R(m11)	m_{11} == (Temp= 70°, Occupied= False	Every 15 second the temperature sensor will send the sensed temperature to the controller and every 5 minutes the occupied sensor will send the data to the controller [77]
T_3	R(VC) OR R(TS) OR R(Temp) OR R(Occupied)	VC= "Increase Temperature" OR TS= ↻ OR Temp= 70° OR Occupied= False	Once the data received from the sensor CReady will send them to Check state
T_4	R(VC == "Increase Temperature") OR R(TS == ↻) OR R(Temp $\leq$ 70°) OR R(Occupied == False)	VC= "Increase Temperature" OR TS= ↻ OR Temp= 69° OR Occupied= False	We need to compare the received data with the defined Threshold
T_6	-	-	This is an extra transition
T_7	R(FD)	FD= Increase Temperature	The controller final decision to the actuator to increase the temperature

Table (4.7) ATP_1 with input values

Execute and Evaluate

Finally the executable test paths generated from the previous step can be executed and the result can be evaluated. The execution can be done manually or using any testbed or simulators. For example, the intelligent home project (IHome) [69] or open smart home simulator (OpenSHS) [12]. The results should be presented in a form of validation report to document the result of the test execution.

4.2.2 System Level Testing

Generate Test Paths

As a first step in testing the Mobile App we need to generate test paths. A test path is a sequence of transitions through the aggregate FSM and through each lower level FSM. FSMApp's [10] first generates test paths through each FSM based on edge coverage criteria [13]. We generate the test sequences for each cluster that satisfy edge coverage. Tables 4.8 to 4.15 show test paths for each cluster as sequences of nodes. The corresponding graphs are mentioned in the caption. Nodes in bold indicate the node is a cluster node.

Test Path ID	**Test Path**	**Length**
1	[Home – **Control** – Home – Exit]	4

Table (4.8) Home Page Test Path of Fig 3.12

Aggregate Test Paths

At this step we will use the generated test paths from each FSM and aggregate them into test sequences for the whole model. In [13] a number of aggregation criteria have been proposed such as, all-combinations, each choice and base choice

Test Path ID	**Test Path**	**Length**
1	[Control – Control Thermostat – **Mode** – Control Thermostat – Main Exit]	5
2	[Control – Control Thermostat – **Eco** – Control Thermostat – Main Exit]	5
3	[Control – Control Thermostat – **Fan** – Control Thermostat – Main Exit]	5
4	[Control – Control Thermostat – **History** – Control Thermostat – Main Exit]	5
5	[Control – Control Thermostat – **Schedule** – Control Thermostat – Main Exit]	5

Table (4.9) Control Thermostat Page Test Paths of Fig 3.13

Test Path ID	**Test Path**	**Length**
1	[Mode – Select Mode – Mode – Main Exit]	4
2	[Mode – Select Mode – cancel – Main Exit]	4

Table (4.10) Mode Page Test Paths of Fig 3.14

Test Path ID	**Test Path**	**Length**
1	[Eco – Select Eco Mode – Eco – Main Exit]	4
2	[Eco – Select Eco Mode – cancel – Main Exit]	4

Table (4.11) Eco Mode Page Test Paths of Fig 3.15

Test Path ID	**Test Path**	**Length**
1	[Fan – Select Fan Speed – Fan – Main Exit]	4
2	[Fan – Select Fan Speed – cancel – Main Exit]	4

Table (4.12) Fan Page Test Paths of Fig 3.16

Test Path ID	**Test Path**	**Length**
1	[History – View History – History – Main Exit]	4
2	[History – View History – cancel – Main Exit]	4

Table (4.13) History Page Test Paths of Fig 3.17

Test Path ID	**Test Path**	**Length**
1	[Schedule – **Set up Schedule** – Schedule – Main Exit]	4
2	[Schedule – **Set up Schedule** – cancel – Main Exit]	4

Table (4.14) Schedule Page Test Paths of Fig 3.18

Test Path ID	**Test Path**	**Length**
1	[Set up Schedule – Update – Set up Schedule – Main Exit]	4
2	[Set up Schedule – Update – cancel – Main Exit]	4

Table (4.15) Set Up Schedule Page Test Paths of Fig 3.19

coverage. In our proposed approach we will apply all-combinations coverage despite the fact that this is the most expensive aggregation coverage criteria. The input of this step is the generated test paths and the output is a set of aggregated paths. To show the process of aggregation procedure we adopted Algorithm 1 in FSMApps approach [10]. It takes AFSMs test paths from previous Phase1 as an input and output a set of aggregated test paths. The algorithm starts by saving AFSM test paths into a new inputList. Line 2 loops through every test path in the inputList. Line 3 takes one test path from the list. Then, we iterate through the nods in the currentPath to check if the node is cluster node. If we found a cluster node, then a new loop changes the cluster node by all the cluster paths to create new partially aggregates.

Algorithm 1 Aggregated Test Paths

```
Input : AFMS and Cluster Test Paths
Result: outputList = Set of Aggregated Paths
inputList = AFSM paths
 while inputList has next path do
    currentPath = get one path from inputList
     pathDone = true
     for i = 1 to Length(currentPath) do
        if node_i is cluster node then
            for j = 1 to Length(Cluster paths) do
                replace node_i with cluster path_j and add new path into inputList
            end
            add list paths to inputList
             remove currentPath from inputList
             i = length (currentPath) +1
             pathDone = false
        end
    end
    if pathDone is true then
        Move currentPath into outputList
    end
end
```

For example, we can aggregate the test path in the top level model AFSM [Home - **Control** - Home - Exit] in Table 4.8 as follows: The bold node (Control) in the test path is a cluster node. We should replace this node by the Control Thermostat test paths from Table 4.9. This will result in new five test paths:

1. Home - Control - Control Thermostat - **Mode** - Control Thermostat - Main Exit - Home - Exit
2. Home - Control – Control Thermostat – **Eco** – Control Thermostat – Main Exit - Home - Exit
3. Home - Control – Control Thermostat – **Fan** – Control Thermostat – Main Exit - Home - Exit
4. Home - Control – Control Thermostat – **History** – Control Thermostat – Main Exit - Home - Exit
5. Home - Control – Control Thermostat – **Schedule** – Control Thermostat – Main Exit - Home - Exit

The first test sill has one cluster node (Mode). While second test has (Eco) cluster node. The third test has (Fan) cluster node. Forth test path has (History) cluster node and the last test has (Schedule) cluster node. We will replace these cluster nodes with the desired tests paths and the result will be as following:

1. Home - Control – Control Thermostat – Mode – Select Mode – Mode – Main Exit – Control Thermostat – Main Exit - Home - Exit
2. Home - Control – Control Thermostat – Mode – Select Mode – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

3. Home - Control – Control Thermostat – Eco – Select Eco Mode – Eco – Main Exit – Control Thermostat – Main Exit - Home - Exit

4. Home - Control – Control Thermostat – Eco – Select Eco Mode – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

5. Home - Control – Control Thermostat – Fan – Select Fan Speed – Fan – Main Exit – Control Thermostat – Main Exit - Home - Exit

6. Home - Control – Control Thermostat – Fan – Select Fan Speed – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

7. Home - Control – Control Thermostat – History – View History – History – Main Exit – Control Thermostat – Main Exit - Home - Exit

8. Home - Control – Control Thermostat – History – View History – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

9. Home - Control – Control Thermostat – Schedule – **Set up Schedule** – Schedule – Main Exit – Control Thermostat – Main Exit - Home - Exit

10. Home - Control – Control Thermostat – Schedule – **Set up Schedule** – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

There is still one cluster node in test 9 and 10 and after replacing them we got:

1. Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – Set up Schedule – Main Exit – Schedule – Main Exit – Control Thermostat – Main Exit - Home - Exit

2. Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – cancel – Main Exit – Schedule – Main Exit – Control Thermostat – Main Exit - Home - Exit

3. Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – Set up Schedule – Main Exit – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

4. Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – cancel – Main Exit – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit

Table 4.16 shows the aggregated test paths of the control thermostat in the Nest App as sequences of nodes. The first column shows the test path id. The second column shows the aggregated abstract test paths. The last column shows the length of each test path in terms of number of nodes in that path. The length of all the test paths in control thermostat is 144 nodes. Paths 9, 10, 11 and 12 is the longest paths with 14 nodes, and the rest are the shortest with 11 nodes.

Choose Input Domains

To convert the abstract test paths generated in the previous step into executable test, we should select inputs. At this step, test designers are free in how to select test values. They could use specific input domain covering partitions or select input values randomly from a possible list of inputs. The test designer chooses values for related inputs. As a result of this step, we will have a set of executable tests paths. Table 4.17 shows the set of inputs for test path 6. The first column shows the transition id, the second column shows the constraint sequence, and the third column shows the input values that meet the constraints. The last column has explanation for each value.

Test 6 in Table 4.16 has no input and eight actions. The actions are select fan speed from list of choices, click manage products button (ManagePbutton), click

ID	Abstract Test Path	Length
1	[Home - Control – Control Thermostat – Mode – Select Mode – Mode – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
2	[Home - Control – Control Thermostat – Mode – Select Mode – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
3	[Home - Control – Control Thermostat – Eco – Select Eco Mode – Eco – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
4	[Home - Control – Control Thermostat – Eco – Select Eco Mode – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
5	[Home - Control – Control Thermostat – Fan – Select Fan Speed – Fan – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
6	[Home - Control – Control Thermostat – Fan – Select Fan Speed – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
7	[Home - Control – Control Thermostat – History – View History – History – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
8	[Home - Control – Control Thermostat – History – View History – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit]	11
9	[Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – Set up Schedule – Main Exit – Schedule – Main Exit – Control Thermostat – Main Exit - Home - Exit]	14
10	[Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – cancel – Main Exit – Schedule – Main Exit – Control Thermostat – Main Exit - Home - Exit]	14
11	[Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – Set up Schedule – Main Exit – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit]	14
12	[Home - Control – Control Thermostat – Schedule – Set up Schedule – Update – cancel – Main Exit – cancel – Main Exit – Control Thermostat – Main Exit - Home - Exit]	14

Table (4.16) Aggregated Abstract Test Paths

T ID	Constraints	Values	Explain
A_1	R(ManagePbutton)	ManagePbutton=click	Click on manage product button to access manage products screen
B_1	R(ManageTbutton)	ManageTbutton=click	Click on manage thermostat button to access manage thermostat screen
B_4 H_1	R(Fanbutton) C1(High, Medium, Low)	Fanbutton=click High	Click on fan button Select fan speed from multiple choices to change the fan speed
H_3	S ((O(C1(High ,Medium, Low))), R (Cancelbutton))	High Cancelbutton=click	Select fan speed from multiple choices then click on the cancel button or just click on the cancel button to cancel and ignore the changes
H_4	R(Backbutton)	Backbutton=click	Click on back button to return to previous screen
B_7	none	none	Back to previous screen without any input constraint
B_8	R(Backbutton)	Backbutton=click	Click on back button to return to previous screen
A_3	none	none	Exit the app without any input constraint

Table (4.17) Aggregated Test Path 6 with input values

manage thermostat button (ManageTbutton), click fan button (Fanbutton), click cancel button (Canclebutton) to cancel and click back arrow to return to previous screen and that used twice (buttonBack).

Execute and Validate

We can use any Mobile App automatic tool to run the generated test cases. Selenium [2] is an open source to test Web and Mobile application. Selenium provides test domain-specific languages such as Java, Python, C, and PHP for writing tests. Selenium is compatible with Windows, Linux, and Mac. Therefore we can use Selenium to execute the tests and generate a reports of the results.

4.2.3 Interaction Testing

This contribution is published in the 2021 International Conference on Computational Science and Computational Intelligence, (Albahli et al. [8]. Interaction can be done on two levels: Devices Interaction or Device-Mobile App Interaction and in both cases we will generate test paths and aggregate them.

Devices Interaction

Fig 3.21 shows the model of devices interaction and Table 3.24 lists and explains the transitions. We will generate tests from this model using edge coverage criteria [13]. Table 4.18 shows the test paths generated from Fig 3.21. Device_1 and Device_2 consist of sensors, controllers and actuators and each device will be replaced with the test paths generated as in Section 4.2.1. The only difference is that we will consider the input constraints for this interaction, by sending the status of each device to the other device via message on the transition.

Transition ID	Test Path
DI_1	[$Device_1$, $Device_2$, $Device_1$]

Table (4.18) Test path for interactions between devices

Device and Mobile App Interaction

Fig 3.22 shows the model of device and Mobile App interaction and Table 3.25 lists and explains the transitions. We will generate test from this model using edge coverage criteria [13]. Table 4.19 shows the test path generated from Fig 3.22. $Device_1$ consists of sensor, controller and actuator and each device will be replaced with the test paths generated as in Section 4.2.1. While the Mobile App is explained in Section 4.2.2, the only difference is that we will consider the input constraints for this interaction.

Transition ID	Test Path
DI_1	[$Device_1$, App_1, $Device_1$]

Table (4.19) Test path for interactions between device and Mobile App

Chapter 5

Smart Home Systems Evolution

5.1 Problem Statement

When a software is evolved, regression testing ensures that new and modified features work properly and that existing features continue to function as expected. The retest all strategy will require retesting the entire system, anticipating that modifications could have affected and produced faults at any possible point in the system. On the other hand, a selective regression testing technique implies that only certain areas of the software were affected by the changes. The following is a typical selective regression test procedure [93]:

- Recognize system changes.
- Determine whether old test cases, T will be valid for the new software version. We get T' from this set by removing all tests that are no longer applicable.
- T' should be used to test the updated software.
- Create new set of test cases, T" to test aspects of the product that aren't fully covered by T'.

- Use T" to run the updated program.

5.2 Evolution in Smart Home System:

5.2.1 Recognize System Changes

Table 5.1 shows the possible evolution in smart home system. SHS can evolve by adding new device into the system, updating an existing with newer version or different brand or removing a device. Each one of these changes can happen to devices with or without interactions with other devices. Changes also can be classified as changes in the Mobile App or local in the device. This results in 12 possible evolution in total. We will address each change and show the changes in models and test paths. All possible changes are classified as:

Type of Evolution	**Device Interaction**	**Changes Location**
Add New Device Modify Device Remove Device	Without Device Interaction With Device Interaction	Mobile App Local

Table (5.1) Evolution in SHS

1. Add a device.
   (a) The new device does not interact with another device in SHS:
      i. The new device needs a new mobile app.
      ii. The new device will use existing app (with modification).
   (b) The new device interacts with another device in SHS:
      i. The new device needs a new mobile app.
      ii. The new device will use existing app (with modification).

2. Modify a device.

    (a) The modified device does not interact with another device in SHS:

        i. The modified device uses a stand-alone mobile app.

        ii. The modified device uses an existing app that control other devices.

    (b) The modified device interacts with another device in SHS:

        i. The modified device uses a stand-alone mobile app.

        ii. The modified device uses an existing app that control other devices.

3. Remove a device.

    (a) The removed device does not interact with another device in SHS:

        i. The removed device uses a stand-alone mobile app.

        ii. The removed device uses an existing app that control other devices.

    (b) The removed device interacts with another device in SHS:

        i. The removed device uses a stand-alone mobile app.

        ii. The removed device uses an existing app that control other devices.

To illustrate this evolution and show how to efficiently and easily create models and test artifacts when technology changes, we will build a smart home model of one device. Then we will handle each case and evaluate how easily these changes can be done using the proposed approach. We will show how these changes affect the current models and the generated test paths. We will apply a selective black-box model-based regression testing based on the change. Existing tests and test requirements are classified as reusable, retestable, or obsolete [18, 20, 21]. We demonstrate how obsolete tests can be corrected by partial regeneration, as well as how more tests are generated to match coverage requirements of the modified model. We do not

have to distinguish between retestable tests (test that must rerun) and reusable tests (tests that do not have to be rerun) when we adapt an existing set of models and tests, because these tests are still working and need to be executed. Andrews et al. [19], proposed a cost-benifit trade off framework for FSMWeb models that helps choose between selective and brute force regression testing. We will adopt this framework with modification since our models is different form FSMWeb and they have interaction between models. Moreover, in our proposed models, changes are only on transitions information not in the models themselves. Issues of using [19] would mean simply no obsolete tests, if transition change means adding new conditions or inputs. Moreover, new property would not be tested at all. Therefore, we proposed general test rules for our models by first defining model changes and then classifying tests after the changes:

Model changes:

1. Node change: add/remove a device will require add/remove nodes.

2. Transition change: modify in/out transition (modify transition inputs and/or conditions), add in/out transition with or without a new node, and remove in/out transition with or without a new node.

5.2.2 Classification of tests after model changes:

Based on the changes in the models, tests can be classified as obsolete, reusable, or retestable.

Define T as the set of transitions t in a model and N as the set of nodes n. T' is then the set of modified transitions t' in a model and N' is the set of modified nodes n'. TP is the set of test paths tp.

Test classification rules:

- Obsolete Test Cases:

  Test become obsolete when a node or transition removed. Let $N_O = \{n|n \in N;$ n is removed $\}$ and the set of obsolete tests due to node changes becomes: $O_N = \{tp_a| \exists\ n \in N_O:\ tp_a$ visit n $\}$.

  Let $T_O = \{t|t \in T;$ t is removed$\}$ and the set of obsolete tests due to transition changes becomes: $O_T = \{tp_a\ |tp_a$ visit $t \in T_O\ \}$

  The entire obsolete test $O_{AT} = O_N \cup O_T$

- Retestable Test Cases:

  Tests become retestable when they are still valid and test portions of the system that has changes in the current model that are close to the changes. We need to define a coverage criterion to define which part of the model/test need to be retestable. When a transition t modified, we will assume the start and end nodes of the modified transition are affected. Therefore, the tests visited these nodes are retestable. Let $T_R = \{t|t \in T;$ t is changed $\}$ and the affected nodes $N_{tm} = \{n|t' \in T,\ t' = (n_s,\ n_e)\ \}$.

  The set of retestable tests due to transition changes is:

  $R_T = \{tp|\exists\ tp_a \in N_{tm} :\ tp_a$ visit t$\}$

- Reusable Test Cases:

  Reusable tests are neither obsolete nor retestable.

5.2.3 Examples of Smart Home Evolution

Case 1: Add New Device, Without device interaction, New Mobile App

Let's start with a smart home system with a single device, for instant, smart thermostat. We modeled this device in Chapter 3 and generate test paths from it in Chapter 4. Fig 5.1 shows the model of the smart thermostat. Table 5.2 shows the aggregated test paths generated from smart thermostat.

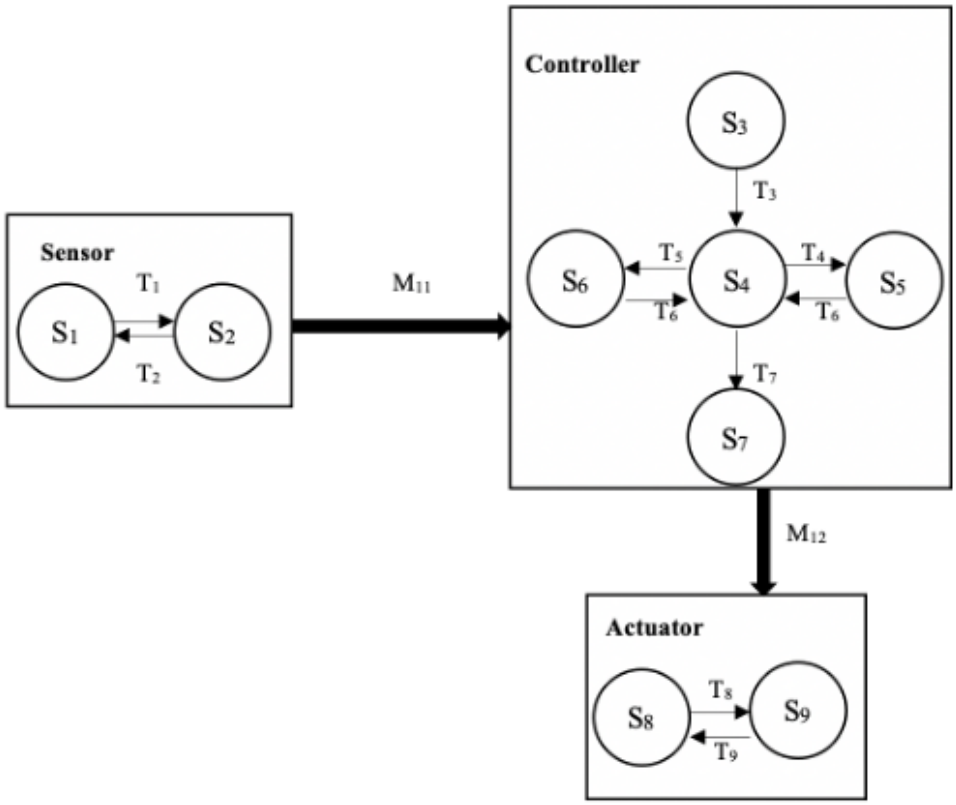

Figure (5.1) Smart Thermostat CEFSM Model

In this section we will add a new device that has no interactions with the existing smart home system. We will see how this change affect the current models and which test classification rules are needed.

Test Path ID	**Test Path**
ATP_1	$[S_1 \xrightarrow{T_1} S_2 \xrightarrow{T_2} S_1]$, $[S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_4} S_5 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7]$
ATP_2	$[S_1 \xrightarrow{T_1} S_2 \xrightarrow{T_2} S_1]$, $[S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_5} S_6 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7]$
ATP_3	$[S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_4} S_5 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7]$, $[S_8 \xrightarrow{T_8} S_9 \xrightarrow{T_9} S_8]$
ATP_4	$[S_3 \xrightarrow{T_3} S_4 \xrightarrow{T_5} S_6 \xrightarrow{T_6} S_4 \xrightarrow{T_7} S_7]$, $[S_8 \xrightarrow{T_8} S_9 \xrightarrow{T_9} S_8]$

Table (5.2) Aggregated Smart Thermostat Test Paths

- Model Changes

  Adding new device that has no interaction with the current system will be just use a model from a reusable test-ready models built it Chapter 3. For example, we decided to add Amazon Echo Dot to the smart home system. Amazon Echo Dot is modeled in fig 3.8 and Table 5.3 shows the aggregated test paths generated from that device.

Test Path ID	**Test Path**
ATP_5	$[S_1 \xrightarrow{T_{10}} S_2 \xrightarrow{T_{11}} S_1]$, $[S_3 \xrightarrow{T_{12}} S_4 \xrightarrow{T_{13}} S_5 \xrightarrow{T_{15}} S_4 \xrightarrow{T_{16}} S_7]$
ATP_6	$[S_1 \xrightarrow{T_{10}} S_2 \xrightarrow{T_{11}} S_1]$, $[S_3 \xrightarrow{T_{12}} S_4 \xrightarrow{T_{14}} S_6 \xrightarrow{T_{15}} S_4 \xrightarrow{T_{16}} S_7]$
ATP_7	$[S_3 \xrightarrow{T_{12}} S_4 \xrightarrow{T_{13}} S_5 \xrightarrow{T_{15}} S_4 \xrightarrow{T_{16}} S_7]$, $[S_8 \xrightarrow{T_{17}} S_9 \xrightarrow{T_{18}} S_8]$
ATP_8	$[S_3 \xrightarrow{T_{12}} S_4 \xrightarrow{T_{14}} S_6 \xrightarrow{T_{15}} S_4 \xrightarrow{T_{16}} S_7]$, $[S_8 \xrightarrow{T_{17}} S_9 \xrightarrow{T_{18}} S_8]$

Table (5.3) Aggregated Amazon Echo Dot Test Paths

Fig 5.2 shows the block diagram of the SHS with two devices and the interaction between the Mobile apps and the devices. D_1 is the existing device (Thermostat) and D_2 is the new added device (Amazon Echo Dot) and App_1 and App_2 are the two mobile applications that control them respectively. As we mentioned these devices have no interaction between them.

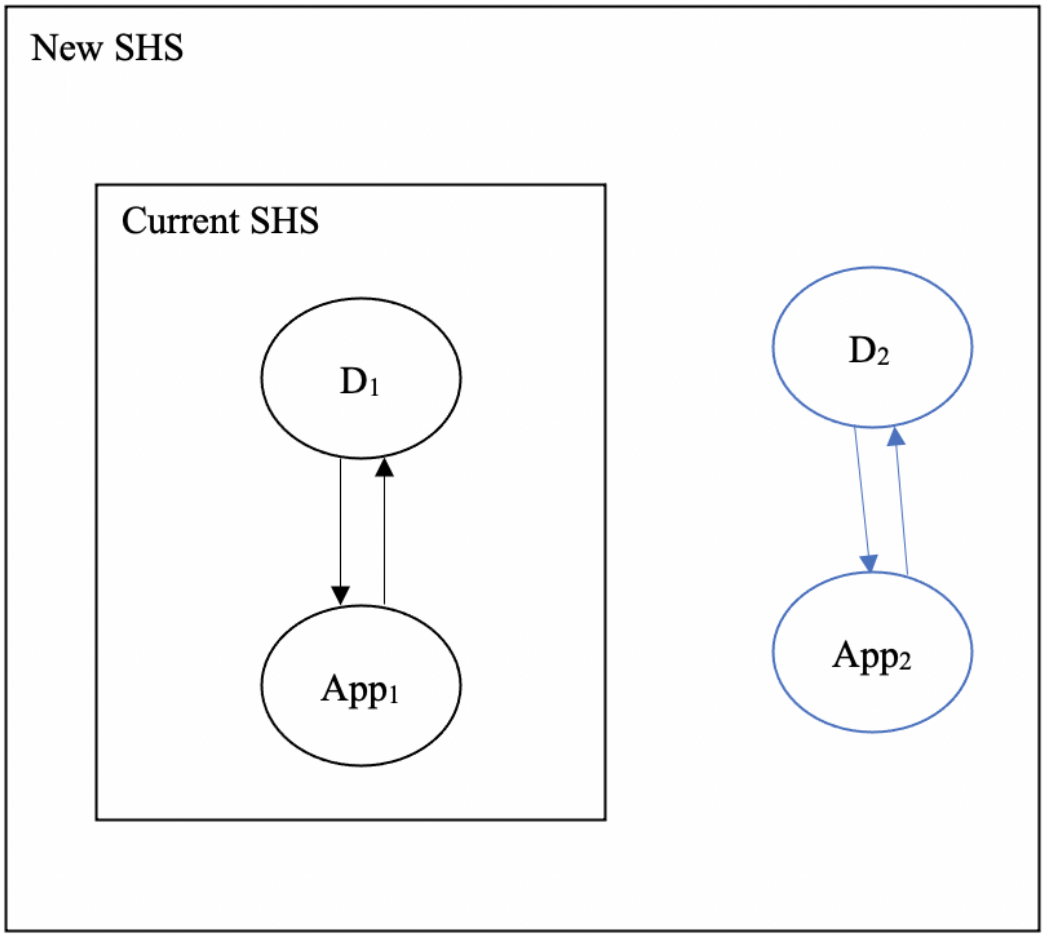

Figure (5.2) Block Diagram of SHS After Adding New Device Without Interactions (New App)

There are two models changes in this case:

1. Node change: Add new nodes for the new device and the new Mobile App.
2. Transition change: Add in/out transitions with new nodes.

- Classification of tests after model changes

  In this case we can keep the smart thermostat tests and we only need to add the test paths for the new Amazon Echo Dot. Therefore, we do not have

any obsolete or retestable test cases. The smart thermostat test cases are considered reusable test cases and the Amazon Echo Dot tests are new.

Case 2: Add New Device, Without device interaction, Use existing app

- Model Changes

  In this case we add a new devices that has no interaction with other devices in the current system, but it used an existing mobile app that controls another existing device. For example, two different Nest devices using the same mobile application to control them but there is no interaction between them.

  Fig 5.3 shows the block diagram of the SHS with two devices and the interaction between the Mobile apps and the devices. D_1 is the existing device and D_2 is the new added device and App_1 is the mobile application that controls both devices. As we mentioned, these devices have no interaction between them. Adding in model are in blue color modifications are in bold blue.

  There are two models changes in this case:

  1. Node Change: Add new nodes for device. Add new LAP and Cluster nodes within the current app to control the new device.
  2. Transition change: Add in/out transitions within the new nodes in device. Add in/out transitions within the current app. Modify in/out transitions (modify transition inputs and/or conditions) within the current app.

- Classification of tests after model changes

  In this case we can keep the current devices tests and we only need to add the test paths for the new device. As well as new tests for the new app screens that control the new device. We do not have any obsolete test cases, any change

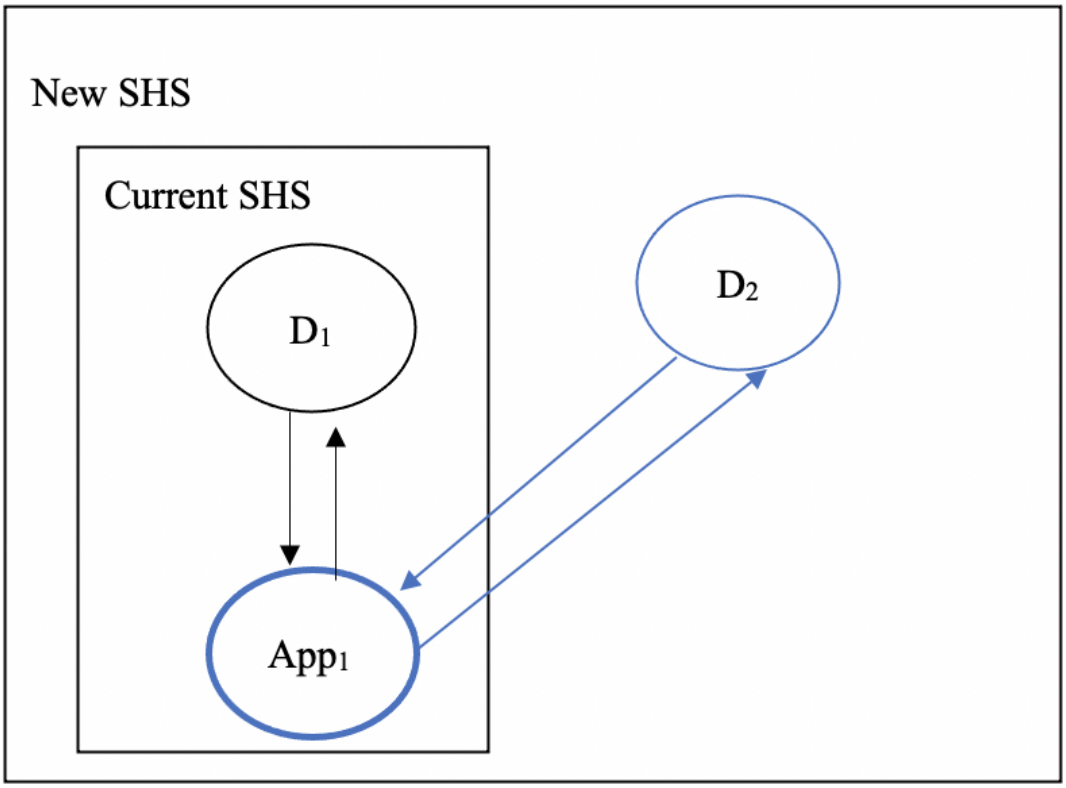

Figure (5.3) Block Diagram of SHS After Adding New Device Without Interactions (Modified App)

in the transition and the start and end nodes of the modified transition are affected and tests for these nodes are retestable, other tests are reusable.

Case 3: Add New Device, With device interaction, New Mobile App

- Model Changes

  In this case we add a new device that has interaction with other devices in the current system and it uses a new mobile app. For example, the current smart home system has two devices (Smart Thermostat and Amazon Echo Dot) we decided to add new device (Smart Lock) which possibly interact with Amazon Echo Dot and has its own Mobile App to control it. August Wi-Fi

Smart Lock is a smart lock that you can easily install to your front door and control it from a mobile application. Via the smart phone application you can control smart lock in your home while you are in home or remotely when you are away. You can automate lock after certain amount of time after door close, and unlock automatically when you arrive home. At any time, you can check your smart lock activity by one click on the smart phone application to see all activities listed by time. Fig 5.4 shows the block diagram of the SHS with three devices and the interaction between them. Also, the interaction between the Mobile apps and the devices. D_1 and D_2 are the existing devices (Thermostat and Amazon Echo Dot) respectively. D_3 is the new added device (Smart Lock). App_1, App_2 and App_3 are the three mobile applications that control these devices. As we mentioned D_2 and D_3 have interaction between them. All the adding in the system are in blue color and the modifications are in bold blue.

There are two models changes in this case:

1. Node Change: Add new nodes for the new device and new Mobile App.
2. Transition change: Add in/out transitions within the new nodes in device. Add in/out transitions within the new nodes in the new Mobile App. Add in/out transitions between the two devices for device interactions. Modify in/out transitions within the nodes in the current device and Mobile App.

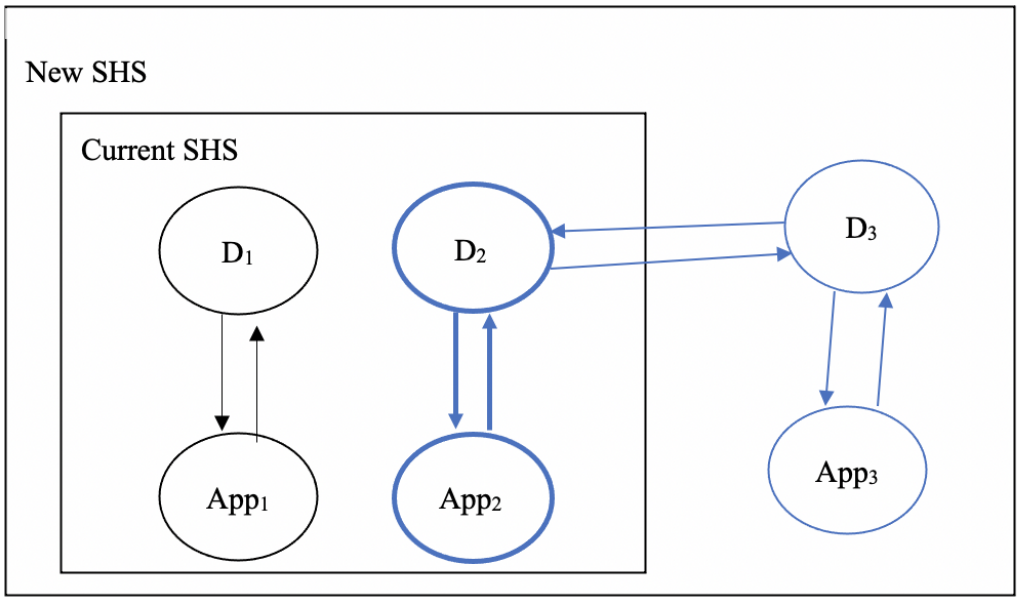

Figure (5.4) Block Diagram of SHS After Adding New Device With Interactions (New App)

- Classification of tests after model changes

  In this case we can keep the current device D_1 tests. We need to add the test paths for the new device D_3. As well as new tests for the new app App_3. We do not have any obsolete test cases. Due to adding new transitions between D_2 and D_3 we need to find all the affected transitions and the start and end nodes in D_2 and App_2 and tests for these nodes are retestable.

Case 4: Add New Device, With device interaction, Use existing app

- Model Changes

  In this case we add a new devices that has interaction with other device in the current system and it uses a current device app. For example, the current smart home system has two devices (Smart Thermostat and Amazon Echo Dot) we decided to add new device (Smart Lock) witch possibly interact with

Amazon Echo Dot and has its own Mobile App to control it but also you can control this device from Amazon Echo Dot App by adding this device in Alexa App. You can easily ask "Alexa, is my front door open?" and Alexa should check and answer your question. You can also ask Alexa to lock/unlock your door and this should reflect necessarily information on both devices and Apps.

Fig 5.5 shows the block diagram of the SHS with three devices and the interaction between them. Also, the interaction between the Mobile apps and the devices. D_1 and D_2 are the existing devices (Thermostat and Amazon Echo Dot) respectively. D_3 is the new added device (Smart Lock). App_1 and App_2 are the two current mobile applications that control the devices. As we mentioned D_2 and D_3 have interaction between them. D_3 can be also controlled via App_2. All the adding in the system are in blue color and the modifications are in bold blue.

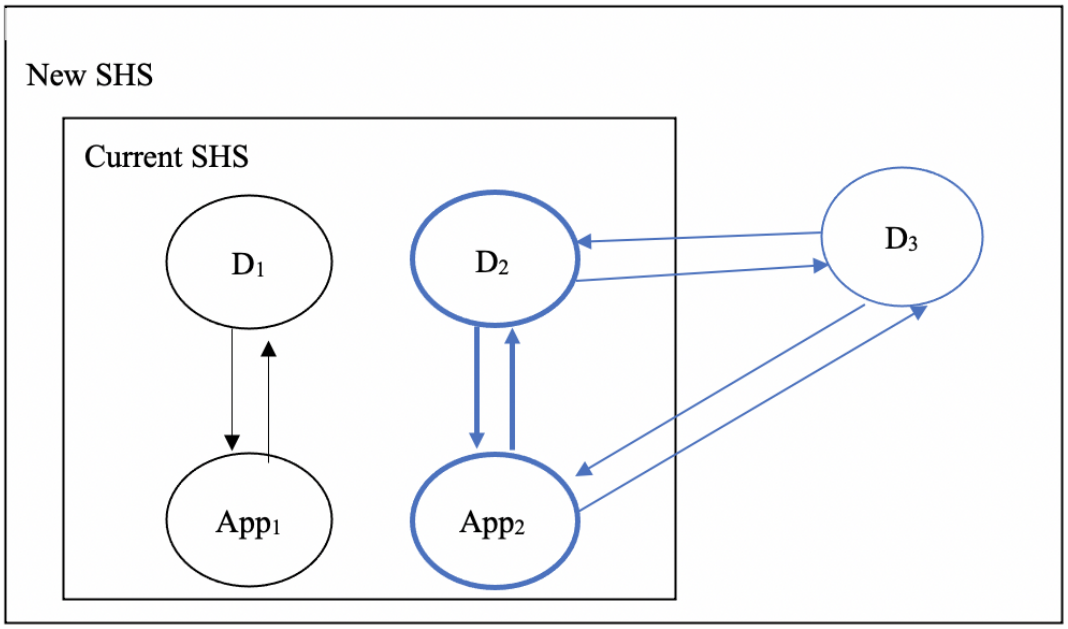

Figure (5.5) Block Diagram of SHS After Adding New Device With Interactions (Modified App)

There are two models changes in this case:

1. Node Change: Add new nodes for the new device. Add new LAP and Cluster nodes within the current app to control the new device.
2. Transition change: Add in/out transitions within the new nodes in device. Add in/out transitions between the two devices for device interactions. Modify in/out transitions (modify transition inputs and/or conditions) within the current app.

- Classification of tests after model changes

  In this case we can keep the current devices D_1, App_1 and some of D_2 tests. We need to add the test paths for the new device D_3. As well as new tests for the new app screens in App_2 that control the new device. We do not have any obsolete test cases. Due to adding new transitions between D_2 and D_3 we need to find all the affected transitions and the start and end nodes in D_2 and App_2 tests for these nodes are retestable.

Case 5: Modify a device, Without device interaction, Uses stand-alone App

In this case, the current smart home system has two devices not interacting; we decided to upgrade one of the devices to a newer version or use a different manufacturer. We will remove an existing device and replace it with a new device, or we will need just to update the transition information (inputs and/or conditions).

- Model Changes

  Fig 5.6 shows the block diagram of two devices D_1 not changes and D_2 is modified to newer version. Modifying a device that has no interaction with

the current system will be changing the inputs and/or the conditions on the transition. We do not need to change or add any nodes. For example, we decided to upgrade our Amazon Echo Dot to Amazon Echo Dot with clock. We will need to modify the inputs on some transitions on that device. All the modifications in the system are in blue color and bold.

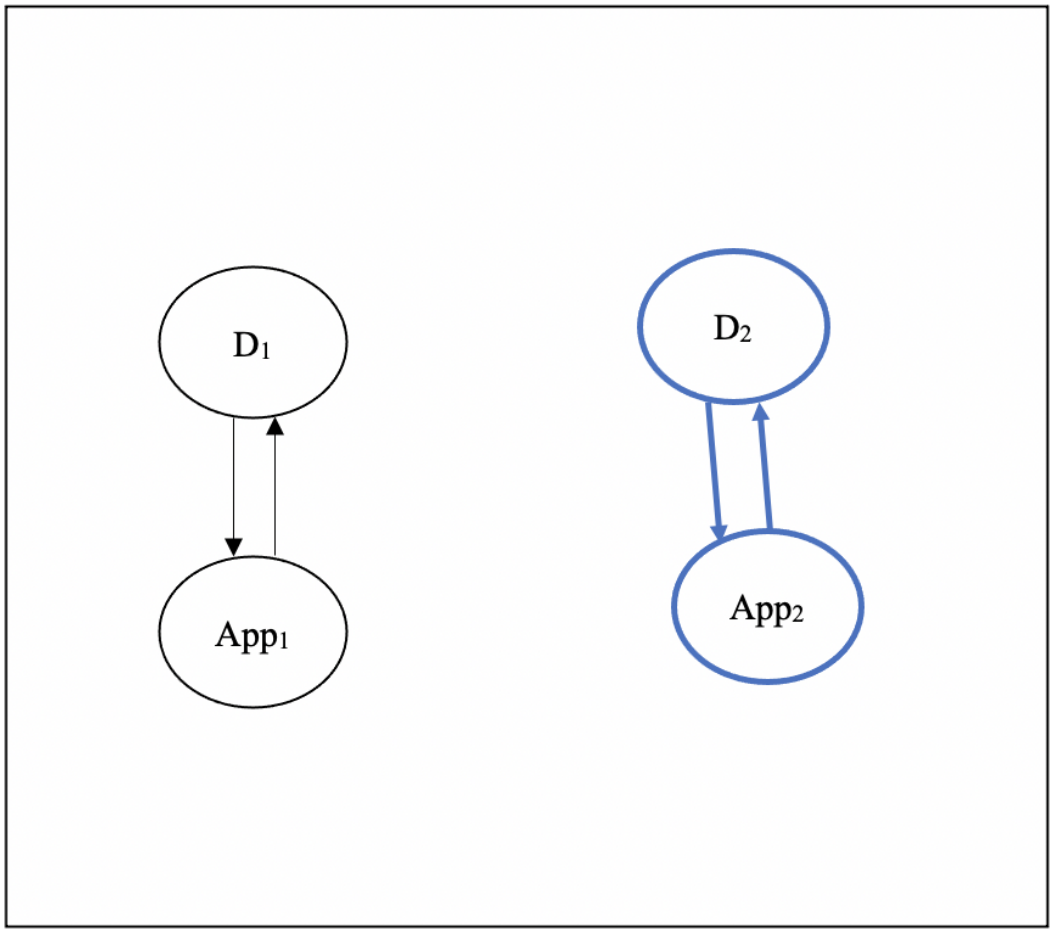

Figure (5.6) Block Diagram of SHS After Modifying a Device Without Interactions (Stand-alone App)

There are two models changes in this case:

1. Node change: No node change for the device. Add new LAP and Cluster nodes within the stand-alone app to control the new features in the modified device.
2. Transition change: Modify in/out transition with inputs and/or conditions in device and App.

- Classification of tests after model changes

  In this case we can keep the current devices tests and we only need to retest the retestable test cases which has the modified inputs/conditions transitions and the visited start and end nodes of that transition in device and App. We do not have any obsolete tests.

Case 6: Modify a Device, Without device interaction, Use existing app

- Model Changes

  In this case we modify a devices that has no interaction with other devices in the current system but it used an existing mobile app that control another existing device. For example, two different Nest devices using the same mobile application to control them but there is no interaction between them and you decide to upgrade one of them to newer version.

  Fig 5.7 shows the block diagram of the SHS with two devices and the interaction between the Mobile apps and the devices. D_1 is the existing device and D_2 is the modified device and App_1 is the mobile application that controls both devices. As we mentioned these devices have no interaction between them. All the modifications in the system are in blue color and bold.

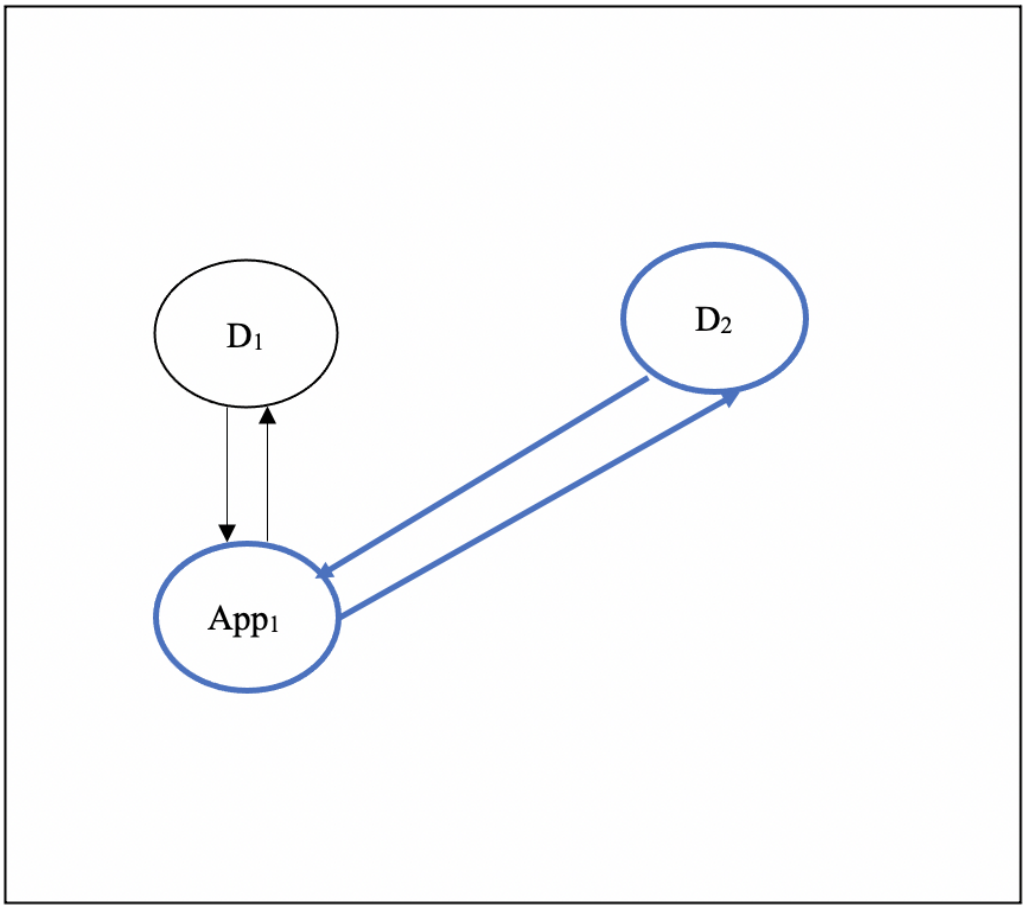

Figure (5.7) Block Diagram of SHS After Modifying a Device Without Interactions (Current App)

There are two models changes in this case:

1. Node Change: No new nodes for the device. Add new LAP and Cluster nodes within the current app to control the new device.
2. Transition change: Modify in/out transition with inputs and/or conditions in device and current App. Add in/out transition within the current app.

- Classification of tests after model changes

  In this case we can keep the current device tests and we only need to modify the test paths for the modified device. As well as new tests for the current app screens that control the new device. We do not have any obsolete test cases, any change in the transition and the start and end nodes of the modified transition are affected and tests for these nodes are retestable, other tests are reusable.

Case 7: Modify a device, With device interaction, Uses stand-alone App

If we have a current smart home system with two devices interact with each other then we decided to upgrade one of the devices to a newer version or use a different manufacture. We will just need to update the transition information (inputs and/or conditions) in the modified device. This will also affect the interaction between the two devices as well as the other some transitions in the other device.

- Model Changes

  Fig 5.8 shows the block diagram of two devices D_1 with no changes and D_2 with modifications. Modifying a device that has interaction with a current device will be changing the inputs and/or the conditions on the transitions of the modified device. We also might need to modify the interaction transitions, and some transitions in D_1. We do not need to change or add any nodes. All the changes are in the information on the transitions. The changes in the system are in blue color and bold.

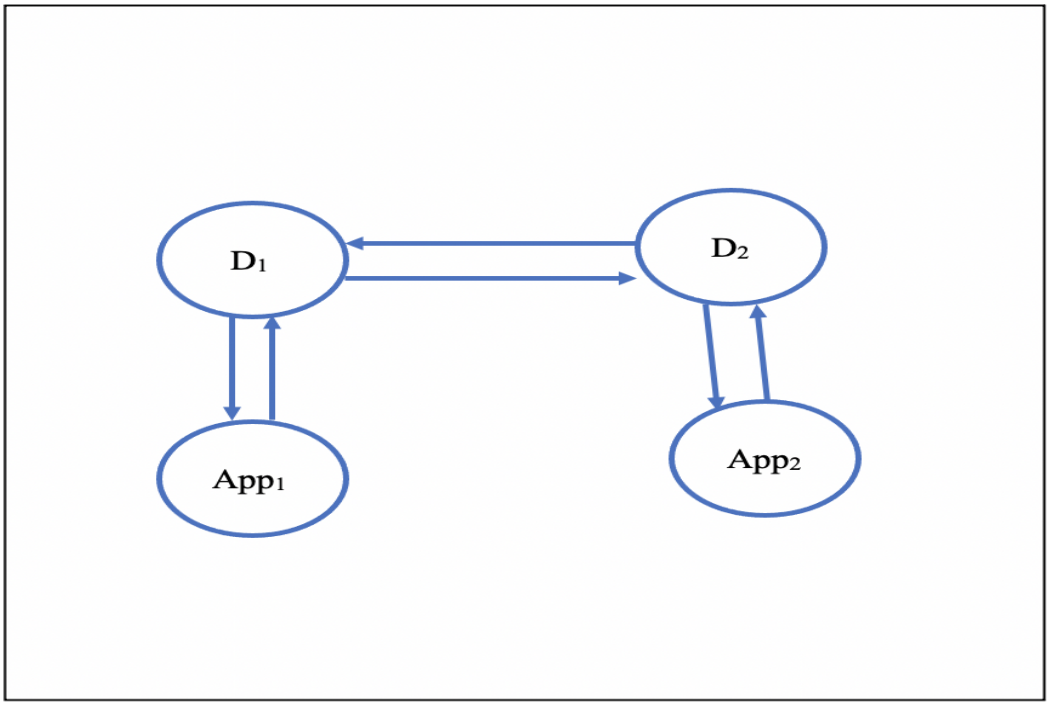

Figure (5.8) Block Diagram of SHS After Modifying a Device With Interactions (Stand-alone App)

There are two models changes in this case:

1. Node change: No node change for the device. Add new LAP and Cluster nodes within the stand-alone app to control the new features in the modified device.

2. Transition change: Modify in/out transition with inputs and/or conditions in the modified device, App of the modified device, and interaction between the devices. Add in/out transition with inputs and/or conditions in the new App screens. Modify in/out transition with inputs and/or conditions in the unmodified device and App.

- Classification of tests after model changes

  In this case devices, Apps test paths are retestable for any change in the transition and start and end nodes. Interaction test paths also retestable. There are no obsolete tests.

Case 8: Modify a Device, With device interaction, Use existing app

- Model Changes

  In this case we modify a devices that has interaction with other devices in the current system but it used an existing mobile app that control another existing device. Fig 5.9 shows the block diagram of the SHS with two devices and the interaction between the Mobile apps and the devices. D_1 is the existing device and D_2 is the modified device and App_1 is the mobile application that controls both devices. As we mentioned these devices have interaction between them. All the changes in the system are in blue color and bold.

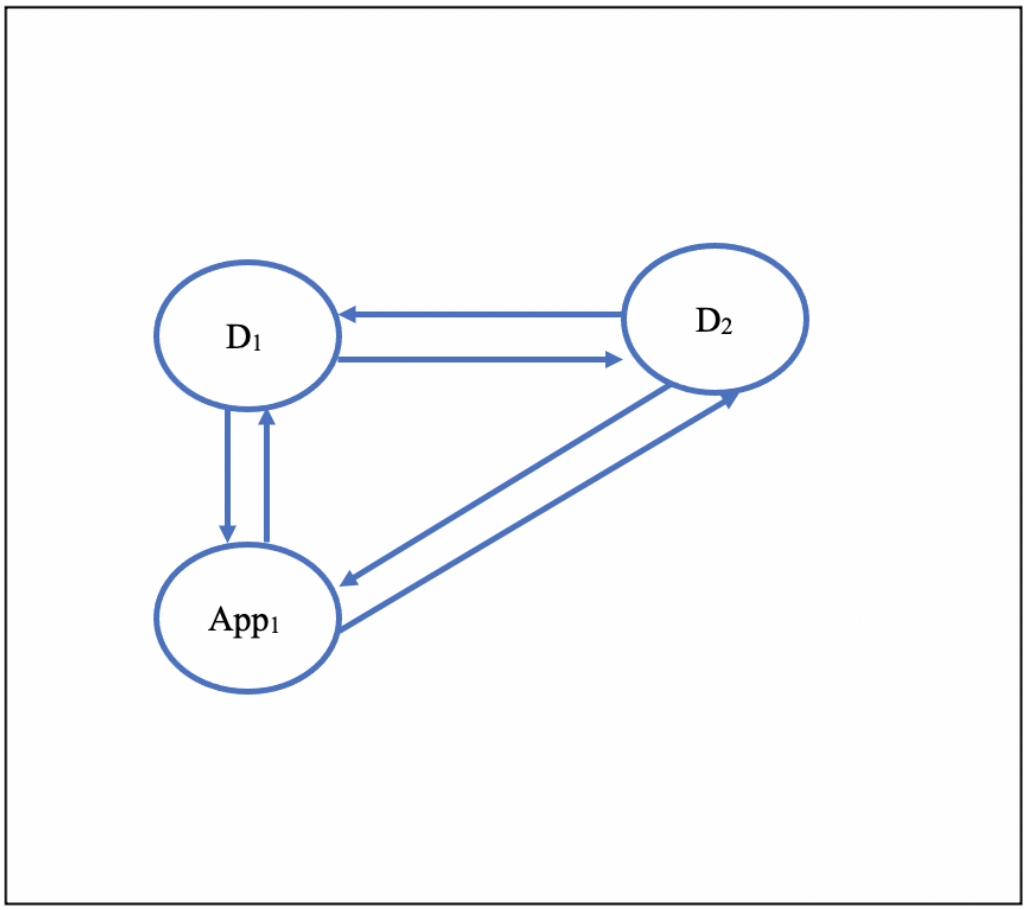

Figure (5.9) Block Diagram of SHS After Modifying a Device With Interactions (Current App)

There are two models changes in this case:

1. Node Change: No new nodes for the device. Add new LAP and Cluster nodes within the current app to control the new device.
2. Transition change: Modify in/out transition with inputs and/or conditions in devices, current App and interaction between the devices. Add in/out transition within the current app.

- Classification of tests after model changes

  In this case we need to modify the test paths for the modified device, modified app and interaction between devices. As well as new tests for the current app screens that control the new device must be added. We do not have any obsolete test cases, any change in the transition and the start and end nodes of the modified transition are affected and tests for these nodes are retestable, other tests are reusable.

Case 9: Remove a device, Without device interaction, Uses stand-alone App

In this case, we have a current smart home system with two devices not interact with each other then we decided to remove one of the devices. We will remove an existing device and all the test paths related to that device. Other parts of the system are not affected.

- Model Changes

  Fig 5.10 shows the block diagram of two devices D_1 and D_2 and we decided to remove D_2 with the application control it. Changes with the red cross represent removed parts.

  There are two model changes in this case:

  1. Node change: Remove all of the removed device nodes and the application control that device nodes.
  2. Transition change: Remove in/out transition in the removed device and App.

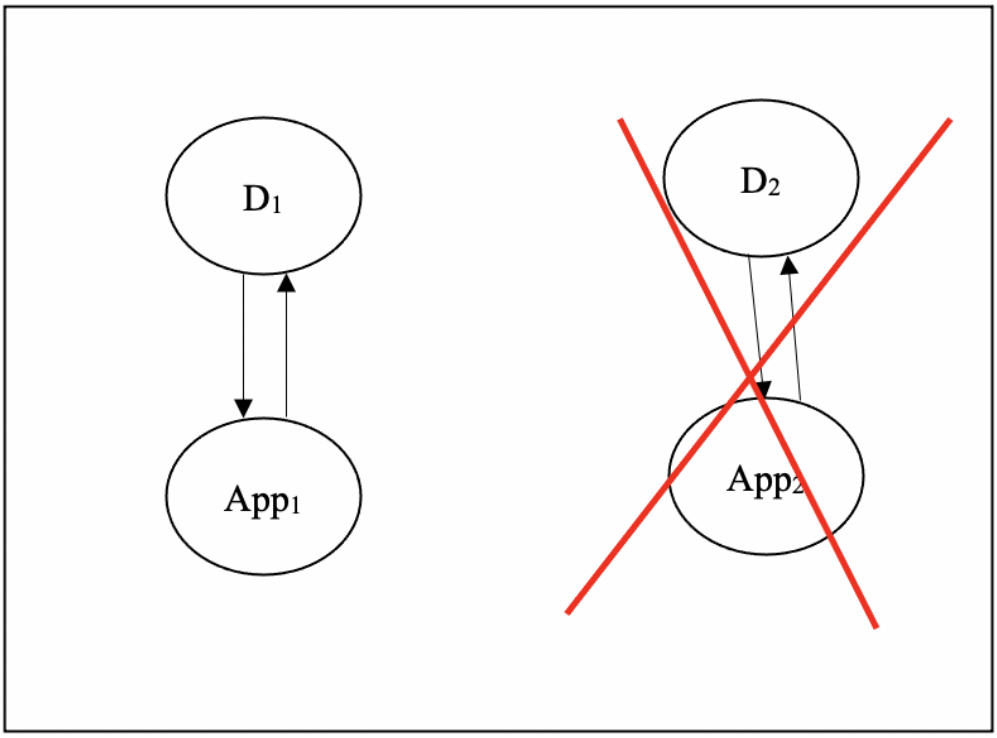

Figure (5.10) Block Diagram of SHS After Removing a Device Without Interactions (Stand-alone App)

- Classification of tests after model changes

  In this case we can keep the current device and application tests. We only need to consider the deleted device and application test paths are obsolete tests. No retestable test cases.

Case 10: Remove a Device, Without device interaction, Use existing app

- Model Changes

  In this case we remove a devices that has no interaction with other devices in the current system but it used an existing mobile app that control another existing device.

Fig 5.13 shows the block diagram of the SHS with two devices and the interaction between the Mobile apps and the devices. D_1 is the existing device and D_2 is the removed device and App_1 is the mobile application that controls both devices. As we mentioned these devices have no interaction between them. All the changes in the system are in blue color and bold. Red color is for removed items.

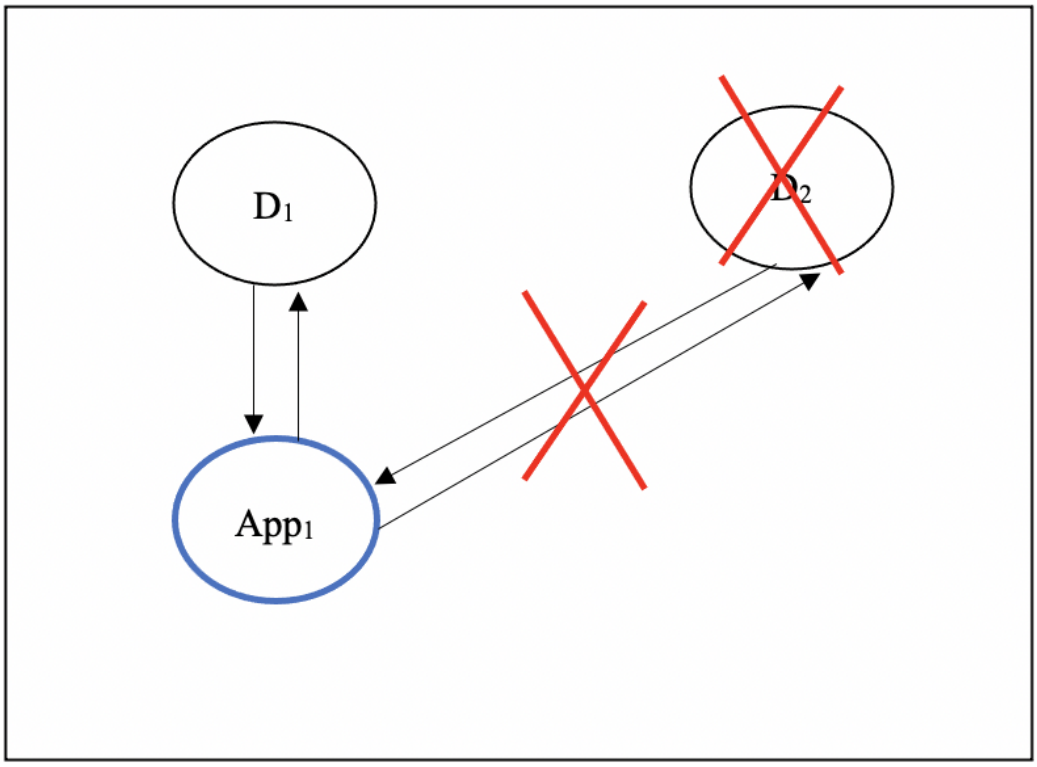

Figure (5.11) Block Diagram of SHS After Removing a Device Without Interactions (Current App)

There are two models changes in this case:

1. Node Change: Remove all of the removed device nodes. Remove some LAP and Cluster nodes within the current app that control the removed device.

2. Transition change: Remove in/out transition in the removed device. Remove in/out transition within the current app and between the application and the removed device. Remove in/out transition between the removed device and the application control it.

- Classification of tests after model changes

  In this case we only need to remove the test paths for the removed device. We also need to remove some test paths for the current app that control the removed device. All the removed test paths are obsolete test cases. We need to retest the application, the current device and interaction between them to make sure that none of the tests affected by removing the removed device controlling pages.

Case 11: Remove a device, With device interaction, Uses stand-alone App

If we have a current smart home system with two devices interact with each other then we decided to remove one of the devices. In this case we will remove an existing device and all the test paths related to that device. We need to retest the existing device or part of it because it might be affected by removing the interaction transitions.

- Model Changes

  Fig 5.12 shows the block diagram of two devices D_1 and D_2 and the interaction between them. We decided to remove D_2 with the application control it. As well as the interaction transitions between the two devices. Removed tests have red crosses and modified part of the system are shown in bold blue color.

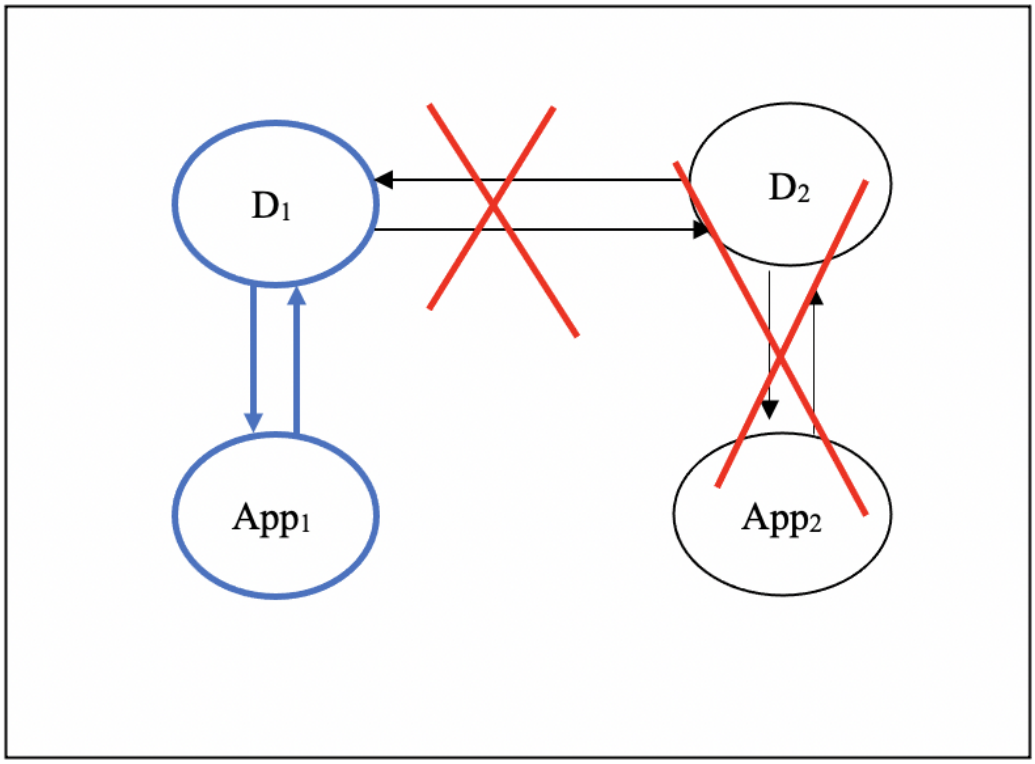

Figure (5.12) Block Diagram of SHS After Removing a Device With Interactions (Stand-alone App)

There are two models changes in this case:

1. Node change: Remove all of the removed device and application nodes.
2. Transition change: Remove in/out transition in the removed device and App. Remove in/out transition for the interaction.

- Classification of tests after model changes

  The current device D_1 and its controller App_1 are affected by removing D_2 nodes therefore their tests are retestable. We need to consider the deleted device and application test paths as obsolete tests.

Case 12: Remove a Device, With device interaction, Use existing app

- Model Changes

  In this case we remove a device that interact with other device in the current system but it used an existing mobile app that control another existing device.

  Fig 5.13 shows the block diagram of the SHS with two devices and the interaction between the Mobile apps and the devices. D_1 is the existing device and D_2 is the removed device and App_1 is the mobile application that controls both devices. As we mentioned these devices interact with each other. All the modification in the system are in blue color and bold. Red cross is for removed items.

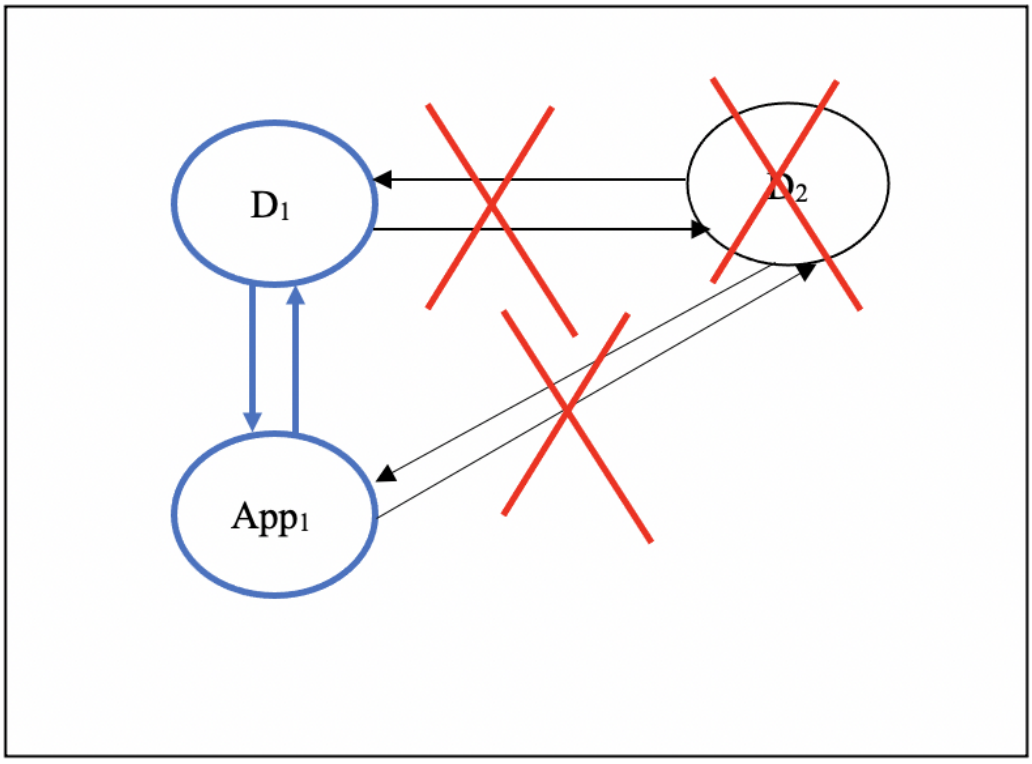

Figure (5.13) Block Diagram of SHS After Removing a Device With Interactions (Current App)

There are two models changes in this case:

1. Node Change: Remove all of the removed device nodes. Remove some LAP and Cluster nodes within the current app that control the removed device.
2. Transition change: Remove in/out transition in the deleted device. Remove in/out transition within the current app and between the application and the removed device. Remove in/out transition between the two devices. Modify in/out transition in the current device and App.

- Classification of tests after model changes

  In this case we need to remove the test paths for the removed device. As well as removing tests for the current app that control the removed device. Also, we remove the test paths between the two devices. All the removed test paths are obsolete test cases. We need to retest the application to make sure that non of the tests affected by removing the removed device controlling pages.

Table 5.4 summarizes the 12 cases of SHS evolution. The first column represents the case number, the second column has the old system components, third column shows the new system components, forth column has the model changes, fifth column shows the transition changes, and finally the last column represents the test classifications.

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
Add New Device without Interaction Between Devices					
1	D_1, App_1	D_1, App_1, D_2,	Add D_2, App_2	Add new transitions within D_2 - Add new transitions within App_2 - Add new transitions between D_2 and App_2	D_1 and App_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - D_2 and App_2 tests are new - Interaction between D_2 and App_2 tests are new - No obsolete tests or retestable

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
2	D_1, App_1	D_1, App_1, D_2	Add D_2 and Modify App_1	Add new transitions within D_2 - Add new transitions within App_1 - Modify transitions within App_1 - Add new transitions between D_2 and App_1	D_1 and App_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - D_2 and some of App_1 tests are new - Interaction between D_2 and App_1 tests are new - No obsolete tests

Add New Device with Interaction Between Devices

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
3	D_1, App_1, D_2, App_2	D_1, App_1, D_2, App_2, D_3, App_3	Add D_3, App_3 and Modify D_2, App_2	Add new transitions within D_3 - Add new transitions within App_3 - Add new transitions between D_3 and App_3 - Add new transitions between D_2 and D_3 - Modify transitions within D_2 and App_2 - Modify transitions between D_2 and App_2	D_1 and App_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - D_3 and App_3 tests are new - Interaction between D_3 and App_3 tests are new - Interaction between D_2 and D_3 tests are new - Some of D_2 and App_2 tests are retestable - Interaction between D_2 and App_2 tests are retestable - No obsolete tests

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
4	D_1, App_1, D_2, App_2	D_1, App_1, D_2, App_2, D_3	Add D_3 and Modify D_2, App_2	Add new transitions within D_3 - Add new transitions within App_2 - Add new transitions between D_3 and App_2 - Add new transitions between D_2 and D_3 - Modify transitions within D_2 and App_2 - Modify transitions between D_2 and App_2	D_1 and App_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - D_3 and some of App_2 tests are new - Interaction between D_3 and App_2 tests are new - Interaction between D_2 and D_3 tests are new - Some of D_2 and App_2 tests are retestable - Interaction between D_2 and App_2 tests are retestable - No obsolete tests

Modify Device without Interaction Between Devices

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
5	D_1, App_1, D_2, App_2	D_1, App_1, D_2, App_2	Modify D_2, App_2	Modify transitions within D_2 - Modify transitions within App_2 - Modify transitions between D_2 and App_2	D_1 and App_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - Some of D_2 and App_2 tests are retestable - Interaction between D_2 and App_2 tests are retestable - No obsolete tests

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
6	D_1, App_1, D_2	D_1, App_1, D_2	Modify D_2, App_1	Modify transitions within D_2 - Add new transitions within App_1 - Modify transitions within App_1 - Modify transitions between D_2 and App_1 - Modify transitions within D_2 and App_2 - Modify transitions between D_2 and App_2	D_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - Some of App_1 tests are new - Some of D_2 and App_1 tests are retestable - Interaction between D_2 and App_1 tests are retestable - No obsolete tests

Modify Device with Interaction Between Devices

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
7	D_1, App_1, D_2, App_2	D_1, App_1, D_2, App_2	Modify D_2, App_2, D_1, App_1	Modify transitions within D_2 - Modify transitions within App_2 - Modify transitions between D_2 and App_2 - Modify transitions between D_1 and D_2 - Modify transitions within D_1 and App_1	Some of D_1, App_1, D_2 and App_2 tests are retestable - Interaction between D_1 and App_1 tests are retestable - Interaction between D_2 and App_2 tests are retestable - Interaction between D_1 and D_2 tests are retestable - No obsolete tests

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
8	D_1, App_1, D_2	D_1, App_1, D_2	Modify D_1, D_2, App_1	Modify transitions within D_2 - Add new transitions within App_1 - Modify transitions within D_1 and App_1 - Modify transitions between D_1 and D_2	Some of App_1 tests are new - Some of D_1, D_2 and App_1 tests are retestable - Interaction between D_1 and App_1 tests are retestable - Interaction between D_2 and App_1 tests are retestable - Interaction between D_1 and D_2 tests are retestable - No obsolete tests

Remove Device without Interaction Between Devices

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
9	D_1, App_1, D_2, App_2	D_1, App_1	Remove D_2, App_2	Remove transitions within D_2 - Remove transitions within App_2 - Remove transitions between D_2 and App_2	D_1 and App_1 tests are reusable - Interaction between D_1 and App_1 tests are reusable - D_2 and App_2 tests are obsolete - Interaction between D_2 and App_2 tests are obsolete

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
10	D_1, App_1, D_2	D_1, App_1	Remove D_2 and Modify App_1	Remove transitions within D_2 - Modify transitions within App_1 - Remove transitions between D_2 and App_1	D_1 tests are reusable - Some of App_1 tests are retestable - Some of App_1 tests are obsolete - Interaction between D_1 and App_1 tests are retestable - D_2 tests are obsolete - Interaction between D_2 and App_1 tests are obsolete

Remove Device with Interaction Between Devices

Case	Old Model	New Model	Model Change	Transition Change	Test Classification
11	D_1, App_1, D_2, App_2	D_1, App_1	Remove D_2, App_2	Remove transitions within D_2 - Remove transitions within App_2 - Remove transitions between D_2 and App_2 - Remove transitions between D_1 and D_2 - Modify transitions within D_1 - Modify transitions within App_1 - Modify transitions between D_1 and App_1	D_1 and App_1 tests are retestable - Interaction between D_1 and App_1 tests are retestable - D_2 and App_2 tests are obsolete - Interaction between D_2 and App_2 tests are obsolete - Interaction between D_1 and D_2 tests are obsolete.

Case	**Old Model**	**New Model**	**Model Change**	**Transition Change**	**Test Classification**
12	D_1, App_1, D_2	D_1, App_1	Remove D_2 and Modify D_1 and App_1	Remove transitions within D_2 - Modify transitions within App_1 - Remove transitions between D_2 and App_1 - Remove transitions between D_1 and D_2	D_1 tests are retestable - Some of App_1 tests are retestable - Some of App_1 tests are obsolete - Interaction between D_1 and App_1 tests are retestable - D_2 tests are obsolete - Interaction between D_2 and App_1 tests are obsolete - Interaction between D_1 and D_2 tests are obsolete

Table (5.4) Summarise Evolution's cases in SHS

Chapter 6

Future Work

This dissertation could be extended in different ways as future work:

6.1 Model other components in the Smart Home System

As well as device components, devices and Mobile Apps, there are different components in the SHS that could be modeled in the future. Actors are one possible component. The main actor of the system is the home owner. The home owner could assign different actors in the SHS and give them different privileges. For example, they could assign child access that allows children to control specific devices, or prevents them from accessing some other devices such as smart kitchen devices or devices with payments. Guest access is another temporary access that can be created for a certain time. Frequent guest is another type of access that can be created for close friends or family members who visit often.

6.2 New system domain

This dissertation is focused on SHS as one of IoT applications. We plan to apply this approach in different IoT applications such as health monitoring, smart farming, self-driving cars, fitness tracking and other IoT applications.

6.3 Automation

Automation tools might be built to automate models and tests generation processes which could reduce the cost of generating the models and tests; choose the transition information; and execute the generated tests. Currently, there is no tool to generate reusable test-ready models of SHS or test sequences, and the process is done manually.

6.4 Execution

The generated tests could be executed in the future using the suggested tool in this dissertation.

6.5 Effectiveness

More case studies could be applied to study the effectiveness of reusable test-ready models of smart home systems. The selective black-box model-based regression testing could be compared to the Brute-Force regression testing to show which one is preferable in case of evolution in smart home system.

Chapter 7

Conclusion

This dissertation proposed Reusable Test-Ready Models of Smart Home Systems. It models the devices components (Sensors, Controllers and Actuators) using Extended Finite State Machines (EFSM). The devices and device interactions are modeled using Communicating Extended Finite State Machines (CEFSM). Finally, the system controllers (mobile applications) are modeled using FSMApp. The Reusable Test-Ready Models can be used for different devices within the smart home with modification to the transition's information. We developed a Model-Based Testing technique for these non-heterogeneous Reusable Test-Ready Models which includes testing criteria. Our proposed approach performs three levels of testing: device testing, device interaction testing, and system testing. In this dissertation we consider the rapid evolution in the SHS and the possibility of adding new devices within the smart home system, update existing devices, and/or remove current devices. We take into consideration that these changes might be in the device and/or the mobile application. The evolution could be within devices that interact with each other or in devices without interaction. We developed and validated a selective black-box model-based regression testing based on the changes. Existing tests and

test requirements are classified as reusable, retestable, or obsolete depending on the change in the SHS. We consider the different possible cases and for each case we first define the changes in the models and then classify tests after model changes. As we explained in the previous chapter, this dissertation can be extended in the future to model different SHS components such as actors. Moreover, we can expand the approach to test different IoT applications. Also, we can automate the process of generating models and tests. Executing the current generated tests in this dissertation is another possible future work. Finally, the effectiveness of reusable test-ready models of smart home systems can be examined by conducting more case studies.

Bibliography

[1] Nest protect detects smoke and carbon monoxide (co) user's guide. `https://nest.com/support/images/misc-assets/Nest-Protect-(Wired-120V)-User-s-Guide.pdf`.

[2] Selenium. `https://www.selenium.dev/`. Accessed: 2022-01-30.

[3] M Abdelgawad, S McLeod, A Andrews, and J Xiao. On the Scalability of Concurrency Coverage Criteria for Model-based Testing of Autonomous Robotic Systems. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, pages 3–9. The Steering Committee of The World Congress in Computer Science, Computer ..., 2018.

[4] Mahmoud Abdelgawad, Sterling McLeod, Anneliese Andrews, and Jing Xiao. Model-based testing of real-time adaptive motion planning (RAMP). In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 162–169. IEEE, 2016.

[5] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. Model-based Testing as a Service for IoT Platforms. In *International Symposium on Leveraging Applications of Formal Methods*, pages 727–742. Springer, 2016.

[6] Vangalur S Alagar, Kasilingam Periyasamy, and Kasilingan Periyasamy. *Specification of Software Systems*. Springer, 2011.

[7] Afnan Albahli and Al Haddad Ahmed Andrews, Anneliese. Reusable Test-Ready Models of Smart Home Systems. presented at The 23rd International Conference on Internet Computing & IoT, 2022.

[8] Afnan Albahli and Anneliese Andrews. Model-Based Testing of Smart Home Systems. presented at The 22nd International Conference on Internet Computing & IoT, 2021.

[9] Afnan Albahli and Anneliese Andrews. Model-Based Testing of Smart Home Systems Using EFSM and CEFSM. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1824–1829. IEEE, 2021.

[10] Ahmed Fawzi ALhaddad. *Improvements of and Extensions to FSMWeb: Testing Mobile Apps*. PhD thesis, University of Denver, 2019.

[11] Mohammad Amin Alipour. Fault injection in the internet of things applications. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Testing Embedded and Cyber-Physical Systems*, pages 9–11. ACM, 2017.

[12] Nasser Alshammari, Talal Alshammari, Mohamed Sedky, Justin Champion, and Carolin Bauer. Openshs: Open smart home simulator. *Sensors*, 17(5):1003, 2017.

[13] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2016.

[14] Anneliese Andrews, Mahmoud Abdelgawad, and Ahmed Gario. Active World Model for Testing Autonomous Systems Using CEFSM. In *MoDeVVa@ MoDELS*, pages 1–10, 2015.

[15] Anneliese Andrews, Mahmoud Abdelgawad, and Ahmed Gario. Towards world model-based test generation in autonomous systems. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 1–12. IEEE, 2015.

[16] Anneliese Andrews, Mahmoud Abdelgawad, and Ahmed Gario. World model for testing autonomous systems using petri nets. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, pages 65–69. IEEE, 2016.

[17] Anneliese Andrews, Mahmoud Abdelgawad, and Ahmed Gario. World model for testing urban search and rescue (usar) robots using petri nets. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 663–670. IEEE, 2016.

[18] Anneliese Andrews, Seif Azghandi, and Orest Pilskalns. Regression Testing of Web Applications Using FSMWeb. In *Proceedings of the International Conference on Software Engineering and Applications*, 2010.

[19] Anneliese Andrews and Hyunsook Do. Trade-off analysis for selective versus brute-force regression testing in fsmweb. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, pages 184–192. IEEE, 2014.

[20] Anneliese Andrews, Salwa Elakeili, and Ahmed Alhaddad. Selective regression testing of safety-critical systems: a black box approach. In *2015 IEEE Inter-*

national Conference on Software Quality, Reliability and Security-Companion, pages 22–31. IEEE, 2015.

[21] Anneliese Andrews, Salwa Elakeili, and Salah Boukhris. Fail-Safe Test Generation in Safety Critical Systems. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, pages 49–56. IEEE, 2014.

[22] Anneliese Andrews, Ahmed Gario, and Salwa Elakeili. A Testing Methodology for Safety Critical Systems. *Software Testing, Verification and Reliability*, 2014.

[23] Anneliese A Andrews, Jeff Offutt, and Roger T Alexander. Testing Web applications by modeling with FSMs. *Software & Systems Modeling*, 4(3):326–345, 2005.

[24] Kelly Androutsopoulos, David Clark, Mark Harman, Robert M Hierons, Zheng Li, and Laurence Tratt. Amorphous Slicing of Extended Finite State Machines. *IEEE Transactions on Software Engineering*, 39(7):892–909, 2012.

[25] José A Asensio, Javier Criado, Nicolás Padilla, and Luis Iribarne. A safe approach using virtual devices to evaluate home automation architectures prior installations. In *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 140–145. IEEE, 2017.

[26] José Andrés Asensio, Javier Criado, Nicolás Padilla, and Luis Iribarne. Emulating home automation installations through component-based web technology. *Future Generation Computer Systems*, 93:777–791, 2019.

[27] Tatiana Balikhina, Ali Al Maqousi, AbdElkarim AlBanna, and Fadi Shhadeh. System architecture for smart home meter. In *2017 2nd International Confer-*

ence on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS), pages 1–5. IEEE, 2017.

[28] Sergiy Boroday, Alex Petrenko, Roland Groz, and Yves-Marie Quemener. Test Generation for CEFSM Combining Specification and Fault Coverage. In *Testing of Communicating Systems XIV*, pages 355–371. Springer, 2002.

[29] C Bourhfir, E Aboulhamid, Rachida Dssouli, and Nathalie Rico. A test case generation approach for conformance testing of SDL systems. *Computer Communications*, 24(3-4):319–333, 2001.

[30] C Bourhfir, Rachida Dssouli, E Aboulhamid, and Nathalie Rico. A guided incremental test case generation procedure for conformance testing for CEFSM specified protocols. In *Testing of Communicating Systems*, pages 279–294. Springer, 1998.

[31] Miroslav Bures, Tomas Cerny, and Bestoun S Ahmed. Internet of Things: Current Challenges in the Quality Assurance and Testing Methods. In *International conference on information science and applications*, pages 625–634. Springer, 2018.

[32] Kwang-Ting Cheng and Avinash S Krishnakumar. Automatic Functional Test Generation Using the Extended Finite State Machine Model. In *30th ACM/IEEE Design Automation Conference*, pages 86–91. IEEE, 1993.

[33] Hyunji Chung, Jungheum Park, and Sangjin Lee. Digital forensic approaches for Amazon Alexa ecosystem. *Digital Investigation*, 22:S15–S25, 2017.

[34] Lucio Ciabattoni, Gionata Cimini, Massimo Grisostomi, Gianluca Ippoliti, and Sauro Longhi. An interoperable framework for home automation design,

testing and control. In *22nd Mediterranean Conference on Control and Automation*, pages 1049–1054. IEEE, 2014.

[35] Francesco Coda, Carlo Ghezzi, Giovanni Vigna, and Franca Garzotto. Towards a software engineering approach to Web site development. In *Proceedings Ninth International Workshop on Software Specification and Design*, pages 8–17. IEEE, 1998.

[36] E De Buyser, Elias De Coninck, Bart Dhoedt, and Pieter Simoens. Exploring the Potential of Combining Smart Glasses and Consumer-grade EEG/EMG Headsets for Controlling IoT Appliances in the Smart Home. 2016.

[37] Statista Research Department. Number of Smart Homes in the United States 2017-2025, Jul 2021.

[38] Giuseppe Di Guglielmo, Franco Fummi, Cristina Marconcini, and Graziano Pravadelli. A Pseudo-Deterministic Functional ATPG Based on EFSM Traversing. In *2005 Sixth International Workshop on Microprocessor Test and Verification*, pages 70–75. IEEE, 2005.

[39] Arilo C Dias-Neto and Guilherme H Travassos. A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges. In *Advances in Computers*, volume 80, pages 45–120. Elsevier, 2010.

[40] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[41] Pranay P Gaikwad, Jyotsna P Gabhane, and Snehal S Golait. A survey based on Smart Homes system using Internet-of-Things. In *2015 International Con-*

ference on Computation of Power, Energy, Information and Communication (ICCPEIC), pages 0330–0335. IEEE, 2015.

[42] Aiman Gannous and Anneliese Andrews. Integrating Safety Certification Into Model-Based Testing of Safety-Critical Systems. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 250–260. IEEE, 2019.

[43] Aiman Gannous, Anneliese Andrews, and Lamees Alhazzaa. Robustness Testing of Safety-critical Systems: A Portable Insulin Pump Application. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1736–1742. IEEE, 2020.

[44] Jerry Z Gao, David Kung, Pei Hsia, Yasufumi Toyoshima, and Cris Chen. Object state testing for object-oriented programs. In *Proceedings Nineteenth Annual International Computer Software and Applications Conference (COMPSAC'95)*, pages 232–238. IEEE, 1995.

[45] Cristian González García, Daniel Meana-Llorián, Juan Manuel Cueva Lovelle, et al. A review about Smart Objects, Sensors, and Actuators. *International Journal of Interactive Multimedia & Artificial Intelligence*, 4(3), 2017.

[46] Ahmed Gario, Anneliese Andrews, and Seana Hagerman. Testing of safety-critical systems: An aerospace launch application. In *2014 IEEE Aerospace Conference*, pages 1–17. IEEE, 2014.

[47] David Gesvindr and Barbora Buhnova. Paasarch: Quality evaluation tool for paas cloud applications using generated prototypes. In *2019 ieee international conference on software architecture companion (icsa-c)*, pages 170–173. IEEE, 2019.

[48] David Gesvindr, Ondrej Gasior, and Barbora Buhnova. Architecture design evaluation of PaaS cloud applications using generated prototypes: PaaSArch Cloud Prototyper tool. *Journal of Systems and Software*, 169:110701, 2020.

[49] Anna Katrina Gomez and SimiKamini Bajaj. Challenges of Testing Complex Internet of Things (IoT) Devices and Systems. In *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, pages 1–4. IEEE, 2019.

[50] Rinu Gour. Top 10 Uses of the Internet of Things, Nov 2018.

[51] Wassila Guebli and Abdelkader Belkhir. Providing Services in an Intelligent Environment: Smart Home Simulator based WoT. In *Proceedings of the International Conference on Internet of things and Cloud Computing*, page 61. ACM, 2016.

[52] Jon D Hagar. Software Test Architectures and Advanced Support Environments for IoT. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 252–256. IEEE, 2018.

[53] Olaf Henniger, Miao Lu, and Hasan Ural. Automatic Generation of Test Purposes for Testing Distributed Systems. In *International Workshop on Formal Approaches to Software Testing*, pages 178–191. Springer, 2003.

[54] Anders Hessel and Paul Pettersson. A Global Algorithm for Model-Based Test Suite Generation. *Electronic Notes in Theoretical Computer Science*, 190(2):47–59, 2007.

[55] Robert M Hierons. Reaching and Distinguishing States of Distributed Systems. *SIAM Journal on Computing*, 39(8):3480–3500, 2010.

[56] William E Howden. Methodology for the Generation of Program Test Data. *IEEE Transactions on computers*, 100(5):554–560, 1975.

[57] Weizheng Hu, Yonggang Wen, Kyle Guan, Guangyu Jin, and King Jet Tseng. iTCM: Toward Learning-Based Thermal Comfort Modeling via Pervasive Sensing for Smart Buildings. *IEEE Internet of Things Journal*, 5(5):4164–4177, 2018.

[58] JC Huang. An Approach to Program Testing. *ACM Computing Surveys (CSUR)*, 7(3):113–128, 1975.

[59] Tongcheng Huang, Xiao Liang, Liang Ronglong, and Wang Zhiqi. Design and Development for Smart Home Control System Based on WeChat Platform. In *2018 2nd International Conference on Data Science and Business Analytics (ICDSBA)*, pages 278–282. IEEE, 2018.

[60] IDC. IoT Growth Demands Rethink of Long-Term Storage Strategies, Jul 2020.

[61] Samireh Jalali and Claes Wohlin. Systematic literature studies: Database searches vs. backward snowballing. In *Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement*, pages 29–38. IEEE, 2012.

[62] Abdul Salam Kalaji, Robert Mark Hierons, and Stephen Swift. Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM). In *2009 international conference on software testing verification and validation*, pages 230–239. IEEE, 2009.

[63] Murad Khan, Sadia Din, Sohail Jabbar, Moneeb Gohar, Hemant Ghayvat, and SC Mukhopadhyay. Context-aware low power intelligent Smarthome based on the Internet of things. *Computers & Electrical Engineering*, 52:208–222, 2016.

[64] Barbara Kitchenham and Stuart Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering. 2007.

[65] Gábor Kovács, Zoltán Pap, and Gyula Csopaki. Automatic test selection based on CEFSM specifications. *Acta Cybernetica*, 15(4):583–599, 2002.

[66] Kate Kozuch. The best smart home devices in 2022. *Tom's Guide*, May 2022.

[67] David Kung, Nimish Suchak, Jerry Gao, Pei Hsia, Yasufumi Toyoshima, and Chris Chen. On object state testing. In *Proceedings Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94)*, pages 222–227. IEEE, 1994.

[68] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[69] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelley XQ Zhang. The intelligent home testbed. *environment*, 2:15, 1999.

[70] J Jenny Li and W Eric Wong. Automatic test generation from communicating extended finite state machine (CEFSM)-based models. In *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002*, pages 181–185. IEEE, 2002.

[71] Knud Lasse Lueth. State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. *IoT Analytics*, Nov 2020.

[72] Chengwen Luo, Jiawei Wu, Jianqiang Li, Jia Wang, Weitao Xu, Zhong Ming, Bo Wei, Wei Li, and Albert Y Zomaya. Gait Recognition as a Service for Unobtrusive User Identification in Smart Spaces. *ACM Transactions on Internet of Things*, 1(1):1–21, 2020.

[73] Somayya Madakam, R Ramaswamy, and Siddharth Tripathi. Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*, 3(05):164, 2015.

[74] Chuck Martin. Smart home technology hits 69% penetration in u.s. *MediaPost*, Sep 2019.

[75] Mirella Martínez, Anna I Esparcia-Alcázar, Tanja EJ Vos, Pekka Aho, and Joan Fons i Cors. Towards Automated Testing of the Internet of Things: Results Obtained with the TESTAR Tool. In *International Symposium on Leveraging Applications of Formal Methods*, pages 375–385. Springer, 2018.

[76] Daniel Meana-Llorián, Cristian González García, B Cristina Pelayo G-bustelo, Juan Manuel Cueva Lovelle, and Nestor Garcia-Fernandez. IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions. *Future Generation Computer Systems*, 76:275–284, 2017.

[77] Hubert Miles. How do ecobee room sensors work (explained), 2021.

[78] Saad Mubeen, Sara Abbaspour Asadollah, Alessandro Vittorio Papadopoulos, Mohammad Ashjaei, Hongyu Pei-Breivold, and Moris Behnam. Management of Service Level Agreements for Cloud Services in IoT: A Systematic Mapping Study. *IEEE Access*, 6:30184–30207, 2017.

[79] Binh Minh Nguyen, Huan Phan, Duong Quang Ha, and Giang Nguyen. An Information-centric Approach for Slice Monitoring from Edge Devices to Clouds. *Procedia computer science*, 130:326–335, 2018.

[80] Cu D Nguyen, Alessandro Marchetto, and Paolo Tonella. Combining model-based and combinatorial testing for effective test case generation. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 100–110, 2012.

[81] A Jefferson Offutt and Roland H Untch. Mutation 2000: Uniting the orthogonal. *Mutation testing for the new century*, pages 34–44, 2001.

[82] A Jefferson Offutt, Yiwei Xiong, and Shaoying Liu. Criteria for generating specification-based tests. In *Proceedings Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'99)(Cat. No. PR00434)*, pages 119–129. IEEE, 1999.

[83] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, and Paul Ammann. Generating test data from state-based specifications. *Software testing, verification and reliability*, 13(1):25–53, 2003.

[84] Dario Olianas, Maurizio Leotta, and Filippo Ricca. Matter: A tool for generating end-to-end iot test scripts. *Software Quality Journal*, pages 1–35, 2021.

[85] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[86] Simone Pimont and Jean-Claude Rault. A software reliability assessment based on a structural and behavioral analysis of programs. In *Proceedings*

of the 2nd international conference on Software engineering, pages 486–491, 1976.

[87] Jeremy Pitt, Ada Diaconescu, and Aikaterini Bourazeri. Democratisation of the SmartGrid and the active participation of prosumers. In *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, pages 1707–1714. IEEE, 2017.

[88] Brien Posey, Sharon Shea, and Ivy Wigmore. What is fog computing? - definition from iot agenda, Oct 2021.

[89] ET Powner and F Yalcinkaya. From basic sensors to intelligent sensors: definitions and examples. *Sensor Review*, 1995.

[90] Geraldo P Rocha Filho, Rodolfo I Meneguette, Guilherme Maia, Gustavo Pessin, Vinícius P Gonçalves, Li Weigang, Jó Ueyama, and Leandro A Villas. A fog-enabled smart home solution for decision-making using smart objects. *Future Generation Computer Systems*, 103:18–27, 2020.

[91] Gérald Rocher, Jean-Yves Tigli, and Stéphane Lavirotte. Probabilistic Models Toward Controlling Smart-* Environments. *IEEE Access*, 5:12338–12352, 2017.

[92] Philipp Rosenkranz, Matthias Wählisch, Emmanuel Baccelli, and Ludwig Ortmann. A Distributed Test System Architecture for Open-Source IoT Software. In *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*, pages 43–48, 2015.

[93] Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. *IEEE Transactions on software engineering*, 22(8):529–551, 1996.

[94] B Sasikala, M Rajanarajana, and B Geethavani. Internet of Things: A survey on security issues analysis and countermeasures. *system*, 8:9, 2017.

[95] Yiran Shen, Wen Hu, Mingrui Yang, Bo Wei, Simon Lucey, and Chun Tung Chou. Face recognition on smartphones via optimised sparse representation classification. In *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, pages 237–248. IEEE, 2014.

[96] Pavle Skocir, Petar Krivic, Matea Tomeljak, Mario Kusek, and Gordan Jezic. Activity Detection in Smart Home Environment. *Procedia Computer Science*, 96:672–681, 2016.

[97] Tamir Sobeih, Nick Whittaker, and Liangxiu Han. DIVINE: Building a Wearable Device for Intelligent Control of Environment Using Google Glass. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 1280–1285. IEEE, 2015.

[98] Robert A Sowah, Kwaku O Apeadu, Abdul Ofoli, Koudjo Koumadi, Amevi Acakpovi, and Stephen K Armoo. Interoperability of Heterogeneous Appliances in Home Automation Using theAllJoyn Framework. In *2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*, pages 1–9. IEEE, 2018.

[99] Ming Tao, Kaoru Ota, and Mianxiong Dong. Ontology-based data semantic management and application in IoT-and cloud-enabled smart homes. *Future generation computer systems*, 76:528–539, 2017.

[100] Ming Tao, Chao Qu, Wenhong Wei, Bin Zhou, and Shuqiang Huang. Hybrid Cloud Architecture for Cross-Platform Interoperability in Smart Homes. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 608–617. Springer, 2018.

[101] Artem Tulenkov, Anzhelika Parkhomenko, and Aleksandr Sokolyanskii. Evaluation and Selection of IoT Service for Smart House System Big Data Processing. In *2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT)*, volume 2, pages 124–129. IEEE, 2019.

[102] Mark Utting and Bruno Legeard. *Practical model-based testing: a tools approach*. Elsevier, 2010.

[103] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software testing, verification and reliability*, 22(5):297–312, 2012.

[104] Jeff Voas, Rick Kuhn, and Phil Laplante. Testing IoT Systems. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 48–52. IEEE Computer Society, 2018.

[105] Sieng Wong, Chia Yee Ooi, Yuan Wen Hau, Muhammad N Marsono, and Nasir Shaikh-Husin. Feasible transition path generation for EFSM-based system testing. In *2013 ieee international symposium on circuits and systems (iscas)*, pages 1724–1727. IEEE, 2013.

[106] Xia Yin, Jiangyuan Yao, Zhiliang Wang, Xingang Shi, Jun Bi, and Jianping Wu. Modeling and Testing of Network Protocols with Parallel State Machines. *IEICE TRANSACTIONS on Information and Systems*, 98(12):2091–2104, 2015.

[107] Lin Yuan. Study of Smart Home System Based on Cloud Computing and the Key Technologies. In *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 968–972. IEEE, 2015.

[108] Muhammad Mu'izzudeen Yusri, Shahreen Kasim, Rohayanti Hassan, Zubaile Abdullah, Husni Ruslai, Kamaruzzaman Jahidin, and Mohammad Syafwan Arshad. Smart mirror for smart life. In *2017 6th ICT International Student Project Conference (ICT-ISPC)*, pages 1–5. IEEE, 2017.

[109] Justyna Zander, Ina Schieferdecker, and Pieter J Mosterman. *Model-based testing for embedded systems*. CRC press, 2017.

[110] ZhiHao Zhang, DongMing Yuan, and HeFei Hu. Multi-Layer Modeling of OpenFlow based on EFSM. In *2016 4th International Conference on Machinery, Materials and Information Technology Applications*, pages 524–529. Atlantis Press, 2017.

[111] Meftah Zouai, Okba Kazar, Belgacem Haba, and Hamza Saouli. Smart house simulation based multi-agent system and internet of things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, pages 201–203. IEEE, 2017.