Design and Validation of a Context-Aware Publish-Subscribe Model

by

Akshat Kumar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2015

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

A system is said to be context-aware if it can extract, interpret and use contextual information to adapt its functionality and enhance its utility. Context awareness allows the application to gain sensitivity for many environmental parameters that are beyond the reach of conventional systems. Human factors related to context include information about the user (knowledge of habits, emotional state), the user's social environment (co-location of others, social interaction, group dynamics), and the user's tasks (spontaneous activity, engaged tasks, general goals). With access to this contextual information, there are many exciting possibilities for applications involving direct human interaction.

Software modelling is one of the first steps in the life cycle of a software system. Software models can lead to the discovery of errors in a system, which is useful as the early discovery of such flaws can enable the designers to update the inexpensive system model. By not using system models before the development of the full scale system, we risk the discovery of major problems later on in the lifecycle, which will be more expensive to fix.

Validation of any software system is an essential part of the development lifecycle. The validation of context-aware systems is especially challenging as the input range of the system is loosely defined. But despite this it is very important to validate context-aware systems thoroughly because it is possible that a subset of possible inputs to the system can be part of a failure-critical user interaction. Modeling and validation are important activities in the development or enhancement of all software systems. While software modelling helps check the properties of the systems before actual development, software validation is essential for ensuring the quality of the software based on the original software requirements.

This thesis focuses on the modeling and the validation of formal case study design models for context-aware systems based on the event based and publish-subscribe pattern. The study validates formal case study design models against relevant properties using a model checker.

Acknowledgements

I would like to thank Professor Paulo Alencar, for his patience, understanding, kindness and guidance. I learned a lot while working with him and I am proud to be his student. I thank Eduardo Barrenechea and Rolando Blanco for their help at times of need. I am thankful to Professor Daniel Berry for serving as my co-supervisor. I also thank Professor Donald Cowan and Professor Gladimir Baranoski for agreeing to serve as members of my thesis committee.

Immense gratitude towards my family, my parents Meera and Ashok, and my brother Anugrah for their unconditional love.

Finally, thanks to my friends Peng Peng and Gaurav Gupta for sharing their wisdom and inspiring optimism.

Dedication

Dedicated to my grandfather Shri Krishna Chandra Vaishya

Contents

List of Tables

List of Figures

Chapter 1

1. Introduction

With the advent of mobile devices and the internet, it has become possible and in many cases necessary to adapt services to the context or particular situation of the user. Although the literature on context-aware systems presents a variety of definitions for context, a well-accepted definition proposed by Dey [1], Abowd *et al.* [2] and Hong *et al.* [3] asserts that context is simply the situation of an entity. For example, human factors related to context include *information on the user* (e.g. knowledge of habits, emotional state), the *user's social environment* (e.g. co-location of others, social interaction, group dynamics) and the *user's tasks* (e.g. spontaneous activity, engaged tasks, general goals) [4].

Systems that can extract contextual information, interpret it and use it to adapt the system's functionality and enhance system utility are labelled as *context-aware systems* [5, 6]. One of the first examples of context-aware system was presented by Schilit and Theimer [7]. In their research, the authors described a system in which users interact with many different mobile and stationary computing platforms, and argued that such a system can adapt according to environmental factors such as the location of use, the collection of nearby objects, as well as the changes to those objects over time. Previous research has demonstrated that context awareness allows systems to be sensitive to factors that were beyond the reach of conventional systems. As reported by Hong *et al.* [3], context-awareness offers unique advantages since it can reduce input costs, enhance the user experience and enable context sharing.

Much research in the related area of *event-based systems* has focused on *Publish-Subscribe (PubSub)* systems Jacobsen et al. [8], describes the publish-subscribe approach as a messaging design pattern that is commonly used for decoupling different components of a system in which the senders of messages, called publishers, do not send messages directly to specific receivers, called subscribers. Birman and Joseph [9] have shown that in PubSub, the publishers can generate events with little or no information about the existence or number of subscribers that have some interest in the event, and the subscribers on the other hand do not need to know about the publishers that exist in the system. Overall, in recent years research into the use of the PubSub pattern in the area of context-aware systems has attracted attention [10].

This thesis presents, a novel approach for modeling and validating software systems that support both context-awareness and the publish-subscribe design pattern. By using formal validation techniques, the study has been able provide formal proofs for relevant properties of the formal case study design models.

1.1 Background

Software modeling and validation are important activities for the development or enhancement of all software systems. Software modelling also helps validate the properties of the systems before its actual development. Software validation is the process that ensures that the model is free from failures and complies with the requirements [11] .

Software modelling involves the creation and analysis of the description of any aspect of a software system. The software model can comprise postulates, data and interfaces presented as a mathematical description of an entity. The use of the system models for validation can result in the early discovery of bugs in the system [12, 13]. By not using system models before the development of the full scale system we risk the discovery of bugs later on in the lifecycle, which can be much more expensive to fix [14].

Proper *software validation*, is an essential part in the successful development of software systems. Software validation is the process that ensures that the model is free from failures and complies with its requirements [11], unlike *software verification*, which determines whether or not the products of a given phase of a software development process fulfill the specifications established in the previous phase [11]. However this thesis focuses on the validation of the software model against the requirements. The importance of validation is further enhanced in the case of failure-critical applications. Accidents caused due to software faults in such areas can lead to irreversible loss or harm. Vieira *et al.* [15] argued that the validation of context-aware systems is especially challenging as the input range of the system is loosely defined. But despite these difficulties, it is argued that it is very important to validate context-aware system models thoroughly because it is possible that a subset of all possible inputs to the system can result in the failure of a critical operation. There is a growing need to automate the validations of all software and, according to Costa et al. [16], the rate of technological advances makes automated validation of system models more desirable.

1.2 Related areas

1.2.1 Context-aware systems

A system is said to be context-aware if it can extract, interpret and use contextual information to adapt its functionality and enhance its utility [5, 6]. Context awareness can enable users of the system to handle vast amount of information by understanding the user's situation as it guides the user through otherwise complicated processes and help users adapt to their environment effortlessly by helping systems to segregate "what is important?" and "what is not important?" [17, 18].

These systems need to process dynamic and ambiguous contextual information and manage large datasets in real-time. For processing this information, they need to interpret and reason about contextual information. In addition, any context-aware system must be able to understand the contextual information and perform reasoning about the user's situation and needs [4]. There are essentially three main strategies for context reasoning namely, the ontological, rule-based and distributed approaches [19, 20, 21].

1.2.2 Distributed event-based systems

Several publications that appeared in recent years document the rise in the popularity of event-based systems owing to immergence of the internet and mobile computing platforms. Event-based systems are made up of components called publishers and subscribers that have no direct connection to each other. An event-based system is different from other systems in that the components of an event-based system know little or nothing about each other and, more specifically, these components don't communicate by direct method invocation. This allows these systems to become more *loosely* coupled. Helmer *et al.* [22] report that event based systems, which rely on asynchronous *communication*, are more scalable and more reliable than other systems. In contrast to synchronous communication processes, asynchronous processes do not have to stop working while waiting for data. Generally, this means that event sources push data about event publishers to event subscribers.

Barrenechea *et al.* [23] described three main characteristics of distributed event-based systems, namely time decoupling, space decoupling and synchronization decoupling. *Time decoupling* enables publishers and subscribers to publish and receive events at the different times. A publisher can publish an event even if the subscriber is offline. Conversely, a subscriber may be notified of events even if the publisher is offline. *Space decoupling* means the publisher and subscriber do not need to know of each other's existence. A publisher can send out events even when no subscribers are available to receive the event and, conversely, a subscriber can exist in the system even when no publisher exists in the system.

Synchronization decoupling enables subscribers to be notified asynchronously by publishers through event messages. This implies that a publisher does not need to wait for a subscriber to receive events.

1.2.3 Mobile systems

Advances in mobile computing and distributed systems have been the driving force behind the high penetration rate of mobile computing devices into new geographical areas. Currently, mobile devices are the prevalent personal computing devices used in all parts of the world, and mobile internet usage has outpaced desktop/laptop usage worldwide [24]. However, the introduction of mobile systems is presenting new challenges to system design. For example, software adaptability has become a major concern since applications running over mobile platforms need to be more adaptable as the environmental conditions of mobile devices change. In addition, environmental changes typically did not need to be considered in the case of traditional fixed computing platforms.

Previous research clearly indicates that context awareness can be especially useful for mobile computing platforms, as it can help these platforms adapt to changes in their environments. For several years researchers have devoted their efforts to translating lessons learned from research results on context awareness into mobile systems. As a consequence, many mobile platforms provide sensors that can be used by context-aware applications running on these platforms. According to Pascoe and Lyons [25, 26] today the world is at the tipping point of widespread adoption of a new kind of platform based on wearable computers. Though the idea has been around for some time now, it is only now that this class of devices has been made available to consumers. Wearable computers are being recognized as the technology that will drive the next big advances in the field.

1.3 Problem statement

This work is motivated by three major problems in the area of context-aware applications. These problems are discussed in the following sub-sections.

1.3.1 Lack of models for context-aware publish-subscribe (CAPS) systems

In recent years, research into context-aware systems has become very popular, but studies in the topic are still lacking. In my view, many of the previous studies in the field of context-aware systems and ubiquitous computing have focused on improvement of applications and aspects involved in direct interaction with the users. According to Baldauf [4], there has been limited effort dedicated to addressing the problems related to the underlying framework abstractions and models such as the development and use of standard communication protocols and the enforcement of standard privacy and security norms. To

the best of my knowledge, there is a lack of research on applying the principles of publish-subscribe architectures to context-aware systems.

Eugster *et al.* [27] report that large-scale systems can achieve component decoupling by using variations of a publish-subscribe architecture. Jacobsen *et al.* [8] has shown that aspects of publish-subscribe patterns can be effectively used to enhance many features of context-aware systems. There is also a lack of research on the development of design practices that can lead to better compatibility between components developed by different vendors. This is a major drawback for the area as the full utility of context-aware systems will only be delivered to the users when devices can exchange information seamlessly without compatibility issues.

1.3.2 Need of rigorous representations for CAPS systems

Thus, more research efforts are needed to understand better the design abstractions and models of CAPS systems. The provision of CAPS models can help us to understand better, the basic components, relationships and interactions underlying such systems. In essence, I believe that there is a need for formal specification techniques for CAPS systems.

Hierons *et al.* and Gaudel [12, 13] state that formal specifications are mathematically-based techniques whose purpose is to help with the implementation of systems and software. They are used to describe a system, to analyze its behavior, and to aid in its design by validating key properties of interest through rigorous and effective reasoning tools [12]. Formal specifications are a great way to reduce the ambiguity about the behaviour of a system, and formal specification techniques can assist in the modelling, design and implementation of reliable software system [28].

It is difficult to judge a design on its own. It can only ever be considered suitable with respect to some desired behaviour. Using formal specification can minimize the ambiguity regarding the desired behaviour of a system. The process of defining a formal specification itself ensures to a large extent that, all aspects of the system behaviour have been considered deeply, this can assist in the discovery of any ambiguities or gaps in system behaviour. Unlike informal specifications, formal specification can be validated using formal validation techniques. According to my review, there has been no previous work in the area that used formal specifications for the validation of models for CAPS systems.

1.3.3 Lack of validation approaches for CAPS systems

A key deficiency in most of the frameworks proposed in the literature in the area of context-aware systems is that none of them provide formal proofs for relevant structural and behavioral properties. Formal validation refers to proving or disproving the correctness of intended algorithms underlying a

system with respect to a certain formal specification or property [29]. With formal specifications, it is possible to use automated formal validation techniques to demonstrate that a system model is correct with respect to its requirements. This allows incorrect system designs to be revised before any major investments have been made in developing an actual system [14]. Formal validation has been widely used in the development of safety-critical systems in many areas, e.g., banking, healthcare and law-enforcement. The motivation behind the use of formal validation in these areas is to improve the robustness of the system and to predict the behaviour of the system in all possible conditions.

Elements from context-aware systems are slowly being adopted in many areas that involve direct interaction with the user. The motivation behind this adoption is reducing the cost of user input, enhancing user experience and taking advantage of collective context. However, these systems need to behave as expected, especially those that run on safety-critical environments. In many cases there are properties that must be satisfied, and the assurances provided by formal validation techniques can ensure that this is the case.

1.4 Proposed approach

This thesis introduces a new approach for modeling and formal validation for a CAPS system. Figure 1 presents the procedure followed in the course of the study.

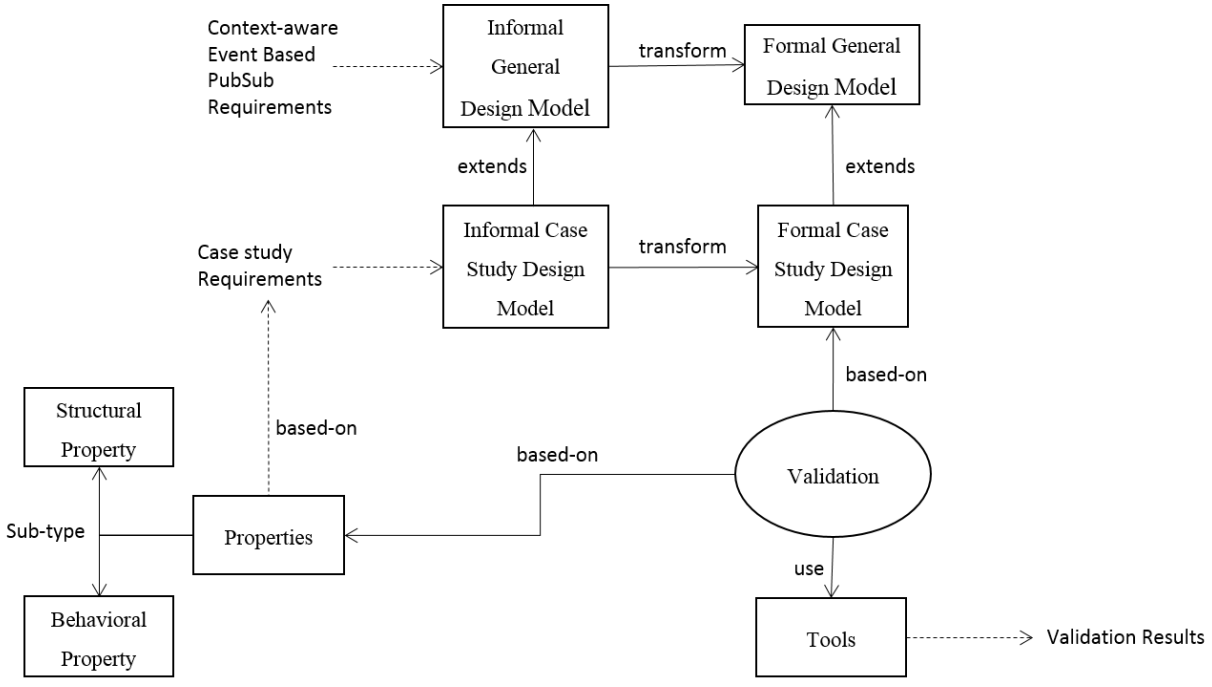

Figure 1 Proposed Approach

The approach consists of four steps:

1. The first step of the study is to construct an informal general design model based on the CAPS requirements, and then to produce informal case study design models based on the case study requirements.
2. Based on the informal design models, the formal general design model is developed, followed by the formal case study design models, which are extensions of the formal general design model.
3. Based on the case study requirements, relevant structural and behavioral properties are identified.
4. Finally, using a model checker, the formal case study design models are validated against the selected properties.

1.4.1 Designing CAPS models

The study started by designing an informal general design model using Unified Modelling Language (UML). The informal design model defines the different components included in the general design model and outlined the capabilities of each of these participant components. The informal design model also provides the relationships and interactions between the components of the model. Section 3.1.1.1 will present the informal general design model in more detail. More details regarding the informal case study design models are presented in Chapter 4.

1.4.2 Formal specification of the CAPS models

Based on the approach presented in Figure 1 the approach uses the informal general design model for developing its formal specification. The formal case study design models were developed by extending the formal general design model. This thesis explores a new approach for the automated formal validation of the formal case study design models. This is a novel approach, as this is the first study to examine the use of formal validation of CAPS models based on the modular notion of environments and on three levels of filtering (i.e., type, content and context filtering). Further details about the formal general design model are presented in Section 3.1.2.1 and the details about the formal case-studies design models are presented in Chapter 4. The source code for the formal general design model can be found in Appendix A and the source code for formal case study design models is provided in Appendix B.

1.4.3 Validation of CAPS case study design models

In order to validate the formal case study design models, this study validated the models with respect to specific properties based on the case study requirements. These properties were subdivided into structural and behavioral properties. Structural properties define the relationships between the components, and

behavioral properties describe the range of valid operations that can be performed by a component. The properties were validated for each case study individually using a model checker. The properties that were validated using the simulations were defined using a platform specific language called XL.

The syntax of the process and property definition language used by the model checker is presented in Section 3.2.3. Further details regarding the properties validated against the formal case study design models are presented in Sections 4.1.2 and 4.2.2. The source code of the properties using the property and process definition language is presented in Appendix C.

1.4.4 Evaluation

To evaluate the approach, the study uses the formal specification of the model and conducted two case studies based on it.

The formal design models are based on Prolog and process calculus expressions. The validation uses a model checker provided by XMC. Prolog definitions are used to define structural relationships, and the process calculus expressions are used to encode application behavior.

The case studies involved modeling of real-world scenarios. Each case study extends the model and provides basis for the validation of the model based on the properties. The properties are defined based on the requirements of the case study.

The first case study involves two components, a cellphone and a calendar application. This case study considers the behavioral adaptation of the cellphone triggered by events from the calendar application.

The second case study involves the eight components listed in Section 4.2.1.1. This case study investigates the behavioral adaptation in the other components in the environment triggered by events from the cellphone.

1.5 Contributions

This thesis introduces an innovative approach for modeling and formal validation for CAPS systems. By using formal validation techniques, the study has been able to provide automated formal proofs of relevant properties of the formal case study design model based on its requirements. The contributions of the thesis include:

- Providing an informal general design model for CAPS systems;
- Developing the formal specification of the general CAPS design models;
- Developing case studies by extending the informal and formal general design model; and

- Conducting the formal validation of case study design models based on structural and behavioral properties.

1.6 Outline of dissertation

The remainder of the thesis is organized as follows. C*hapter 2* outlines existing research work related to the thesis. *Chapter* 3 explains the proposed approach, and describes the different entities and relationships present in the general design model, the informal general design model and the kinds of properties that will be used using formal validation. C*hapter 3* also outlines the implementation of the general design model using a formal specification language and gives a brief description of how the properties are implemented, then it presents the syntax of the process and the property description languages used for the validation. *Chapter* 4 provides the outline of the case study, and presents an example driven description about their formal specification and their properties. *Chapter* 5 completes the thesis by providing conclusions and possible areas for future investigation.

Chapter 2

2. Related work

This work is based on a substantial body of previous work performed by the research community in related areas. This chapter will mention previous work that helped us in the definition of this study. The rest of the chapter will be divided into sections focusing on work from individual areas.

2.1 Context-aware systems

The area of context-aware systems is highly researched and there has also been significant development in other areas that share many of the concerns that were originally introduced by consideration of context-aware systems area. This section will present relevant work from research into context-aware systems.

Context-aware systems can be designed and implemented in many ways. Each approach depends on special requirements and conditions such as the location of sensors (local or remote), the number of possible users, the type of resources (high-end-PCs or small mobile devices) or the flexibility of the system. Furthermore, the method of acquisition of contextual information is very important when designing context-aware systems because it can predefine the architectural style of the system.

A context-aware system can use three types of sensors: physical sensors, virtual sensors and logical sensors [30] for the acquisition of contextual information. *Physical sensors* are hardware sensors capable of capturing physical data like light, audio, motion or location. Location has received a large amount of research, but other types of context have also recently been explored. *Virtual sensors* capture contextual information from software application, the operating system or network. Detection of a new appointment on a calendar application is an example of a virtual sensor. Some systems have used virtual sensors to infer some properties of the physical sensors. For example Wi-Fi networks are widely used today to infer location [31]. Virtual sensors have not received as much research attention as physical sensors although they seem promising. The third type of sensor that appears in the literature talks about is a *Logical sensor*. These sensors combine information to derive a higher-level of context. There has not been much research on logical sensors.

Chen, Finin & Joshi [32] present three approaches for the acquisition of contextual information. The *Direct sensor access* approach, which is applicable to devices or platforms that have sensors built-in.

Drivers for the sensors are usually hardwired into the application, so they are tightly coupled. Hence the approach is not suitable for distributed systems because of their inability to support components that are capable of managing multiple concurrent sensors. A *Middleware infrastructure* approach introduces a layered architecture to context-aware systems with the intention of hiding low-level sensor details. In comparison to direct access, this assists in extensibility since the client code does not need to be modified and it also simplifies reusability of hardware dependent sensing code because of the strict encapsulation. A *Context server* distributed approach extends the middleware based architecture by moving contextual data to a context server to facilitate concurrent multiple access. This approach also removes computational requirements from the clients as most of the computation intensive operations are performed by the context server.

Winograd [33] proposed a similar classification of three approaches for context management. First *Widgets* are software components that provide a public interface for hardware sensors. This approach is tightly coupled, it increases efficiency but is not robust in relation to component failures. Second, *Networked services*, which resembles the context server architecture proposed in [32], except that a global widget manager is used to find networked services. Third, *Blackboard model*, that represents a data-centric view like publish-subscribe. In this approach the processes post messages to a shared media, the called blackboard, and subscribe to it to be notified when some specified event occurs.

Contextual information needs to be encapsulated as context models. A context model stores context data in a machine consumable form. Strang and Linnhoff-Popien [34] classified most popular context modelling approaches. *Key-Value models* represent the simplest data structure for context modelling. *Markup scheme models* use a hierarchical data structure consisting of markup tags with attributes. *Graphical models* use modelling standards like UML by Sheng and Benatallah [35] and an extension of ORM by Hendricksen *et al.* [36] for representing context models. *Object oriented models* encapsulate the details of context processing and representation. A popular example of this approach is Hydrogen presented by Hofer *et al.* [37]. *Logic based models*, as proposed by McCarthy and Buvac [38]. The logic approach uses facts, expressions and rules to define a formal context model. *Ontology based models* are a very promising instrument for modelling contextual information because of their formal expressiveness and their potential to aid in the reasoning about contextual information.

Context-aware systems dealing with location information are widespread and the demand for them is growing because of the increasing spread of mobile devices. Example of previous work in this area can be found in Priyantha *et al.* [39], Espinoza *et al.* [40], Burrell and Gay [41] and Kerer *et al.* [42]. These papers present different techniques for sensing the location of the user including GPS satellites, mobile phone towers, badge proximity detectors, cameras, magnetic card readers and barcode readers. These

systems use only one aspect of context, namely location information. The use of different types of context atoms such as noise, light and location allows the combination to form high-level context objects, which are necessary to build more adaptive, useful and user-friendly systems. Muñoz *et al.* [43] demonstrated this adaptation by extending an instant messaging system with context-awareness to support information management within a hospital setting.

Context-aware systems are designed for dealing with special types of context. They are well-suited for specific conditions. The goal of most frameworks proposed for context-aware systems is to simplify the development by providing an abstract framework. The architecture *Context Managing Framework* presented by Korpipää *et al.* [44] is derived from a classical hierarchical infrastructure with some centralized components. It is useful to overcome memory and processor constraints of small mobile devices but provides one single point of failure and thereby lacks robustness. The *Service Oriented Context-Aware Middleware* (SOCAM) project introduced by Gu *et al.* [45] uses a central server as well, called a context interpreter. *Context Awareness Sub-Structure* (CASS) presented in Fahy and Clarke [46] is based on an extensible centralised middleware approach. *Context Broker Architecture* (CoBrA) Chen *et al.* [32, 47], it is an agent based architecture for supporting context-aware computing. A central component of CoBrA is an intelligent context broker that maintains and manages a shared contextual model on behalf of a community of agents. Another framework based on a layered architecture is built in the *Hydrogen project* Hofer *et al.* [37]. The *Hydrogen project* distinguishes between a remote and a local context. The remote context represents information that is known to another device, while the local context represents the knowledge our own device is aware of. The *Sentient Object Model* proposed by Biegel and Cahill [48] is an example of a context-aware middleware approach. A sentient object is an encapsulated entity consisting of three main parts: Sensory capture, Context hierarchy and Inference engine. The *Gaia project* presented by Roman *et al.* [49] is another middleware infrastructure. It aims at supporting the development and execution of portable applications for interactive physical systems (active spaces).

Sehic, Nastic & Vögler [50] propose a programing model for context-aware applications called *Entity-Adaptation*. Their motivation for this work is to improve the modularity of the source code and make more of the components in a context-aware application reusable. The researchers plan to achieve this by decoupling the code of the context-aware application with the boilerplate code used for interacting with the physical environment. The researchers also introduce a framework that provides the stakeholders with support for the development and execution of context-aware applications developed using the entity-adaptation programing model called the CAPA framework. The researchers [50] use one case study for the demonstration and evaluation of the Entity-Adaptation programing model and the CAPA framework.

The case study considered the design, development and deployment an application for a building management system. The previous system in place was in form of two separate systems that were later combined into a single system.

2.2 Publish-subscribe event based systems

Event-based systems have emerged as a key technology for achieving scalable information dissemination. In particular, they have been used as communication backbone within publish/subscribe (PubSub) communication systems. Their aim is to reduce the network and computational overhead for event diffusion to a set of interested recipients.

There are a number of PubSub architectures with different fundamental characteristics. One possible decomposition as proposed classifies the systems into the general categories of subject-based or content-based systems [51]. In *subject-based systems*, a message belongs to one of a fixed set of groups, channels, or topics. Subscription targets a group, channel, or topic, and the user receives all events that are associated with that group. For example, in a subject-based system for stock trading, a participant could select one or two stocks then subscribe based on stock name if that were one of valid groups. In contrast *content-based* systems are not constrained to the notion that a message must belong to a particular group. Instead, the decision on where to direct a message is made on a message-by-message basis based on a query or predicate issued by a subscriber. Liu and Palae [51] also present other classifications for PubSub systems are presented based on System Architecture, Matching Algorithm, Event distribution scheme, Reliability and Security.

Guruduth *et al.* [52] presented a PubSub system called *Gryphon*, targeted towards the distribution of large volumes of data to thousands of clients. It is a content-based publish subscribe system with a client-server architecture. Events are matched to subscribers using a matching tree that is constructed in the preprocessing phase before the system is operational.

Rowstron *et al.* [53] presented *Scribe*, a subject-based PubSub system. It is built on *Pastry* [54], a generic peer-to-peer object location and routing substrate as layer on the Internet. For Scribe, each node can act as a publisher, a root of a multicast tree, and a subscriber to a topic, a node within a tree, or any reasonable combination of the roles. Scribe's matching algorithm is based on numeric keys and node IDs, each node in the network has a unique numeric identifier (node ID) and the numeric key is a number which can be used to identify the topic.

Zhuang *et al.* [55] presented *Bayeux*, a multicast event distribution system that scales to arbitrarily large receiver groups while tolerating failures in routers and network links. It is based on *Tapestry* [56] that provides a decentralized peer-to-peer architecture. Each Tapestry node can assume the roles of server, router, and client. Bayeux allows messages to locate objects and route them across an arbitrarily-sized network. For Bayeux a multicast system needs only to duplicate a packet when the receiver node identifiers become divergent.

Siena proposed by Carzaniga [57] is a content-based scalable event-notification. Its architecture is client-server based which allows two types of clients, publishers and subscribers, exchange messages through a Siena server. Matching in Siena is accomplished at the server with a Binary Decision Diagram [58]. Geoffrey and Pallickara [59] presented *NaradaBrokering*, it can be categorized as a content-based publish subscribe system with a hierarchical topology of servers for event dissemination within a cell and peer to peer traffic between cells. Slominski, Aleksander, *et al.* [60] presented *XMessages*, which is a hybrid subject-based and content-based PubSub system using a client server graph architecture. The filtering is based on the content of messages.

Padres presented by Fidler *et al.* [8] is a popular a content-based PubSub system which uses distributed middleware. Padres has been designed to support several novel concepts in PubSub that are necessary in the workflow management. Matching in padres is performed using a powerful rule engine to enable composite subscriptions.

2.3 Mobile systems

Mobile systems have limited resources, such as battery life, network bandwidth, storage capacity, and processor performance. These restrictions may be alleviated by computation off-loading: sending heavy computation to resourceful servers and receiving the results from these servers [61].

High performance for multimedia tasks is required while preserving battery life. Energy conservation in computing platforms was non-existent prior to the development of the mobile computing platforms, as a result this problem has received a significant amount of research. There have been attempts to reduce the energy consumption of all aspects of mobile computing platforms. Simultaneously there have been a lot of improvements in the capabilities of the batteries that are used to power the mobile devices. Barr [62] presented approach to reduce the number of bits transmitted by developing a special compression technique that consumes less energy. Krashinsky and Balakrishnan [63] present a new protocol as an enhancement to the popular power saving variant of IEEE 802.11, their motivation is the removal of the mandatory performance degradation that is a related effect of the power saving mode.

Smit *et al.* [64] presented *Chameleon,* an approach in which reconfiguration is applied dynamically at various levels of a mobile system. Their major motivation was to have an energy-efficient system while achieving an adequate Quality of Service for applications. In *Chameleon,* the granularity of reconfiguration is chosen in accordance with the computation model of the task to be performed [65]. The philosophy used is that operations on data should be at the place where it is most energy efficient and minimises the required communication. Partitioning is an important architectural decision, which dictates where applications can run, where data can be stored, and the complexity of the mobile system and the cost of communication services. Their approach is based on a dynamic matching of the architecture and the application.

Energy has become an important quality metric for mobile apps. Recent studies by Gui *et al.* [66] have shown that energy related complaints can represent a significant source of user unhappiness with mobile applications. Hao *et al.* [67] propose a new approach, *eCalc*, which can help the developer by providing code-level estimates of energy consumption. eCalc achieves this using estimation techniques based on program analysis of the mobile application.

Li *et al.* [68] propose an approach for reducing display energy by automatically changing the color schemes used by a web app so that the pages consume less energy when displayed on OLED displays used by energy constrained devices. The first step is a static analysis that examines the server-side code of the web app and builds a model representing the potential HTML content that it could generate. In the second step, the approach parses this model to identify the visual relationships between tags. In the third step of the approach is to determine the colors of the text and background of the different HTML elements.

2.4 Formal validation

Formal validation of hardware and software systems has gained popularity in industry since the advent of the famous “Pentium bug” in 1994, which caused Intel to recall faulty chips and take a loss of $475 million [69]. The model checking problem involves the construction of an abstract model of a system or subsystem, followed by the construction of a specification formula and finally establishing if the model semantically implements the specification.

Model checking is a technique to validate a system description against a specification [70]. Given a system description and a logical specification, the model checking algorithm either proves that the system description satisfies the specification, or reports a counterexample that violates the specification. The input to a software model checker is the program source or system description, and a temporal safety

property. The specification is usually given by program instrumentation that defines a monitor automaton, which models if a program execution violates the desired property.

Holzmann & Joshi [71] present a software verification approach. This novel approach allows the application code to be verified directly by the model checker. This means that developers do not need to implement a model in the application in the language of the model checker. However, the approach does require the developers to write a 'test-harness' in the language of the model checker. The model checker being used for the approach of Holzmann & Joshi is the *SPIN* model checker.

Jhala *et al.* [70] present a model checker for software validation called *BLAST*, for programs written in the C programming language. Blast constructs, explores, and refines abstractions of the program state space based on lazy predicate abstraction and interpolation-based predicate discovery.

XSB [72] & XMC [73] have previously been used for rigorous analysis of software design patterns [74]. The author tries to address the problems in ensuring the integrity and reliability of these composed systems because of their complex software topologies, interactions, and transactions. The author is concerned about the consequences of the complex interactions between different patterns that might be used in a large software system. The author focuses on object-oriented patterns, and argues that since most existing techniques for modeling software design components such as UML are based on informal design notations they lack the support to analyze the pattern interactions formally.

Chapter 3

3. Proposed approach

This thesis introduces an innovative approach for modeling and formal validation for CAPS systems. Recently some work has been done in applying publish subscribe design to context-aware systems. Blanco [10] proposed an adaptable component model for context-aware systems based on a publish-subscribe design and distributed event-based systems. However, the study does not consider neither formal models nor the properties these models should satisfy. In contrast, this thesis presents an original approach based on formal validation techniques.

The CAPS models are specified in XSB Prolog and a variant of process calculus supported by the XMC model checker process language. The validation of the formal CAPS models relies on structural and behavioral properties that the formal models should satisfy. These properties are implemented in the XL scripting language [75] and the validation is automated by using the XMC model checker [73]. The approach is evaluated through two case studies.

3.1 Modeling

As per the study approach represented in Figure 1, we will be using both informal and formal models in this study.

3.1.1 Informal models

The informal models are based on the requirements of the general model and the requirements of the case study respectively.

3.1.1.1 General informal design model

The general design model comprises basic elements that communicate with each other by using events. In this model the basic logical elements are *environment* and *component*. An environment can be said to be a special kind of component. Hence, it is correct to say that *components* are the root elements of the general design model.

The general design model needs to identify relationships between the elements. The relationships define the capabilities of elements. The relationships are also used to determine the complex relative-roles that can exist because of multiple interconnected components. Relations are also needed to associate a component to its context.

The intent of a component to publish or share a particular kind of information with other components needs to be identified by the general design model. In this case the component produces information in the form of events [76]. The general design model be able to determine whether a component is interested in receiving a particular kind of information. In this case the component is said to be a consumer of information in the form of events.

The ability of the general design model to isolate the context of each component and to combine the context of multiple components, are very important for context-aware systems. For example, in the case of an environment containing two components, the context of the environment is the combination of the contextual information of the individual components and the context of the environment itself.

Finally, the general design model needed to support different types of events. This is achieved by using event types for defining events prior to the transmission of the event, as events are thought as instances of event schemas. An event is said to be of a certain type if the key-value contained in event is an exact match for the event schema defined for the type.

Environments and components

The most basic element of the general design model is the *component*. A component is used to group relevant contextual information. A component can be a producer of information in which case it has the role of *event-publisher*. A component can also be a consumer of information, in which case it is said to have the role of *event-subscriber*. A component can have multiple roles at the same time, thus enabling a component to have a rich behavior.

Another element is the environment, which is a special kind of component. An environment has some extra features. The first feature is that an environment can contain other components and environments, thus enabling the formation of complex hierarchies of components and environments leading to rich context based interactions. Using this capability, environments can used to define regions by grouping components based on location, utility or other properties. The second distinguishing feature of environments is that environments can filter propagation of events to its subordinate components by using context filters. In Figure 2 show the relation between a component and an environment using a UML class diagram.

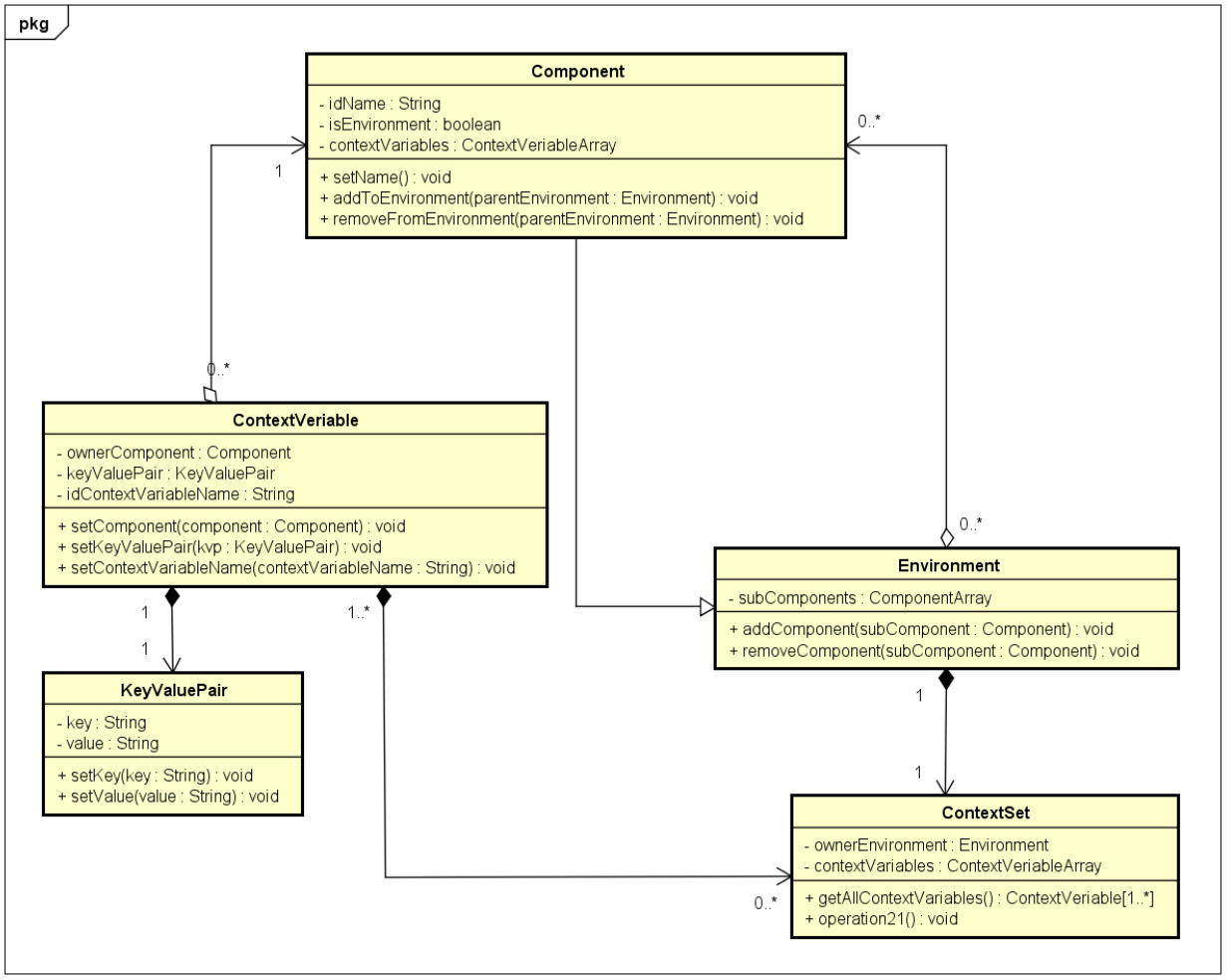

Figure 2 Design environment and component

Event schemas and events

Events are the only medium of communication used by the general design model. Events are used for both carrying data and signaling. In case of data transfer, the data itself is broken into chunks and then the data piggybacks on an event as its payload. When each event reaches its destination the receiver can reassemble the parts into the whole message. In the case of signaling the payload information is generally much smaller than in the case of data transfer. This means that in most cases the entire message can be conveyed to the receiver through a single event. Figure 3 highlights the differences between an event schema and an event.

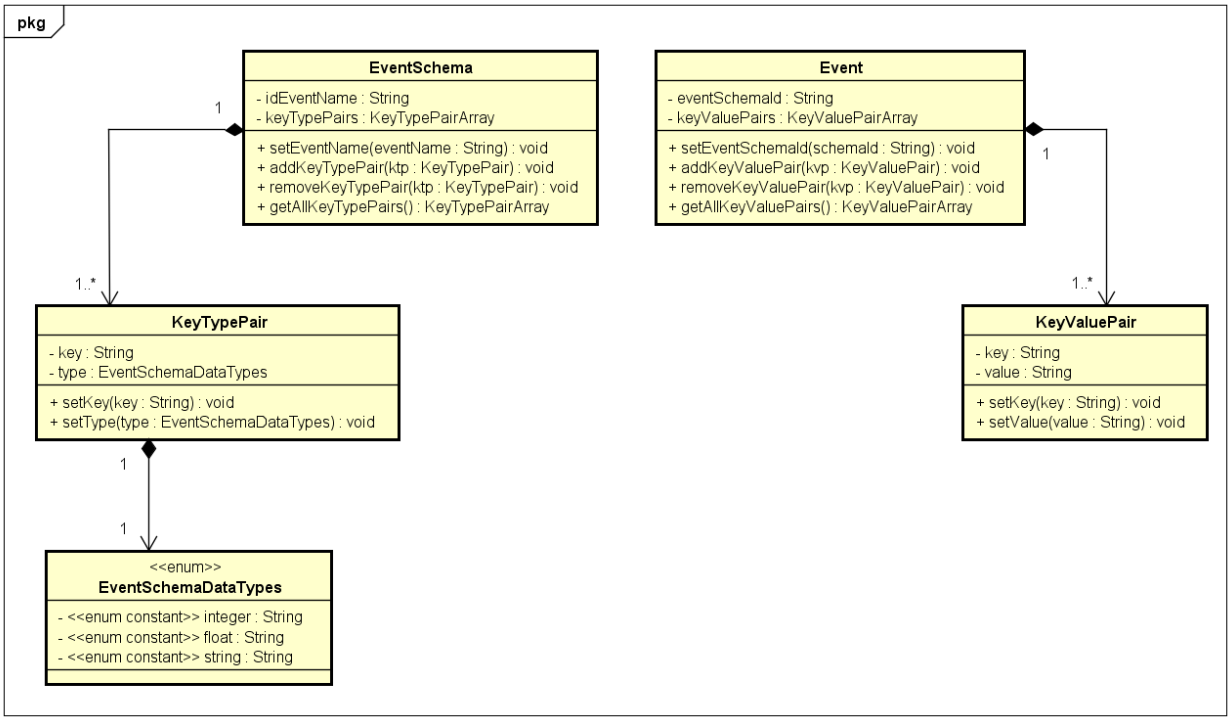

Figure 3 Design event schema and events

Each event is associated with a single event-type. An event-type is defined by a single event-schema [77]. Every event-schema contains a name for the event-type and one or more key-value pairs as attributes. The event name is the unique identifier used to identify an event-type. It should be a string that can contain lowercase alphabets and digits, but the name cannot have a digit as its first character.

Each event attribute is a key-value pair, where the key is the identifier for the corresponding value with the event instance as the scope. In the case of an event-schema, all the values represent their types. The event schema can use exactly one of the three permissible types which are integer, float and string.

Context elements and context sets

Previous work in the area has shown that the definition of *context* is highly dynamic [78, 79]. One of the few facts regarding context that is widely accepted by the community is that context of an entity can be formulated from the contexts of its subordinate components. By using this idea, context can be thought of as divided into hierarchical levels, where each successive level contains information that is based on lower levels of contextual information. For instance let's consider a four way traffic intersection that has multiple cars on each side of the road. Each car in itself has its own context that is fully sufficient for all the systems that are dependent on it. The context of an individual car can contain information like the

amount of remaining fuel, the identity of the driver, identity of the passengers, the destination address, the source address, historical data about the car, etc. The overarching context of the intersection contains information that is derived from the context of individual cars, and some additional information available only to the intersection. It can contain information like the number of cars waiting at the intersection, the side of the intersection that has the highest amount of traffic. In this case, the context of the intersection can be said to be at a higher level than that of the individual cars. Figure 4 represents the relation between environments, context sets and context variables.

Certain contextual elements can be derived indirectly by reasoning based on events from different sources that are otherwise unrelated by using logical sensors [30]. For example, it might be safe to infer that the user is in a meeting if based on the location event from the users phone and a calendar schedule event from the users planner. If the event from the planner signals that a meeting has started and the location event signals that the user's current location is in the office, it is safe to say that the user is present in the meeting. However, the user's phone and the calendar would on their own be unable to make such an inference as the information is not available to any single one of the components, but only to an overarching environment.

Context-elements can be grouped together to define *Context-sets.* A context-set can contain zero or more key-value pairs from different context-elements. The context-set of a component is the collection of all context-elements that are comprise its context. Referring back to the traffic intersection example, the context-set of the traffic intersection will contain all the context-elements from the individual context-sets of all the cars on the traffic intersection. The context-set of an environment is the super-set of all the context-sets of its subordinate components and environments.

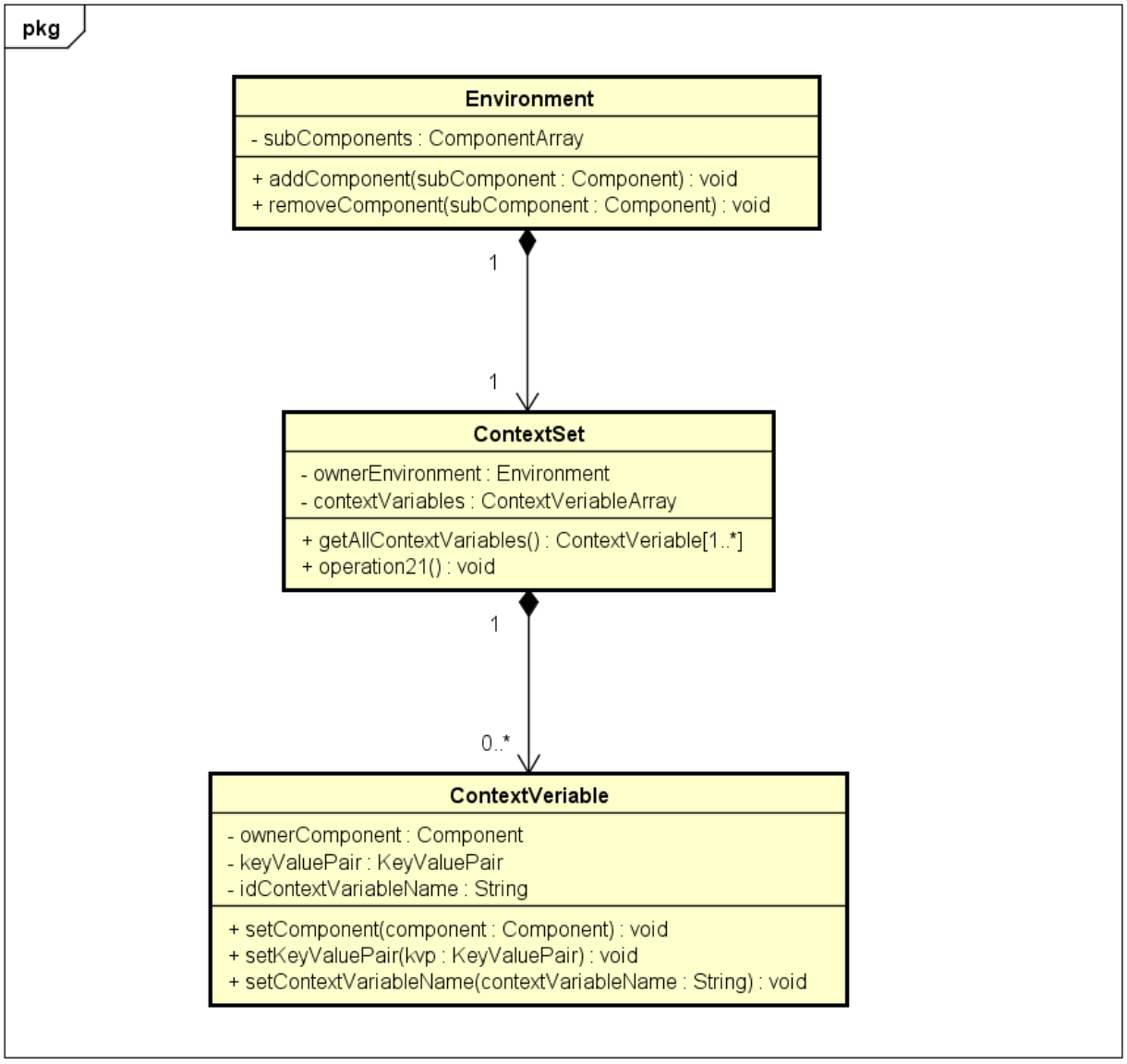

Figure 4 Design context elements and context set

Advertisements and subscriptions

As in the case object oriented languages, where objects are instances of classes, in this model events are instances of event-schemas. This means that when an event is said to be of a certain type, then it is guaranteed to contain all the information mentioned by the *event-schema and* in the correct format. These guarantees are important as it means that the component receiving the event will never receive a malformed event that has no utility for the component. Figure 5 highlights the differences in the composition of advertisements and subscriptions with respect to filters.

Components can express their interest in receiving the events of a specific event-schema by means of event subscription. An event subscription is a contract between a component and an environment that contains the component. It requires the environment to forward events of a specific type to the component, subject to some conditions. Apart from uniquely identifying the component and the environment, the subscription is bound to a single event-schema. This means that the environment is not allowed to forward events of a different type to the component. If for example a component has a single subscription for the event type 'x', then it can only receive the events of that of type 'x' and no other type. Event subscription can also contain an expression for context filter and a separate expression for content filter associated with the subscription. The filter expressions can be empty, in which case the respective filter acts like an all-pass filter.

The subscription context filter is computed by the environment and uses the context-set of the environment rather than the context-set of the component. This means that the context variables that are not accessible by the component can be used for the context filter. This is useful for enabling the context filter to exploit a wider set of environmental conditions. For example, if the climate control is interested in the temperature event, but only when the user is in the vicinity of the room, conventionally such a feature will need the modification of the climate control to have a built-in proximity sensor however, by using the general design model, such features can be easily implemented without the modification of the component's individual capabilities and by utilizing the already available information, the climate control can subscribe to the temperature event and use a context filter that uses the location event from the user's mobile phone for achieving the feature. Figure 7 depicts the details about the composition of the context filter. Figure 8 highlights the differences in the terms used to specify content and context filters.

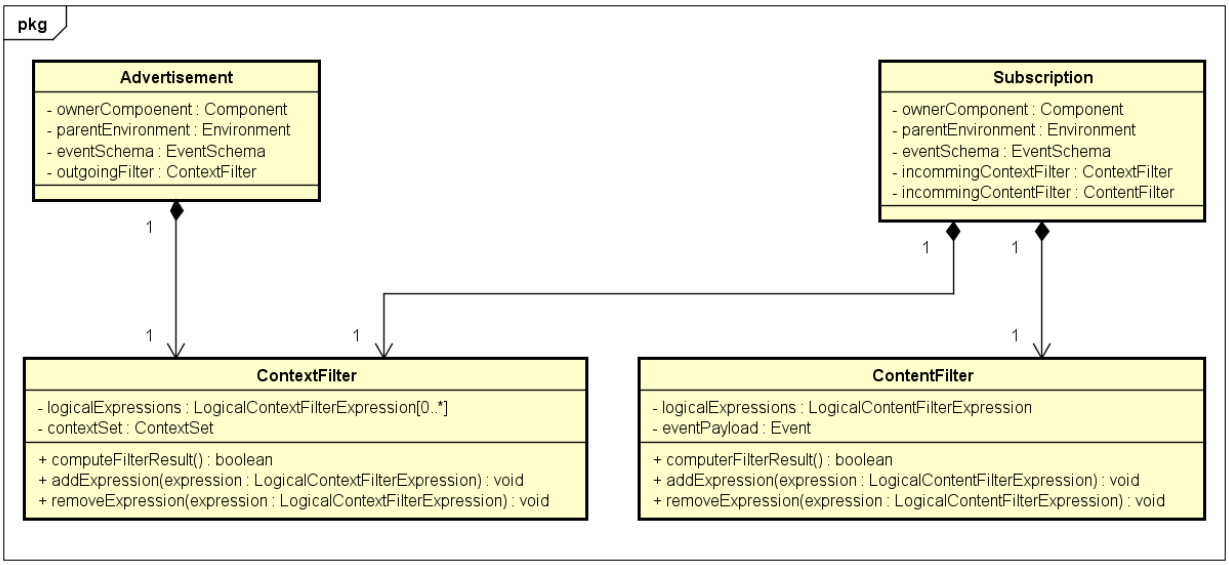

Figure 5 Design advertisements and subscriptions

The subscription content filter is computed by the component itself based on the contents of the event. These filters are useful when the component needs to narrow down its interest in an event based on the contents of the event. For example in the climate control example, if the climate control is interested in the temperature event only when the temperature is beyond a certain threshold limit, then it can use the content filter that subjects the contents of the event to the condition that verifies that the temperature is beyond the threshold. Figure 6 presents the details regarding the composition of the content filter. Figure 8 highlights the differences in the terms used to specify content and context filters.

Components can communicate their ability and desire to publish certain kinds of information by using advertisements. An advertisement is a contract between a component and an environment that contains the component. It requires the environment to forward events of a certain type published by the component. Each advertisement uniquely identifies the participating component, environment and also the type of the event to be published, i.e. the component is not allowed to publish types of events other than those mentioned in the advertisement. For example, a component has a single advertisement for publishing the events of type ‘x’, then it is only allowed to publish events of type ‘x’ and no other. The advertisement can also provide a filter expression for the context filter associated with the advertisement. The context filter expression can be empty, in which case the filter acts like an all-pass filter.

The advertisement context filter is computed by the environment and uses the context-set of the environment rather than the context-set of the component. This means that the context variables that are not accessible by the component can be used for the context filter. This is useful to enable the context filter to be dependent on rich environmental conditions. For example, if an office intercom system is

interested in connecting the call only when the user is not in a meeting, conventionally such a feature will need the modification of the intercom system to incorporate the information about the user's schedule however, by using the general design model, such features can be easily implemented without the modification of the component's individual capabilities and by utilizing the already available information. The intercom system can advertise the call event with the appropriate context filter expression that uses the schedule events from the user's calendar. If an event does not pass the advertisement context filter, it is not propagated further. This means that the event publisher has control of the events and can decide the conditions in which the events published by it are relevant. Figure 7 depicts the details about the composition of the context filter.

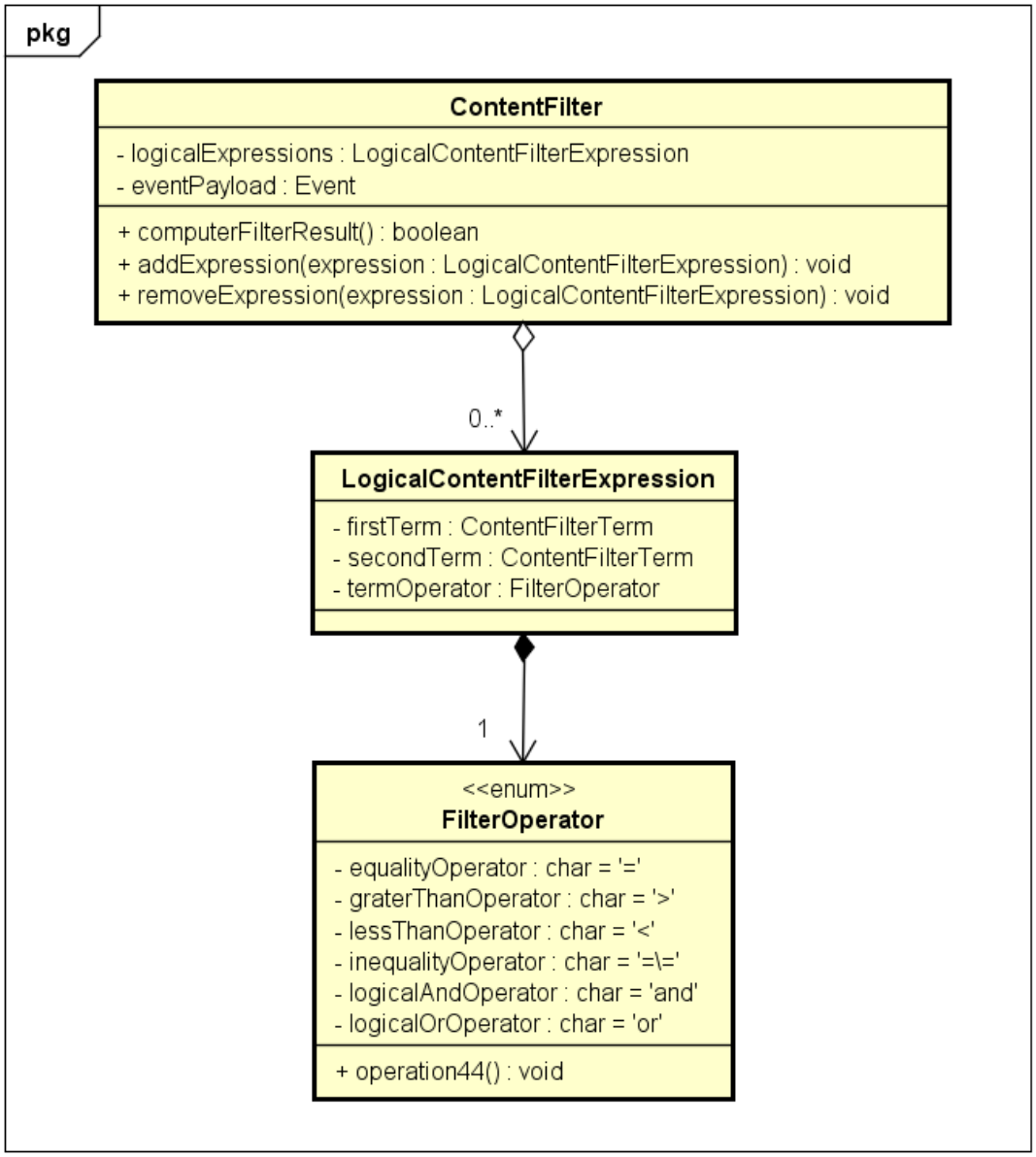

Figure 6 Content filter design

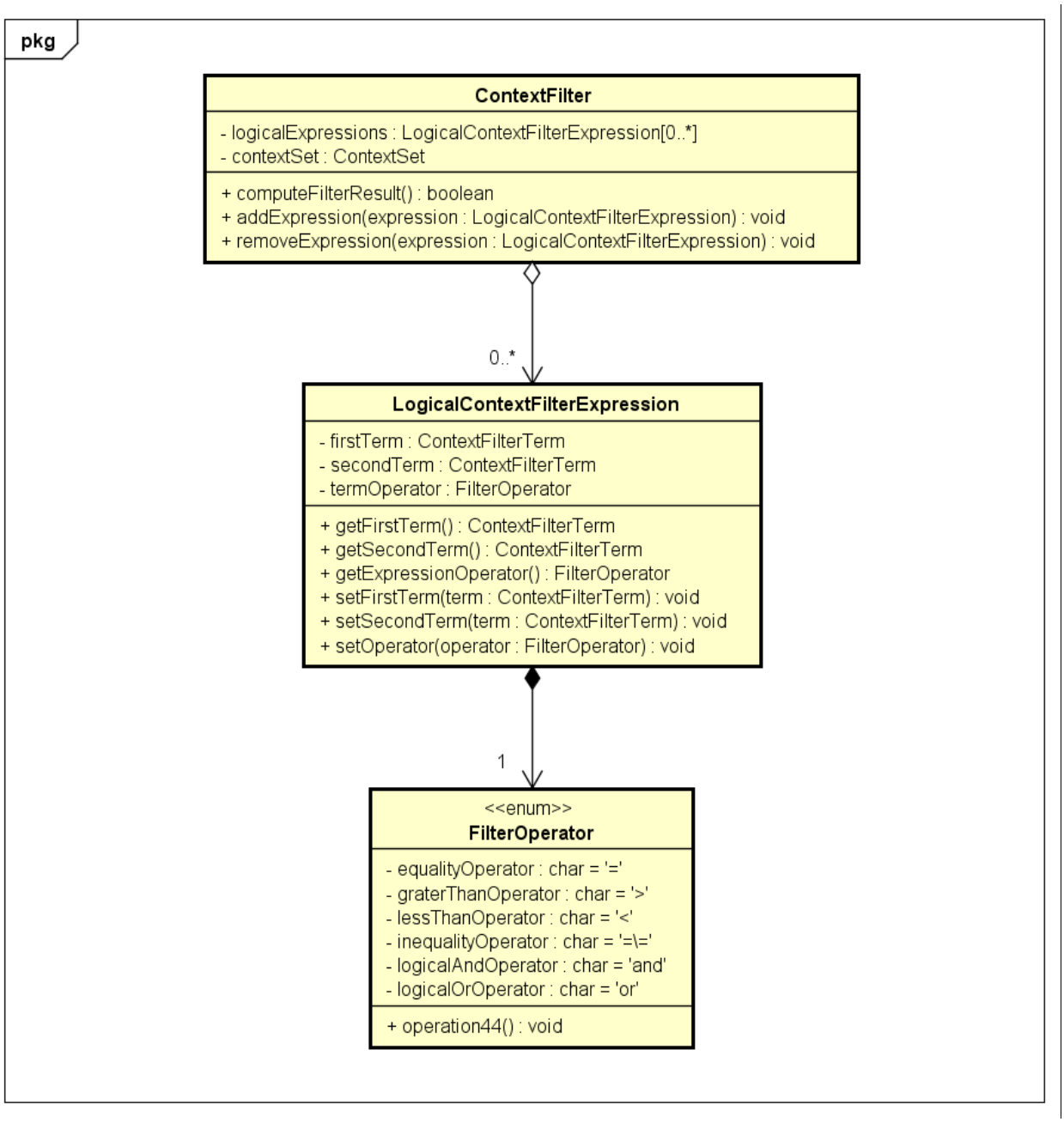

Figure 7 Context filter design

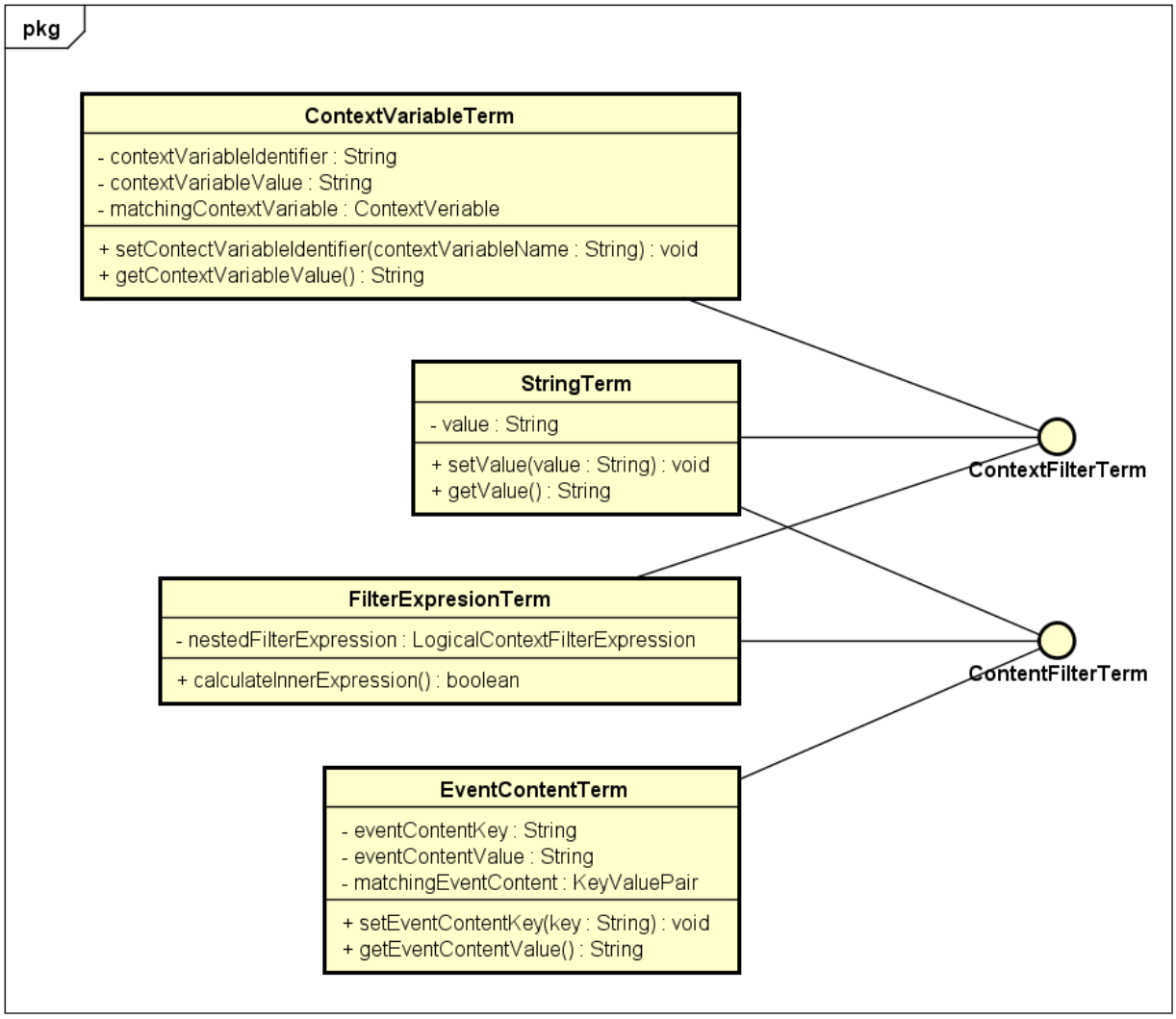

Figure 8 Content & context filter terms

3.1.1.2 Case study informal design models

As shown in Figure 1, the informal case study design models need to be based on specific case study requirements. The informal case study design models are extensions of the informal general design model. More details regarding the informal models of the case studies are presented in Chapter 4.

3.1.2 Formal models

The formal general design model and the case study design models are based on the informal general design model and the informal case study design models respectively. In addition, according to Figure 1, the formal case study design models are extensions of the formal general design model.

3.1.2.1 General formal design model

This section will present the formal general design model. These specifications are represented in XSB-Prolog [80].

Brief discussions about XSB Prolog have been presented in the previous chapters. The language manual defines XSB as a research oriented Logic Programming System. The most prominent way XSB-prolog is different from other popular implementations of Prolog like SWI-Prolog, GNU-Prolog is that XSB uses *tabled resolution.*

Tabled resolution is useful for recursive query computation, allowing programs to terminate correctly in many cases where other versions of Prolog could not. This makes XSB very useful for applications that require parsing, program analysis, model-checking, data mining, diagnosis and temporal reasoning.

The formal specification of the model has been divided into six files with names listed below,

1) creation_script_db.P
2) nested_list_approach.P
3) context_filter.P
4) xmc_utils.P

The source code for the above files is provided in Appendix A. Apart for the source code the only thing needed to reproduce the study are the XSB-prolog and XMC platforms. Files (5) and (6) are specific to the case studies that were used to validate the properties of the general design model based on real-world scenarios. The description about the contents of each file is presented below.

3.1.2.1.1 creation_script_db

The file provides the definitions for the predicates that can be used to create a hierarchy of components and environments, and it also provides an API that can be used to define different elements and relations involved in the case study.

The implementation for the basic protocols for event dissemination through the hierarchy of the components are also defined in the file. The predicates that are a part of the event distribution protocols use more specialized predicates provided by *nested_list_approach* file.

Table 1 provides some of the file's predicates, for aiding better understanding about it's contents. For more details about the file, please refer to *Appendix A.1*, which contains the complete implementation of the file in *XSB-prolog*.

Predicate Signature	**Description**
`create_environment`	This predicate is used to create a new environment. The predicate is a part of the interface that this file defines for the definition of the environment-component hierarchy. After the successful execution of this predicate the environment thus created can be used in the creation of other elements or relationships.
`create_subscription`	This predicate is used to create a new relation between the elements that have been previously defined. This predicate is a part of the set of predicates that define the interface for the creation of components and components. After the successful execution of the predicate a subscription relation is defined between the event schema and the component.
`simulate_event_component`	This predicate is used when there is a need to trigger an event in the component, the predicate will start the dissemination of the event through the hierarchy of components that have be previously defined. The predicate completes its execution after the distribution of the entire hierarchy has been completed through the consideration of all the components and the relations.

`get_reachable_environments`	This predicate is part of the internal predicates that are used by other higher level predicates for the dissemination of an event at each component that the event encounters. The event dissemination protocol decides on its next component by using recursive calls of this predicate for each component the event visits. The predicate enforces the protocol for the event distribution.

Table 1 Sample predicates for file creation_script_db

3.1.2.1.2 nested_list_approach

This file provides predicates that are used by higher level predicates in the *creation_script_db* file. The file defines specialized predicates that are used to handle the nested list data structure that has been used for the implementation of multiple concepts in the modelling of the general design model.

Table 2 provides some of the predicates contained in the *creation_script_db*, for aiding better understanding about the contents of the file. For more details about the file, please refer to *Appendix A.2*, which contains the complete implementation of the file in *XSB-prolog*.

Predicate Signature	**Description**
`unify_event`	This predicate is used to validate the event before it is disseminated through the component hierarchy. The predicate completes successfully if the event instance matches the event schema that has been previously defined. The predicate requires both the event instance and event schema to be in the nested list format. The predicate compares the length of the event instance and the event schema before the more complicated matching of individual key-value pair contained in the nested list.
`compare_elements`	This predicate is used by the `unify_event` predicate that is described previously. This predicate compares iterates the nested lists by making recursive calls to itself.

`flatten2`	This predicate is used to convert the nested list into a flat list of depth of one. This is useful for many higher level functionalities.

Table 2 Sample predicates for file nested_list_approach

3.1.2.1.3 context_filter

The predicates in this file are used to compute the result of the *content-filter* and the *context-filter* based on the filter expression that has been previously defined. The file contains predicates that can be used for computing both the content and context filters. However, the two filters need to be handled in slightly different way. The content filter is computed against the contents of the event instance itself. The context filter is computed against the context-set of the parent environment.

Presented in Table 3 are samples of some of the predicates contained in the *context_filter*, for aid in a better understanding about the file's contents. For more details about the file, please refer to *Appendix A.3*, which contains the complete implementation of the file in *XSB-prolog*.

Predicate Signature	**Description**
`can_pass_event`	This predicate is used to compute the result of a content-filter, the filter is computed by using the filter expression and the contents of the event instance. The predicate completes execution if the event is found to pass the filter.
`faltten_to_key_value_pairs`	This predicate is used to convert the nested list into a flat list of key-value pairs of maximum depth of two. This predicate is used by the `can_pass_event` predicate.
`compute_logic_molecule`	This predicate is used to compute the result of a single expression involving 2 terms and an operation.
`compute_logic`	This predicate is used to compute a simple expression involving a single king of operation and 2 terms.

Table 3 Sample predicates for file context_filter

3.1.2.1.4 xmc_utils

The predicates in this file are used by the XMC processes to interact with the XSB code. The predicates are used for making the implementation of XMC process simpler. The file enhances code reuse as the predicates contained are used by XMC for both the case studies.

Presented in Table 4 are some of the predicates contained in the *xmc_utils*, for aiding better understanding about the contents of the file. For more details about the file, please refer to *Appendix A.4*, which contains the complete implementation of the file in *XSB-Prolog*

Predicate Signature	**Description**
`assertTriggPredicate`	This predicate is used for recording that some event was received by a component. This predicate is executed by the event response predicates defined in case study specific files.
`cleanupComponentTriggPredicates`	This predicate is used by XMC process to clear the database of any previous records about receiving events by a component. This predicate is useful when the XMC property requires multiple instances of event reception.
`hasComponentReceivedResult`	This predicate is used by the XMC to check if a component has received an event of a specific kind. The predicate checks the XSB database for records that could be previously stored by `assertTriggPredicate`.
`no_duplicates`	This predicate completes successful execution if the list provided to it does not have any duplicate members.

Table 4 Sample predicates for file xmc_utils

3.1.2.2 Case study formal design models

As shown in Figure 1, the formal case study design models need to be based on the informal case study design models. The formal case study design models are extensions of the formal general design model. More details regarding the formal case study design models are presented in Chapter 4.

3.2 Validation

This section will describe how the formal case study design models are validated. The validation uses a model checker to validate properties of these models. These properties are based on the case study requirements.

The model checker used in the validation is XMC [73]. The XMC model checker supports the specification of both processes and properties for the validation of the models. The formal general design model is achieved by using an end-user logic programming language Prolog, and the process and the property specifications are achieved by using the platform specific language of the model checker called XL.

The validation was done using the model checker provided by XMC. The XMC model checker is seamlessly able to integrate with the general design model which is defined using XSB. The compatibility between XSB and XMC was the primary reason behind the choice of language.

The XMC system contains a compiler for XL, a model checker for XL and μ-calculus. XMC itself is implemented using XSB Prolog, which supports the ability to integrate XSB models with XMC. This study used XMC v1.0 and XSB v2.6. Based on my experience, installing XMC was not trivial, as it was found that most of the XMC versions were compatible with only specific XSB versions. The selection of the computing platform for the XMC-XSB system was also more complicated than expected. The reason for this might be that the system is actively being developed and lacks adequate documentation. After attempting to compile the XMC-XSB system successfully on many platforms, the entire environment was setup successfully by using an older version of Linux, Redhat version 9, nicknamed *Shrike*.

The XL scripting language has two distinct uses, the first is the definition of processes based on the value passing Calculus of Communicating Systems (CCS) [81], and the second is a property definition language based on μ-calculus. The XL syntax that governs both the aspects of the language is presented in section 3.2.3. The source code for the formal general design model and the properties are presented in further detail Section 3.1.2.1.

3.2.1 Structural properties

Structural properties define the relationships between the components. A structural property can be considered to have the following form:

$$H \leftarrow B_1, \ldots, B_n, n \geq 0 ,$$

H, $B_1, \ldots, B_n$ are atoms. H is the head of the property, whereas, $B_1, \ldots, B_n$ are the body of the property.

3.2.2 Behavioral properties

Behavioral properties describe the range of valid operations that can be performed by a component. A behavioural property ϕ is defined using μ-calculus expressions. The properties are expressed using fixed point equations, where μ is the operator used to signify a least fixed point equation and the ν operator is used for the greatest fixed point equations. The least fixed point equation starts its computation from the minimal element and then expands to a greater element iteratively. For the greatest fixed point equation the computation starts from the greatest element and then reducing iteratively.

The modal μ-calculus supports the use of constants (tt) and (ff), and the standard logical connectors' disjunction ($\vee$) and conjunction ($\wedge$). The modal μ-calculus supports actions. The box modality ($[a]\phi$) represents that the behavioral property ϕ holds for all the single-step reachable states by an action called a transition. The diamond modality ($<a>\phi$) represents that it is possible for an action to lead to a state for which ϕ holds true. It is common practice to use the ν formulae for safety properties that require something to never happen, whereas it is also a common practice to use the μ formulae for liveliness properties that require something to happen. In short, it is safe to say that ν is for safety and μ is for liveness.

The syntax of μ-calculus is as follows,

$$\phi ::= Z \mid tt \mid ff \mid \phi \wedge \phi \mid \phi \vee \phi \mid <A>\phi \mid [A]\phi \mid \mu Z.\phi \mid \nu Z.\phi ,$$

where Z is a set of formula variables, A is a set of actions, tt and ff are propositional constants, $\wedge$ and $\vee$ are logical connectors, $<A>\phi$ denotes that possibly after action A, the formula ϕ holds, $[A]\phi$ asserts that after action A, the formula ϕ will always hold, the formula $\mu Z.\phi$ and $\nu Z.\phi$ stand for the least fixed point and the greatest fixed point.

3.2.3 XMC syntax

The XMC system contains a compiler for XL. The XL language performs two distinct functions. The first is the definition of processes based on the value passing Calculus of Communicating Systems (CCS) [81], and the second is a property definition language based on μ-calculus. This chapter presents the syntax of the XL language.

XL is a language for specifying asynchronous concurrent systems. It inherits the parallel composition (written as '|'), and choice operators ('#'), the notion of channels, input ('!') and output ('?') actions, and synchronization from Milner's value-passing calculus. XL also has a sequential composition (';'), generalizing CCS's prefix operation, and a built-in conditional construct ('if'). Complex processes may be defined starting from the elementary actions using these composition operations. Process definitions may be recursive; In fact, as in CCS, recursion is the sole mechanism for defining iterative processes. Processes take zero or more parameters.

3.2.3.1 XL Process Definition Language

One of the basic objects in XL is a process. Processes are defined using the infix ::= operator. The left hand side of a definition is a process term, the name of the process, along with a set of variables that form the parameters of the process. The right hand side is a process expression. Process definitions are terminated by a period ('.').

A process p with n parameters is invoked as $p(t_1, t_{2...} t_n)$, where t_i are arbitrary terms whose types are consistent with the corresponding parameters in the definition of p. A process is distinguished by its name and the number of parameters it takes. In other words, one can define many distinct processes with name p as long as the number of parameters are all different. By convention, a process p that takes n parameters is written as p/n.

Computation

An XL process can intermingle with computation. All variables in XL are single-assignment variables. Computation is specified as a Prolog predicate, either defined in the same specification file using inline Prolog or imported from a different file or library. Prolog predicates may be defined in an XL specification, by enclosing the definition between a '{*' and '*}'.

Terms can be compared using the following relational operations: ==, \== for testing equality and inequality of arbitrary terms; >=, =<, > and < for inequalities over terms representing integers; = for unification of two terms, and \= for testing the inability to unify of two terms. Relational expressions may be built from these base comparisons using conjunction (/\), disjunction (\/) or negation (~).

Communication

Written as C?A or C!A, these expressions denote values (A) read from or written to a channel (C). For a channel C that is used for signalling (i.e., no values are passed along it), XL uses C?* and C!* to denote sending and receiving signals along them.

The expression C!A , the value A needs to be associated with a value when the output function is enabled. The input expression C?B, most often B is simply a variable when B is a term, synchronization occurs only when the value specified in the corresponding output action on channel C unifies with B.

3.2.3.2 XL Property Specification Language

XMC currently supports properties specified in alternation-free modal mu-calculus. The mu-calculus formulas are written in an equational form, and supports parameters and values. Properties are specified using fixed point equations, where each equation is of the form *f* + = *e* or *f* - =*e*. The operator + = denotes a *least fixed point* equation and - = denotes a *greatest fixed point* equation. These symbols reflect the computation itself: least fixed points are computed by starting from the minimal element and iteratively expand it; greatest fixed points start from the maximal element and iteratively reduce it.

The left hand side of a fixed point equation denotes the (parameterized) name of the formula, and the right hand side (RHS) is the expression denoting the formula itself. The RHS is a Boolean expression (using connectives ∨ and ∧ denoting disjunction and conjunction respectively) of modal expressions, and two base propositions: *tt* denoting true and *ff* denoting false. Modal expressions are formed by prefixing a mu-calculus expression with a modality: < > denoting diamond, or existential modality and [ ] denoting box, or universal modality.

Chapter 4

4. Case studies

This chapter presents the two case studies to illustrate the study approach. For each case study, first I present the informal case study design models and use real world scenarios that these cases involve, and then I formalize these models to obtain their formal specifications. After that, I define properties specific to each case study, and validate the formal models with respect to their properties. The specification for the each of the formal case study design models is presented in Appendix B.

The properties are specified using the XMC XL property specification language, and the scenarios are specified using the XMC XL process specification language. The properties are validated against the case study specifications. Because the XMC model checker is built upon the XSB Prolog, the XMC XL processes use specific XSB Prolog predicates. Figure 9 highlights within a box, the different activities involved in the validation of the formal case study design models.

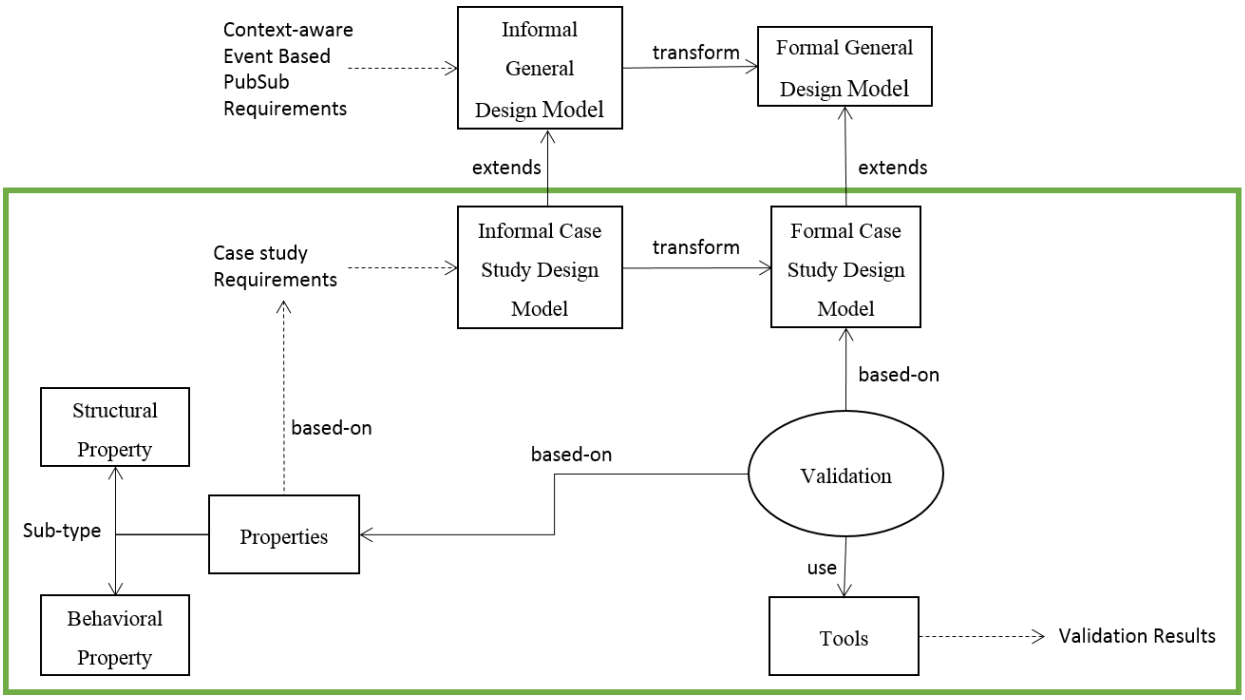

Figure 9 Approach for case study validation

4.1 Cellphone behavior adaptation

This case study involves the investigation of the mechanisms involved in changes in the behaviour of a cellphone triggered by events from other components in the system. Figure 10 depicts the arrangement of the components involved in the case study and, as shown in Figure 10, the cellphone contains the calendar application. Figure 11 presents the events that are involved in the case study.

4.1.1 Informal model

4.1.1.1 Participants

- Cellphone
- Calendar Application

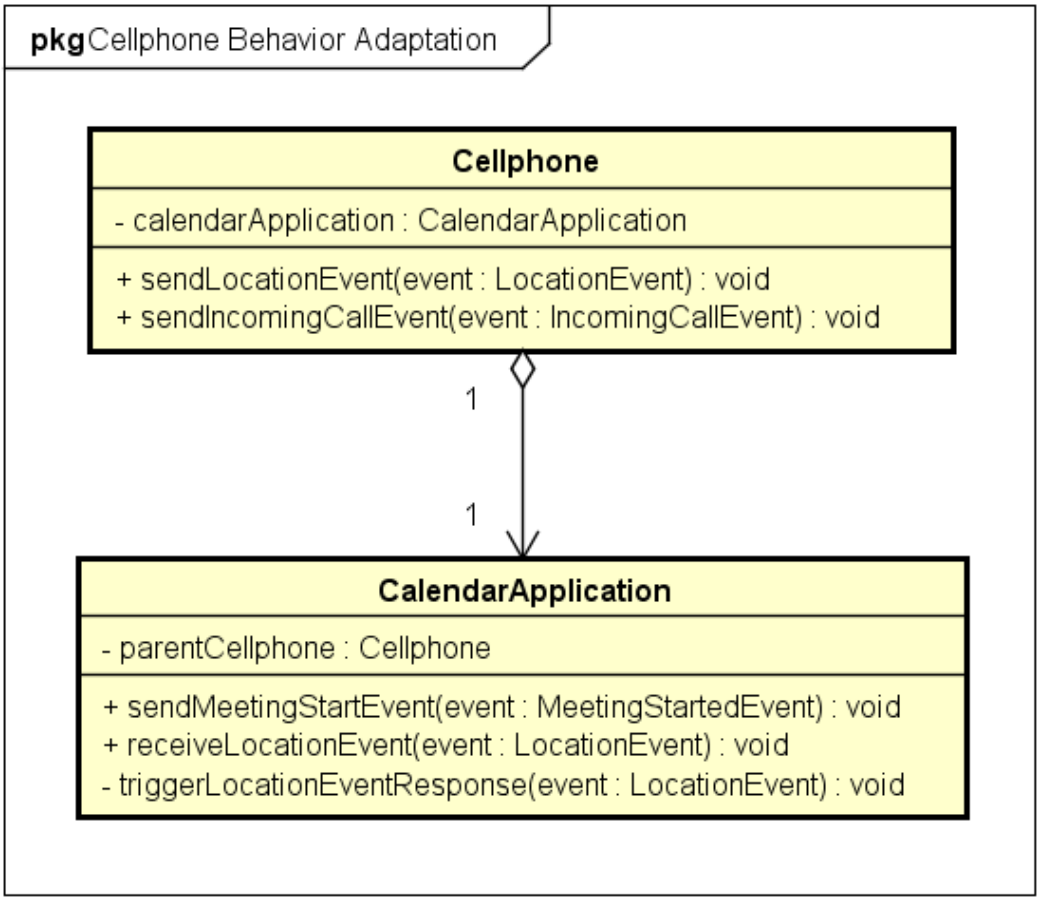

Figure 10 Cellphone behaviour adaptation - Components

4.1.1.2 Scenario

- The calendar application generates the *meeting started* event. This happens when the time changes to the time of a scheduled meeting.
- The cellphone generates the *location event* to signal a change in the user's location.
- The cellphone generates the *incoming call event.*
    - If the meeting has not started yet i.e., the *meeting started event* has not yet been received, then the call will be allowed to pass through to the user.
    - If the meeting started event has already been received, but the latest location event has signalled that the user is not at the office, then the call is allowed to pass through to the user.
    - If the meeting started event has already been received and the latest location event signals that the user is at the office and the call is not from a preferred caller, then the call is not allowed to flow to the user and is blocked by the cellphone.
    - If the meeting started event has already been received and the latest location event signals that the user is at the office and the call is from a preferred caller, then the call is allowed to flow to the user.

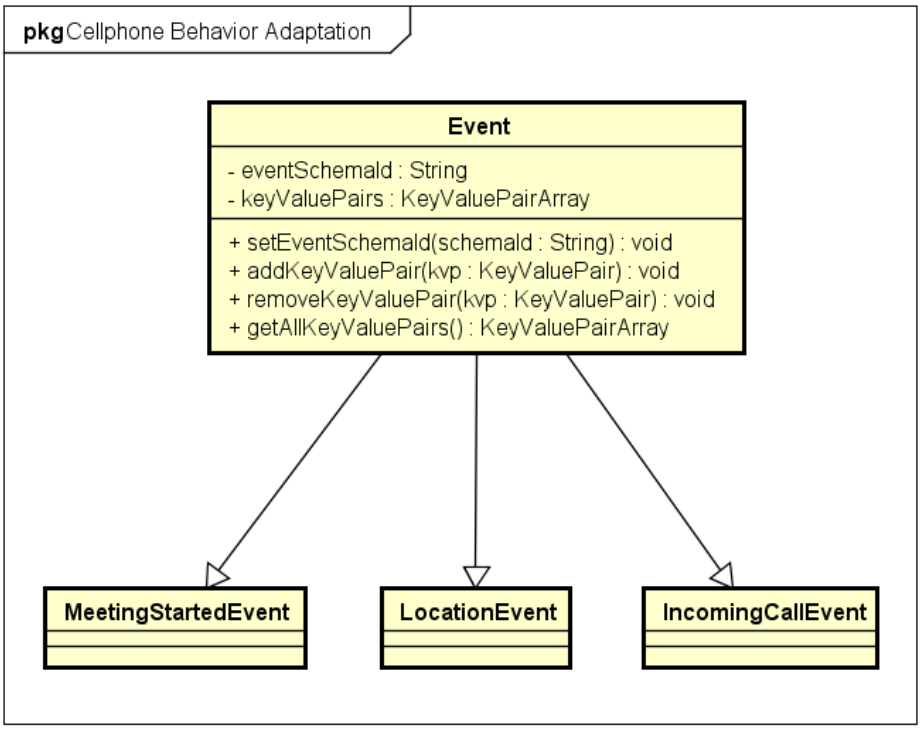

Figure 11 Cellphone behaviour adaptation - Events

4.1.2 Formal model and validation

The formal model for the cellphone behaviour adaptation is validated against the properties relevant to this case. The XMC properties and processes for the case study are defined in the *t1.xl* file. Table 5 presents some of the process and properties that are specified in this file for this case-study. For more details, please refer to the *Appendix C.1,* which contains the complete specifications in XL.

XMC Process	XMC Property	Description
`send_before_receive_after_cs_one`	`causality_cs_one`	This property verifies that an event can only be received by the target component after it has been sent by the sending component.
`runtime_validity_cs_one`	`runtime_validity_property_cs_one`	This property checks if the hierarchy of components is consistent while events are travelling between components.
`unique_components_cs_one`	`uniqueComponent_cs_one`	This property verifies that all the components in the hierarchy are unique. This means that all the components have a unique name.
`unique_subscriptions_cs_one`	`uniqueSubscription_cs_one`	This property verifies that all the advertisement event schema-component relations are unique. This means that there should be exactly one advertisement relation involving an event schema and a component.
`chk_advertisement_relation_cs_one`	`valid_adv_relations_cs_one`	This property verifies that an advertisement can only be created for the event schemas that have been defined prior to the creation of the advertisement.

chk_env_component_relation_cs_one	valid_env_component_relations_cs_one	This property verifies the validity of the environment-component hierarchy by ensuring that an environment cannot contain a component that has not been properly defined prior to the definition of the relation.
chk_event_without_subscription_cs_one	never_receive_event_without_subscription_cs_one	This property verifies that for a component to receive any event it needs to be properly defined prior to sending the event through the component hierarchy.

Table 5 Sample properties for case study cellphone behaviour adaptation

4.1.2.1 Formal validation

This section presents the specification of some properties and processes and the automated validation of some of the process specifications against the properties using XMC. To illustrate how I perform the automated formal validation, I provide the definition of two properties that were validated. First, the definition of the process `send_before_receive_after_cs_one` is provided, which represents the scenario in which events are sent and received:

```
send_before_receive_after_cs_one ::=
    chk_send_event
    ; chk_receive_event
    .
```

The property called `causality_cs_one` is specified as follows:

```
causality_cs_one += [recvnew]tt \/ ([-]causality_cs_one /\ [sendnew]tt).
```

When `causality_cs_one` was validated against the process specification it was found to be true.

Second, the definition of the process `runtime_validity_cs_one` is provided, which represents the scenario where we check the structural validity of the model while an event is sent:

```
runtime_validity_cs_one ::=
```

```
chk_send_event

| chk_relations_duplicates_cs_one

.
```

The property called `runtime_validity_property_cs_one` is specified as follows:

```
runtime_duplicates_property_cs_one += unique_all_cs_one.

runtime_relations_property_cs_one += all_valid_relations_cs_one.
runtime_validity_property_cs_one += unique_all_cs_one /\
all_valid_relations_cs_one.
```

When `runtime_validity_property_cs_one` was validated against the process specification it was found to be true. Some of the other properties that were found to be true are provided in Table 6.

Description	**XMC Property**
A component that is not an environment can not contain components	`valid_env_component_relations_cs_one`
A component cannot receive an event type for which the event schema has not been defined	`never_receive_undefined_event_cs_one`
A component cannot send an event type for which the event schema has not been defined	`never_send_undefined_event_cs_one`
A subscription cannot be created for an event type for which the event schema has not been defined	`valid_subs_relations_cs_one`
An advertisement cannot be created for an event type for which the event schema has not been defined	`valid_adv_relations_cs_one`
A subscription cannot be created for a component that is not defined	`valid_subs_relations_cs_one`
An advertisement cannot be created for a component that is not defined	`valid_adv_relations_cs_one`
All events can have one and only one definition of event schema	`uniqueEvent_cs_one`
All advertisement relations should be unique	`uniqueAdvertisement_cs_one`
All subscription relations should be unique	`uniqueSubscription_cs_one`
All environment-component relationships should be unique	`uniqueEnvironmentComponentRelation_cs_one`

Table 6 Cellphone behaviour adaptation other properties

For further details about the specification of other processes and properties used in the validation of this case study model please refer to Appendix C.1.

4.2 Cellphone as controller

This case study, observes the scenario in which the cellphone triggers adaptations in other components. Initial events from the cellphone trigger a chain of events between components that leads to a multifaceted change the system. Figure 12 shows the relationships between the components involved in the case study and, as shown in Figure 12, the case study involves numerous nested environments. Figure 13 shows the events used in this case study.

4.2.1 Informal model

4.2.1.1 Participants

- User's Cellphone
- Alarm application
- True Remote Control Application (TRCA)
- Smart Home Controller Unit (SHCU)
- Home Security System
- Home Climate Control Unit
- Car Computer Controller Unit (CCCU)
- Car Engine Starter

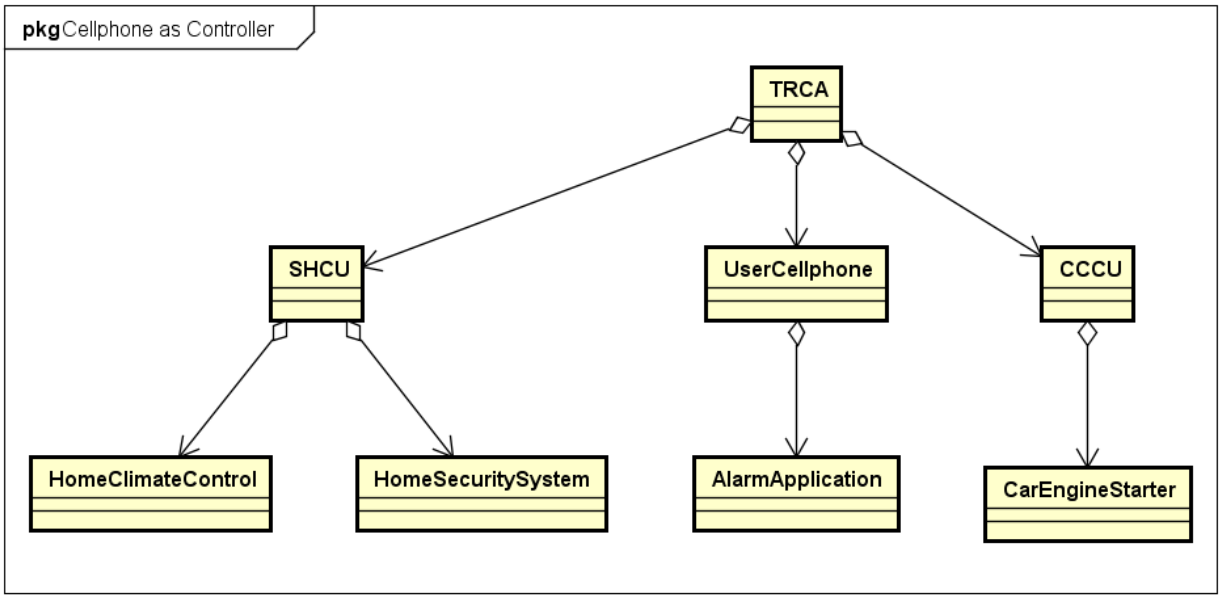

Figure 12 Cellphone as a controller - Components

4.2.1.2 Scenario

- The alarm application sends an *Alarm Event.*
- The cellphone generates the *Cellphone Time Change Event*. This signals that some time has passed after the alarm had gone off.
- The cellphone generates the *Location Event.* This signals that the user is still at home after the alarm event and the time changed event.
- The TRCA generates the *Remote Starter Action Event*. This signals the CCCU to start the car engine to prepare for the users departure.
- The CCCU responds to the remote starter event with the *Result of Remote Starter Event.* This event carries as its payload if the engine was successfully started by the car engine starter.
- The TRCA responds to the result of remote starter event with the *Notify User of Car Engine Started Event*. This event signals to the user cellphone that the car engine has started and it's ready for their departure.
- The user starts driving away from home. This is detected by the *Cellphone Subsequent Location Event,* and the event contains multiple successive locations of the user. The user is assumed to be moving away from home if the successive locations are further and further away from home.
- After the detection that the user is moving away from home, the TRCA generates two instances of *Trigger Home Action Event*. One of the instances carries instructions for the activation of the home security system, while the other caries temperature information for the home climate controller.
- On receiving the *Trigger Home Action Event* for security system, the SHCU activates the home security system and generates the *Result of Home Action Event. T*his event carries as a payload if the home security system has been successfully activated.
- On receiving the Trigger Home Action Event for climate control system, the SHCU configures the climate control component and then generates the *Result of Home Action Event*. This event carries as a payload if the climate control was successfully configured.
- The TRCA receives the Result of Home Action Event for security system it generates the *Notify User of Home Action Event.* This event notifies the user cellphone if the security system has been activated successfully.

- The TRCA receives the Result of Home Action Event for climate control system it generates the *Notify User of Home Action Event.* This event notifies the user cellphone if the climate control system has been configured successfully.

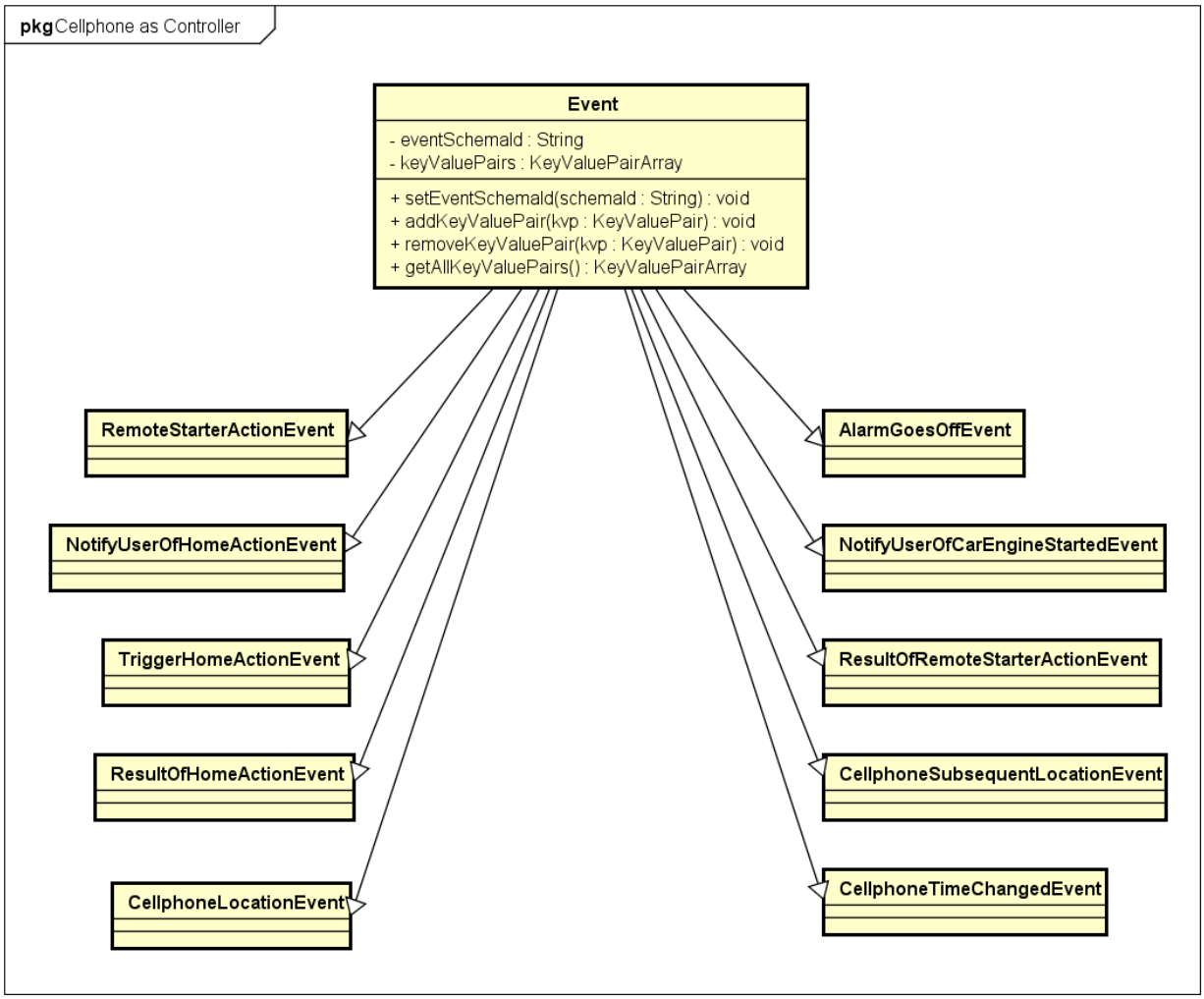

Figure 13 Cellphone as a controller - Events

4.2.2 Formal model and validation

The formal model for the cellphone as controller is validated against the properties relevant to this case. The XMC properties for the case study are defined in the *t2.xl* file. In Table 7 presents some of the process-property pairs defined in this file for the case-study. For more details please refer to the *Appendix C.2*, which contains the complete implementation in XL.

XMC Process	XMC Property	Description
`fidelity_chk`	`fidelity`	This property verifies that the contents of the events are not altered in any was while the event id transiting through the component hierarchy. This property ensures that no information is lost while the event is in transition.
`chk_disconnected_component`	`chk_component_connection`	This property checks an aspect of the event dissemination protocol defined by the general design model by checking if an orphan component can receive an event.
`unique_events`	`uniqueEvent`	This property verifies if all the event schemas are unique, this means that all the event schemas need to have unique names.
`testContextFilter(1)`	`filterNotBlocking`	This property verifies if the context filter of the receiving component functions properly. The process sends an instance of an event schema that should not be blocked according to the receiving components subscription.
`testContextFilter(0)`	`filterBlocking`	This property verifies if the context filter of the receiving

		component functions properly. The process sends an instance of an event schema that should be blocked according to the receiving components subscription.

Table 7 Sample properties for case study cellphone as controller

4.2.2.1 Formal validation

This section presents the specification of some properties and processes and the automated validation of some of the process specifications against the properties using XMC. To illustrate how I perform the automated formal validation, I provide the definition of two properties that were validated. First, the definition of the process `fidelity_chk` is provided, which represents the scenario in which the contents of events are checked to have not changed between sending and receiving the event:

```
fidelity_chk ::=
    send_event_for_fidelity
    ; receive_event_for_fidelity
    .
```

The property called `fidelity` is specified as follows:

```
fidelity += never_fidelity_fail.
never_fidelity_fail -= [fidelityTestFailed]ff /\ [-]never_fidelity_fail.
```

When `fidelity` was validated against the process specification it was found to be true.

Second, the definition of the process `chk_disconnected_component` is provided, which represents the scenario where a disconnected component is checked to not be able to receive any events:

```
chk_disconnected_component ::=
    disconnect_component
    ; chk_disconnected_component_send_notify_event_disconnected
    ; reconnect_component
```

The property called `chk_component_connection` is specified as follows:

```
possible_event_received_when_connected += <eventByComponentWhileConnected>tt
\/ <->possible_event_received_when_connected.

never_event_received_when_disconnected -=
[eventByComponentWhileDisconnected]ff /\ [-
]never_event_received_when_disconnected.

chk_component_connection += possible_event_received_when_connected /\
never_event_received_when_disconnected.
```

When `chk_component_connection` was validated against the process specification it was found to be true. Some of the other properties that were found to be true are provided in Table 8.

Description	**XMC Property**
An event can only be received after it has been sent	`causality`
The contents of the event should not be altered once it has been sent	`fidelity`
An event cannot be received if it not sent	`never_receive_spontaneous_event`
Event cannot be received if the target component is not subscribed for it	`never_receive_event_without_subscription`
Event cannot be sent if the component does not hold an advertisement of the event type	`never_send_event_without_advertisement`
Event cannot be received if it doesn't pass the context filter associated with the receiving component's subscription for the event type.	`filterBlocking`
Event cannot be received if it does not pass the content filter associated with receiving component's subscription for the event type	`contentFilterBlocking`
Event cannot be sent if the event does not match the event schema defined for the event type	`never_able_to_send_event_with_invalid_schema`

Table 8 Cellphone as controller other properties

For further details about the specification of other processes and properties used in the validation of this case study model please refer to Appendix C.2.

Chapter 5

5. Conclusion and future work

This study has demonstrated a novel approach for designing and validating of a context-aware publish-subscribe model. It uses a model checker to provide formal proofs of the properties related to the case studies. By using formal verification, it is hoped to encourage future work in terms of the exploration of additional cases studies and other properties. To my knowledge, this is the first study to investigate the application of formal mechanisms for the validation of a modular CAPS model based on the notion of environments and different levels of filters (e.g., type, content, and context).

The study also showcased the XSB-Prolog and XMC platforms and their capabilities to be used for the validation of frameworks and general design models that are often proposed in the research community. It is believed that the use of formal proofs for supporting claims in areas relating to software engineering will inspire greater confidence from the industry towards new ideas proposed by the researchers. A result supported by the formal proof obtained by a model checker will significantly assuage any doubts that are faced by the industry practitioners.

Below are some suggestions on possible topics for future work.

5.1 Additional case studies

More research effort can be focused on further validation of the general design model. The implementation of more case studies can be done with minimal effort by using the API developed in this thesis. Further validation can also be done by validating more advanced properties. Adding more properties to the validation can lead to improvement of the general design model and discovery of more areas of applications that are a good match for the properties.

5.2 Enriching the filter definition language

A promising future enhancement can be enriching the filter language that is used for the context and the content filters. The enhancement of the filter language will enable more complex adaptation rules to be implemented. This enrichment can be done by the introduction of more specialized operators or routines that can be used in the expression language.

5.3 Enhancement of legacy systems

Another area of future work is the enhancement of the general design model to be able to perform feature enhancement of legacy systems easily by using wrappers. A wrapper can enforce content and context filters to adapt the behaviour of legacy systems without requiring any changes to their own source code. Further research can explore the feasibility of the general design model in the context of Internet of Things (IoT) [82]. I feel that the general design model can provide feature enhancement to the cheap, low power sensors and actuators involved in IoT.

5.4 Development of a reusable framework

Finally, effort can be allocated towards the development of a reusable framework based on the general design model using widely adopted languages such as Java. The implementation of such a framework can support the development of software systems based on the general design model that use real-world components.

Appendix A General design model XSB source code

A.1 creation_script_db

```
% PREDICATE TO CREATE ENVIRONMENT WITH A LIST OF CONTEXT VARIABLES
create_environment(Environment_name, Context_var_list):-
    %not(environment_instance(Environment_name)),
    assert(environment_instance(Environment_name)),
    % CREATE A COMPONENT INSTANCE FOR EACH ENVIRONMENT SO THAT THE ENVIRONMENT CAN BE
NESTED INSIDE ANOTHER ENVIRONMENT
    assert(component_instance(Environment_name)),
    % ASSOSIATE CONTEXT VARIABLES TO THE NEW ENVIRONMENT
    add_context_variable_list_environment(Environment_name,Context_var_list).

% PREDICATE TO CREATE A COMPONENTWITH A LIST OF CONTEXT VARIABLES
create_component(Component_name, Context_variable_list):-
    %not(component_instance(Component_name)),
    assert(component_instance(Component_name)),
    % ASSOSIATE THE CONTEXT VARIABLES TO THE NEW CONTEXT

add_component_context_variable_list(Component_name,Context_variable_list).

% PREDICATE TO DECLARE A NEW EVENT SCHEMA, THE EVENT SCHEMA IS REPRESENTED IN A NESTED
LIST FORMAT WHERE DATA IS REPRESENTED AS KV PAIR
create_event_schema(Event_id,Event_schema_list):-
    assert(event_schema(Event_id,Event_schema_list)).

% PREDICATE TO CREATE AN ADVERTISEMENT FOR A COMPONENT, AN ADVERTISEMENT IS ASSOSIATED
WITH A COMPONENT, AN EVENT TYPE, AN ENVIRONMENT AND A CONTEXT FILTER. AN ENVIRONMENT
IS NEEDED BECAUSE ADVERTISEMENTS ARE NOT SHARED BETWEEN ENVIRONMENTS. THEREFORE IF A
sADVERTISEMENTS FOR EACH PARENT ENVIRONMENT. THE ADVERTISEMENT ALSO STORES A CONTEXT
FILTER EXPRESSION THAT AN EVENT NEEDS TO PASS BEFORE REACHING THE IMMEDIATE PARENT
ENVIRONMENT.
create_advertisement(Component_name,Environment_name,Event_id,Context_filter_
expression):-
    % CHECKING IF THE COMPONENT ALREADY EXISTS
    component_instance(Component_name),
    % CHECKING IF THE ENVIRONMENT EXISTS
    environment_instance(Environment_name),
    % CHECKING IF THE COMPONENT IS CONTAINED IN THE ENVIRONMENT
    environment_component(Environment_name,Component_name),
    % CHECKING IF THE EVENT TYPE EXISTS
    event_schema(Event_id,_),


assert(event_advertisement_entry(Component_name,Environment_name,Event_id,Con
text_filter_expression)).

% PREDICATE TO CREATE AN ADVERTISEMENT FOR AN ENVIRONMENT. . THE ADVERTISEMENT ALSO
STORES A CONTEXT FILTER EXPRESSION THAT AN EVENT NEEDS TO PASS BEFORE REACHING THE
IMMEDIATE PARENT ENVIRONMENT.
create_advertisement(Environment_name,Event_id,Context_filter_expression):-
    % CHECKING IF THE ENVIRONMENT EXISTS
    environment_instance(Environment_name),
    % CHECKING IF THE EVENT TYPE EXISTS
    event_schema(Event_id,_),
```

```
assert(event_advertisement_entry(Environment_name,Environment_name,Event_id,C
ontext_filter_expression)).

% PREDICATE TO CREATE A SUBSCRIPTION FOR A COMPONENT, A SUSCRIPTION IS ASSOSIATED WITH
A COMPONENT, AN ENVIRONMENT, AN EVENT TYPE, A CONTEXT FILTER AND A CONTENT FILTER. AN
ENVIRONMENT IS NEEDED AS SUBSCRIPTIONS ARE NOT SHARD AMONG ENVIRONMENTS. THEREFORE IF
A COMPONENT WANTS TO SUBSCRIBE TO AN EVENT FROM MULTIPLE PARENTS IT WILL NEED MULTIPLE
SUBSCRIPTIONS FOR EACH PARENT ENVIRONMENT. A CONTENT FILTER IS EVALUATED BASED ON THE
CONTENT FILTER EXPRESSION AND THE CONTENTS OF THE EVENT. A CONTEXT FILTER IS EVALUATED
BASED ON THE CONTEXT FILTER EXPRESSION AND THE VALUES OF THE CONTEXT VARIABLES HOSTED
BY THE PARENT ENVIRONMENTS.
create_subscription(Component_name,Environment_name,Event_id,Content_filter_e
xpression,Context_filter_expression):-
    % CHECKING IF THE COMPONENT EXISTS
    component_instance(Component_name),
    % CHECKING IF THE EVENT TYPE EXISTS
    event_schema(Event_id,_),


assert(event_subscription_entry(Component_name,Environment_name,Event_id,Cont
ent_filter_expression,Context_filter_expression)).

add_component_context_variable_list(Component_name,[]).

% THESE PREDICATES ARE USED TO ITERATE THE LIST OF CONTEXT VARIABLES OF COMPONENTS
add_component_context_variable_list(Component_name,[H|T]):-
    add_component_context_variable(Component_name,H),
    add_component_context_variable_list(Component_name,T).

% THIS PREDICATE IS USED TO STORE EACH CONTEXT VARIABLE
add_component_context_variable(Component_name,[Key,Value|[]]):-
    assert(context_variable(Component_name,[Key,Value])).

% THIS PREDICATE IS USED TO ADD A COMPONENT TO AN ENVIRONMENT. THIS PREDICATE CAN ALSO
BE USED TO NEST ENVIRONMENTS AS EACH ENVIRONMENT IS ALSO ASSERTTED AS A COMPONENT.
add_component_to_environment(Environment_name, Component_name):-
    % CHECK IF ENVIRONMENT EXISTS
    environment_instance(Environment_name),
    % CHECK IF THE COMPONENT EXISTS
    component_instance(Component_name),
    assert(environment_component(Environment_name,Component_name)),

findall(X,context_variable(Component_name,X),Component_context_variable_list)
,
    write(Component_context_variable_list),nl.

% THIS PREDICATE IS USED TO ITTERATE THE LIST OF ENVIRONMENT CONTEXT VARIABLES
add_context_variable_list_environment(Environment_name,[H|T]):-
    assert(context_variable(Environment_name,H)),
    add_context_variable_list_environment(Environment_name,T).

add_context_variable_list_environment(Environment_name,[]).

% THIS PREDICATE IS USED TO ADD A LIST OF COMPONENTS TO ENVIRONMENT
add_component_list_environment(Environment_name,[H|T]):-
    add_component_to_environment(Environment_name,H),
    add_component_list_environment(Environment_name,T).
```

```
add_component_list_environment(Environment_name,[]).

% THIS PREDIACTE IS USED FOR GETING ALL THE EVENT TYPES FOR AN ENVIRONMENTS
SUBCOMPONENTS FOR THAT HAVE ADVERTISEMENTS
get_event_id_list_advertisement_environment(Environment_name,Event_id_list):-

findall(X,environment_event_advertisement(Environment_name,X),Event_id_list).

% THIS PREDICATE IS TRUE IS THE ENVIRONMENT OR ONE OF ITS COMPONENTS HAS AN
ADVERTISEMENTS FOR THE EVENT TYPE
environment_event_advertisement(Environment_name,Event_id):-
    % CHECK IF THE ENVIRONMENT EXISTS
    environment_instance(Environment_name),
    % CHECK IF THE EVENT TYPE EXISTS
    event_schema(Event_id,_),

    % CHECK IF THE COMPONENT IS CONTAINED INSIDE THE ENVIRONMENT
    environment_component(Environment_name,Component_name),

event_advertisement_entry(Component_name,Environment_name,Event_id,Context_fi
lter_expression).

% THIS PREDICATE IS TRUE IS THE ENVIRONMENT OR ONE OF ITS COMPONENTS HAS AN
SUBSCRIPTIONS FOR THE EVENT TYPE
environment_event_subscription(Environment_name,Event_id):-
    %CHECK IF THE ENVIRONMENT EXISTS
    environment_instance(Environment_name),
    % CHECK IF THE EVENT TYPE EXISTS
    event_schema(Event_id,_),

    % CHECK IF THE COMPONENT IS CONTAINED INSIDE THE ENVIRONMENT
    environment_component(Environment_name,Component_name),

event_subscription_entry(Component_name,Environment_name,Event_id,Content_fil
ter_expression,Context_filter_expression).

% PREDICATE TO CHECK IF ANY OF THE N DEGREE CHILD OF THE COMPONENT HAS A SUBSCRIPTION
FOR THIS KIND OF EVENT
component_event_subscription(Component_name,Event_id):-

event_subscription_entry(Component_name,Environment_name,Event_id,Content_fil
ter_expression,Context_filter_expression).

component_event_subscription(Component_name,Event_id):-
    % GET A CHILD CONTAINED INSIDE THE COMPONENT
    environment_component(Component_name,Sub_component_name),
    component_event_subscription(Sub_component_name,Event_id).

% PREDICATE TO CHECK IF ANY OF THE N DEGREE CHILD OF THE COMPONENT HAS AN
ADVERTISEMENT FOR THIS KIND OF EVENT
component_event_advertisement(Component_name,Environment_name,Event_id):-

event_advertisement_entry(Component_name,Environment_name,Event_id,Context_fi
lter_expression).

component_event_advertisement(Component_name,Environment_name,Event_id):-
    environment_component(Component_name,Sub_component_name),
```

```
component_event_advertisement(Sub_component_name,Component_name,Event_id).

% PREDICATE TO GET ALL THE FIRST DEGREE COMPONENTS OF AN ENVIRONMENT
get_component_list_environment(Environment_name,Component_name_list):-
    findall(X,environment_component(Environment_name,X),Component_name_list).

% THIS PREDICATE IS USED TO SIMULATE AN EVENT ORIGINATION FROM A COMPONENT
simulate_event_component(Component_name,Environment_name,Event_id,Event_data_
list):-
    component_instance(Component_name),

    % TYPE FILTER START, THIS IS A CHECK IF THE COMPONENT HAS AN ADVERTISEMENT FOR
THIS TYPE OF EVENT

event_advertisement_entry(Component_name,Environment_name,Event_id,Context_fi
lter_expression),
    % TYPE FILTER END

    % CONTEX FILTER START, SINCE THE OUTGOING EVENT S ONLY NEED TO PASS THE CONTEXT
FILTER AND NOT THE CONTENT FILTER

findall(X,get_context_variables(Environment_name,_,X),Environment_context_var
iable_list),
    write('Context filter expression -
'),write(Context_filter_expression),nl,
    write('Context filter data'),write(Environment_context_variable_list),nl,
    compute(Context_filter_expression,Environment_context_variable_list),
    write('Context filter passed'),nl,
    % CONTEX FILTER END

    % VALIDATE THE EVENT TO ONE OF THE KNOWN EVENT SCHEMAS
    [Event_type|Et] = Event_data_list,
    event_schema(Event_type,Event_schema),

    % VALIDATE IT FURTHER BY A DATA TYPE CHECK OS OF THE VALUES OF THE KV PAIRS
    !,unify_event(Event_data_list,Event_schema),

    % THE BELOW IS USED TO THE ACTIONS THAT ARE TAKEN BY THE COMPONENT ON THE ARRIVAL
OF THIS TYPE OF EVENT

!,trigger_component_event_response(Component_name,Event_type,Event_data_list)
,

    % WE NOW MOVE TO THE ENVIRONMRNT THAT CONTAINS THE COMPONENT

get_reachable_environments(Event_type,Event_data_list,[Environment_name],Comp
onent_name,[Component_name],Reachable_environment_list),
    nl,write('Event '),write(Event_id),write(' Visited List --
>'),write(Reachable_environment_list),nl.

% THIS PREDICATE IS USED TO SIMULATE AN EVENT ORIGINATION FROM AN ENVIRONMENT
simulate_event_environment(Environment_name,Event_id,Event_data_list):-
    environment_instance(Environment_name),
    % TYPE FILTER START

event_advertisement_entry(Environment_name,Environment_name,Event_id,Context_
filter_expression),
```

```
    % TYPE FILTER END

    % VALIDATE THE EVENT TO ONE OF THE KNOWN EVENT SCHEMAS
    [Event_type|Et] = Event_data_list,
    event_schema(Event_id,Event_schema),
    % VALIDATE IT FURTHER BY A DATA TYPE CHECK OS OF THE VALUES OF THE KV PAIRS
    !,unify_event(Event_data_list,Event_schema),

        % THE BELOW IS USED TO THE ACTIONS THAT ARE TAKEN BY THE COMPONENT ON THE
ARRIVAL OF THIS TYPE OF EVENT

!,trigger_component_event_response(Component_name,Event_type,Event_data_list)
,

    % WE NOW MOVE TO THE ENVIRONMRNT THAT CONTAINS THE ENVIRONMENT

get_reachable_environments(Event_id,Event_data_list,[Environment_name],Enviro
nment_name,[],Reachable_environment_list),
    nl,write('Event '),write(Event_id),write(' Visited List --
>'),write(Reachable_environment_list),nl.

process_event_environment(Environment_name,Event_id,Event_data_list,Calling_c
omponent_name):-

event_subscription_entry(Environment_name,Environment_name,Event_id,Content_f
ilter_expression,Context_filter_expression),
    write('processing event '),write(Event_id),write(' for environment
'),write(Environment_name),nl,
    write('calling component '),write(Calling_component_name),nl,

    % CONTEX VARIABLE START
    (environment_component(Environment_name,Calling_component_name)
    ->

findall(X,get_context_variables(Environment_name,_,X),Environment_context_var
iable_list)
    ;

findall(X,get_context_variables(Calling_component_name,_,X),Environment_conte
xt_variable_list)
    ),
    write('Context filter expression -
'),write(Context_filter_expression),nl,
    write('Context filter data'),write(Environment_context_variable_list),nl,
    compute(Context_filter_expression,Environment_context_variable_list),
    write('Context filter passed'),nl,
    % CONTEX VARIABLE END

    % CONTENT FILTER START
    can_pass_event(Event_data_list,Content_filter_expression),
    write('**Passed** Component Content Filter -
'),write(Environment_name),nl,
    % CONTENT FILTER END


trigger_component_event_response(Environment_name,Event_id,Event_data_list).
```

```
process_event_environment(Environment_name,Event_id,Event_data_list,Calling_component_name).

% ONE OF THE DEFINITIONS OF THE A RECURSIVE PREDICATE FOR ITERATING THROUGH ALL THE COMPONENTS IN AN ENVIRONMENT
get_reachable_environments(Event_type,Event_data_list,[],Calling_component_name,I,I).

% ONE OF THE DEFINITIONS OF THE A RECURSIVE PREDICATE FOR ITERATING THROUGH ALL THE COMPONENTS IN AN ENVIRONMENT
get_reachable_environments(Event_type,Event_data_list,[Rh|Rt],Calling_component_name,I,O):-

get_reachable_environments(Event_type,Event_data_list,Rh,Calling_component_name,I,O2),

get_reachable_environments(Event_type,Event_data_list,Rt,Calling_component_name,O2,O).

% THIS PREDICATE IS USED TO COMPUTE THE EFFECT OF AN EVENT ON REACHING AN ENVIRONMENT
get_reachable_environments(Event_type,Event_data_list,Component_name,Calling_component_name,I,O):-
    environment_instance(Component_name),
    % CHECKING THAT THE ENVIRONMENT HAS NOT BEEN VISITIED BEFORE
    not(member(Component_name,I)),
    write('Visiting Environment - '),write(Component_name),nl,
    % write('_-> Already Visited - '),write(I),nl,
    % ADD THE NAME OF THIS ENVIRONMENT IN THE LIST TO PREVENT VISITING IT AGAIN
    append(I,[Component_name],I2),


process_event_environment(Component_name,Event_type,Event_data_list,Calling_component_name),

    % FIND ALL THE ENVIRONMENTS THAT CONTAIN THIS ENVIRONMENT
    % THIS CALL IS SPREADING THE ENENT UPWARDS TOWARDS THE PARENT ENVIRONMENTS
    findall(X,environment_component(X,Component_name),Environment_name_list),
    % write('|-> Parent Environments - '),write(Environment_name_list),nl,

    % THE METHOD RECURSIVELY TO VISIT PARENT ENVIRONMENT

get_reachable_environments(Event_type,Event_data_list,Environment_name_list,Component_name,I2,I3),
    % write('@@'),write(I3),nl,

    % WE ARE NOW LOOKING FOA ALL THE COMPONENTS AND ENVIRONMENTS CONTAINED IN THIS ENVIRONMENT

findall(X,environment_component_event_subscription(Component_name,X,Event_type),Sub_component_name_list),
    % write('|-> Sub of '),write(Component_name),write(' -
'),write(Sub_component_name_list),nl,

    % NOW WE ARE GOING TO SPRED THE EVENT DOWNWARDS TOWARDS THIS ENVIRONMENTS CHILDRREN COMPONENTS AND ENVIRONEMENTS

get_reachable_environments(Event_type,Event_data_list,Sub_component_name_list,Component_name,I3,O).
```

```
% THIS PREDICATE IS USED TO COMPUTE THE EFFECT OF AN EVENT ON REACHING A COMPONENT
get_reachable_environments(Event_type,Event_data_list,Component_name,Calling_
component_name,I,O):-
    not(environment_instance(Component_name)),
    component_instance(Component_name),
    % CHECKING THAT THE ENVIRONMENT HAS NOT BEEN VISITIED BEFORE
    not(member(Component_name,I)),
    append(I,[Component_name],O),

    % TYPE FILTER START, CHECKING IF THE COMPONENT HAS SUBSCRIBED TO THIS TYPE OF
EVENT

event_subscription_entry(Component_name,Environment_name,Event_type,Content_f
ilter_expression,Context_filter_expression),
    % TYPE FILTER END

    write('##Matching## Component Found - '),write(Component_name),nl,

    % CONTEX VARIABLE START

findall(X,get_context_variables(Calling_component_name,_,X),Environment_conte
xt_variable_list),
    write('Context filter expression -
'),write(Context_filter_expression),nl,
    write('Context filter data'),write(Environment_context_variable_list),nl,
    compute(Context_filter_expression,Environment_context_variable_list),
    write('Context filter passed'),nl,
    % CONTEX VARIABLE END

    % CONTENT FILTER START
    can_pass_event(Event_data_list,Content_filter_expression),
    write('**Passed** Component Content Filter - '),write(Component_name),nl,
    % CONTENT FILTER END

    write('Visiting Component - '),write(Component_name),nl,

    % NOW WE EXECUTE THE PREDICATE THAT SIGNALS THAT THE EVENT HAS BEEN ACCEPTED BY
THE COMPONENT, IT IS ALSO WHERE THE COMPONENT SERVICES THE EVENT

trigger_component_event_response(Component_name,Event_type,Event_data_list).

% ONE OF THE DEFINITIONS, IT IS USED TO PREVENT THE BREAK IN THE RECURSION CHAIN IF
THE COMPONENT DOES NOT SUBSCRIBE TO A EVENT TYPE
get_reachable_environments(Event_type,Event_data_list,Component_name,Calling_
component_name,I,O):-
    not(environment_instance(Component_name)),
    component_instance(Component_name),
    not(member(Component_name,I)),
    append(I,[Component_name],O),
    write('Visiting Component - '),write(Component_name),nl,

not(event_subscription_entry(Component_name,Environment_name,Event_type,Conte
nt_filter_expression,Context_filter_expression)).

% THIS PREDICATE IS USED AS AN ALL PASS DEFINITION FOR THE RECURSIVE PREDICATE
get_reachable_environments(Event_type,Event_data_list,Component_name,Calling_
component_name,I,I).
```

```
get_context_variables(Environment_name,Component_name,Environment_context_var
iable):-
    Component_name = Environment_name,
    context_variable(Environment_name,Environment_context_variable).

environment_component_event_subscription(Environment_name,Component_name,Even
t_id):-
    environment_component(Environment_name,Component_name),
    event_subscription_entry(Component_name,Environment_name,Event_id,_,_).

environment_component_event_subscription(Environment_name,Component_name,Even
t_id):-
    environment_component(Environment_name,Component_name),
    environment_instance(Component_name).

% THIS PREDICATE IS USED TO FETCH THE CONTEXT VARIABLES OF AN ENVIRONMENT, AND THE
COMPONENTS CONTAINED WITHIN IT
get_context_variables(Environment_name,Component_name,Environment_context_var
iable):-
    environment_instance(Environment_name),
    ancestor(Environment_name,Component_name),
    context_variable(Component_name,Environment_context_variable).

% THIS PREDICATE IS USED TO GET A CONTEXT VARIABLE CREATED FOR AN ENVIRONMENT
get_context_variables(Environment_name,Component_name,Environment_context_var
iable):-
    context_variable(Environment_name,Environment_context_variable).

get_context_variables(Environment_name,Component_name,Environment_context_var
iable):-
    environment_instance(Environment_name),
    ancestor(Environment_name,Sub_environment_name),
    context_variable(Sub_environment_name,Environment_context_variable).

% USED TO FIND THE ACESTORY OF A COMPONENT OR ENVIRONMENT
ancestor(Environment_name,Component_name):-
    environment_component(Environment_name,Component_name).

% USED TO FIND THE ACESTORY OF A COMPONENT OR ENVIRONMENT
ancestor(Environment_name,Component_name):-
    environment_component(Sub_environment_name,Component_name),
    ancestor(Environment_name,Sub_environment_name).

% THE PREDICATE IS USED TO CAHNGE THE VALUE OF A CONTEXT VARIABLE
change_context_variable_value(Context_variable_name,Context_variable_new_valu
e):-
    context_variable(Component_name,[Context_variable_name,Old_value]),

    retract(context_variable(Component_name,[Context_variable_name,_])),
    write('Setting new value '),write(Context_variable_new_value),write(' for
context variable '),write(Context_variable_name),nl,

assert(context_variable(Component_name,[Context_variable_name,Context_variabl
e_new_value])).
```

A.2 nested_list_approach

```
% THIS PREDICATE IS USED TO CREATE NESTED EVENT SCHEMAS, IT GENERATES SCHEMAS FOR
EVENTS THAT CONTAIN OTHER EVENTS. THIS IS NO LONGER DIRECTLY USED NOW BUT IT IS A
USEFULT UTILITY WHEN THINKING ABOUT LARGE EVENT SCHEMAS.
lower_event([Key,Value|[]],I,O):-
    % CHECK IF THE KEY IS AN DEFINED EVENT TYPE
    not(event_schema([Key|SchemaTail])),
    % CHECK IF THE VALUE IS AN DEFINED EVENT TYPE
    not(event_schema([Value|SchemaTail])),
    not(is_list(Key)),
    not(is_list(Value)),

    %nl,write(Key),write('-'),write(Value),nl,
    Kvp = [Key,Value],
    append(I,[Kvp],O).

lower_event([Head|[]],I,O):-
    lower_event(Head,I,O).

lower_event([Head|Tail],I,O):-
    lower_event(Head,I,Os1),
    lower_event(Tail,Os1,O).

lower_event(Type,I,O):-
    % FETCH THE SCHEMA DEFINITTION OF THE SCHEMA TYPE
    event_schema([Type|SchemaTail]),
    %write(Type),write('->'),
    append(I,[Type],Os1),
    %write([SchemaTail]),nl,
    % RESOLVE THE COMPONENTS OF THE SCHEMA DEFINITION FOR OTHER NESTED TYPES
    lower_event(SchemaTail,[],Os2),
    append(Os1,[Os2],O).

lower_event(Type,I,O):-
    not(event_schema([Type|SchemaTail])),
    %write(Type),write('*-'),nl,
    append(I,[Type],O).

step_forward([Head|[]]).

step_forward([Head|Tail]):-
    %write(Head),
    lower_event(Head),
    step_forward(Tail).

% THIS IS THE ENTRY PREDICATE FOR THE EVENT SCHEMA GENERATOR
top_event(Type,O):-
    % write('Resolving Schema'),nl,
    lower_event(Type,[],O).

% THIS PREDICATE IS USED TO CHECK IF THE PROVIDED EVENT AND THE SCHEMA UNIFY
unify_event(Event,Schema):-
    % write('Start'),nl,
    [Eh|Et] = Event,

    % For computing the schema internally, this is no longer needed but can be a
helpful utility.
    % top_event(Eh,Schema),
```

```
    % write('Schema Decided'),nl,

    % EASIEST CHECK FIRST, CHECK IF THE LENGHTS OF THE 2 ARE SAME, IF NOT WE DO NOT
NEED TO CHECK FURTHER
    !,unify_length(Event,Schema),
    % write('length compared'),nl,
    % IF THE LENGHS MATCH, WE MOVE ON THE ELEMENTS OF THE EVENT AND COMPARE THE KEYS
AND THE TYPE OF VALUES
    !,compare_elements(Event,Schema).
    % write('Schema compared'),nl.

% THIS PREDICATE IS USED TO ITERRATE THE EVENT AND THE SCHEMA TOGATHER
compare_elements([Eh|[]],[Sh|[]]):-
    %write('End'),nl,
    !,compare_elements(Eh,Sh).

compare_elements([Eh|Et],[Sh|St]):-
    %write('middle'),nl,
    !,compare_elements(Eh,Sh),
    compare_elements(Et,St).

% CHECK IF THE EVENT ELEMENT IS AN INTEGER
compare_elements(Eventelement,integer):-
    %write('comparing-integer => '),write(Eventelement),write('='),write(integer),nl,
    !,integer(Eventelement).

% CHECK IF THE EVENT ELEMENT IS A FLOAT
compare_elements(Eventelement,float):-
    %write('comparing-float => '),write(Eventelement),write('='),write(float),nl,
    !,float(Eventelement).

% CHECK IF THE EVENT ELEMENT IS A STRING
compare_elements(Eventelement,string):-
    %write('comparing-string => '),write(Eventelement),write('='),write(string),nl,
    !,atom(Eventelement).

% THIS DEFINITION OF THE PREDICATE ENSURES THAT ALL THE KEYS ARE THE SAME BETWEEN THE
EVENT AND THE SCHEMA
compare_elements(Sameelement,Sameelement).

% THIS PREDICATE IS SUCESSFULL IF THE LENGHT OD THE 2 NESTED LISTS ARE THE SAME
unify_length(Event,Schema):-
    % FLATTEN THE SCHEMA NESTED LIST TO A FLAT LIST
    flatten2(Schema,FlatSchema),
    % FLATTEN THE EVENT NESTED LIST TO A FLAT LIST
    flatten2(Event,FlatEvent),
    % FIND THE LENGHT OF THE SCHEMA FLAT LIST
    length(FlatSchema,Lengthschema),
    % FIND THE LENGHT OF THE EVENT FLAT LIST
    length(FlatEvent,Lengthevent),
    % UNIFY THE 2 LENGHTS
    Lengthschema=Lengthevent.

% UTILITY PREDICATE TO FLATTEN A NESTED LIST
flatten2([], []) :- !.
flatten2([L|Ls], FlatL) :-
    !,
    flatten2(L, NewL),
    flatten2(Ls, NewLs),
```

```
    append(NewL, NewLs, FlatL).
flatten2(L, [L]).
```

A.3 context_filter

```
% THIS PREDICATE IS USED TO CHECK IS AN EVENT CAN PASS THROUGH THE FILTER
can_pass_event(Event,Filter_expression):-
    % WE FALTTEN THE EVENT FROM A NESTED LIST TO A LIST OF KEY-VALUE PAIRS
    faltten_to_key_value_pairs(Event,[],Flat_kv_pairs),
    % write(Flat_kv_pairs),nl,

%replace_keys_by_values(Context_filter_expression,Flat_kv_pairs,Logic_filled_expressio
n),
    compute(Filter_expression,Flat_kv_pairs).

% THESE PREDICATES ARE USED TO FLATTEN THE NESTED LIST OF KV PAIRS INTO A FLAT LIST OF
KV PAIRS
faltten_to_key_value_pairs([H|[]],I,Flat_kv_pairs):-
    % write('(2)-> '),write(H),write(' '),
    faltten_to_key_value_pairs(H,I,Flat_kv_pairs).

faltten_to_key_value_pairs([H|T],I,Flat_kv_pairs):-
    % write('(1)-> '),write(H),write(' '),
    faltten_to_key_value_pairs(H,I,O1),
    faltten_to_key_value_pairs(T,O1,Flat_kv_pairs).

faltten_to_key_value_pairs([Key,Value|[T]],I,Flat_kv_pairs):-
    not(is_list(Key)),
    is_list(Value),
    % write('(4)-> '),write(Key),nl,
    faltten_to_key_value_pairs(Value,I,O1),
    faltten_to_key_value_pairs(T,O1,Flat_kv_pairs).

faltten_to_key_value_pairs([Key,Value|[]],I,Flat_kv_pairs):-
    not(is_list(Key)),
    not(is_list(Value)),
    % write('(5)-> '),write(Key),nl,
    Kvp = [Key,Value],
    append(I,[Kvp],Flat_kv_pairs).

faltten_to_key_value_pairs([Key,Value|[]],I,Flat_kv_pairs):-
    not(is_list(Key)),
    is_list(Value),
    % write('(6)-> '),write(Key),nl,
    faltten_to_key_value_pairs(Value,I,Flat_kv_pairs).

% compute([3,=,3],Result).
% compute([[3,=,3],&,[3,=,3]]).
% compute([[[5,=,3],or,[5,=,3]],or,[[5,=,5],or,[5,=,3]]]).
% THIS PREDICATE PRODUCES A BOOLEAN OUTPUT FOR THE INPUT OF FILTER EXPRESSION AND LIST
OF KEY-VALUE PAIRS
compute(Filter_expression,Flat_kv_pairs):-
    % write('starting filter computation'),nl,
    compute_logic_molecule(Filter_expression,Flat_kv_pairs).

% THIS A SHORT EXECUTION PATH FOR IF THE FILTER EXPRESSION IS NOT PROVIDED FOR A
COMPONENT
compute([],_).
```

```
% THIS PREDICATE IS USED TO COMPUTE THE FITER EXPRESSION RECURSIVELY, IT BREAKS THE
FILTER EXPRESSION INTO NESTED LISTS OF 3 ELEMENTS EACH
compute_logic_molecule([Term_1,Operator,Term_2|[]],Flat_kv_pairs):-
    !,
    % CHECKING IF THE OPERATOR IS SUPPORTED
    member(Operator,[>,<,=<,>=,and,or,=\=,=]),
    % write('term1 = '),write(Term_1),nl,
    % write('term2 = '),write(Term_2),nl,

    % CHECKING IF EITHER THE FIRST OR THE SECOND TERM OF THE EXPRESSION IS A LIST IN
ITS SELF, THIS WOULD MEAN THAT THE EXPRESSION IS A NESTED EXPRESSION, IN HTIS CASE THE
PREDICATE IS CALLED ON THE INNER EXPRESSION RECURSIVELY
    (not(is_list(Term_1))-
>get_value_for_key(Term_1,Flat_kv_pairs,R1);(compute_logic_molecule(Term_1,Fl
at_kv_pairs)->R1 = true;R1 = fail)),
    (not(is_list(Term_2))-
>R2=Term_2;(compute_logic_molecule(Term_2,Flat_kv_pairs)->R2 = true;R2 =
fail)),

    compute_logic(R1,Operator,R2).

% THIS PREDICATE IS USED TO FETCH THE VALUE CORRESPONDING TO A KEY FROM A LIST OF KEY-
VALUE PAIRS.
get_value_for_key(R1,[[Key,Value]|T],V1):-
    % write('looking for '),write(R1),write(' comparing with '),write(Key),nl,
    (R1==Key->V1 = Value;get_value_for_key(R1,T,V1)).

get_value_for_key(R1,[[Key,Value]|[]],V1):-
    % write('looking for '),write(R1),write(' comparing with '),write(Key),nl,
    (R1==Key->V1 = Value;fail).

% THIS PREDICATE HANDLES THE EQUALITY OPERATOR
compute_logic(R1,=,R2):-
    % write(R1),write('='),write(R2),nl,
    (  R1 == R2
    -> true
    ;  fail
     ).

% THIS PREDICATE HANDLES THE > OPERATOR
compute_logic(R1,>,R2):-
    % write(R1),write('>'),write(R2),nl,
    (  R1 > R2
    -> true
    ;  fail
     ).

% THIS PREDICATE HANDLES THE < OPERATOR
compute_logic(R1,<,R2):-
    % write(R1),write('<'),write(R2),nl,
    (  R1 < R2
    -> true
    ;  fail
     ).

% THIS PREDICATE HANDLES THE =< OPERATOR
compute_logic(R1,=<,R2):-
    % write(R1),write('=<'),write(R2),nl,
    (  R1 =< R2
```

```
     -> true
     ;  fail
      ).

% THIS PREDICATE HANDLES THE >= OPERATOR
compute_logic(R1,>=,R2):-
     % write(R1),write('>='),write(R2),nl,
     (  R1 >= R2
     -> true
     ;  fail
      ).

% THIS PREDICATE HANDLES THE =/= INEQUALITY OPERATOR
compute_logic(R1,=\=,R2):-
     % write(R1),write('=\='),write(R2),nl,
     (  \+(R1 == R2)
     -> true
     ;  fail
      ).

% THIS PREDICATE HANDLES THE AND OPERATOR
compute_logic(R1,and,R2):-
     % write(R1),write(' and '),write(R2),nl,
     (  R1,R2
     -> true
     ;  fail
      ).

% THIS PREDICATE HANDLES THE OR OPERATOR
compute_logic(R1,or,R2):-
     % write(R1),write(' or '),write(R2),nl,
     (  R1;R2
     -> true
     ;  fail
      ).
```

A.4 xmc_utils

```
assertTriggPredicate(Component_name,Event_id,Event_data):-
    assert(triggered(Component_name,Event_id,Event_data)).

cleanupAllTriggPredicates :-
    retract(triggered(_,_,_)).

cleanupComponentTriggPredicates(Component_name):-
    retract(triggered(Component_name,_,_)).

cleanupComponentTriggPredicates(Component_name).

cleanupComponentEventTriggPredicates(Component_name,Event_id):-
    retract(triggered(Component_name,Event_id,_)).

hasComponentReceived(Component_name,Event_id,Event_data):-
    triggered(Component_name,Event_id,Event_data).

hasComponentReceived(Component_name,Event_id):-
    triggered(Component_name,Event_id,_).

hasComponentReceivedResult(Component_name,Event_id, Result):-
    triggered(Component_name,Event_id, _),
    Result = 1.

hasComponentReceivedResult(Component_name,Event_id, Result):-
    Result = 0.

distance(X1,Y1,X2,Y2,D):-
    D is sqrt(((X2-X1) * (X2-X1)) + ((Y2-Y1) * (Y2-Y1))).

no_duplicates(L) :-
    setof(X, member(X, L), Set),
    length(Set, Len),
    length(L, Len).

all_environemtns(In_list,Out_environment_name_list):-
    findall(X,environment_instance(X),Environment_name_list),
    append(In_list,Environment_name_list,Out_environment_name_list).

all_components(In_list,Out_component_name_list):-
    findall(X,component_instance(X),Component_name_list),
    append(In_list,Component_name_list,Out_component_name_list).

all_event_names(In_list,Out_event_name_list):-
    findall(X,event_schema(X,_),Event_name_list),
    append(In_list,Event_name_list,Out_event_name_list).

all_advertisements(In_list,Out_advertisement_list):-
    findall(X(Y),event_advertisement_entry(X,_,Y,_),Advertisement_list),
    append(In_list,Advertisement_list,Out_advertisement_list).

all_subscriptions(In_list,Out_subscription_list):-
    findall(X(Y),event_subscription_entry(X,_,Y,_,_),Subscription_list),
```

```
    append(In_list,Subscription_list,Out_subscription_list).

all_context_variables(In_list,Out_context_variable_list):-
    findall(X,context_variable(_,[X,_]),Context_variable_list),
    append(In_list,Context_variable_list,Out_context_variable_list).

all_environment_component_relation(In_list,Environment_component_relation_lis
t):-
    findall(X(Y),environment_component(X,Y),Relation_list),
    append(In_list,Relation_list,Environment_component_relation_list).

all_environment_component_relation_as_nested_lists(In_list,Environment_compon
ent_relation_list):-
    findall([X,Y],environment_component(X,Y),Relation_list),
    append(In_list,Relation_list,Environment_component_relation_list).

validate_env_component_relation_list([H|T]):-
    validate_env_relation(H),
    validate_env_component_relation_list(T).

validate_env_component_relation_list([H|[]]):-
    validate_env_relation(H).

validate_env_relation([Environment_name,Component_name]):-
    environment_instance(Environment_name),
    component_instance(Component_name).

all_advertisements_relation_as_nested_lists(In_list,Advertisement_component_e
vent_relation_list):-

findall([Component_name,Event_id,Environment_name],event_advertisement_entry(
Component_name,Environment_name,Event_id,_),Relation_list),

append(In_list,Relation_list,Advertisement_component_event_relation_list).

validate_advertisement_relation_list([H|T]):-
    validate_advertisement_relation(H),
    validate_advertisement_relation_list(T).

validate_advertisement_relation_list([]).

validate_advertisement_relation([Component_name,Event_id,Environment_name]):-
    component_instance(Component_name),
    environment_instance(Environment_name),
    event_schema(Event_id,_).

all_subscription_relation_as_nested_lists(In_list,Subscription_component_even
t_relation_list):-

findall([Component_name,Event_id,Environment_name],event_subscription_entry(C
omponent_name,Environment_name,Event_id,_,_),Relation_list),
    append(In_list,Relation_list,Subscription_component_event_relation_list).

validate_subscription_relation_list([H|T]):-
    validate_subscription_relation(H),
    validate_subscription_relation_list(T).
```

```
validate_subscription_relation_list([]).

validate_subscription_relation([Component_name,Event_id,Environment_name]):-
    component_instance(Component_name),
    environment_instance(Environment_name),
    event_schema(Event_id,_).

all_context_variable_relation_as_nested_lists(In_list,Context_variable_relati
on_list):-

findall(Component_name,context_variable(Component_name,_),Component_list),
    append(In_list,Component_list,Context_variable_relation_list).

validate_context_variable_relation_list([H|T]):-
    component_instance(H),
    validate_context_variable_relation_list(T).

validate_context_variable_relation_list([]).

validate_no_dual_membership_list([H|T]):-
    validate_no_dual_membership(H),
    validate_no_dual_membership_list(T).

validate_no_dual_membership_list([]).

validate_no_dual_membership([Component_name,Event_id,Environment_name]):-
    (
not(event_advertisement_entry(Component_name,Environment_name,Event_id,_))
    -> true
    ;  fail
     ).
```

Appendix B Case study XSB source code

B.1 cellphone_behavior_adaptation_case_study_specific_behavior

```
trigger_component_event_response(users_phone,locationEvent,Event_data):-
    write('users_phone reacting to location event'),nl,
    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),

    get_value_for_key(latitude,Flat_kv_pairs,Latitude_value),
    get_value_for_key(longitude,Flat_kv_pairs,Longitude_value),
    write('Latitude - '),write(Latitude_value),nl,
    write('Longitude - '),write(Longitude_value),nl,

    distance(Latitude_value,Longitude_value,5,5,Distance),
    write('Distance calculated - '),write(Distance),nl,
    change_context_variable_value(distanceToWorkContextElement,Distance),

    assertTriggPredicate(users_phone,locationEvent,Event_data).

trigger_component_event_response(calendar_application,meetingStartedEvent,Eve
nt_data):-
    write('calendar_application reacting to meeting started event'),nl,
    change_context_variable_value(meetingInProgressContextElement,1),


assertTriggPredicate(calendar_application,meetingStartedEvent,Event_data).

% INCOMING CALL EVENT START
trigger_component_event_response(users_phone,incomingCallEvent,Event_data):-
    % Meeting has started
    context_variable(_,[meetingInProgressContextElement,1]),
    % The user is at work
    context_variable(_,[distanceToWorkContextElement,0.0]),

    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),
    get_value_for_key(callerName,Flat_kv_pairs,Caller_name),

    % Caller is in the preferred list
    member(Caller_name,[akshat,anugrah]),

    change_context_variable_value(inBusinessMeetingContextElement,1),

    write('users_phone reacting to incoming call event'),nl,
    write('BusinessMeetingPreferredCallerProfile - LET IT THROUGH'),nl,

    assertTriggPredicate(users_phone,incomingCallEvent,Event_data).

trigger_component_event_response(users_phone,incomingCallEvent,Event_data):-
    % Meeting has started
    context_variable(_,[meetingInProgressContextElement,1]),
    % The user is at work
    context_variable(_,[distanceToWorkContextElement,0.0]),

    change_context_variable_value(inBusinessMeetingContextElement,1),
```

```
    write('users_phone reacting to incoming call event'),nl,
    write('BusinessMeetingProfile - BLOCK CALL'),nl.

trigger_component_event_response(users_phone,incomingCallEvent,Event_data):-

    write('users_phone reacting to incoming call event'),nl,
    write('NotInMeetingProfile - NORMAL CALL'),nl,

    assertTriggPredicate(users_phone,incomingCallEvent,Event_data).
% INCOMING CALL EVENT END

trigger_component_event_response(Component_name,Event_id,Event_data):-
    component_instance(Component_name),
    event_schema(Event_id,_),
    assertTriggPredicate(Component_name,Event_id,Event_data).

trigger_component_event_response(Component_name,Event_id,Event_data).
```

B.2 cellphone_as_controller_specific_behaviour

```
trigger_component_event_response(true_remote_control_application,alarmGoesOff
Event,Event_data):-
    write('Component true_remote_control_application reacting to the
alarmGoesOffEvent.'),nl,
    change_context_variable_value(alarmOccured,1),
    write('AFTER 1 MINUTE PASSES...'),nl,

assertTriggPredicate(true_remote_control_application,alarmGoesOffEvent,Event_
data),

simulate_event_environment(user_cellphone,cellphoneTimeChangedEvent,[cellphon
eTimeChangedEvent,[[dateTime,hammerTime]]]).

trigger_component_event_response(true_remote_control_application,cellphoneTim
eChangedEvent,Event_data):-
    write('Component true_remote_control_application reacting to the
cellphoneTimeChangedEvent.'),nl,
    context_variable(Component_name,[alarmOccured,1]),
    change_context_variable_value(activationTimeHasElapsedContextElement,1),

assertTriggPredicate(true_remote_control_application,cellphoneTimeChangedEven
t,Event_data),

simulate_event_environment(user_cellphone,cellphoneLocationEvent,[cellphoneLo
cationEvent,[[latitude,0.0],[longitude,0.0]]]).

trigger_component_event_response(true_remote_control_application,cellphoneLoc
ationEvent,Event_data):-
    write('Component true_remote_control_application reacting to the
cellphoneLocationEvent.'),nl,
    context_variable(Component_name,[alarmOccured,1]),

context_variable(Component_name,[activationTimeHasElapsedContextElement,1]),
```

```
    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),
    get_value_for_key(latitude,Flat_kv_pairs,Latitude_value),
    get_value_for_key(longitude,Flat_kv_pairs,Longitude_value),
    distance(Latitude_value,Longitude_value,0,0,Distance),
    write('Distance from home calculated - '),write(Distance),nl,
    Distance = 0.0,

    change_context_variable_value(mustStartCarEngineContextElement,1),

assertTriggPredicate(true_remote_control_application,cellphoneLocationEvent,E
vent_data),

simulate_event_environment(true_remote_control_application,remoteStarterActio
nEvent,[remoteStarterActionEvent,[[action,start]]]).

trigger_component_event_response(user_cellphone,notifyUserOfCarEngineStartedE
vent,Event_data):-
    write('@@@ Component user_cellphone reacting to the
notifyUserOfCarEngineStartedEvent. @@@'),nl,

assertTriggPredicate(user_cellphone,notifyUserOfCarEngineStartedEvent,Event_d
ata).

trigger_component_event_response(car_computer_control_unit,remoteStarterActio
nEvent,Event_data):-
    write('Component car_computer_control_unit reacting to the
remoteStarterActionEvent.'),nl,

    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),
    get_value_for_key(action,Flat_kv_pairs,Action_type),
    Action_type = start,

    write('ATTEMPTING TO START CAR ENGINE.'),nl,
    write('Triggering ResultOfRemoteStarterAction Event.'),nl,

assertTriggPredicate(car_computer_control_unit,remoteStarterActionEvent,Event
_data),

simulate_event_environment(car_computer_control_unit,resultOfRemoteStarterAct
ionEvent,[resultOfRemoteStarterActionEvent,[[action,start],[result,1]]]).

trigger_component_event_response(true_remote_control_application,resultOfRemo
teStarterActionEvent,Event_data):-
    write('Component true_remote_control_application reacting to the
resultOfRemoteStarterActionEvent.'),nl,
    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),
    get_value_for_key(result,Flat_kv_pairs,Action_result),

    write('Triggering NotifyUserOfCarEngineStarted Event.'),nl,

assertTriggPredicate(true_remote_control_application,resultOfRemoteStarterAct
ionEvent,Event_data),

simulate_event_environment(true_remote_control_application,notifyUserOfCarEng
ineStartedEvent,[notifyUserOfCarEngineStartedEvent,[[result,Action_result]]])
.
```

```
trigger_component_event_response(true_remote_control_application,cellphoneSub
sequentLocationEvent,Event_data):-
    write('Component true_remote_control_application reacting to the
cellphoneSubsequentLocationEvent.'),nl,
    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),

    get_value_for_key(latitude0,Flat_kv_pairs,Latitude_value0),
    get_value_for_key(latitude1,Flat_kv_pairs,Latitude_value1),
    get_value_for_key(latitude2,Flat_kv_pairs,Latitude_value2),

    get_value_for_key(longitude0,Flat_kv_pairs,Longitude_value0),
    get_value_for_key(longitude1,Flat_kv_pairs,Longitude_value1),
    get_value_for_key(longitude2,Flat_kv_pairs,Longitude_value2),

    distance(Latitude_value0,Longitude_value0,0,0,Distance0),
    distance(Latitude_value1,Longitude_value1,0,0,Distance1),
    distance(Latitude_value2,Longitude_value2,0,0,Distance2),

    write('Distance0 - '),write(Distance0),nl,
    write('Distance1 - '),write(Distance1),nl,
    write('Distance2 - '),write(Distance2),nl,


assertTriggPredicate(true_remote_control_application,cellphoneSubsequentLocat
ionEvent,Event_data),
    (
    Distance0 < Distance1,Distance1 < Distance2
    ->
    write('Cellphone moving away from Home'),nl,
    change_context_variable_value(mustTriggerHomeActionsContextElement,1),

simulate_event_environment(true_remote_control_application,triggerHomeActionE
vent,[triggerHomeActionEvent,[[action,setTemprature],[value,23]]]),

simulate_event_environment(true_remote_control_application,triggerHomeActionE
vent,[triggerHomeActionEvent,[[action,engageSecurityAlarm],[value,1]]])
    ;
    % pass without doing anything
    write('Cellphone NOT moving away from Home'),nl).

trigger_component_event_response(smart_home_controller_unit,triggerHomeAction
Event,[triggerHomeActionEvent,[[action,setTemprature],[value,Value]]]):-
    write('Component smart_home_controller_unit reacting to the
triggerHomeActionEvent for setting temperature.'),nl,
    write('SETTING TEMPRATURE TO '),write(Value),nl,

assertTriggPredicate(smart_home_controller_unit,triggerHomeActionEvent,Event_
data),

simulate event environment(smart home controller unit,resultOfHomeActionEvent
,[resultOfHomeActionEvent,[[action,setTemprature],[value,Value],[result,1]]])
.

trigger_component_event_response(smart_home_controller_unit,triggerHomeAction
Event,[triggerHomeActionEvent,[[action,engageSecurityAlarm],[value,Value]]]):
-
```

```
    write('Component smart_home_controller_unit reacting to the
triggerHomeActionEvent for engage security alarm.'),nl,
    write('ENGAGING ALARM SYSTEM'),nl,

assertTriggPredicate(smart_home_controller_unit,triggerHomeActionEvent,Event_
data),

simulate_event_environment(smart_home_controller_unit,resultOfHomeActionEvent
,[resultOfHomeActionEvent,[[action,engageSecurityAlarm],[value,Value],[result
,1]]]).

trigger_component_event_response(smart_home_controller_unit,triggerHomeAction
Event,Event_data):-

assertTriggPredicate(smart_home_controller_unit,triggerHomeActionEvent,Event_
data),
    write('Component smart_home_controller_unit reacting to the
triggerHomeActionEvent of UNKNOWN type'),nl.

trigger_component_event_response(true_remote_control_application,resultOfHome
ActionEvent,[resultOfHomeActionEvent,[[action,engageSecurityAlarm],[value,Val
ue],[result,Result]]]):-
    write('Component true_remote_control_application reacting to the
resultOfHomeActionEvent for engageSecurityAlarm.'),nl,

assertTriggPredicate(true_remote_control_application,resultOfHomeActionEvent,
Event_data),

simulate_event_environment(true_remote_control_application,notifyUserOfHomeAc
tionEvent,[notifyUserOfHomeActionEvent,[[action,engageSecurityAlarm],[value,V
alue],[result,Result]]]).

trigger_component_event_response(true_remote_control_application,resultOfHome
ActionEvent,[resultOfHomeActionEvent,[[action,setTemprature],[value,Value],[r
esult,Result]]]):-
    write('Component true_remote_control_application reacting to the
resultOfHomeActionEvent for setTemprature.'),nl,

assertTriggPredicate(true_remote_control_application,resultOfHomeActionEvent,
Event_data),

simulate_event_environment(true_remote_control_application,notifyUserOfHomeAc
tionEvent,[notifyUserOfHomeActionEvent,[[action,setTempraturesetTemprature],[
value,Value],[result,Result]]]).

trigger_component_event_response(user_cellphone,notifyUserOfHomeActionEvent,E
vent_data):-
    write('@@@ Component user_cellphone reacting to the
notifyUserOfHomeActionEvent '),
    faltten_to_key_value_pairs(Event_data,[],Flat_kv_pairs),
    get_value_for_key(action,Flat_kv_pairs,Action_value),
    write('For '),write(Action_value), write('. @@@'),nl,

assertTriggPredicate(user_cellphone,notifyUserOfHomeActionEvent,Event_data).

trigger_component_event_response(Component_name,Event_id,Event_data):-
    component_instance(Component_name),
```

```
    event_schema(Event_id,_),
    assertTriggPredicate(Component_name,Event_id,Event_data).

trigger_component_event_response(Component_name,Event_id,Event_data).
```

Appendix C Case study XMC source code

C.1 Cellphone behavior adaptation

```
% CHECK THAT NO LOSS IS ENCOUNTERED WHILE SENDING THE EVENTS, AND THAT IT IS
POSSIBLE THAT THE EVENT IS RECEIVED
% mck(chk_send_receive,lossless)

chk_send_event ::=

simulate_event_component(calendar_application,users_phone,meetingStartedEvent
,[meetingStartedEvent,[[meetingName,board_meeting],[date,sfsdfs],[startTime,s
dfsdf],[duration,1]]])
    ; action(sendnew)
    .

chk_receive_event ::=

hasComponentReceivedResult(calendar_application,meetingStartedEvent,Result);
    if (Result==1)
    then
    {
        action(recvnew)
    }
    .

chk_send_receive ::=
    chk_send_event
    ; chk_receive_event
    .

losslesstx += [-sendnew]losslesstx /\ <->tt.
losslessrx += <recvnew>tt \/ <->losslessrx.
lossless += losslesstx /\ losslessrx.

% CHECK THAT NO LOSS IS ENCOUNTERED WHILE SENDING THE EVENTS, AND THAT IT IS
POSSIBLE THAT THE EVENT IS RECEIVED

% RECEIVER CONTEXT FILTER

chk_receiver_context_filter_unblocked ::=
    change_context_variable_value(name,phone)
    ;
simulate_event_environment(users_phone,locationEvent,[locationEvent,[[latitud
e,1],[longitude,1]]])
    ; chk_receiver_context_filter_received_unblocked.

chk_receiver_context_filter_blocked ::=
    change_context_variable_value(name,someotherphone)
    ;
simulate_event_environment(users_phone,locationEvent,[locationEvent,[[latitud
e,1],[longitude,1]]])
    ; chk_receiver_context_filter_received_blocked.

chk_receiver_context_filter_received_blocked ::=
```

```
    hasComponentReceivedResult(calendar_application,locationEvent,Result)
    ; if (Result==1)
    then
    {
        action(receivedLocationEventBlocked)
    }
    .

chk_receiver_context_filter_received_unblocked ::=
    hasComponentReceivedResult(users_phone,locationEvent,Result)
    ; if (Result==1)
    then
    {
        action(receivedLocationEventUnblocked)
    }
    .

never_receive_blocked_event -= [receivedLocationEventBlocked]ff /\ [-
]never_receive_blocked_event.
possibly_receive_unblocked_event += <receivedLocationEventUnblocked>tt \/ <-
>possibly_receive_unblocked_event.

% RECEIVER CONTEXT FILTER

% UNIQUE COMPONENTS
% mck(chk_duplicates,unique_all).

chk_duplicates_cs_one ::=
    unique_environments_cs_one
    ; unique_components_cs_one
    ; unique_events_cs_one
    ; unique_advertisements_cs_one
    ; unique_subscriptions_cs_one
    ; unique_context_variables_cs_one
    ; unique_environment_component_relation_cs_one
    .

unique_environments_cs_one ::=
    all_environemtns([],All_env_list)
    ; if (no_duplicates(All_env_list))
    then
    {
        action(noDuplicateEnv)
    }
    .

unique_components_cs_one ::=
    all_components([],All_component_list)
    ; if (no_duplicates(All_component_list))
    then
    {
        action(noDuplicateComponent)
    }
    .

unique_events_cs_one ::=
    all_event_names([],All_event_list)
```

```
    ; if (no_duplicates(All_event_list))
    then
    {
        action(noDuplicateEvent)
    }
    .

unique_advertisements_cs_one ::=
    all_advertisements([],All_advertisement_list)
    ; if (no_duplicates(All_advertisement_list))
    then
    {
        action(noDuplicateAdvertisement)
    }
    .

unique_subscriptions_cs_one ::=
    all_subscriptions([],All_subscription_list)
    ; if (no_duplicates(All_subscription_list))
    then
    {
        action(noDuplicateSubscription)
    }
    .

unique_context_variables_cs_one ::=
    all_context_variables([],All_context_variable_list)
    ; if (no_duplicates(All_context_variable_list))
    then
    {
        action(noDuplicateContextVariable)
    }
    .

unique_environment_component_relation_cs_one ::=

all_environment_component_relation([],All_environment_component_relation_list
)
    ; if (no_duplicates(All_environment_component_relation_list))
    then
    {
        action(noDuplicateEnvironmentComponentRelation)
    }
    .

unique_all_cs_one += uniqueEnv_cs_one /\ uniqueComponent_cs_one /\
uniqueEvent_cs_one /\ uniqueAdvertisement_cs_one /\ uniqueSubscription_cs_one
/\ uniqueContextVariable_cs_one /\ uniqueEnvironmentComponentRelation_cs_one.

uniqueEnv_cs_one += [-noDuplicateEnv]uniqueEnv_cs_one /\ <->tt.
uniqueComponent_cs_one += [-noDuplicateComponent]uniqueComponent_cs_one /\ <-
>tt.
uniqueEvent_cs_one += [-noDuplicateEvent]uniqueEvent_cs_one /\ <->tt.
uniqueAdvertisement_cs_one += [-
noDuplicateAdvertisement]uniqueAdvertisement_cs_one /\ <->tt.
uniqueSubscription_cs_one += [-
noDuplicateSubscription]uniqueSubscription_cs_one /\ <->tt.
```

```
uniqueContextVariable_cs_one += [-
noDuplicateContextVariable]uniqueContextVariable_cs_one /\ <->tt.
uniqueEnvironmentComponentRelation_cs_one += [-
noDuplicateEnvironmentComponentRelation]uniqueEnvironmentComponentRelation_cs
_one /\ <->tt.

% UNIQUE COMPONENTS

% VALIDATE RELATIONS

chk_all_relations_cs_one ::=
    chk_env_component_relation_cs_one
    ; chk_advertisement_relation_cs_one
    ; chk_subscription_relation_cs_one
    ; chk_context_variable_relation_cs_one
    ; chk_for_dual_role_cs_one
    .

chk_env_component_relation_cs_one ::=

all_environment_component_relation_as_nested_lists([],All_environment_compone
nt_relation_list)
    ; if
(validate_env_component_relation_list(All_environment_component_relation_list
))
    then
    {
        action(allValidEnvComponentRelation)
    }
    .

chk_advertisement_relation_cs_one ::=

all_advertisements_relation_as_nested_lists([],Advertisement_component_event_
relation_list)
    ;
if(validate_advertisement_relation_list(Advertisement_component_event_relatio
n_list))
    then
    {
        action(allValidAdvertisements)
    }
    .

chk_subscription_relation_cs_one ::=

all_subscription_relation_as_nested_lists([],Subscription_component_event_rel
ation_list)
    ;
if(validate_subscription_relation_list(Subscription_component_event_relation_
list))
    then
    {
        action(allValidSubscriptions)
    }
    .
```

```
chk_context_variable_relation_cs_one ::=

all_context_variable_relation_as_nested_lists([],Context_variable_relation_li
st)
    ;
if(validate_context_variable_relation_list(Context_variable_relation_list))
    then
    {
        action(allValidContextVariable)
    }
    .

chk_for_dual_role_cs_one ::=

all_subscription_relation_as_nested_lists([],Subscription_component_event_rel
ation_list)
    ;
if(validate_no_dual_membership_list(Subscription_component_event_relation_lis
t))
    then
    {
        action(allValidNoDualMembership)
    }
    .

all_valid_relations_cs_one += valid_env_component_relations_cs_one /\
valid_adv_relations_cs_one /\ valid_subs_relations_cs_one /\
valid_context_variable_relations_cs_one /\ valid_no_dual_membership_cs_one.

valid_env_component_relations_cs_one += [-
allValidEnvComponentRelation]valid_env_component_relations_cs_one /\ <->tt.
valid_adv_relations_cs_one += [-
allValidAdvertisements]valid_adv_relations_cs_one /\ <->tt.
valid_subs_relations_cs_one += [-
allValidSubscriptions]valid_subs_relations_cs_one /\ <->tt.
valid_context_variable_relations_cs_one += [-
allValidContextVariable]valid_context_variable_relations_cs_one /\ <->tt.
valid_no_dual_membership_cs_one += [-
allValidNoDualMembership]valid_no_dual_membership_cs_one /\ <->tt.

% CAUSALITY
% mck(send_before_receive_after,causality).

send_before_receive_after_cs_one ::=
    chk_send_event
    ; chk_receive_event
    .

causality_cs_one += [recvnew]tt \/ ([-]causality /\ [sendnew]tt).

% CAUSALITY

% SPONTANEOUS EVENT
% mck(chk_spontaneous_event, never_receive_spontaneous_event)

chk_spontaneous_event_cs_one ::=
    cleanupComponentTriggPredicates(calendar_application)
```

```
    ; hasComponentReceivedResult(calendar_application,locationEvent,Result)
    ; if(Result == 1)
    then
    {
        action(spontaneousEventReceived)
    }
    .

never_receive_spontaneous_event_cs_one -= [spontaneousEventReceived]ff /\ [-
]never_receive_spontaneous_event.

% SPONTANEOUS EVENT

% EVENT WITHOUT ADVERTISEMENT
% mck(chk_event_without_advertisement,
never_send_event_without_advertisement)

chk_event_without_advertisement_cs_one ::=
    if
(not(simulate_event_environment(users_phone,meetingStartedEvent,[meetingStart
edEvent,[[meetingName,board_meeting],[date,sfsdfs],[startTime,sdfsdf],[durati
on,1]]])))
    then
    {
        action(unableToSendEventWithoutAdvertisement)
    }
    .

never_send_event_without_advertisement_cs_one += [-
unableToSendEventWithoutAdvertisement]never_send_event_without_advertisement
/\ <->tt.

% EVENT WITHOUT ADVERTISEMENT

% EVENT WITH INVALID SCHEMA

chk_invalid_event_schema_cs_one ::=
    if
(not(simulate_event_component(calendar_application,users_phone,meetingStarted
Event,[meetingStartedEvent,[[meetingName,board_meeting],[date,sfsdfs],[startT
ime,sdfsdf],[duration,xx]]])))
    then
    {
        action(unableToSendEventWithInvalidSchema)
    }
    .

never_able_to_send_event_with_invalid_schema_cs_one += [-
unableToSendEventWithInvalidSchema]never_able_to_send_event_with_invalid_sche
ma /\ <-> tt.

% EVENT WITH INVALID SCHEMA

% NON EXISTANT COMPONENT CANNOT SEND EVENT

chk_cannot_send_non_exist_component_cs_one ::=
```

```
if(not(simulate_event_component(non_existent_component,users_phone,meetingSta
rtedEvent,[meetingStartedEvent,[[meetingName,board_meeting],[date,sfsdfs],[st
artTime,sdfsdf],[duration,1]]]))
    then
    {
        action(nonExistentComponentUnableToSendEvent)
    }
    .

never_send_event_non_existent_component_cs_one += [-
nonExistentComponentUnableToSendEvent]never_send_event_non_existent_component
/\ <->tt.

% NON EXISTANT COMPONENT CANNOT SEND EVENT

% COMPONENT CANNOT SEND EVENT THAT IS NOT DEFINED

chk_cannot_send_non_exist_event_cs_one ::=
    if
(not(simulate_event_environment(calendar_application,nonExistentEvent,[nonExi
stentEvent,[[action,99],[value,23]]])))
    then
    {
        action(unableToSendUndefinedEvent)
    }
    .

never_send_undefined_event_cs_one += [-
unableToSendUndefinedEvent]never_send_undefined_event /\ <->tt.

% COMPONENT CANNOT SEND EVENT THAT IS NOT DEFINED

% FIDELITY
% mck(fidelity_chk,fidelity).
send_event_for_fidelity_cs_one ::=

simulate_event_component(calendar_application,users_phone,meetingStartedEvent
,[meetingStartedEvent,[[meetingName,board_meeting],[date,sfsdfs],[startTime,s
dfsdf],[duration,1]]])
    ;
assert(event_fidelity_chk(users_phone,meetingStartedEvent,[meetingStartedEven
t,[[meetingName,board_meeting],[date,sfsdfs],[startTime,sdfsdf],[duration,1]]
]))
    .

receive_event_for_fidelity_cs_one ::=
    hasComponentReceivedResult(users_phone,meetingStartedEvent,Result)
    ; if (Result==1)
    then
    {
        triggered(users_phone,meetingStartedEvent, Event_data)
        ;
event_fidelity_chk(users_phone,meetingStartedEvent,Event_data_stored)

        ; if (Event_data_stored == Event_data)
        then
```

```
        {
            action(fidelityTestPassed)
        }
        else
        {
            action(fidelityTestFailed)
        }
    }
    .

fidelity_chk_cs_one ::=
    send_event_for_fidelity_cs_one
    ; receive_event_for_fidelity_cs_one
    .

fidelity_cs_one += never_fidelity_fail.
possible_fidelity_pass_cs_one += <fidelityTestPassed>tt \/ <-
>possible_fidelity_pass.
never_fidelity_fail_cs_one -= [fidelityTestFailed]ff /\ [-
]never_fidelity_fail.

% FIDELITY

% VALID STATE WHILE EVENT IS TRAVERSED
% mck(runtime_validity,runtime_validity_property).

runtime_duplicates_cs_one ::=
    chk_send_event
    | chk_duplicates_cs_one
    .

runtime_relations_cs_one ::=
    chk_send_event
    | chk_all_relations_cs_one
    .

runtime_validity_cs_one ::=
    chk_send_event
    | chk_relations_duplicates_cs_one
    .

chk_relations_duplicates_cs_one ::=
    chk_all_relations_cs_one
    ; chk_duplicates_cs_one
    .

runtime_duplicates_property_cs_one += unique_all_cs_one.
runtime_relations_property_cs_one += all_valid_relations_cs_one.
runtime_validity_property_cs_one += unique_all_cs_one /\
all_valid_relations_cs_one.

% VALID STATE WHILE EVENT IS TRAVERSED

% EVENT WITHOUT SUBSCRIPTION

chk_event_without_subscription_cs_one ::=
```

```
simulate_event_environment(users_phone,incomingCallEvent,[incomingCallEvent,[
[callerName,akshat],[callingNumber,a99]]])
    ;
hasComponentReceivedResult(calendar_application,incomingCallEvent,Result)
    ; if (Result == 1)
    then
    {
        action(receivedEventWithoutSubscription)
    }
    .

never_receive_event_without_subscription_cs_one -=
[receivedEventWithoutSubscription]ff /\ [-
]never_receive_event_without_subscription_cs_one.

% EVENT WITHOUT SUBSCRIPTION

% CONTENT FILTER CHECK

chk_content_filter_blocking_cs_one ::=
    chk_content_filter_send_blocked_event_cs_one
    ; chk_content_filter_receive_blocked_event_cs_one
    .

chk_content_filter_send_blocked_event_cs_one ::=

simulate_event_environment(users_phone,locationEvent,[locationEvent,[[latitud
e,5],[longitude,1]]])
    .

chk_content_filter_receive_blocked_event_cs_one ::=
    hasComponentReceivedResult(calendar_application,locationEvent,Result)
    ; if (Result==1)
    then
    {
        action(receivedBlockedEvent)
    }
    .

chk_receiver_content_filter_blocking_cs_one -= [receivedBlockedEvent]ff /\ [-
]chk_receiver_content_filter_blocking.

% CONTENT FILTER CHECK

% DISCONNECTED COMPONENT

chk_disconnected_component_cs_one ::=
    disconnect_component_cs_one
    ; chk_disconnected_component_send_cs_one
    ; reconnect_component_cs_one
    .

disconnect_component_cs_one ::=
    retract(environment_component(users_phone,calendar_application))
    .
```

```
reconnect_component_cs_one ::=
    assert(environment_component(users_phone,calendar_application))
    .

chk_disconnected_component_send_cs_one ::=
    cleanupComponentTriggPredicates(calendar_application)
    ;
simulate_event_environment(users_phone,locationEvent,[locationEvent,[[latitud
e,1],[longitude,1]]])
    ; hasComponentReceivedResult(calendar_application,locationEvent,Result)
    ; if(Result == 1)
    then
    {
        action(eventByComponentWhileDisconnected)
    }
    .

never_event_received_when_disconnected_cs_one -=
[eventByComponentWhileDisconnected]ff /\ [-
]never_event_received_when_disconnected_cs_one.

% DISCONNECTED COMPONENT

% UNDEFINED COMPONENT CANNOT RECEIVE EVENT

chk_undefined_component_cannot_receive_event_cs_one ::=
    cleanupComponentTriggPredicates(undefined_component)
    ;
simulate_event_environment(users_phone,locationEvent,[locationEvent,[[latitud
e,1],[longitude,1]]])
    ; hasComponentReceivedResult(undefined_component,locationEvent,Result)
    ; if(Result == 1)
    then
    {
        action(undefinedComponentReceivedEvent)
    }
    .

never_receive_event_undefined_component_cs_one -=
[undefinedComponentReceivedEvent]ff /\ [-
]never_receive_event_undefined_component_cs_one.

% UNDEFINED COMPONENT CANNOT RECEIVE EVENT

% COMPONENT CANNOT RECEIVE AN EVENT FOR WHICH A SCHEMA IS NOT DEFINED

chk_cannot_receive_undefined_event_cs_one ::=
    cleanupComponentTriggPredicates(users_phone)
    ; hasComponentReceivedResult(users_phone,undefined_event_type,Result)
    ; if(Result == 1)
    then
    {
        action(undefinedEventReceived)
    }
    .
```

```
never_receive_undefined_event_cs_one -= [undefinedEventReceived]ff /\ [-
]never_receive_undefined_event_cs_one.
```

C.2 Cellphone as controller

```
% RECEIVER CONTEXT FILTER
% mck(testContextFilter(1),filterNotBlocking)
% mck(testContextFilter(0),filterBlocking)

send_trigger_home_action_event ::=

simulate_event_environment(true_remote_control_application,triggerHomeActionE
vent,[triggerHomeActionEvent,[[action,setTemprature],[value,23]]])
    ; action(sendTriggerHomeActionEvent)
    .

chk_received_trigger_home_action_event ::=

hasComponentReceivedResult(smart_home_controller_unit,triggerHomeActionEvent,
Result)
    ; if (Result==1)
    then
    {
        action(receivedTriggerHomeActionEvent)
    }
    .

testContextFilter(Pass) ::=
    change_context_variable_value(mustTriggerHomeActionsContextElement,Pass)
    ; send_trigger_home_action_event
    | chk_received_trigger_home_action_event
    .

filterBlocking += contextFilterBlock1 /\ contextFilterBlock2.
contextFilterBlock1 += <sendTriggerHomeActionEvent>tt \/ <-
>contextFilterBlock1.
contextFilterBlock2 -= [receivedTriggerHomeActionEvent]ff /\ [-
]contextFilterBlock2.

filterNotBlocking += contextFilterBlock1 /\ contextFilterNonBlock1.
contextFilterNonBlock1 += <receivedTriggerHomeActionEvent>tt \/ <-
>contextFilterNonBlock1.

% RECEIVER CONTEXT FILTER

% BLOCKING RECEIVER CONTENT FILTER
% mck(testContentFilter,contentFilterBlocking).
% mck(testContentFilterNotBlocked,contentFilterNotBlocking).

send_alarm_event_blocked ::=

simulate_event_component(alarm_application,user_cellphone,alarmGoesOffEvent,[
alarmGoesOffEvent,[[dateTime,sometime]]])
    ; action(sendBlockedAlarmEvent)
    .
```

```
send_alarm_event_not_blocked ::=

simulate_event_component(alarm_application,user_cellphone,alarmGoesOffEvent,[
alarmGoesOffEvent,[[dateTime,othertime]]])
    ; action(sendBlockedAlarmEvent)
    .

chk_received_alarm_event ::=

hasComponentReceivedResult(true_remote_control_application,alarmGoesOffEvent,
Result)
    ; if (Result==1)
    then
    {
        action(receivedAlarmEvent)
    }
    .

testContentFilter ::=
    send_alarm_event_blocked
    ; chk_received_alarm_event
    .

testContentFilterNotBlocked ::=
    send_alarm_event_not_blocked
    ; chk_received_alarm_event
    .

contentFilterBlocking += contentFilterBlock1 /\ contentFilterBlock2.
contentFilterBlock1 += <sendBlockedAlarmEvent>tt \/ <->contentFilterBlock1.
contentFilterBlock2 -= [receivedAlarmEvent]ff /\ [-]contentFilterBlock2.

contentFilterNotBlocking += contentFilterBlock1 /\ contentFilterNonBlock1.
contentFilterNonBlock1 += <receivedAlarmEvent>tt \/ <-
>contentFilterNonBlock1.

% BLOCKING RECEIVER CONTENT FILTER

% UNIQUE COMPONENTS
% mck(chk_duplicates,unique_all).

chk_duplicates ::=
    unique_environments
    ; unique_components
    ; unique_events
    ; unique_advertisements
    ; unique_subscriptions
    ; unique_context_variables
    ; unique_environment_component_relation.

unique_environments ::=
    all_environemtns([],All_env_list)
    ; if (no_duplicates(All_env_list))
    then
    {
        action(noDuplicateEnv)
```

```
    }
    .

unique_components ::=
    all_components([],All_component_list)
    ; if (no_duplicates(All_component_list))
    then
    {
        action(noDuplicateComponent)
    }
    .

unique_events ::=
    all_event_names([],All_event_list)
    ; if (no_duplicates(All_event_list))
    then
    {
        action(noDuplicateEvent)
    }
    .

unique_advertisements ::=
    all_advertisements([],All_advertisement_list)
    ; if (no_duplicates(All_advertisement_list))
    then
    {
        action(noDuplicateAdvertisement)
    }
    .

unique_subscriptions ::=
    all_subscriptions([],All_subscription_list)
    ; if (no_duplicates(All_subscription_list))
    then
    {
        action(noDuplicateSubscription)
    }
    .

unique_context_variables ::=
    all_context_variables([],All_context_variable_list)
    ; if (no_duplicates(All_context_variable_list))
    then
    {
        action(noDuplicateContextVariable)
    }
    .

unique_environment_component_relation ::=

all_environment_component_relation([],All_environment_component_relation_list
)
    ; if (no_duplicates(All_environment_component_relation_list))
    then
    {
        action(noDuplicateEnvironmentComponentRelation)
    }
```

```
    .

unique_all += uniqueEnv /\ uniqueComponent /\ uniqueEvent /\
uniqueAdvertisement /\ uniqueSubscription /\ uniqueContextVariable /\
uniqueEnvironmentComponentRelation.

uniqueEnv += [-noDuplicateEnv]uniqueEnv /\ <->tt.
uniqueComponent += [-noDuplicateComponent]uniqueComponent /\ <->tt.
uniqueEvent += [-noDuplicateEvent]uniqueEvent /\ <->tt.
uniqueAdvertisement += [-noDuplicateAdvertisement]uniqueAdvertisement /\ <-
>tt.
uniqueSubscription += [-noDuplicateSubscription]uniqueSubscription /\ <->tt.
uniqueContextVariable += [-noDuplicateContextVariable]uniqueContextVariable
/\ <->tt.
uniqueEnvironmentComponentRelation += [-
noDuplicateEnvironmentComponentRelation]uniqueEnvironmentComponentRelation /\
<->tt.

% UNIQUE COMPONENTS

% VALID RELATIONS = ROLE INTEGRITY + ACTION INEGRITY
% mck(chk_all_relations,all_valid_relations).

chk_all_relations ::=
    chk_env_component_relation
    ; chk_advertisement_relation
    ; chk_subscription_relation
    ; chk_context_variable_relation
    ; chk_for_dual_role
    .

chk_env_component_relation ::=

all_environment_component_relation_as_nested_lists([],All_environment_compone
nt_relation_list)
    ; if
(validate_env_component_relation_list(All_environment_component_relation_list
))
    then
    {
        action(allValidEnvComponentRelation)
    }
    .

chk_advertisement_relation ::=

all_advertisements_relation_as_nested_lists([],Advertisement_component_event_
relation_list)
    ;
if(validate_advertisement_relation_list(Advertisement_component_event_relatio
n_list))
    then
    {
        action(allValidAdvertisements)
    }
    .
```

```
chk_subscription_relation ::=

all_subscription_relation_as_nested_lists([],Subscription_component_event_rel
ation_list)
    ;
if(validate_subscription_relation_list(Subscription_component_event_relation_
list))
    then
    {
        action(allValidSubscriptions)
    }
    .

chk_context_variable_relation ::=

all_context_variable_relation_as_nested_lists([],Context_variable_relation_li
st)
    ;
if(validate_context_variable_relation_list(Context_variable_relation_list))
    then
    {
        action(allValidContextVariable)
    }
    .

chk_for_dual_role ::=

all_subscription_relation_as_nested_lists([],Subscription_component_event_rel
ation_list)
    ;
if(validate_no_dual_membership_list(Subscription_component_event_relation_lis
t))
    then
    {
        action(allValidNoDualMembership)
    }
    .

all_valid_relations += valid_env_component_relations /\ valid_adv_relations
/\ valid_subs_relations /\ valid_context_variable_relations /\
valid_no_dual_membership.

valid_env_component_relations += [-
allValidEnvComponentRelation]valid_env_component_relations /\ <->tt.
valid_adv_relations += [-allValidAdvertisements]valid_adv_relations /\ <->tt.
valid_subs_relations += [-allValidSubscriptions]valid_subs_relations /\ <-
>tt.
valid_context_variable_relations += [-
allValidContextVariable]valid_context_variable_relations /\ <->tt.
valid_no_dual_membership += [-
allValidNoDualMembership]valid_no_dual_membership /\ <->tt.

% VALID RELATIONS

% FIDELITY
% mck(fidelity_chk,fidelity).
send_event_for_fidelity ::=
```

```
simulate_event_environment(true_remote_control_application,triggerHomeActionE
vent,[triggerHomeActionEvent,[[action,setTemprature],[value,23]]])
    ;
assert(event_fidelity_chk(true_remote_control_application,triggerHomeActionEv
ent,[triggerHomeActionEvent,[[action,setTemprature],[value,23]]]))
    .

receive_event_for_fidelity ::=

hasComponentReceivedResult(smart_home_controller_unit,triggerHomeActionEvent,
Result)
    ; if (Result==1)
    then
    {

triggered(smart_home_controller_unit,triggerHomeActionEvent,Event_data)
        ;
event_fidelity_chk(true_remote_control_application,triggerHomeActionEvent,Eve
nt_data_stored)

        ; if (Event_data_stored == Event_data)
        then
        {
            action(fidelityTestPassed)
        }
        else
        {
            action(fidelityTestFailed)
        }
    }
    .

fidelity_chk ::=
    send_event_for_fidelity
    ; receive_event_for_fidelity
    .

fidelity += never_fidelity_fail.
possible_fidelity_pass += <fidelityTestPassed>tt \/ <-
>possible_fidelity_pass.
never_fidelity_fail -= [fidelityTestFailed]ff /\ [-]never_fidelity_fail.

% FIDELITY

% CAUSALITY
% mck(send_before_receive_after,causality).

send_before_receive_after ::=
    change_context_variable_value(mustTriggerHomeActionsContextElement,Pass)
    ; send_trigger_home_action_event
    ; chk_received_trigger_home_action_event
    .

causality += [receivedTriggerHomeActionEvent]tt \/ ([-]causality /\
[sendTriggerHomeActionEvent]tt).
```

```
% CAUSALITY

% VALID STATE WHILE EVENT IS TRAVERSED
% mck(runtime_validity,runtime_validity_property).

runtime_duplicates ::=
    send_trigger_home_action_event
    | chk_duplicates
    .

runtime_relations ::=
    send_trigger_home_action_event
    | chk_all_relations
    .

runtime_validity ::=
    send_trigger_home_action_event
    | chk_relations_duplicates
    .

chk_relations_duplicates ::=
    chk_all_relations
    ; chk_duplicates
    .

runtime_duplicates_property += unique_all.
runtime_relations_property += all_valid_relations.
runtime_validity_property += unique_all /\ all_valid_relations.

% VALID STATE WHILE EVENT IS TRAVERSED

% SPONTANEOUS EVENT
% mck(chk_spontaneous_event, never_receive_spontaneous_event)

chk_spontaneous_event ::=
    cleanupComponentTriggPredicates(true_remote_control_application)
    ;
hasComponentReceivedResult(true_remote_control_application,alarmGoesOffEvent,
Result)
    ; if(Result == 1)
    then
    {
        action(spontaneousEventReceived)
    }
    .

never_receive_spontaneous_event -= [spontaneousEventReceived]ff /\ [-
]never_receive_spontaneous_event.

% SPONTANEOUS EVENT

% EVENT WITHOUT ADVERTISEMENT
% mck(chk_event_without_advertisement,
never_send_event_without_advertisement)

chk_event_without_advertisement ::=
```

```
    if
(not(simulate_event_component(alarm_application,user_cellphone,resultOfHomeAc
tionEvent,[resultOfHomeActionEvent,[[action,string],[value,string],[result,st
ring]]])))
    then
    {
        action(unableToSendEventWithoutAdvertisement)
    }
    .

never_send_event_without_advertisement += [-
unableToSendEventWithoutAdvertisement]never_send_event_without_advertisement
/\ <->tt.

% EVENT WITHOUT ADVERTISEMENT

% EVENT WITH INVALID SCHEMA

chk_invalid_event_schema ::=
    if
(not(simulate_event_environment(true_remote_control_application,triggerHomeAc
tionEvent,[triggerHomeActionEvent,[[action,99],[value,23]]])))
    then
    {
        action(unableToSendEventWithInvalidSchema)
    }
    .

never_able_to_send_event_with_invalid_schema += [-
unableToSendEventWithInvalidSchema]never_able_to_send_event_with_invalid_sche
ma /\ <-> tt.

% EVENT WITH INVALID SCHEMA'

% EVENT WITHOUT SUBSCRIPTION

chk_event_without_subscription ::=

simulate_event_environment(true_remote_control_application,notifyUserOfCarEng
ineStartedEvent,[notifyUserOfCarEngineStartedEvent,[[result,1]]])
    ;
hasComponentReceivedResult(smart_home_controller_unit,notifyUserOfCarEngineSt
artedEvent,Result)
    ; if (Result == 1)
    then
    {
        action(receivedEventWithoutSubscription)
    }
    .

never_receive_event_without_subscription -=
[receivedEventWithoutSubscription]ff /\ [-
]never_receive_event_without_subscription.

% EVENT WITHOUT SUBSCRIPTION

% NON EXISTANT COMPONENT CANNOT SEND EVENT
```

```
chk_cannot_send_non_exist_component ::=

if(not(simulate_event_component(non_existent_component,user_cellphone,alarmGo
esOffEvent,[alarmGoesOffEvent,[[dateTime,othertime]]])))
    then
    {
        action(nonExistentComponentUnableToSendEvent)
    }
    .

never_send_event_non_existent_component += [-
nonExistentComponentUnableToSendEvent]never_send_event_non_existent_component
/\ <->tt.

% NON EXISTANT COMPONENT CANNOT SEND EVENT

% COMPONENT CANNOT SEND EVENT THAT IS NOT DEFINED

chk_cannot_send_non_exist_event ::=
    if
(not(simulate_event_environment(true_remote_control_application,nonExistentEv
ent,[nonExistentEvent,[[action,99],[value,23]]])))
    then
    {
        action(unableToSendUndefinedEvent)
    }
    .

never_send_undefined_event += [-
unableToSendUndefinedEvent]never_send_undefined_event /\ <->tt.

% COMPONENT CANNOT SEND EVENT THAT IS NOT DEFINED

% COMPONENT CANNOT RECEIVE AN EVENT FOR WHICH A SCHEMA IS NOT DEFINED

chk_cannot_receive_undefined_event ::=
    cleanupComponentTriggPredicates(true_remote_control_application)
    ;
hasComponentReceivedResult(true_remote_control_application,undefined_event_ty
pe,Result)
    ; if(Result == 1)
    then
    {
        action(undefinedEventReceived)
    }
    .

never_receive_undefined_event -= [undefinedEventReceived]ff /\ [-
]never_receive_undefined_event.

% COMPONENT CANNOT RECEIVE AN EVENT FOR WHICH A SCHEMA IS NOT DEFINED

% UNDEFINED COMPONENT CANNOT RECEIVE EVENT

chk_undefined_component_cannot_receive_event ::=
    cleanupComponentTriggPredicates(undefined_component)
```

```
    ;
simulate_event_environment(true_remote_control_application,notifyUserOfCarEng
ineStartedEvent,[notifyUserOfCarEngineStartedEvent,[[result,1]]])
    ;
hasComponentReceivedResult(undefined_component,notifyUserOfCarEngineStartedEv
ent,Result)
    ; if(Result == 1)
    then
    {
        action(undefinedComponentReceivedEvent)
    }
    .

never_receive_event_undefined_component -=
[undefinedComponentReceivedEvent]ff /\ [-
]never_receive_event_undefined_component.

% UNDEFINED COMPONENT CANNOT RECEIVE EVENT

% UNCONNECTED COMPONENT

chk_disconnected_component ::=
    disconnect_component
    ; chk_disconnected_component_send_notify_event_disconnected
    ; reconnect_component
    .

chk_disconnected_component_send_notify_event_connected ::=

simulate_event_environment(true_remote_control_application,notifyUserOfCarEng
ineStartedEvent,[notifyUserOfCarEngineStartedEvent,[[result,1]]])
    ;
hasComponentReceivedResult(user_cellphone,notifyUserOfCarEngineStartedEvent,R
esult)
    ; if(Result == 1)
    then
    {
        action(eventByComponentWhileConnected)
    }
    .

chk_disconnected_component_send_notify_event_disconnected ::=
    cleanupComponentTriggPredicates(user_cellphone)
    ;
simulate_event_environment(true_remote_control_application,notifyUserOfCarEng
ineStartedEvent,[notifyUserOfCarEngineStartedEvent,[[result,1]]])
    ;
hasComponentReceivedResult(user_cellphone,notifyUserOfCarEngineStartedEvent,R
esult)
    ; if(Result == 1)
    then
    {
        action(eventByComponentWhileDisconnected)
    }
    .

disconnect_component ::=
```

```
retract(environment_component(true_remote_control_application,user_cellphone)
)
    .

reconnect_component ::=

assert(environment_component(true_remote_control_application,user_cellphone))
    .

possible_event_received_when_connected += <eventByComponentWhileConnected>tt
\/ <->possible_event_received_when_connected.
never_event_received_when_disconnected -=
[eventByComponentWhileDisconnected]ff /\ [-
]never_event_received_when_disconnected.
chk_component_connection += possible_event_received_when_connected /\
never_event_received_when_disconnected.
```

Bibliography

[1] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing,* pp. 4-7, 2001.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," *Symposium on Handheld and Ubiquitous Computing,* pp. 304-307, 1999.

[3] D. K. Dan Hong and V. Y. S. Chiu, "Requirements Elicitation for the Design of Context-Aware Applications in a Ubiquitous Environment," in *International Conference on Electronic Commerce*, New York, NY, 2005.

[4] M. Baldauf, "A Survey on Context-Aware Systems," International Journal of Ad Hoc and Ubiquitous Computing, Geneva, SWITZERLAND, 2007.

[5] R. J. Robles and T. H. Kim, "Review: Context Aware Tools for Smart Home Development," *International Journal of Smart Home,* 2010.

[6] G. Thyagaraju and U. P. Kulkarni, "Design and Implementation of User Context aware Recommendation Engine for Mobile using Bayesian Network, Fuzzy Logic and Rule Base," *International Journal of Computer Applications,* 2012.

[7] B. Schilit and M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *Network, IEEE,* vol. 8, no. 5, pp. 22 - 32, 1994.

[8] H.-A. Jacobsen, A. Cheung, G. Li, B. Maniymaran, V. Muthusamy and R. S. Kazemzadeh, "The padres distributed publish/subscribe system," in *Feature Interactions in Telecommunications and Software Systems*, 2005.

[9] K. Birman and T. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," *Symposium on Operating systems principles,* vol. 21, no. 5, pp. 123 - 138, 1987.

[10] L. Blanco, An Adaptive Context-Aware Publish Subscribe Component Metamodel, Waterloo, ON, Canada: University of Waterloo, 2015.

[11] J. S. Collofello, Introduction to Software Verification and Validation, Carnegie Mellon University Software Engineering Institute, 1988.

[12] R. M. Hierons, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, H. Zedan, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman and K. Kapoor, "Using Formal Specifications to Support Testing," *ACM Computing Surveys,* vol. 41, no. 2, 2009.

[13] M.-C. Gaudel, "Formal Specification Techniques," in *ICSE*, 1994.

[14] J. Westland, "The Cost Behavior of Software Defects," *Decision Support Systems,* vol. 37, no. 2, pp. 229 - 238, 2004.

[15] V. Vieira, K. Holl and M. Hassel, "A Context Simulator As Testing Support for Mobile Apps," in *Symposium on Applied Computing*, Salamanca, Spain, 2015.

[16] E. S. Da Costa, R. d. A. C. Reis and A. C. Dias-Neto, "Extension of Sikuli Tool to Support Automated Tests to Windows Phone Context-Aware Applications," in *UBICOMM*, 2015.

[17] B. Schilit, N. Adams and R. Want, "Context-Aware Computing Applications," in *Mobile Computing Systems and Applications*, Santa Cruz, California, 1994.

[18] B. Schilit and N. York, "Context-Aware Computing Applications," pp. 85-90, 1995.

[19] P. J. Brown, J. D. Bovey and X. Chen, "Context-Aware Applications: From the Laboratory to the Marketplace," *IEEE Personal Communications,* 1997.

[20] A. R. Ana, Á. M. G. Carvalho and C. G. Ralha, "Agent-Based Architecture for Context-Aware and Personalized Event Recommendation," *Expert Systems with Applications,* vol. 41, no. 2, pp. 563-573, 2014.

[21] H. Guermah, T. Fissaa, H. Hafiddi, M. Nassar and A. Kriouile, "An Ontology Oriented Architecture for Context Aware Services Adaptation," *International Journal of Computer Science Issues,* vol. 11, no. 2, 2014.

[22] S. Helmer, A. Poulovassilis and F. Xhafa, Reasoning in Event-Based Distributed Systems, Springer, 2011.

[23] E. S. Barrenechea, P. S. C. Alencar, R. Blanco and D. Cowan, "Context-Awareness and Adaptation in," University of Waterloo, Waterloo, Ontario, Canada.

[24] R. Chaffey and D. Thurner, Mobile Marketing Strategy Guide, Smart Insights.

[25] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," in *Symposium on wearable computers*, 1996.

[26] K. Lyons, "What can a Dumb Watch Teach a Smartwatch?: Informing the Design of Smartwatches," in *International Symposium on Wearable Computers*, 2015.

[27] P. T. Eugster, P. A. Felber, R. Guerraoui and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Comput. Surv.,* vol. 35, pp. 114--131, 2003.

[28] A. v. Lamsweerde, "Formal Specification: A Roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, Limerick, Ireland, 2000.

[29] A. Sanghavi, "What is Formal Verification?," *EE Times Asia,* 21 May 2010.

[30] P. Ferreira, Distributed Context-Aware Systems, Springer.

[31] A. Uzun, M. Salem and A. Kupper, "Exploiting Location Semantics for Realizing Cross-Referencing Proactive Location-Based Services," *IEEE International Conference on Semantic Computing,* pp. 76-83, 2014.

[32] H. Chen, T. Finin and A. Joshi, "An Ontology for Context-Aware Pervasive Computing Environments," *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review,* vol. 18, pp. 197--207, 2003.

[33] T. Winograd, "Architectures for Context," *Human-Computer Interaction,* vol. 16, no. 2, pp. 401-419, 2001.

[34] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," in *First International Workshop on Advanced Context Modelling*, Nottingham, England, 2004.

[35] Q. Z. Benatallah and S. a. Boualem, "ContextUML: a UML-based modeling language for model-driven development of context-aware Web services," *Mobile Business,* pp. 206 - 212, 2005.

[36] K. Henricksen, J. Indulska and A. Rakotonirainy, "Generating Context Management Infrastructure from High-Level Context Models," in *International Conference on Mobile Data Management*, 2003.

[37] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann and W. Retschitzegger, "Context-Awareness on Mobile Devices - The Hydrogen Approach," *System Sciences,* 2003.

[38] J. McCarthy and S. Buvac, "Formalizing Context," *Fall Symposium on Context in Knowledge Representation and Natural Language,* p. 99–135, 1997.

[39] N. B. Priyantha, A. Chakraborty and H. Balakrishnan, "The Cricket Location-Support System," *nternational Conference on Mobile Computing and Networking,* 2000.

[40] F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore and M. Bylund, "GeoNotes: Social and Navigational Aspects of Location-Based Information Systems," in *Ubicomp 2001: Ubiquitous Computing*, Springer, 2001, pp. 2-17.

[41] J. Burrell and G. Gay, "E-graffiti: Evaluating Real-World Use of a Context-Aware System," in *Interacting with*, 2002, p. 301–312.

[42] C. Kerer, S. Dustdar, M. Jazayeri, D. Gomes, A. Szego and J. A. B. Caja, "Presence-Aware Infrastructure Using Web Services and RFID Technologies," in *Proceedings of the 2nd European workshop on object orientation and web services*, Oslo, Norway, 2004.

[43] M. A. Muñoz, V. M. Gonzalez, M. Rodríguez and J. Favela, "Supporting Context-Aware Collaboration in a Hospital: An Ethnographic Informed Design," in *Groupware: Design, Implementation, and Use*, Autrans, France, Springer, 2003, pp. 330-344.

[44] P. Korpipää and J. Mäntyjärvi, "An Ontology for Mobile Device Sensor-Based Context Awareness," in *Modeling and Using Context*, Stanford, CA, USA, Springer, 2003, pp. 451-458.

[45] T. Gu, H. K. Pung and D. Q. Zhang, "A Middleware for Building Context-Aware Mobile Services," in *Vehicular Technology Conference*, 2004.

[46] P. Fahy and S. Clarke, "CASS–A Middleware for Mobile Context-Aware Applications," in *Workshop on context awareness, MobiSys*, 2004.

[47] H. Chen, T. Finin and A. Joshi, "Semantic Web in a Pervasive Context-Aware Architecture," 2014.

[48] G. Biegel and V. Cahill, "A Framework for Developing Mobile, Context-Aware Applications," in *Pervasive Computing and Communications*, 2004.

[49] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Nahrstedt., "A Middleware Infrastructure for Active Spaces," *IEEE Pervasive Computing,* pp. 74-83, 2002.

[50] S. N. Sehic and M. Stefan. Vögler, "Entity-Adaptation : A Programming Model for Development of Context-Aware Applications," *Symposium on Applied Computing,* pp. 436-443, 2014.

[51] Y. Liu and B. Plale, "Survey of Publish Subscribe Event Systems," *Indiana University Department of Computer Science,* pp. 1-19, 2003.

[52] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Storm and D. C. Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," *Distributed Computing Systems,* pp. 262 - 272, 1999.

[53] A. Rowstron, A.-M. Kermarrec, M. Castro and P. Druschel, "Scribe: The Design of a Large-Scale Event Notification Infrastructure," in *Networked Group Communication*, London, UK, Springer, 2001, pp. 30-43.

[54] A. Rowstron and P. Druschel, "Pastry: Scalable Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware 2001*, Heidelberg, Germany, Springer, 2001, pp. 329-350.

[55] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz and J. D. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination," in *Network and operating systems support for digital audio and video*, 2001.

[56] B. Y. Zhao, J. Kubiatowicz and A. D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," University of California at Berkeley, Berkeley, CA, USA, 2001.

[57] A. Carzaniga, D. S. Rosenblum and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *Transactions on Computer Systems,* vol. 19, no. 3, pp. 332-383, 2001.

[58] A. Campailla, S. Chaki, E. Clarke, S. Jha and H. Veith, "Efficient Filtering in Publish-Subscribe Systems Using Binary Decision," in *International Conference on Software Engineering*, 2001.

[59] G. C. Fox and S. Pallickara, "An Event Service to Support Grid Computational Environments," in *Concurrency and Computation: Practice and Experience 14*, 2002, pp. 13-15.

[60] A. Slominski, Y. Simmhan, A. Rossi, M. Farrellee and D. Gannon, "Xevents/Xmessages: Application Events and Messaging Framework for Grid," Indiana University, Extreme Lab, 2001.

[61] K. Kumar, J. Liu, Y. H. Lu and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications,* vol. 18, no. 1, pp. 129-140, 2013.

[62] K. C. Barr, "Energy Aware Lossless Data Compression," MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2002.

[63] R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown," *MOBICOM,* 2002.

[64] G. J. Smit, P. J. Havinga, L. T. Smit, P. M. Heysters and M. A. Rosien, "Dynamic Reconfiguration in Mobile Systems," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, Springer, 2002, pp. 171-181.

[65] M. Galster, D. Weyns, D. Tofan, B. Michalik and P. Avgeriou, "Variability in Software Systems - A Systematic Literature Review," IEEE, 2014.

[66] J. G. Halfond, S. Mcilroy, M. Nagappan and W. G. J., "Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers," in *International Conference on Software Engineering*, 2015.

[67] S. Hao, D. Li, W. G. J. Halfond and R. Govindan, "Estimating Android Applications' CPU Energy Usage via Bytecode Profiling," in *Green and Sustainable Software*, 2015.

[68] D. Li, A. H. Tran and W. G. J. Halfond, "Optimizing Display Energy Consumption for Hybrid Android Apps," in *Software Development Lifecycle for Mobile*, 2015.

[69] T. Coe, "Computational Aspects of the Pentium Affair," *Computational Science & Engineering,* vol. 2, no. 1, pp. 18 - 30, 1995.

[70] D. Beyer, T. a. Henzinger, R. Jhala and R. Majumdar, "The Software Model Checker Blast: Applications to Software Engineering," *International Journal on Software Tools for Technology Transfer,* vol. 9, no. 5-6, pp. 505-525, 2007.

[71] G. J. Holzmann and R. Joshi, "Model-Driven Software Verification," in *Model Checking Software*, Barcelona, Spain, Springer, 2004, pp. 76-91.

[72] K. Sagonas, T. Swift and D. S. Warren, "XSB As an Efficient Deductive Database Engine," *SIGMOD Rec.,* vol. 23, pp. 442--453, 1994.

[73] C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, Y. Dong, X. Du, A. Roychoudhury and V. N. Venkatakrishnan, "XMC: A Logic-Programming-Based Verification Toolset," in *Computer Aided Verification*, Chicago, IL, USA, Springer, 2000, pp. 576-580.

[74] J. Dong, Design Component Contracts: Modeling and, Waterloo, Ontario, Canada: University of Waterloo, 2002.

[75] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift and D. S. Warren, "Efficient Model Checking Using Tabled Resolution," in *Computer Aided Verification*, Haifa, Israel, Springer, 1997, pp. 143-154.

[76] R. Blanco, J. Wang and P. Alencar, "A Metamodel for Distributed Event Based Systems," in *Proceedings of the Second International Conference on Distributed Event-Based Systems*, New York, USA, 2008.

[77] R. Blanco and P. Alencar, "Towards Modularization and Composition in Distributed Event Based Systems," University of Waterloo, Waterloo, Ontario, Canada, 2009.

[78] J. W. Kaltz, J. Ziegler and S. Lohmann, "Context-aware Web Engineering: Modeling and Applications," *Informatik,* 2005.

[79] N. Ryan, J. Pascoe and D. Morse, "Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant," in *Computer Applications and Quantitative Methods in Archaeology*, 1997.

[80] T. Swift and d. S. Warren, "XSB: Extending Prolog with Tabled Logic Programming," *Theory and Practice of Logic Programming,* vol. 12, no. Special Issue 1-2, pp. 157--187, 2012.

[81] L. Aceto, K. G. Larsen and A. Ingólfsdóttir, An Introduction to Milner's CCS, 2004.

[82] C. Z. A. Perera, P. Christen and D. Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," *Communications Surveys & Tutorials,* vol. 16, no. 1, pp. 414 - 454, 2014.