An Adaptive Context-Aware Publish–Subscribe Component Metamodel

by

Luis Blanco

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

There are plenty of solutions in which the functionality for context awareness is separated from the domain-specific functionality. Also, there are multiple ways of modeling, gathering, and processing context-related information; however there are only two main types of context dissemination, also known as context propagation: event based, or query based.

This means that most context-aware systems use events and require publish–subscribe mechanisms. However, a few include context as part of the event-propagation process; also, a few context models and metamodels consider events when modeling context-related data, even though such data are propagated though events.

This document proposes a metamodel that introduces the publish–subscribe logic of distributed event-based systems (DEBSs) to the modeling of context-related information and to the modeling of context-dependent interactions and adaptations. This approach has two main advantages: on one hand, no assumptions need to be made regarding the type of context or the domain of the application when considering interactions and adaptations that depend on context.

On the other hand, by introducing context-related elements when considering event propagation, the DEBS middleware can filter events based on context and stop their propagation. Also, including other features of DEBSs such as event enrichment and transformations, will make for a more efficient dissemination mechanism.

The metamodel suggested here could be used to model context, and context interaction between components. It can also be used when designing DEBS middleware that would take into account context when propagating events.

Acknowledgements

First and foremost I would like to thank Professor Paulo Alencar, for his guidance, support, and understanding during this very difficult time. It has been a true honor working under his tutelage and I am looking forward to continuing working with him in the future. I would also like to thank Eduardo Barrenechea and Rolando Blanco, for their help, feedback, and ideas. Also thanks to Professor Daniel Berry for serving as co-supervisor; and Professors Donald Cowan and Professor Ladan Tahvildari for agreeing to read this thesis and for their valuable feedback.

Special thanks to my beautiful Wife for her unconditional love, support, and patience; and to my family: my parents German and Consuelo, and my brothers Rolando and Mauricio, also for their support.

Finally, thanks to my best friend Mauricio Chavez for his words of wisdom and encouragement. I hope the tough times are soon left behind.

Dedication

Dedicated to my amazing wife Golnar.

Table of Contents

List of Figures

Chapter 1
Introduction

Dey [1] defines a context-aware system as a system that "uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". Dey also defines context as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [1]. According to Shilit et al. [2]: "such context-aware systems adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time". Finally, according to Brown [3]: "the adjective 'context-aware' is attached to applications that are mainly driven by the user's context. They tend to be mobile applications, because it is here that the user's context changes most rapidly and it is here that there is the greatest need for context-aware behavior". Brown [3] also defines context as: "location, identities of the people around the user, the time of day, season, temperature, etc."

In conclusion, context-awareness makes reference to the ability of monitoring context, and a context-aware system is capable of adapting its behavior depending on variables and situations that are dynamic.

A distributed event-based system (DEBS) is middleware that allows independent components to communicate via events. Components called *publishers* advertise events, while components called *subscribers*, express their interest in an event via subscriptions. Subscribers will only receive notifications of events to which they have subscribed. [4, 5]

This thesis focuses on modeling context and adaptation to that context, using the publish–subscribe logic from DEBSs as the mechanism to define and propagate context information and trigger adaptation to context.

The rest of this first chapter introduces terms and technologies closely related to context-awareness; then provides a more detailed overview of context-aware systems and DEBSs. After the introduction to related areas, the thesis delimits the problem it concentrates on, introduces the proposed approach to alleviate the problem and enumerates the contributions brought by the proposed solution; the chapter concludes with an overview of the structure of the rest of this thesis.

1.1 Related Areas

1.1.1 Adaptive, Mobile, and Ubiquitous Technologies

Adaptive computing is a very vast field that includes context-aware systems; in general *adaptive computing* makes reference to systems that can adapt by themselves as result of changes to their environments [6]. Examples of adaptive computing can be found everywhere and in a wide range of domains such as: the dynamic allocation of hardware resources and immediate system migration depending on the demands of software systems [7], chips that can change their configuration at run-time [8], and systems that assist patients in everyday tasks [9].

Another technology closely related to context-aware systems is that of *ubiquitous computing*, also known as *pervasive computing*. The term ubiquitous computing was first introduced in 1991 by Mark Weiser [10] and states: "ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user."

Thanks to the access that a multitude of devices now have to the internet, and particularly to wireless access, and also thanks to advances in electronics, computing is no longer limited to computers, but also to devices such as appliances, smartphones, devices implanted in humans and animals, wearable computers, etc. These devices are continuously connected and adapt constantly to their environment.

Some devices used in ubiquitous computing may be restricted to specific locations due to limited internet access or the need to be close to other devices, such as readers or other computers. In mobile computing, devices do not have this location restriction making them almost constantly available; however mobile devices are limited by what Sanaei et al. [11] call *resource poverty*, which makes reference to limitations in storage, processing capabilities, power supply, etc. Some of these limitations can be circumvented up to certain degree by concepts such as *Mobile Cloud Computing (MCC)* [11]. These type of technologies open the door for future extra functionality such as complex communication and data sharing between devices. However, they also introduce additional complexity in the design and implementation of applications.

Another interesting concept in the field of ubiquitous computing is that of *the Internet Of Things (IoT)* [12, 13] that encompasses the concept that not only devices that interact with end users connect to the internet, but devices embedded everywhere have the capability of communicating with each

other with little or no user interaction. The vision is to have environments where *anything identifies itself, anything communicates,* and *anything interacts, anywhere,* at *any time*; with the ultimate goal of having objects that "know what we like, what we want, and what we need and act accordingly without explicit instructions" [14].

1.1.2 Context-aware systems

Context-aware applications can be classified into three groups depending on the approach followed to represent and deal with context [15]: in an application with *no context model*, the context-related code is intertwined with the application code; an application with an *implicit context model* uses libraries and toolkits to support context awareness; and an application with an *explicit context model* has a well-defined context manager middleware. In applications that follow the *implicit context model* approach*,* even though the code related to context awareness is separated from the domain-specific code, the calls to that context-related code are inside the domain-specific code, which makes the two tied together, and still hard to separate. The third option is the preferred one for distributed and mobile applications, since it allows for more flexible changes to the context logic, and easier addition of components to the system.

In order to provide context awareness, in particular in *ubiquitous systems* where *context acquisition* is an important aspect, the following areas need to be supported [15, 16]: *context modeling, context acquisition and preprocessing, context dissemination,* and *context reasoning.* These areas are referred to as the *context life cycle*, and even though there are variations on what areas are included in the *context life cycle*, the ones mentioned here encompass sub-areas included by other definitions.

It is worth mentioning that *context reasoning* makes reference to the ability of deducing new and useful information from context-related data [17]. It also deals with issues such as context security and privacy, and inconsistency resolution.

Even though there are multiple ways of specifying context, see the "Related Work" chapter, there are basically just two ways of propagating context: using DEBSs or using queries.

1.1.3 DEBSs

As mentioned earlier on this chapter, a DEBS facilitates the interaction, via events, between independent components. In particular, a DEBS implements the publish–subscribe scheme.

Events are generated by *publishers* and received by *subscribers*. The existence of events is announced by publishers via *advertisements*; and subscribers express their interest on *events* via *subscriptions.* Both advertisements and subscriptions can be complemented with *filters*: events will only be published if the condition on the advertisement filter allows it; and a subscriber will be notified of the occurrence of an event if the condition on the subscription filter is satisfied.

The main characteristics of DEBSs include [5, 18]: *space*, *time,* and *synchronization decoupling*.

Space decoupling means that the publisher and the subscriber don't need to be in the same environment or location. Even more, they don't even need to be aware of the existence of each other.

Time decoupling means that the publisher and the subscriber don't have to interact at the same time. Even more, they don't even communicate directly: it is the event-based middleware that regulates the flow of event information from publishers to subscribers.

Synchronization decoupling means that the communication between publishers and subscribers is not direct. Publishers don't have to wait for any response from subscribers and vice versa.

As well, based on the way subscribers specify the events they are interested in, publish–subscribe systems can be [18, 19]: *topic, content,* and *type based.*

In *topic-based* systems events are organized by topic, and subscribers that expressed their interest in a topic will receive the events belonging to that topic. That interest is expressed via *subscriptions.*

In *content based* systems subscribers express their interest in events depending on the possible values of the attributes of the events. That interest is expressed via *content filters*.

In *type based* systems subscribers express their interest in an event depending on the type of the event as well as on possible values of the attributes of the event.

1.2 Problem

In current context-aware systems, there is a disconnect between the modeling of context-related data and the propagation of that data. On one hand, context-aware applications don't take advantage of the benefits of events when modeling context-aware data, more specifically when modelling interactions and adaptations that depend on context.

On the other hand, event-based middleware layers do not take advantage of filtering context data when propagating events; or features such as event enrichment and transformation. Barrenechea [20]

already considers the notion of having events and context together as part of the event-based middleware that propagates context. This idea is taken and developed into a metamodel that can be used to assist in the design of such middleware, or to introduce publish–subscribe concepts into the modeling of context-aware related data.

1.3 Proposed Approach

The metamodel is based on the principle that events and context elements are not necessarily foreign concepts to each other. An attribute of an event can be calculated from the values of context elements or from the values of attributes of other events. A context element can be calculated from event attributes or from other context elements. Finally, events can be filtered based on context elements or other events.

No assumptions are made as to what type of situations will be modeled, and though that assumption may make the models more complex, the metamodel could be used to model more types of interactions. This means there are no predefined context elements such as location, time, temperature, etc.

1.4 Thesis Outline

Chapter 2 presents work related to the two areas of context awareness that are the focus of this thesis, namely context modeling, and context dissemination. In the first area, the chapter enumerates the main types of available context models. In the second, the chapter concentrates on the mechanisms used to propagate context-related data, and emphasizes event-based middleware. Chapter 3 introduces the proposed metamodel: first outlining its requirements, and then introducing the elements that comprise it. Chapter 4 uses the metamodel to model the context interactions between components present in two case studies. Chapter 5 finishes this thesis by providing conclusions, outlining this work's contributions, and giving suggestions for future work.

Chapter 2
Related Work

As mentioned in Chapter 1, in order to provide context-awareness, four major areas need to be supported, namely *context modeling, context acquisition and preprocessing, context dissemination,* and *context reasoning*. Since this thesis contributes to *context modeling,* and *context dissemination,* this chapter concentrates in related work on those two areas.

2.1 Context Modeling

Context models make reference to tools used to represent context-related information. One way of classifying context models is based on the way data is represented, more specifically, on the type of data structure used to represent context-related data [13, 16, 21]. Using this approach models can be grouped as follows:

2.1.1 Key-Value Based

These type of models use combinations of basic key-value pairs to represent context data [2]. An example of these types of systems is Mobisaic, a system that uses variables and values to represent context; variables are tied to dynamic URLs for easy access from mobile phones [22]. The main advantages of these type of models is that they are easy to modify at the conceptual level: adding, removing elements, is straightforward; but modeling complex structures is difficult and cumbersome. This approach was one of the first used to model context, and it is now rarely used.

2.1.2 Markup Language Based

These type of models use markup languages are based on the Standard Generalized Markup Language standard, such as XML. Validation is supported through schema definitions. Two widely used models of these type are CC/PP and UAProf. CC/PP (Composite Capabilities/Preference Profiles) is a standard that is used for specifying capabilities and preferences that can be applied to *user agents*, also known as devices [23]. UAProf (User Agent Profile) is another standard based on the *user agent* concept, and it is oriented towards wireless devices [24].

The main advantages of these types of models is that they allow easy data retrieval and sharing. However they are limited when trying to model reasoning mechanisms [13].

2.1.3 Graphic Based

It uses standard tools to model context and relationships between the different context elements. Some of the standard tools that can be used are UML (Unified Model Language), ERM (Entity-Relationship Model), and ORM (Object Role Model).

Their main advantage is that they capture relationships together with the specification of context elements.

2.1.4 Object Oriented Based

It uses concepts of object-oriented programming such as inheritance, encapsulation, and re-usability. The main advantage of this approach is that models can easily be integrated to existing systems written in programming languages that support object-oriented concepts. However no ability to reason can be modeled [13].

One example of a system modeled using this modeling approach is GUIDE; a location based system used to provide context aware guide to tourists. The system interacts with the user via their local browser [25].

2.1.5 Ontology Based

Gomez-Perez et al. define ontologies as mechanisms that: "provide a common vocabulary of an area and define, with different levels of formality, the meaning of the terms and the relationships between them" [26].

Tools that support the definition and specification of ontologies, such as OWL (Web Ontology Language) [27], have been used to define context models.

One of the systems that use OWL is CoBrA (Context Broker Architecture), which provides a central context broker that manages individual and independent domain brokers. Each domain broker represents a part of the context-model [28, 29].

2.2 Context Dissemination

Most context-aware systems use one of two methods to propagate context: query based or event based [13]. In query-based propagation context dissemination is initiated by the interested party, called the *consumer*, which makes an explicit request in terms of a query.

In event-based propagation *consumers* express their interest by subscribing to events, and they will be notified when an event of that type occurs and satisfies any filters at both the advertisement and subscription levels. This is the area where DEBSs are used in context awareness.

2.3 Proposed Approach

As mentioned earlier, this thesis presents a metamodel that uses specific concepts from DEBSs. The metamodel will fall into the *Markup Language Based* model category: it uses XML and XML Schema Definitions. What makes this metamodel unique, and different to related work, is precisely the integration of the concepts from DEBSs at the metamodel level.

Introducing these *context dissemination* concepts as modeling tools provides a different approach to context-awareness modeling, with the following advantages:

1. It guarantees that context awareness modeled using the metamodel has the same advantages of DEBSs, namely *space*, *time,* and *synchronization decoupling*.
2. It provides a great deal of flexibility by allowing the mixture of events with context elements.
3. A unique characteristic, compared to other markup language based models, is that, by using concepts from DEBSs, the metamodel provides a mechanism to model not only context data but also the flow of context. That flow is modeled via advertisements, subscriptions, and filters.
4. By introducing the concept of features, the metamodel also provides a mechanism to specify adaptation to context.
5. When designing a context-aware DEBS middleware layer, the metamodel provides a guide as to what elements and functionality may be required.

Chapter 3
An Adaptive Context-Aware Publish–Subscribe Component Metamodel

This chapter presents a metamodel for adaptive, context-aware publish–subscribe components. It is fitting to use XML Schema Definitions (.xsd files) to specify the metamodel. Then, in Chapter 4, the .xsd files are used in XML documents to model specific case studies.

3.1 General Requirements

The metamodel has to provide the means to identify the elements that interact via context and events. Those elements are *components* and *environments.*

It also needs to provide elements to define events, and allow components to advertise or subscribe to such events. Events are defined via *events* and *event schemas*, and *event contracts* are the mechanisms used by components to advertise events or to subscribe to events.

In order to provide as much flexibility as possible, *calculated event attributes* are introduced. A *calculated event attribute* allows for the definition of an event attribute that depends on other events or context elements. A *calculated event attributes* also allows for the specification of enrichment, and transformation of existing events and context elements. Enriching means adding extra information to existing event attributes and context elements; transforming means the new event attribute is the result of operations applied to existing event attributes and context elements.

In conjunction with events the model must have elements to model context. *Context elements*, provide that capability. A context element can be calculated via events, and via other context elements.

In order to model adaptation to context and events, components expose *features* and those features are grouped in *feature profiles*. Environments select, depending on context or event filter conditions, the proper feature profiles using a *feature profiles selector*.

The final requirement to provide adaptation is to allow filtering at the event, context, and profiles selector level.

Even though a metamodel is being specified, i.e.: the elements in the metamodel represent classes, elements that actually represent instances of classes are also included. This is done so filters, and

context elements can be modeled. See the “Event Schemas and Events” section below for more details.

The rest of the chapter provides details on the elements just described, and introduces other auxiliary elements.

3.2 Components and Environments

The first element of the metamodel is the *component* element. A component is used to define the elements whose context is relevant and that interact via context and events with other components. It is also used to expose features so its behavior can adapt to events and context from other components.

The second building block are *environments*; an environment is a special type of component: it contains other components and provides logical regions that restrict the scope of context elements and events. This means a context element is only relevant inside the environment in which it is defined. Also, events are only propagated and visible to components and environments contained in the parent environment. This also means that the context of an element is relevant only to the environment that contains the context element. However, there should be no limitations on how many environments to which a component can belong to; also, environments can be nested inside other environments. Figure 1 shows the specification of components and environments.

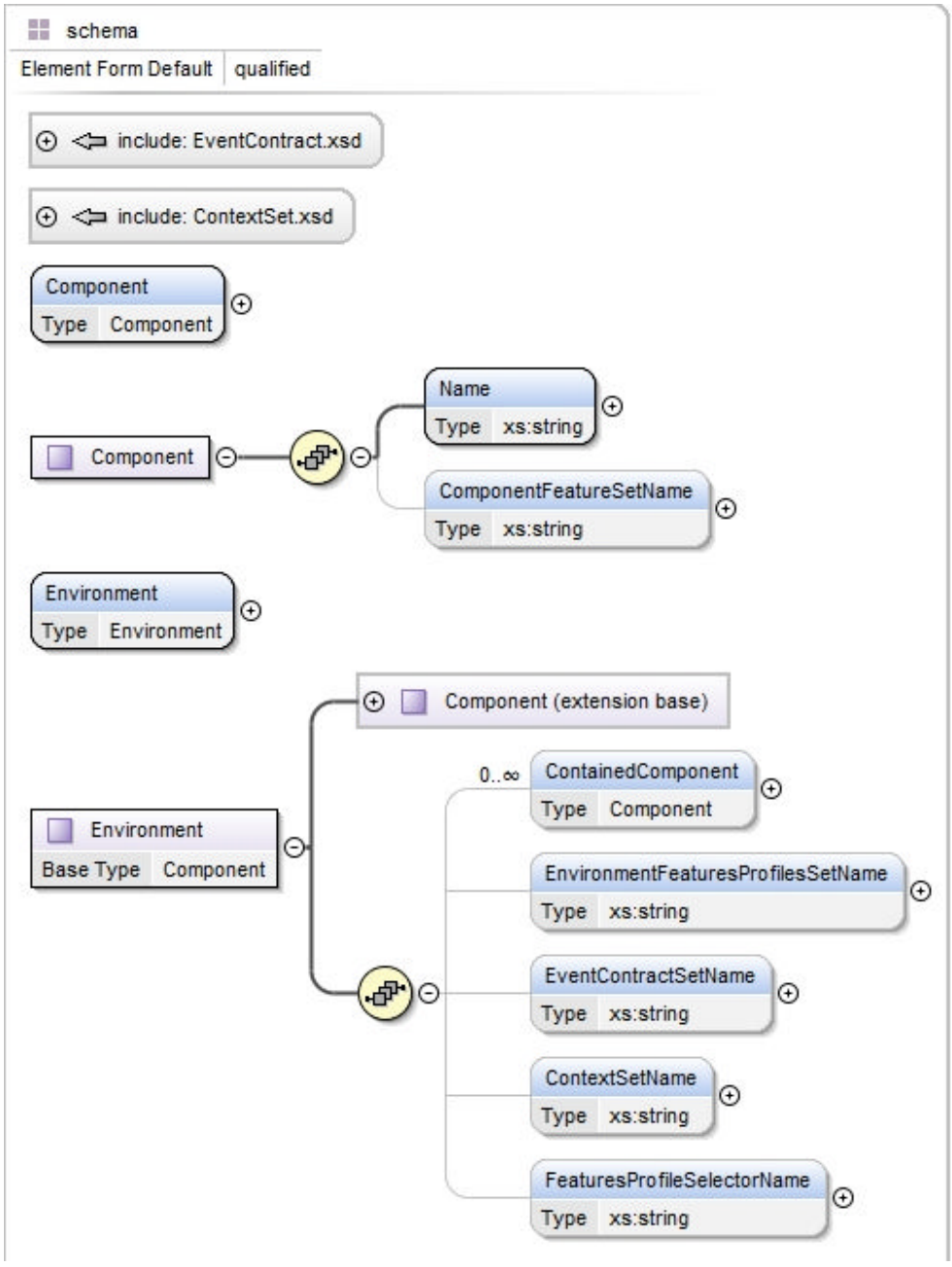

Figure 1 Components and Environments

3.3 Event Schemas and Events

All communication between components is done via events: either directly through advertisements and subscriptions, or indirectly through feature profiles and context elements. In the case of a context

element, its values are calculated from events and other context elements; and in the case of a features profile, the conditions to select which profile to activate will depend on events and context elements.

Events are specified using *event schemas*. An event schema consists of a *name* and one or more *event attribute schemas*. An event attribute schema represents the definition of an attribute of an event. In its simplest form (*simple event attribute schema*) an event attribute consists of a *name* and a *type*. However in order to model more complicated attributes, in addition to simple event attributes, the following event attribute schemas are also provided:

Nested event attribute schema: It allows the definition of nested attributes, that is, attributes that contain other attributes. For example, in the "Case Studies" chapter the *Location* attribute consists of two children attributes: *Latitude* and *Longitude*. In general, a nested attribute consists of a *parent event attribute schema* and one or more *child event attribute schemas.*

Calculated event attribute schema: This type of schema allows for the definition of attributes whose values require calculations. This calculations may involve the values of other event attributes and context elements. Having calculated event attributes allows for the simulation of event operations such as event enrichment, where the calculated attribute adds extra information to an existing event attribute or context element; and event transformation and interpretation, where the calculated attribute is the result of operations to existing event attributes and context elements.

Originally, in order to model calculations, the metamodel contained a set of classes representing arithmetical operations, however these classes cluttered the model, so those objects were replaced with xquery expressions. This de-clutters the metamodel, and it doesn't reduce its effectiveness. In summary, calculated attributes consist of a *name,* a *type*, and a *calculation expression*.

Subsequent event attribute schema: Sometimes it may be required to remember previous values of an event attribute; for instance in the second case study in Chapter 4, it is needed to tell if a cellphone is moving away from a specific point, so subsequent event attributes store the previous values of the location of the cellphone and if the distance to the specific point is increasing it is determined that the cellphone is moving away.

Subsequent event attributes consist of a *name,* a *type*, and information about the attribute whose values need to be remembered: *source event name, source event attribute schema,* and *occurrence index*.

Figure 2 shows the specification of event schemas and event attribute schemas.

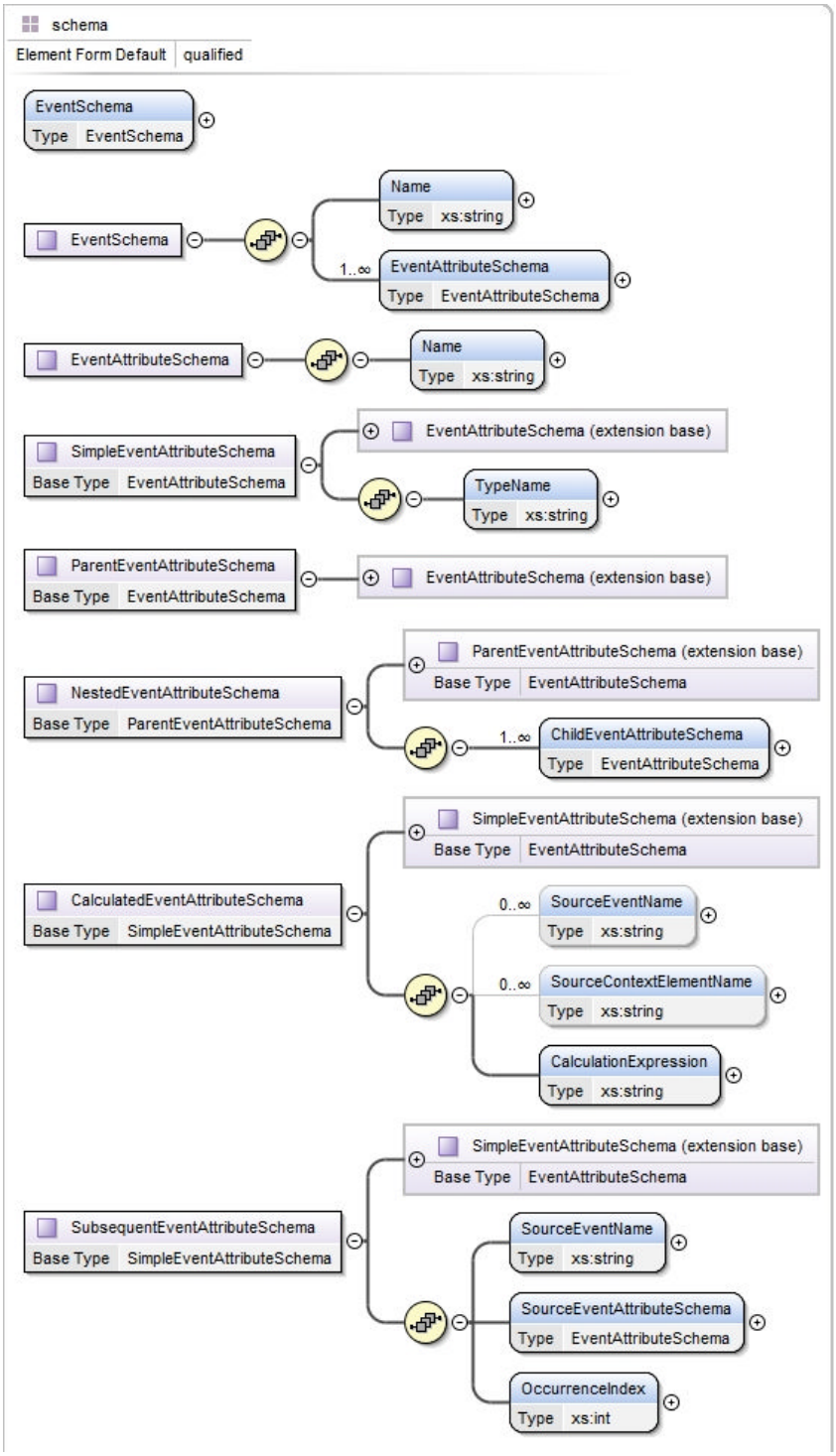

Figure 2 Event Schemas

Conceptually, *events* are instances of event schemas, however, and in order to facilitate their use to specify filters and context elements, *events* are defined as a *name,* and a collection of *event attributes*. An event attribute, with the exception of *nested attributes,* contains a reference to its schema and a value. A *nested attribute* contains a reference to its *parent attribute* and a collection of references to its children attributes. Figure 3 shows the specification of events and event attributes.

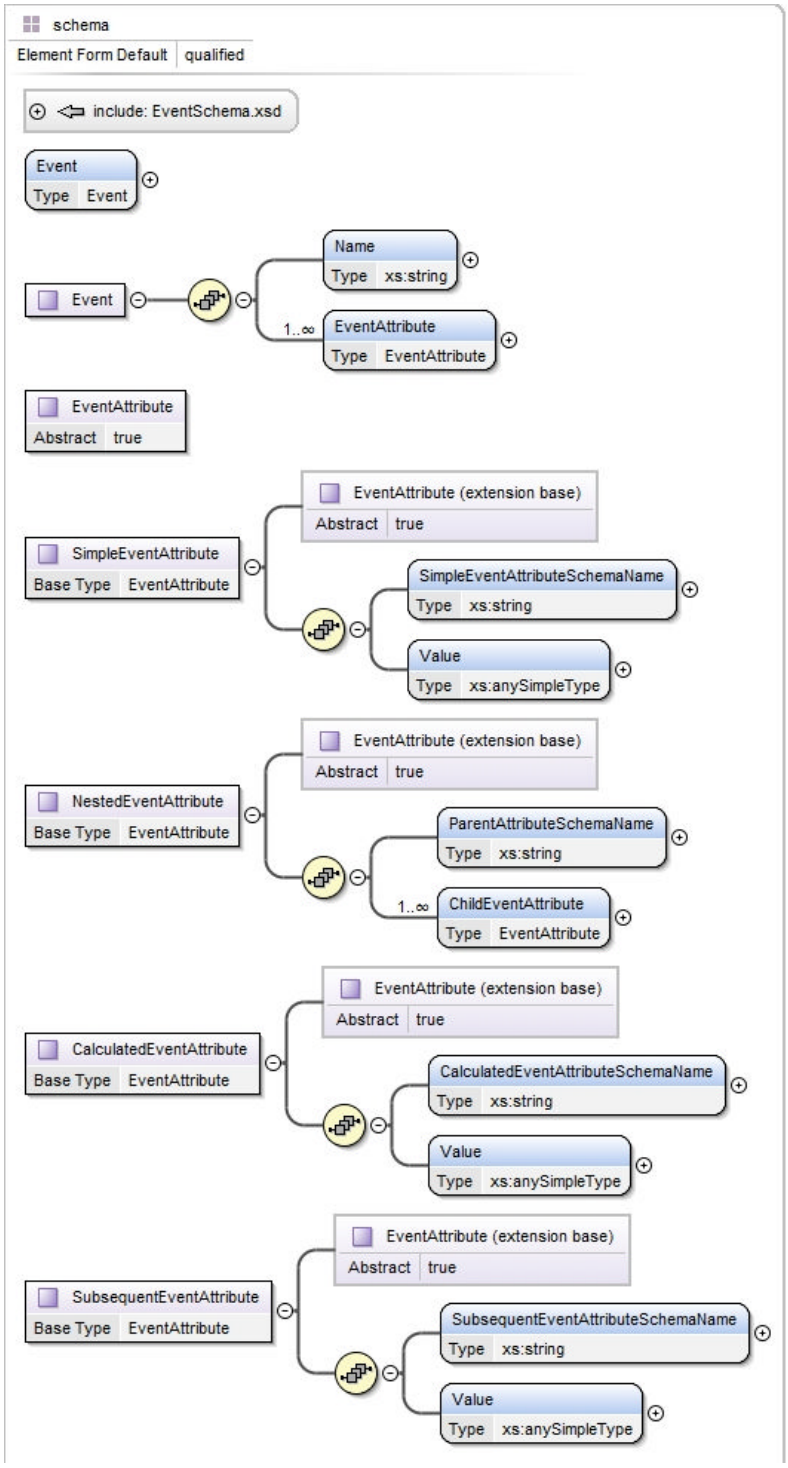

Figure 3 Events

Components expose event schemas using *advertisements* and components express their interest in an event schema with *subscriptions*. Advertisements and subscriptions are types of *event contracts*. In the case of an advertisement the contract specifies the event's name, the component advertising the event, the event's schema, and, if needed, event filters, so the event will be published only in certain cases. For example, don't propagate an event if today is not a working day.

In the case of a subscription, the contract specifies the component that wants to subscribe to an event and the actual event; filters can also be added inside subscriptions, so the component that subscribes to an event will only be notified of the event in certain cases and not every time the event is published.

Advertisements and subscriptions are at the class level, however to represent actual instances of the event *publications* and *notifications* classes are defined. These classes are included so models created with the metamodel can be evaluated. Figure 4 shows the specification of event contracts.

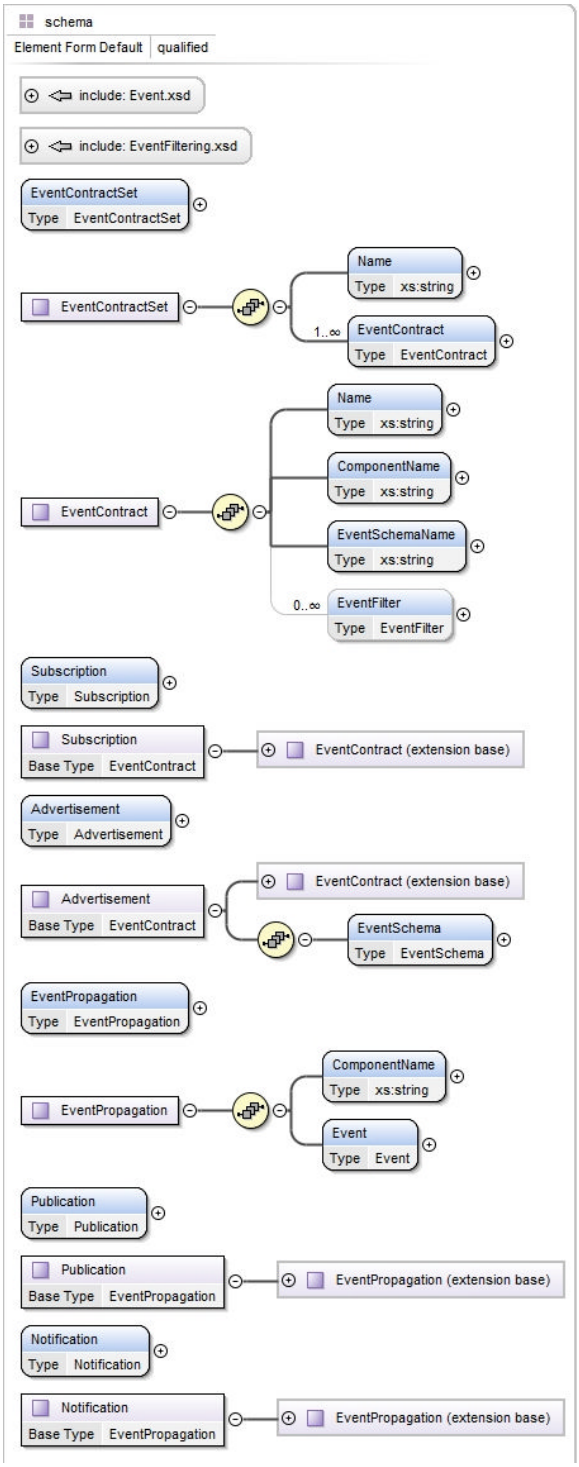

Figure 4 Event Contracts

As mentioned, advertisements and subscriptions can specify *event filters*. Event filters were designed to make them as simple to use and as powerful as possible: filtering expressions can make reference to attributes of the event being filtered, to attributes of other events, or to context elements. This way both content and context filtering are offered.

An event filter consists of a *name*, a collection of references to the events (*source events*) and to the context elements (*source context elements*) used inside the xquery expression that filters the event (*filtering expression).* Figure 5 shows the structure of event filters.

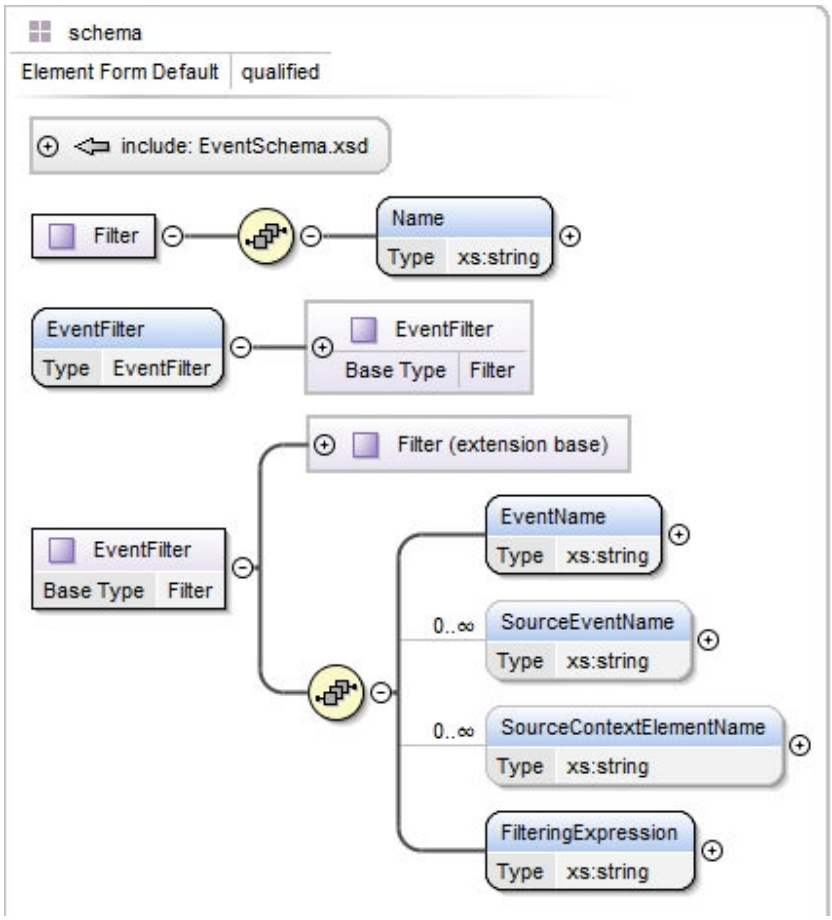

Figure 5 Event Filters

3.4 Context and Context Elements

As shown in Chapter 1, there is not a unique definition of context, and the meaning of context may vary depending on the application or system to be designed. Because of this, when defining elements

to model context, the objective is to be generic, and make as few assumptions as possible. Following this principle, a *context element* is specified as a value that can be calculated via events and other context elements. With this simple concept complex conditions and situations can be modeled. For instance, a context element could to tell if the cellphone of a user is home based on an event published by the phone with its location.

More specifically, a *context element* contains a *name,* a *value type*, a collection of references to the events (*source events*) and to the context elements (*source context elements*) used inside the xquery expression that calculates the value of the context element.

Context elements are then grouped into a *context set*, which can be then referenced by an environment. Figure 6 shows the structure of context sets and elements. Notice that the figure also contains definitions for *context element values* and *context value sets*, these represent instances of objects of type *context element* and *context set*. These classes are included so models created from the metamodel can be evaluated.

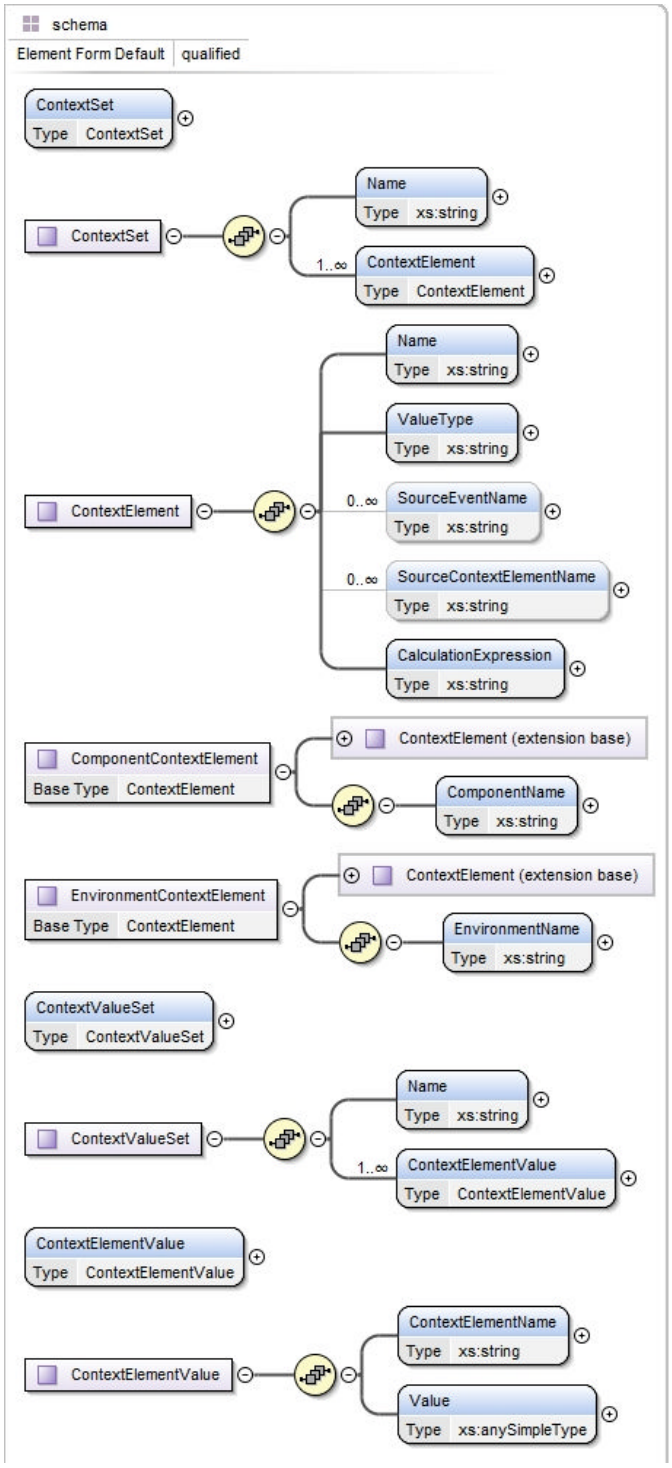

Figure 6 Context

3.5 Features, Features Profiles, and Feature Selectors

Behavior adaptation can be modeled two ways: by selecting features depending on context or event filter conditions; or by a component subscribing to events. The first approach is followed in the first case study in the "Case Studies" chapter, while the second case study follows the second approach.

Components expose *features,* which represent either actions, or settings. For instance a cellphone may expose email or phone call features. Features can be of different types:

A *simple feature* consists of a *name* and a *type*. *A range feature* is numeric and consists of a *name* and a *minimum* and a *maximum value.* An *enumerated feature* consists of a *name* and a list of *possible values*. A *single value enumerated feature* is an enumerated feature that can have only one value, while a *multiple value enumerated feature* can have multiple values. Finally, a *compound feature* consists of a collection of features of any type; for instance a phone call feature may contain volume, ring tone, and vibration mode features.

The features of a component are grouped in a *component feature set.* Figure 7 shows the specification of features and component feature sets.

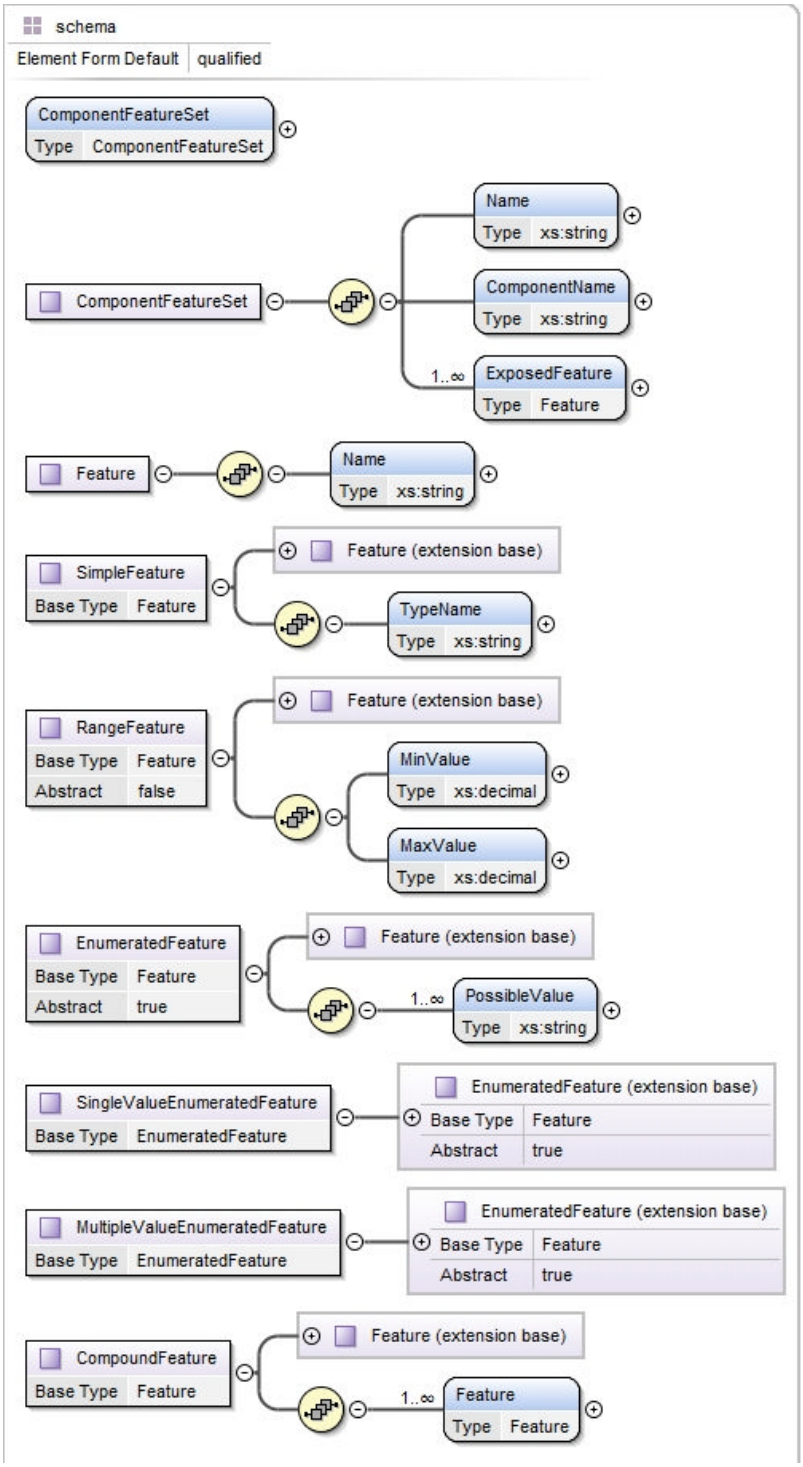

Figure 7 Features and Features Sets

The next step is to define *feature profiles.* A feature profile is a collection of *feature instances.* A *feature instance* is a specific value for a specific *feature*.

For instance if a cellphone exposes a phone call compound feature, that consists of volume, ring tone, and vibration mode features; a *user busy* features profile could set the volume to zero, the ring tone to none and vibration mode to on.

Features profiles are then grouped by component in a *component features profile set*; and those component sets into a single *environment features profile set*.

Figure 8 shows the specification of features profiles and sets.

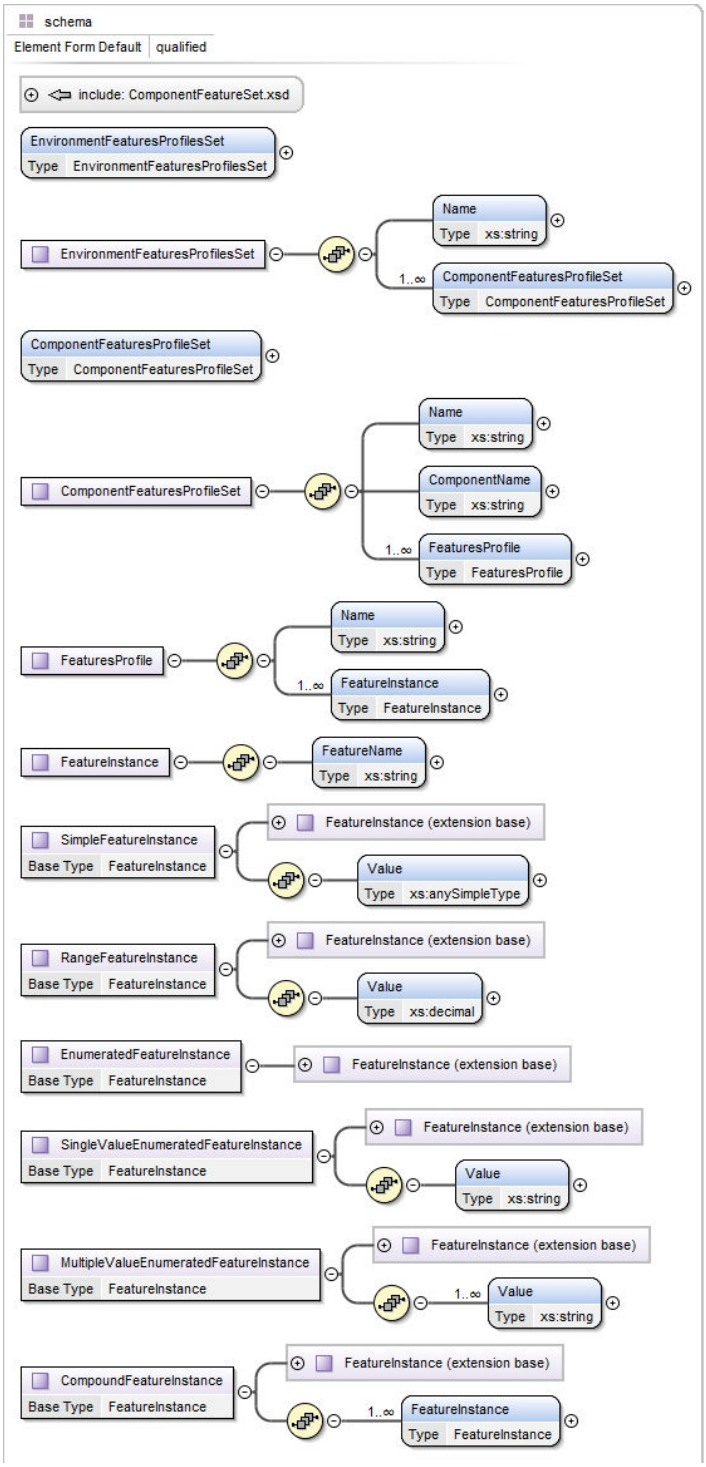

Figure 8 Features Profiles

The final step in the specification of features consists on defining a mechanism that can be used, by environments, to select the proper feature profile depending on given conditions. Those conditions will be calculated using the values of events and context elements.

That mechanism is called a *profile selectors*. A profile selector consists of a *name* and a collection of *feature profile conditions*. Feature profile conditions have a *condition name* and a reference to the *features profile* to be selected if the condition is true. The conditions can be of three types:

A *feature profile if condition* contains, in addition to the attributes of the base type, a *conditional expression* and a collection of the events and context elements needed to calculate if the condition is true or not.

A *feature profile not condition* contains a reference to a *feature profile if condition*. So if the if condition is false, then the features profile specified inside this not-condition will be selected.

A *feature profile default condition:* the features profile specified inside this condition will be selected, if no other condition is true.

Figure 9 shows the structure of features profiles selectors and features profile conditions.

The introduction of features concludes the design of the metamodel. The next chapter uses the metamodel to model two case studies.

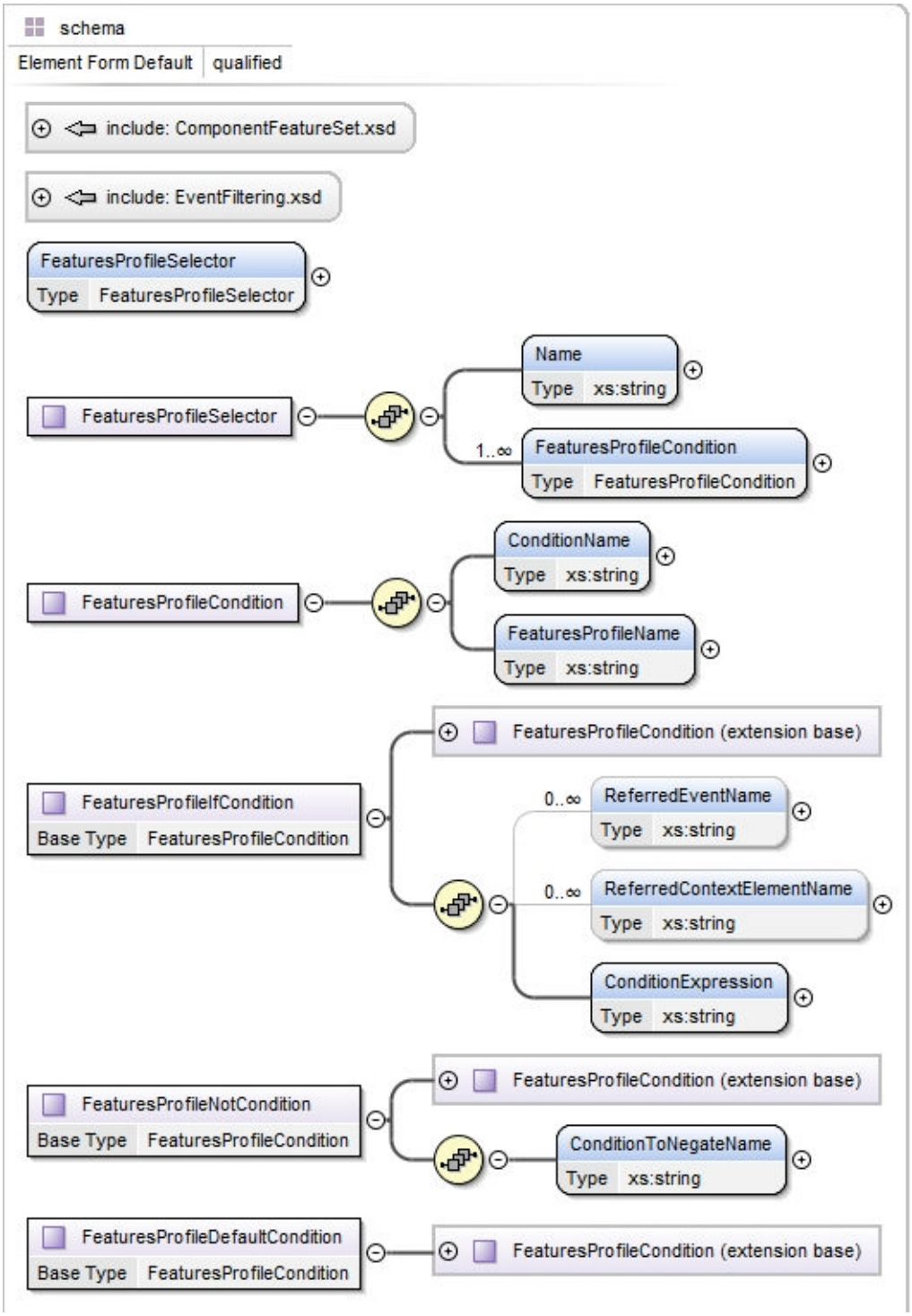

Figure 9 Feature Profile Selectors

Chapter 4
Case Studies

This chapter applies the metamodel to two separate case studies. Each case represents a situation in which context is a primary factor.

As mentioned in the previous chapter, xquery expressions are used to model mathematical expressions; however, since some of the expressions are quite complex and it is outside the scope of this document to show the power of xquery, in such cases, only the purpose of the xquery function instead is noted of having also the actual xquery expression.

4.1 Cellphone Behavior Adaptation Case Study

4.1.1 Description:

In this case study, the cellphone's behavior adapts to the context of the same cellphone. Here the concept of profiles is explored: what features to activate depends on context. To demonstrate the power of the metamodel when implemented at the operating system level, it is assumed that the device has the capability of adapting to context via events without the need of running a local application.

The scenario to be considered is as follows: the user is at work, and his or her calendar indicates a meeting has just started. The cellphone's profile is changed to *in business meeting*, and the behavior of the phone when receiving phone calls changes according to the user's preferences.

4.1.2 Participants:

User's Cellphone

Calendar application

4.1.3 Interaction via Events:

- The Calendar application detects a meeting has just started:
    - A *MeetingStartedEvent* event is generated by the Calendar application, and the user's Cellphone receives the event. This event will be used to calculate the value of the context element *InBusinessMeetingContextElement.* This context element tells if a business meeting is in progress and the user is at work.

- The location of the Cellphone has changed:
    - A *LocationEvent* is generated by the phone. This event will also be used to calculate the value of the context element *InBusinessMeetingContextElement*
- A phone call comes in:
    - A *IncomingCallEvent* is generated
    - The Cellphone:
        - Receives the *IncomingCallEvent*.
        - If there is a meeting going on and the user is at work, i.e., the *InBusinessMeetingContextElement* context element has a value of true:
            - If the call is from somebody not in the *preferred callers list*:
                - To deal with the call the actions specified by the proper features profile (*BusinessMeetingProfile*) are followed. The options are: ignore the call, go directly to voice mail, or put the call through with the proper settings such as ring tone, vibration mode, and volume.
            - If the call is from somebody in the *preferred callers list*:
                - Follow the actions specified by the Exception features profile (*BusinessMeetingPreferredCallerProfile*).
        - If no meeting is going on or the user is not at work:
            - Handle the call as usual (*NotInMeetingProfile*)

4.1.4 Modeling:

4.1.4.1 Components:

In this case study there is a single environment: the user's Cellphone. The environment contains an additional component besides itself: the Calendar application. Below is the specification for the Cellphone environment:

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Component.xsd">
   <Name>UserCellphone</Name>
   <ComponentFeatureSetName>
```

```
      UserCellphoneComponentFeatureSet
   </ComponentFeatureSetName>
   <ContainedComponent>
      <Name>CalendarApplication</Name>
   </ContainedComponent>
   <EnvironmentFeaturesProfilesSetName>
      UserCellphoneEnvironmentFeaturesProfileSet
   </EnvironmentFeaturesProfilesSetName>
   <EventContractSetName>UserCellphoneEventContractSet</EventContractSetName>
   <ContextSetName>UserCellphoneContextSet</ContextSetName>
   <FeaturesProfileSelectorName>
      UserCellphoneEnvironmentFeaturesProfileSelector
   </FeaturesProfileSelectorName>
</Environment>
```

4.1.4.2 Events:

The Calendar application doesn't subscribe to any events and has a single advertisement. The advertisement is for the *MeetingStartedEvent* event. The Cellphone advertises two events: one for the location of the cellphone, *LocationEvent*, and one when an incoming call is detected, *IncomingCallEvent*. Since all events are processed at the environment level no subscriptions are specified for events of interest to the environment. The Cellphone itself subscribes to the *IncomingCallEvent*. This subscription doesn't require any filters either.

The *MeetingStartedEvent* event is used to calculate the *MeetingInProgressContextElement* context element; while the *LocationEvent* event is used to calculate the *DistanceToWorkContextElement* context element. Both, *MeetingInProgressContextElement* and *DistanceToWorkContextElement*, context elements are then used to calculate a third context element: *InBusinessMeetingContextElement* which tells if a business meeting is in progress and the user is at work.

The advertisements, including the corresponding event schemas, and subscriptions for this case study are specified as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<EventContractSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="EventContract.xsd">
```

```
<Name> UserCellphoneEventContractSet </Name>
<EventContract xsi:type = "Advertisement">
   <Name>MeetingStartedEventAdvertisement</Name>
   <ComponentName>CalendarApplication</ComponentName>
   <EventSchemaName>MeetingStartedEvent</EventSchemaName>
   <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>MeetingStartedEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
         <Name>MeetingName</Name>
         <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
         <Name>Date</Name>
         <TypeName>xs:date</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
         <Name>StartTime</Name>
         <TypeName>xs:time</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
         <Name>Duration</Name>
         <TypeName>xs:decimal</TypeName>
      </EventAttributeSchema>
   </EventSchema>
</EventContract>
<EventContract xsi:type = "Advertisement">
   <Name>LocationEventAdvertisement</Name>
   <ComponentName>UserCellphone</ComponentName>
   <EventSchemaName>LocationEvent</EventSchemaName>
   <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>LocationEvent</Name>
      <EventAttributeSchema xsi:type = "NestedEventAttributeSchema">
         <Name>Location</Name>
```

```
            <ChildEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
               <Name>Latitude</Name>
               <TypeName>xs:double</TypeName>
            </ChildEventAttributeSchema>
            <ChildEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
               <Name>Longitude</Name>
               <TypeName>xs:double</TypeName>
            </ChildEventAttributeSchema>
         </EventAttributeSchema>
      </EventSchema>
   </EventContract>
   <EventContract xsi:type = "Advertisement">
      <Name>IncomingCallEventAdvertisement</Name>
      <ComponentName>UserCellphone</ComponentName>
      <EventSchemaName>IncomingCallEvent</EventSchemaName>
      <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="EventSchema.xsd">
         <Name>IncomingCallEvent</Name>
         <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
            <Name>CallerName</Name>
            <TypeName>xs:string</TypeName>
         </EventAttributeSchema>
         <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
            <Name>CallingNumber</Name>
            <TypeName>xs:string</TypeName>
         </EventAttributeSchema>
      </EventSchema>
   </EventContract>
   <EventContract xsi:type = "Subscription">
      <Name>IncomingCallEventCellphoneSub</Name>
      <ComponentName>UserCellphone</ComponentName>
      <EventSchemaName>IncomingCallEvent</EventSchemaName>
   </EventContract>
</EventContractSet>
```

4.1.4.3 Context:

In this case study there are three context elements: the cellphone's distance to work *DistanceToWorkContextElement*; if a meeting is in progress, *MeetingInProgressContextElement*; and an element that combines the two and tells if the cellphone is at work and a meeting is in progress, *InBusinessMeetingContextElement*.

DistanceToWorkContextElement is calculated using the *LocationEvent* event, *MeetingInProgressContextElement* is calculated using the *MeetingStartedEvent*, and *InBusinessMeetingContextElement* is calculated using the other two context elements. Below is the specification of all context elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContextSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ContextSet.xsd">
   <Name>UserCellphoneContextSet</Name>
   <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>DistanceToWorkContextElement</Name>
      <ValueType>xs:double</ValueType>
      <SourceEventName>LocationEvent</SourceEventName>
      <!-- Formula to calculate distance from two latitude/longitude points: -->
      <CalculationExpression>
         Cellphone_DistanceToWorkContextElementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>UserCellphone</EnvironmentName>
   </ContextElement>
   <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>MeetingInProgressContextElement</Name>
      <ValueType>xs:boolean</ValueType>
      <SourceEventName>MeetingStartedEvent</SourceEventName>
      <!-- Formula to check if a meeting is still in progress: -->
      <CalculationExpression>
         Cellphone_MeetingInProgressContextElementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>UserCellphone</EnvironmentName>
```

```
    </ContextElement>
    <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>InBusinessMeetingContextElement</Name>
      <ValueType>xs:boolean</ValueType>
      <SourceContextElementName> DistanceToWorkContextElement
</SourceContextElementName>

<SourceContextElementName>MeetingInProgressContextElement</SourceContextElementName>
      <!-- Formula to check if the value of DistanceToWorkContextElement is less than 1
        and MeetingInProgressContextElement is true: -->
      <CalculationExpression>
        Cellphone_InBusinessMeetingContextElementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>UserCellphone</EnvironmentName>
    </ContextElement>
</ContextSet>
```

And the expressions to calculate the context elements are:

- Cellphone_DistanceToWorkContextElementCalculationExpression.xquery:

```
(: Calculate distance to work using CellphoneLocationEvent :)
declare function local:distance-to-work-in-kms
  ( $latitude as xs:double, $longitude as xs:double, $reflatitude as xs:double, $reflongitude as
xs:double)  as xs:double? {
  1
 } ;

(: From the values of the Location Event, return the distance to work :)
let $doc := fn:doc("CellphoneLocationEvent.xml") (: Document with the values of the current instance
of the event :)
let $event := $doc/Event
let $event_attribute := $event/EventAttribute[@xsi:type = "NestedEventAttribute" and
                                ParentAttributeSchemaName = "Location"]

let $lat_child := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute" and
./SimpleEventAttributeSchemaName = "Latitude"]
```

```
let $lat := $lat_child/Value

let $lon_child := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute" and
./SimpleEventAttributeSchemaName = "Longitude"]
let $lon := $lat_child/Value

return local:distance-to-work-in-kms($lat, $lon, 43.47, -80.54)
```

- Cellphone_MeetingInProgressContextElementCalculationExpression.xquery:

```
(:Get the event if it is happening on a weekday, between 9am and 5pm:)

(:day-of-week function adapted from: http://www.functx.com/ :)
declare function local:day-of-week
  ( $date as xs:anyAtomicType? )  as xs:integer? {
  if (empty($date))
  then ()
  else xs:integer((xs:date($date) – xs:date('1901-01-06'))
        div xs:dayTimeDuration('P1D')) mod 7
 } ;

(: Working day and time :)
(: If the date is a weekday and the time is between 9am and 5 pm, return true :)
declare function local:working-time
  ( $date as xs:anyAtomicType?, $time as xs:anyAtomicType?)  as xs:34oolean {
  let $dow := local:day-of-week($date)
  return
     if (($dow!= 0) and ( $dow != 7)) then
        if ((xs:time($time) >= xs:time("09:00:00"))
           and
           (xs:time($time) <= xs:time("17:00:00"))) then
           true()
        else false()
     else false()
};
```

```
(: adapted from: http://www.functx.com/ :)
declare function local:getDayTimeDuration
  ( $days as xs:decimal? ,
    $hours as xs:decimal? ,
    $minutes as xs:decimal? ,
    $seconds as xs:decimal? )  as xs:dayTimeDuration {

    (xs:dayTimeDuration('P1D') * $days) +
    (xs:dayTimeDuration('PT1H') * $hours) +
    (xs:dayTimeDuration('PT1M') * $minutes) +
    (xs:dayTimeDuration('PT1S') * $seconds)
 } ;

(: If the current time falls in between a meeting, return true :)
let $doc := fn:doc("CellphoneMeetingStartedEvent.xml") (: Document with the values of the current
instance of the event :)
let $event := $doc/Event
let $event_attribute_start_time := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                   SimpleEventAttributeSchemaName = "StartTime"]
let $event_attribute_start_date := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                   SimpleEventAttributeSchemaName = "Date"]
let $event_attribute_duration := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                   SimpleEventAttributeSchemaName = "Duration"]

let $start_datetime := fn:dateTime(xs:date($event_attribute_start_date),
xs:time($event_attribute_start_time))
let $end_datetime := $start_datetime + local:getDayTimeDuration(0,
                                        0,
                                        xs:decimal($event_attribute_duration),
                                        0)
let $curr_time := fn:current-dateTime()
return (: working day and time :)
      local:working-time($event_attribute_start_date, $event_attribute_start_time) and
      (: current time is bigger than $start_datetime and smaller than $end_datetime :)
```

```
    $curr_time >= $start_datetime and
    $curr_time <= $end_datetime
```

- Cellphone_InBusinessMeetingContextElementCalculationExpression.xquery:

```
(:Tell if there is a business meeting going on and the user is at work:)

let $doc_dist := fn:doc("CellphoneDistanceToWorkContexElementValue.xml") (: Document with the
current value of the Distance To Work context :)
let $doc_in_bus := fn:doc("CellphoneMeetingInProgressContextElementValue.xml")  (: Document with
the current value of the Meeting In Progress context :)
let $dist := $doc_dist/Value
let $in_bus := $doc_in_bus/Value
return $in_bus = true() and ($dist < 1)
```

4.1.4.4 Features:

The Cellphone exposes the following features:

```
<?xml version="1.0" encoding="UTF-8"?>
<ComponentFeatureSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ComponentFeatureSet.xsd">
   <Name>UserCellphoneComponentFeatureSet</Name>
   <ComponentName>UserCellphone</ComponentName>
   <ExposedFeature xsi:type= "CompoundFeature">
      <Name>PhoneCall</Name>
      <Feature xsi:type= "RangeFeature">
         <Name>Volume</Name>
         <MinValue>0</MinValue>
         <MaxValue>10</MaxValue>
      </Feature>
      <Feature xsi:type= "SingleValueEnumeratedFeature">
         <Name>VibrationMode</Name>
         <PossibleValue>On</PossibleValue>
         <PossibleValue>Off</PossibleValue>
      </Feature>
      <Feature xsi:type= "SingleValueEnumeratedFeature">
         <Name>Ringtone</Name>
```

```
        <PossibleValue>Ringtone1</PossibleValue>
        <PossibleValue>Ringtone2</PossibleValue>
      </Feature>
      <Feature xsi:type= "SingleValueEnumeratedFeature">
        <Name>Action</Name>
        <PossibleValue>Ignore</PossibleValue>
        <PossibleValue>VoiceMail</PossibleValue>
        <PossibleValue>ReceiveCall</PossibleValue>
      </Feature>
    </ExposedFeature>
</ComponentFeatureSet>
```

Features are then grouped into profiles so a single profile can consist of multiple features, for instance let the phone ring at a certain volume and vibrate at the same time. For this example the set of available profiles are:

```
<?xml version="1.0" encoding="UTF-8"?>
<EnvironmentFeaturesProfilesSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="ComponentFeaturesProfileSet.xsd">
    <Name>UserCellphoneEnvironmentFeaturesProfileSet</Name>
    <ComponentFeaturesProfileSet>
      <Name>UserCellphoneFeaturesProfileSet</Name>
      <ComponentName>UserCellphone</ComponentName>
      <FeaturesProfile>
        <Name>BusinessMeetingProfile</Name>
        <FeatureInstance xsi:type = "CompoundFeatureInstance">
          <FeatureName>PhoneCall</FeatureName>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>Action</FeatureName>
            <Value>VoiceMail</Value>
          </FeatureInstance>
        </FeatureInstance>
      </FeaturesProfile>
      <FeaturesProfile>
        <Name>BusinessMeetingPreferredCallerProfile</Name>
        <FeatureInstance xsi:type = "CompoundFeatureInstance">
```

```
      <FeatureName>PhoneCall</FeatureName>
      <FeatureInstance xsi:type= "RangeFeatureInstance">
        <FeatureName>Volume</FeatureName>
        <Value>3</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
        <FeatureName>VibrationMode</FeatureName>
        <Value>On</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
        <FeatureName>Ringtone</FeatureName>
        <Value>Ringtone2</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
        <FeatureName>Action</FeatureName>
        <Value>ReceiveCall</Value>
      </FeatureInstance>
    </FeatureInstance>
  </FeaturesProfile>
  <FeaturesProfile>
    <Name>NotInMeetingProfile</Name>
    <FeatureInstance xsi:type = "CompoundFeatureInstance">
      <FeatureName>PhoneCall</FeatureName>
      <FeatureInstance xsi:type= "RangeFeatureInstance">
        <FeatureName>Volume</FeatureName>
        <Value>8</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
        <FeatureName>VibrationMode</FeatureName>
        <Value>On</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
        <FeatureName>Ringtone</FeatureName>
        <Value>Ringtone1</Value>
      </FeatureInstance>
```

```
            <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
              <FeatureName>Action</FeatureName>
              <Value>ReceiveCall</Value>
            </FeatureInstance>
          </FeatureInstance>
        </FeaturesProfile>
    </ComponentFeaturesProfileSet>
</EnvironmentFeaturesProfilesSet>
```

What to do after the call is received is decided based on the specified *feature profile selector*. The selector stores a set of profiles and the conditions that must be true for each profile to be selected. In this case, as specified above, there are three profiles and their corresponding conditions: one profile, *BusinessMeetingPreferredCallerProfile*, is used if *InBusinessMeetingContextElement* is true, and the caller is in the *preferred callers list*; another profile, *BusinessMeetingProfile*, to be selected if *InBusinessMeetingContextElement* is also true but the caller is not in the *preferred callers list*; and one, *NotInMeetingProfile*, if no meeting is taking place or we are outside business hours.

As mentioned in the section where the metamodel is introduced, the conditions to select the proper profile can be of three types: *if condition, not condition*, and *default condition*. For this case study there will be two *if conditions* and one *default condition*:

The first condition depends on *InBusinessMeetingContextElement* and the *IncomingCallEvent* event and will check if the user is in a business meeting and the caller is in the *preferred callers list*:

```
<FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
   <ConditionName>BusinessMeetingPreferredCallerProfileCondition</ConditionName>
   <FeaturesProfileName>BusinessMeetingPreferredCallerProfile</FeaturesProfileName>
   <ReferredEventName>IncomingCallEvent</ReferredEventName>
   <ReferredContextElementName>
      InBusinessMeetingContextElement
  </ReferredContextElementName>
  <ConditionExpression>
      CellphoneIncomingCallEvent_CallerInListFilteringExpression.xquery
   </ConditionExpression>
</FeaturesProfileCondition>
```

The condition (xquery expression) that filters the event is CellphoneIncomingCallEvent_CallerInListFilteringExpression.xquery*:*

```
(:Tell if there is a business meeting going on and the user is at work and
the caller is in a predefined list::)

let $doc := fn:doc("CellphoneIncomingCallEvent.xml") (: Document with the values of the current
instance of the event :)
let $doc_in_bus := fn:doc("CellphoneInBusinessMeetingContextElementValue.xml")  (: Document
with the values of the current instance of the In Business Meeting context :)
let $in_bus := $doc_in_bus/Value
let $event := $doc/Event
let $event_attribute := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                    SimpleEventAttributeSchemaName = "CallerName"]
let $list_names := ('Michael Smith', 'Other Caller 1', 'Other Caller 2')
   return
   if ($list_names = xs:string($event_attribute/Value)) then
      if ($in_bus = true()) then
        true()
      else false()
    else
      false()
```

The second *if condition* also depends on *InBusinessMeetingContextElement* and the *IncomingCallEvent* event and will check if the user is in a business meeting and the caller is NOT in the *preferred callers list*:

```
<FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
   <ConditionName>BusinessMeetingNotPreferredCallerProfileCondition</ConditionName>
   <FeaturesProfileName>BusinessMeetingProfile</FeaturesProfileName>
   <ReferredEventName>IncomingCallEvent</ReferredEventName>
   <ReferredContextElementName>
      InBusinessMeetingContextElement
  </ReferredContextElementName>
  <ConditionExpression>
      CellphoneIncomingCallEvent_CallerNotInListFilteringExpression.xquery
```

```
    </ConditionExpression>
</FeaturesProfileCondition>
```

The condition (xquery expression) that filters the event is CellphoneIncomingCallEvent_CallerNotInListFilteringExpression.xquery:

```
(:Tell if there is a business meeting going on and the user is at work and
the caller is NOT in a predefined list::)

let $doc := fn:doc("CellphoneIncomingCallEvent.xml") (: Document with the values of the current
instance of the event :)
let $doc_in_bus := fn:doc("CellphoneInBusinessMeetingContextElementValue.xml")  (: Document
with the values of the current instance of the In Business Meeting context :)
let $in_bus := $doc_in_bus/Value
let $event := $doc/Event
let $event_attribute := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                    SimpleEventAttributeSchemaName = "CallerName"]
let $list_names := ('Michael Smith', 'Other Caller 1', 'Other Caller 2')
    return
    if ($list_names = xs:string($event_attribute/Value)) then
        if ($in_bus = false()) then
          true()
        else
          false()
      else
        false()
```

Finally, the default condition: no meeting is going on, or we are outside business hours, so just process the call as usual, is:

```
<FeaturesProfileCondition xsi:type = "FeaturesProfileDefaultCondition">
   <ConditionName>ElseCondition</ConditionName>
    <FeaturesProfileName>NotInMeetingProfile</FeaturesProfileName>
</FeaturesProfileCondition>
```

The full specification of the features profile selector is:

```
<FeaturesProfileSelector xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="ComponentFeaturesProfileSelector.xsd">
```

```
    <Name>UserCellphoneEnvironmentFeaturesProfileSelector</Name>
    <FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
      <ConditionName>BusinessMeetingPreferredCallerProfileCondition</ConditionName>
      <FeaturesProfileName>BusinessMeetingPreferredCallerProfile</FeaturesProfileName>
      <ReferredEventName>IncomingCallEvent</ReferredEventName>
      <ReferredContextElementName>
        InBusinessMeetingContextElement
      </ReferredContextElementName>
      <ConditionExpression>
        CellphoneIncomingCallEvent_CallerInListFilteringExpression.xquery
      </ConditionExpression>
    </FeaturesProfileCondition>
    <FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
      <ConditionName>BusinessMeetingNotPreferredCallerProfileCondition</ConditionName>
      <FeaturesProfileName>BusinessMeetingProfile</FeaturesProfileName>
      <ReferredEventName>IncomingCallEvent</ReferredEventName>
      <ReferredContextElementName>
        InBusinessMeetingContextElement
      </ReferredContextElementName>
      <ConditionExpression>
        CellphoneIncomingCallEvent_CallerNotInListFilteringExpression.xquery
      </ConditionExpression>
    </FeaturesProfileCondition>
    <FeaturesProfileCondition xsi:type = "FeaturesProfileDefaultCondition">
      <ConditionName>ElseCondition</ConditionName>
      <FeaturesProfileName>NotInMeetingProfile</FeaturesProfileName>
    </FeaturesProfileCondition>
  </FeaturesProfileSelector>
```

4.2 Cellphone as Controller Case Study

4.2.1 Description:

In this case study, instead of the cellphone adapting to context, the cellphone initiates the context adaptation of other devices. In this case study the use of independent environments will be illustrated, so instead of an environment controlling all behavior adaptation via *feature profiles*, events will be used to notify the proper sub-environment that components inside that sub-environment need to adapt.

As per the previous example, it is assumed that the device has the capability of filtering events depending on context, or adapting to context without the need of running a local application. Here, as well, the scenario will first be described, and the interaction that is taking place between the different components will be modeled.

The scenario is as follows: The user's alarm goes off, zz minutes later the user's car is started. The user receives confirmation the car computer controller has started the car. After getting ready to leave, the user gets in the car and as he or she is driving off the security alarm system is engaged and the temperature of the home set to a specific temperature. The user receives confirmation the smart home controller has engaged the alarm system and set the temperature of the home.

4.2.2 Participants:

User's Cellphone

Alarm application

True Remote Control Application (TRCA)

Smart Home Controller Unit (SHCU): For simplicity it is assumed there is a single unit that controls the different devices of the home. In this case the SHCU will control the temperature controller and the security alarm system

Home Security System

Home Climate Control Unit

Car Computer Controller Unit (CCCU): For simplicity it is assumed there is a single unit that controls the different devices of the car. In this scenario the Car Computer Controller Unit will control the engine starter (CCCU) (environment)

Car Engine Starter

Since the purpose of this scenario is to emphasize the use of context, and the use of independent environments; the interaction between controller units and components whose context is irrelevant is omitted. In this case, the interaction between the Smart Home Controller Unit and the specific elements of the home (the Security System and the Climate Control Unit) is omitted, as well as the interaction between the Car Computer Controller Unit and the Engine Starter. These interactions could be modeled using Features and Features Profiles: the Components exposing features and the proper controller selecting them (using a *features profile selector*), the same way it was done in the first case study.

4.2.3 Interaction via Events:

This scenario is divided in two parts: in the first part, the user's alarm goes off, zz minutes later the user's car is turned on. The user receives confirmation the Car Computer Controller has started the car.

In the second part, the user gets in the car and as he or she is driving off the security alarm system is engaged and the temperature of the home set to a specific temperature. The user receives confirmation the Smart Home Controller has engaged the alarm system and set up the temperature of the home.

Part one:

- The Alarm application detects the alarm has gone off:
    - An *AlarmGoesOffEvent* is generated by the Alarm application. This event will be used to calculate the context element *MustStartCarEngineContextElelement.* This context element belongs to the True Remote Control Application environment.
- The cellphone detects the time has changed (a minute has elapsed)
    - A *CellphoneTimeChangedEvent* is generated. This event is also used to calculate the *MustStartCarEngineContextElelement* context element.
- The location of the cellphone has changed:
    - A *CellphoneLocationEvent* is generated by the phone. This event will also be used to calculate the value of the context element *MustStartCarEngineContextElelement*.
- The User's device alarm has gone off, zz minutes have passed, and the user is at home, i.e., *MustStartCarEngineContextElelement* is true:

        - A *RemoteStarterActionEvent* event is generated by the smartphone's True Remote Control Application.
- The Car Computer Controller Unit (CCCU) receives the *RemoteStarterActionEvent* event:
    - The CCCU tries to start the car via the engine starter, then it sends a *ResultOfRemoteStarterActionEvent* event containing a flag indicating if the car was successfully started or not.
- The True Remote Control Application receives the *ResultOfRemoteStarterActionEvent* event:
    - It generates a *NotifyUserOfCarEngineStartedEvent* event.
- The user's mobile device receives the event *NotifyUserOfCarEngineStartedEvent* and the user is notified of the result of the attempt to start the car's engine. The notification is delivered via the notification methods selected by the user (alarm, email, a message box, etc.) The notification methods depend on the result of the attempt.

Part two:

- The user gets in the car and starts driving away from home:
    - Two *TriggerHomeActionEvent* events are generated by the True Remote Control Application. The first one instructs the Smart Home Computer Controller Unit (SHCCU) to set the temperature to a certain value, and the second to engage the security alarm.
- The Smart Home Computer Controller Unit (SHCCU) receives both events, the order in which the events are received is not important:
    - The SHCCU sets the temperature of the home via the home's temperature controller, then it sends a *ResultOfHomeActionEvent* event containing a flag indicating the result of trying to set the temperature of the home at a certain value.
    - The SHCCU engages the security alarm system, and then it sends a *ResultOfHomeActionEvent* event containing a flag indicating if the alarm system was successfully engaged.
- The True Remote Control Application receives both *ResultOfHomeActionEvent* events:
    - It generates two *NotifyUserOfHomeActionEvent* events. Two events are generated in case the user wants different notification methods depending on the type of action to be performed, and the result of each action.

- The user's mobile device receives the *NotifyUserOfHomeActionEvent* events and the user is notified of the result of the attempt to set the home's temperature and the attempt to engage the security alarm system.

4.2.4 Modeling:

In the previous case study the context was used as part of the conditions to select the proper feature profiles. In this case study the context will be used to filter events that trigger actions.

4.2.4.1 Components:

In this case study there are four environments: the user's Cellphone, the True Remote Control Application (TRCA), the Smart Home Controller Unit (SHCU), and the Car Computer Controller Unit (CCCU).

As components there is the Alarm application on the user's Cellphone; for the home there are the Home Security System, and the Home Climate Control Unit; and for the car, the Car Engine Starter.

The user's Cellphone environment deals with notifying the user of result of actions using *features profiles*. The Cellphone environment is specified as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Component.xsd">
   <Name>UserCellphone</Name>
   <ComponentFeatureSetName>
      UserCellphoneComponentFeatureSet
   </ComponentFeatureSetName>
   <ContainedComponent>
      <Name>AlarmApplication</Name>
   </ContainedComponent>
   <EnvironmentFeaturesProfilesSetName>
      UserCellphoneEnvironmentFeaturesProfileSet
   </EnvironmentFeaturesProfilesSetName>
   <FeaturesProfileSelectorName>
      UserCellphoneEnvironmentFeaturesProfileSelector
   </FeaturesProfileSelectorName>
</Environment>
```

Since the interaction between controllers and components is being omitted, the Smart Home Controller Unit (SHCU) environment is specified as:

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Component.xsd">
   <Name>SmartHomeControllerUnit</Name>
   <ContainedComponent>
      <Name>HomeSecuritySystem</Name>
   </ContainedComponent>
   <ContainedComponent>
      <Name>HomeClimateControl</Name>
   </ContainedComponent>
</Environment>
```

The Car Computer Controller Unit environment is specified as:

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Component.xsd">
   <Name>CarComputerControllerUnit</Name>
   <ContainedComponent>
      <Name>CarEngineStarter</Name>
   </ContainedComponent>
</Environment>
```

Finally, the True Remote Control Application is the environment that controls the propagation of events, so it must contain all components and environments that advertise or subscribe to events:

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Component.xsd">
   <Name>TrueRemoteControlApplication</Name>
   <ContainedComponent>
      <Name>AlarmApplication</Name>
   </ContainedComponent>
   <ContainedComponent>
      <Name>UserCellphone</Name>
```

```
    </ContainedComponent>
    <ContainedComponent>
        <Name>SmartHomeControllerUnit</Name>
    </ContainedComponent>
    <ContainedComponent>
        <Name>CarComputerControllerUnit</Name>
    </ContainedComponent>
    <EventContractSetName>
        TrueRemoteControlApplicationContractSet
    </EventContractSetName>
    <ContextSetName>
        TrueRemoteControlApplicationContextSet
    </ContextSetName>
</Environment>
```

4.2.4.2 Events:

Part One:

The advertisements for part one of this case study are: the Alarm application has a single advertisement, and it is for the *AlarmGoesOffEvent* event. The Cellphone advertises two events, one for the location of the cellphone, *CellphoneLocationEvent*, and one when the time of the cellphone changes, *CellphoneTimeChangedEvent*. These two events are used when calculating the *MustStartCarEngineContextElelement*, see the context section for more details on *MustStartCarEngineContextElelement*.

The True Remote Control Application (TRCA) also advertises two events: *RemoteStarterActionEvent*, and *NotifyUserOfCarEngineStartedEvent*.

The Car Computer Controller Unit (CCCU) advertises the *ResultOfRemoteStarterActionEvent* event.

The subscriptions for this part of the case study are: the Alarm application doesn't subscribe to any events. The Cellphone subscribes to the *NotifyUserOfCarEngineStartedEvent* event. The True Remote Control Application (TRCA) subscribes to *ResultOfRemoteStarterActionEvent*; and in order to calculate the *MustStartCarEngineContextElement* context element, it subscribes to *AlarmGoesOff*,

CellphoneLocationEvent and *CellphoneTimeChangedEvent*. Finally, the Car Computer Controller Unit (CCCU) subscribes to *RemoteStarterActionEvent*.

Part two:

The advertisements for part are: as per on part one, the Cellphone advertises its location, however, in order to calculate if the Cellphone is moving away from home, i.e.: if the distance from home is increasing, an event, *CellphoneSubsequentLocationEvent*, that keeps subsequent values of the *CellphoneLocationEvent* event is used. See the Context section for more details on how *MustTriggerHomeActionsContextElement* is calculated.

The True Remote Control Application (TRCA) advertises two events: *TriggerHomeActionEvent*, and *NotifyUserOfHomeActionEvent*.

The Smart Home Controller Unit (SHCU) advertises the *ResultOfHomeActionEvent* event.

The subscriptions for this part of the case study are: the Cellphone subscribes to the *NotifyUserOfHomeActionEvent* event. The True Remote Control Application (TRCA) subscribes to *ResultOfHomeActionEvent*; and in order to calculate the *MustTriggerHomeActionsContextElement* context element, it subscribes to *CellphoneSubsequentLocationEvent*. Finally, the Smart Home Controller Unit (SHCU) subscribes to *TriggerHomeActionEvent*.

The advertisements, including the corresponding event schemas, and subscriptions for this case study are specified as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<EventContractSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="EventContract.xsd">
   <Name>TrueRemoteControlApplicationContractSet</Name>
   <!-- Advertisements for Part One: -->
   <EventContract xsi:type = "Advertisement">
      <Name>AlarmGoesOffEventAdvertisement</Name>
      <ComponentName>AlarmApplication</ComponentName>
      <EventSchemaName>AlarmGoesOffEvent</EventSchemaName>
      <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="EventSchema.xsd">
         <Name>AlarmGoesOffEvent</Name>
         <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
```

```
        <Name>DateTime</Name>
        <TypeName>xs:dateTime</TypeName>
      </EventAttributeSchema>
    </EventSchema>
</EventContract>
<EventContract xsi:type = "Advertisement">
    <Name>CellphoneLocationEventAdvertisement</Name>
    <ComponentName>UserCellphone</ComponentName>
    <EventSchemaName>CellphoneLocationEvent</EventSchemaName>
    <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>CellphoneLocationEvent</Name>
      <EventAttributeSchema xsi:type = "NestedEventAttributeSchema">
        <Name>Location</Name>
        <ChildEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
          <Name>Latitude</Name>
          <TypeName>xs:double</TypeName>
        </ChildEventAttributeSchema>
        <ChildEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
          <Name>Longitude</Name>
          <TypeName>xs:double</TypeName>
        </ChildEventAttributeSchema>
      </EventAttributeSchema>
    </EventSchema>
</EventContract>
<EventContract xsi:type = "Advertisement">
    <Name>CellphoneTimeChangedEventAdvertisement</Name>
    <ComponentName>UserCellphone</ComponentName>
    <EventSchemaName>CellphoneTimeChangedEvent</EventSchemaName>
    <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>CellphoneTimeChangedEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>DateTime</Name>
        <TypeName>xs:dateTime</TypeName>
```

```
      </EventAttributeSchema>
   </EventSchema>
</EventContract>
<EventContract xsi:type = "Advertisement">
   <Name>RemoteStarterActionEventAdvertisement</Name>
   <ComponentName>TrueRemoteControlApplication</ComponentName>
   <EventSchemaName>RemoteStarterActionEvent</EventSchemaName>
   <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>RemoteStarterActionEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
         <Name>Action</Name> <!--StartEngine, StopEngine-->
         <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
   </EventSchema>
</EventContract>
<EventContract xsi:type = "Advertisement">
   <Name>NotifyUserOfCarEngineStartedEventAdvertisement</Name>
   <ComponentName>TrueRemoteControlApplication</ComponentName>
   <EventSchemaName>NotifyUserOfCarEngineStartedEvent</EventSchemaName>
   <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>NotifyUserOfCarEngineStartedEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
         <Name>Result</Name>
         <TypeName>xs:boolean</TypeName>
      </EventAttributeSchema>
   </EventSchema>
</EventContract>
<EventContract xsi:type = "Advertisement">
   <Name>ResultOfRemoteStarterActionEventAdvertisement</Name>
   <ComponentName>CarComputerControllerUnit</ComponentName>
   <EventSchemaName>ResultOfRemoteStarterActionEvent</EventSchemaName>
   <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
```

```
      <Name>ResultOfRemoteStarterActionEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Action</Name> <!--StartEngine, StopEngine-->
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Result</Name>
        <TypeName>xs:boolean</TypeName>
      </EventAttributeSchema>
    </EventSchema>
</EventContract>
<!-- Advertisements for Part Two: -->
<EventContract xsi:type = "Advertisement">
    <Name>CellphoneSubsequentLocationEventAdvertisement</Name>
    <ComponentName>UserCellphone</ComponentName>
    <EventSchemaName>CellphoneSubsequentLocationEvent</EventSchemaName>
    <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>CellphoneSubsequentLocationEvent</Name>
      <EventAttributeSchema xsi:type = "NestedEventAttributeSchema">
        <Name>Location</Name>
        <ChildEventAttributeSchema xsi:type = "SubsequentEventAttributeSchema">
          <Name>Latitude0</Name>
          <TypeName>xs:double</TypeName>
          <SourceEventName>CellphoneLocationEvent</SourceEventName>
          <SourceEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
            <Name>Latitude</Name>
            <TypeName>xs:double</TypeName>
          </SourceEventAttributeSchema>
          <OccurrenceIndex>0</OccurrenceIndex>
        </ChildEventAttributeSchema>
        <ChildEventAttributeSchema xsi:type = "SubsequentEventAttributeSchema">
          <Name>Latitude1</Name>
          <TypeName>xs:double</TypeName>
          <SourceEventName>CellphoneLocationEvent</SourceEventName>
```

```
    <SourceEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
      <Name>Latitude</Name>
      <TypeName>xs:double</TypeName>
    </SourceEventAttributeSchema>
    <OccurrenceIndex>1</OccurrenceIndex>
  </ChildEventAttributeSchema>
  <ChildEventAttributeSchema xsi:type = "SubsequentEventAttributeSchema">
    <Name>Latitude2</Name>
    <TypeName>xs:double</TypeName>
    <SourceEventName>CellphoneLocationEvent</SourceEventName>
    <SourceEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
      <Name>Latitude</Name>
      <TypeName>xs:double</TypeName>
    </SourceEventAttributeSchema>
    <OccurrenceIndex>2</OccurrenceIndex>
  </ChildEventAttributeSchema>
  <ChildEventAttributeSchema xsi:type = "SubsequentEventAttributeSchema">
    <Name>Longitude0</Name>
    <TypeName>xs:double</TypeName>
    <SourceEventName>CellphoneLocationEvent</SourceEventName>
    <SourceEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
      <Name>Longitude</Name>
      <TypeName>xs:double</TypeName>
    </SourceEventAttributeSchema>
    <OccurrenceIndex>0</OccurrenceIndex>
  </ChildEventAttributeSchema>
  <ChildEventAttributeSchema xsi:type = "SubsequentEventAttributeSchema">
    <Name>Longitude1</Name>
    <TypeName>xs:double</TypeName>
    <SourceEventName>CellphoneLocationEvent</SourceEventName>
    <SourceEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
      <Name>Longitude</Name>
      <TypeName>xs:double</TypeName>
    </SourceEventAttributeSchema>
    <OccurrenceIndex>1</OccurrenceIndex>
```

```
        </ChildEventAttributeSchema>
        <ChildEventAttributeSchema xsi:type = "SubsequentEventAttributeSchema">
          <Name>Longitude2</Name>
          <TypeName>xs:double</TypeName>
          <SourceEventName>CellphoneLocationEvent</SourceEventName>
          <SourceEventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
            <Name>Longitude</Name>
            <TypeName>xs:double</TypeName>
          </SourceEventAttributeSchema>
          <OccurrenceIndex>2</OccurrenceIndex>
        </ChildEventAttributeSchema>
      </EventAttributeSchema>
    </EventSchema>
  </EventContract>
  <EventContract xsi:type = "Advertisement">
    <Name>TriggerHomeActionEventAdvertisement</Name>
    <ComponentName>TrueRemoteControlApplication</ComponentName>
    <EventSchemaName>TriggerHomeActionEvent</EventSchemaName>
    <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>TriggerHomeActionEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Action</Name> <!--AlarmSystem, SetTemperature-->
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Value</Name>
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
    </EventSchema>
  </EventContract>
  <EventContract xsi:type = "Advertisement">
    <Name>NotifyUserOfHomeActionEventAdvertisement</Name>
    <ComponentName>TrueRemoteControlApplication</ComponentName>
    <EventSchemaName>NotifyUserOfHomeActionEvent</EventSchemaName>
```

```
    <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>NotifyUserOfHomeActionEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Action</Name>
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Value</Name>
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Result</Name>
        <TypeName>xs:boolean</TypeName>
      </EventAttributeSchema>
    </EventSchema>
  </EventContract>
  <EventContract xsi:type = "Advertisement">
    <Name>ResultOfHomeActionEventAdvertisement</Name>
    <ComponentName>SmartHomeControllerUnit</ComponentName>
    <EventSchemaName>ResultOfHomeActionEvent</EventSchemaName>
    <EventSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="EventSchema.xsd">
      <Name>ResultOfHomeActionEvent</Name>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Action</Name>
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Value</Name>
        <TypeName>xs:string</TypeName>
      </EventAttributeSchema>
      <EventAttributeSchema xsi:type = "SimpleEventAttributeSchema">
        <Name>Result</Name>
        <TypeName>xs:boolean</TypeName>
```

```
      </EventAttributeSchema>
   </EventSchema>
</EventContract>
<!-- Subscriptions for Part One: -->
<EventContract xsi:type = "Subscription">
   <Name>NotifyUserOfCarEngineStartedEventSub</Name>
   <ComponentName>UserCellphone</ComponentName>
   <EventSchemaName>NotifyUserOfCarEngineStartedEvent</EventSchemaName>
</EventContract>
<EventContract xsi:type = "Subscription">
   <Name>AlarmGoesOffEventSub</Name>
   <ComponentName>TrueRemoteControlApplication</ComponentName>
   <EventSchemaName>AlarmGoesOffEvent</EventSchemaName>
</EventContract>
<EventContract xsi:type = "Subscription">
   <Name>CellphoneLocationEventSub</Name>
   <ComponentName>TrueRemoteControlApplication</ComponentName>
   <EventSchemaName>CellphoneLocationEvent</EventSchemaName>
</EventContract>
<EventContract xsi:type = "Subscription">
   <Name>CellphoneTimeChangedEventSub</Name>
   <ComponentName>TrueRemoteControlApplication</ComponentName>
   <EventSchemaName>CellphoneTimeChangedEvent</EventSchemaName>
</EventContract>
<EventContract xsi:type = "Subscription">
   <Name>ResultOfRemoteStarterActionEventSub</Name>
   <ComponentName>TrueRemoteControlApplication</ComponentName>
   <EventSchemaName>ResultOfRemoteStarterActionEvent</EventSchemaName>
</EventContract>
<EventContract xsi:type = "Subscription">
   <Name>RemoteStarterActionEventSub</Name>
   <ComponentName>CarComputerControllerUnit</ComponentName>
   <EventSchemaName>RemoteStarterActionEvent</EventSchemaName>
</EventContract>
<!-- Subscriptions for Part Two: -->
```

```
    <EventContract xsi:type = "Subscription">
      <Name>NotifyUserOfHomeActionEventSub</Name>
      <ComponentName>UserCellphone</ComponentName>
      <EventSchemaName>NotifyUserOfHomeActionEvent</EventSchemaName>
    </EventContract>
    <EventContract xsi:type = "Subscription">
      <Name>ResultOfHomeActionEventSub</Name>
      <ComponentName>TrueRemoteControlApplication</ComponentName>
      <EventSchemaName>ResultOfRemoteStarterActionEvent</EventSchemaName>
    </EventContract>
    <EventContract xsi:type = "Subscription">
      <Name>CellphoneSubsequentLocationEventSub</Name>
      <ComponentName>TrueRemoteControlApplication</ComponentName>
      <EventSchemaName>CellphoneSubsequentLocationEvent</EventSchemaName>
    </EventContract>
    <EventContract xsi:type = "Subscription">
      <Name>TriggerHomeActionEventSub</Name>
      <ComponentName>SmartHomeControllerUnit</ComponentName>
      <EventSchemaName>TriggerHomeActionEvent</EventSchemaName>
    </EventContract>
</EventContractSet>
```

4.2.4.3 Context:

In part one of this case study there is one main context element: *MustStartCarEngineContextElement*. This context element depends on multiple conditions: the alarm has gone off, zz minutes have passed since the alarm went off, and the user is at home.

In order to make the calculation of the context element easier the context element is divided into two other context elements: first, *ActivationTimeHasElapsedContextElement,* which depends on the *AlarmGoesOffEvent* event and the *CellphoneTimeChangedEvent* event, and indicates if zz minutes have passed since the alarm went off; and second, *CellphoneAtHomeContextElelement,* which depends on the *CellphoneLocationEvent* event and indicates if the cellphone is at home. So *MustStartCarEngineContextElement* hence depends on *ActivationTimeHasElapsedContextElement* and *CellphoneAtHomeContextElelement*.

In part two there is one context element *MustTriggerHomeActionsContextElement*, which depends on the *CellphoneSubsequentLocationEvent* event, and indicates if the Cellphone's location started at home, and is moving away from the home. *CellphoneSubsequentLocationEvent* stores subsequent values of the *CellphoneLocationEvent*, those values are used to calculate the distance of the cellphone to the home, if those distances are increasing it is deduced the cellphone is moving away from the home.

Here is the specification of all context elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContextSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ContextSet.xsd">
   <Name>TrueRemoteControlApplicationContextSet </Name>
   <!-- Context Elements for Part One: -->
   <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>ActivationTimeHasElapsedContextElement</Name>
      <ValueType>xs:boolean</ValueType>
      <SourceEventName>AlarmGoesOffEvent</SourceEventName>
      <SourceEventName>CellphoneTimeChangedEvent</SourceEventName>
      <CalculationExpression>
         CellAsContrl_ActivationTimeHasElapsedContextElementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>TrueRemoteControlApplication</EnvironmentName>
   </ContextElement>
   <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>CellphoneAtHomeContextElelement</Name>
      <ValueType>xs:boolean</ValueType>
      <SourceEventName>CellphoneLocationEvent</SourceEventName>
      <!-- Formula to check if the distance of the cellphone to home is less than 1 -->
      <CalculationExpression>
         CellAsContrl_CellphoneAtHomeContextElelementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>TrueRemoteControlApplication</EnvironmentName>
   </ContextElement>
   <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>MustStartCarEngineContextElement</Name>
```

```
      <ValueType>xs:boolean</ValueType>
      <SourceContextElementName>
         ActivationTimeHasElapsedContextElement
      </SourceContextElementName>
      <SourceContextElementName>
         CellphoneAtHomeContextElelement
      </SourceContextElementName>
      <!-- Formula to check if the cell phone is at home
         and enough time has passed since the alarm went off: -->
      <CalculationExpression>
         CellAsContrl_MustStartCarEngineContextElementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>TrueRemoteControlApplication</EnvironmentName>
   </ContextElement>
   <!-- Context Elements for Part Two: -->
   <ContextElement xsi:type = "EnvironmentContextElement">
      <Name>MustTriggerHomeActionsContextElement</Name>
      <ValueType>xs:boolean</ValueType>
      <SourceEventName>
         CellphoneSubsequentLocationEvent
      </SourceEventName>
      <!-- Formula to check if the cell phone was at home, and it is
         now moving away: -->
      <CalculationExpression>
         CellAsContrl_MustTriggerHomeActionsContextElementCalculationExpression.xquery
      </CalculationExpression>
      <EnvironmentName>TrueRemoteControlApplication</EnvironmentName>
   </ContextElement>
</ContextSet>
```

And the expressions to calculate the context elements are:

- CellAsContrl_ActivationTimeHasElapsedContextElementCalculationExpression.xquery:

```
(:Tell if a specific number of minutes has elapsed since the alarm went off:)

(: adapted from: http://www.functx.com/ :)
```

```
declare function local:getDayTimeDuration
  ( $days as xs:decimal? ,
    $hours as xs:decimal? ,
    $minutes as xs:decimal? ,
    $seconds as xs:decimal? )  as xs:dayTimeDuration {

    (xs:dayTimeDuration('P1D') * $days) +
    (xs:dayTimeDuration('PT1H') * $hours) +
    (xs:dayTimeDuration('PT1M') * $minutes) +
    (xs:dayTimeDuration('PT1S') * $seconds)
 } ;

let $doc_alarm := fn:doc("AlarmGoesOffEvent.xml") (: Document with the current value of the Alarm
Goes Off event :)
let $event_alarm := $doc_alarm/Event
let $event_attribute_alarm_dt := $event_alarm/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                 SimpleEventAttributeSchemaName = "DateTime"]
let $alarm_dt := $event_attribute_alarm_dt/Value

let $doc_time := fn:doc("CellphoneTimeChangedEvent.xml") (: Document with the current value of the
Cellphone Time Changed event :)
let $event_time := $doc_time/Event
let $event_attribute_time_dt := $event_time/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                SimpleEventAttributeSchemaName = "DateTime"]
let $curr_dt := $event_attribute_time_dt/Value

let $elapsed_datetime := xs:dateTime($alarm_dt) + local:getDayTimeDuration(0,
                                                  0,
                                                  xs:decimal(15),
                                                  0)

return if ($alarm_dt) then
          $elapsed_datetime <= xs:dateTime($curr_dt)
       else
          false()
```

- CellAsContrl_CellphoneAtHomeContextElelementCalculationExpression.xquery:

```
(: Calculate distance to home :)
declare function local:distance-to-home-in-kms
  ( $latitude as xs:double, $longitude as xs:double, $reflatitude as xs:double, $reflongitude as
xs:double)  as xs:double? {
  1
 } ;

(: From the values of the Location Event, return if the distance to home is less than 1 :)
let $doc := fn:doc("CellphoneLocationEvent.xml") (: Document with the values of the current instance
of the event :)
let $event := $doc/Event
let $event_attribute := $event/EventAttribute[@xsi:type = "NestedEventAttribute" and
                                ParentAttributeSchemaName = "Location"]

let $lat_child := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute" and
./SimpleEventAttributeSchemaName = "Latitude"]
let $lat := $lat_child/Value

let $lon_child := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute" and
./SimpleEventAttributeSchemaName = "Longitude"]
let $lon := $lat_child/Value

return local:distance-to-home-in-kms($lat, $lon, 43.47, -80.54) < 1
```

- CellAsContrl_MustStartCarEngineContextElementCalculationExpression.xquery:

```
(:Tell if the cell phone is at home and enough time has passed since the alarm went off:)

let $doc_home := fn:doc("CellphoneAtHomeContexElementValue.xml") (: Document with the current
value of the Cellphone At Home context :)
let $doc_enough_elapsed_time := fn:doc("ActivationTimeHasElapsedContextElementValue.xml")  (:
Document with the current value of the Meeting In Progress context :)
let $at_home := $doc_home/Value
let $enough_time := $doc_enough_elapsed_time/Value
return ($at_home = true()) and ($enough_time = true())
```

- CellAsContrl_MustTriggerHomeActionsContextElementCalculationExpression.xquery:

```
(: Calculate distance to home :)
declare function local:distance-to-home-in-kms
  ( $latitude as xs:double, $longitude as xs:double, $reflatitude as xs:double, $reflongitude as
xs:double)  as xs:double? {
  1
 } ;

(: From the value of the first occurrence of the Location Event tell if the distance to home is less than
1,
and in subsequent occurrences the distance to home has increased :)
let $doc := fn:doc("CellphoneSubsequentLocationEvent.xml") (: Document with the values of the
current instance of the event :)
let $event := $doc/Event
let $event_attribute := $event/EventAttribute[@xsi:type = "NestedEventAttribute" and
                                ParentAttributeSchemaName = "Location"]
let $lat_child0 := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute"
                                        and ./SimpleEventAttributeSchemaName = "Latitude0"]
let $lat0 := $lat_child0/Value

let $lat_child1 := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute"
                                        and ./SimpleEventAttributeSchemaName = "Latitude1"]
let $lat1 := $lat_child1/Value

let $lat_child2 := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute"
                                        and ./SimpleEventAttributeSchemaName = "Latitude2"]
let $lat2 := $lat_child2/Value

let $lon_child0 := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute"
                                        and ./SimpleEventAttributeSchemaName = "Longitude0"]
let $lon0 := $lon_child0/Value

let $lon_child1 := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute"
                                        and ./SimpleEventAttributeSchemaName = "Longitude1"]
let $lon1 := $lon_child1/Value
```

```
let $lon_child2 := $event_attribute/ChildEventAttribute[@xsi:type = "SimpleEventAttribute"
                                        and ./SimpleEventAttributeSchemaName = "Longitude2"]
let $lon2 := $lon_child2/Value

let $dist0 := local:distance-to-home-in-kms($lat0, $lon0, 43.47, -80.54)
let $dist1 := local:distance-to-home-in-kms($lat1, $lon1, 43.47, -80.54)
let $dist2 := local:distance-to-home-in-kms($lat2, $lon2, 43.47, -80.54)

return
   ($dist0 < 1)
   and
   ($dist1 > $dist0)
   and
   ($dist2 > $dist1)
```

4.2.4.4 Features:

For this case study only the Cellphone exposes features, and there are used to notify the user of results of actions initiated by the True Remote Control Application:

```
<?xml version="1.0" encoding="UTF-8"?>
<ComponentFeatureSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ComponentFeatureSet.xsd">
   <Name>UserCellphoneComponentFeatureSet</Name>
    <ComponentName>UserCellphone</ComponentName>
   <ExposedFeature xsi:type= "CompoundFeature">
      <Name>Email</Name>
      <Feature xsi:type= "SimpleFeature">
         <Name>To</Name>
         <TypeName>xs:string</TypeName>
      </Feature>
      <Feature xsi:type= "SimpleFeature">
         <Name>From</Name>
         <TypeName>xs:string</TypeName>
      </Feature>
      <Feature xsi:type= "SimpleFeature">
```

```
      <Name>Subject</Name>
      <TypeName>xs:string</TypeName>
   </Feature>
   <Feature xsi:type= "SimpleFeature">
      <Name>Message</Name>
      <TypeName>xs:string</TypeName>
   </Feature>
</ExposedFeature>
<ExposedFeature xsi:type= "CompoundFeature">
   <Name>MessageBox</Name>
   <Feature xsi:type= "SimpleFeature">
      <Name>Title</Name>
      <TypeName>xs:string</TypeName>
   </Feature>
   <Feature xsi:type= "SimpleFeature">
      <Name>Message</Name>
      <TypeName>xs:string</TypeName>
   </Feature>
</ExposedFeature>
<ExposedFeature xsi:type= "CompoundFeature">
   <Name>SMS</Name>
   <Feature xsi:type= "SimpleFeature">
      <Name>From</Name>
      <TypeName>xs:string</TypeName>
   </Feature>
   <Feature xsi:type= "SimpleFeature">
      <Name>Message</Name>
      <TypeName>xs:string</TypeName>
   </Feature>
</ExposedFeature>
<ExposedFeature xsi:type= "CompoundFeature">
   <Name>Alarm</Name>
   <Feature xsi:type= "SimpleFeature">
      <Name>Message</Name>
      <TypeName>xs:string</TypeName>
```

```
      </Feature>
      <Feature xsi:type= "SingleValueEnumeratedFeature">
        <Name>Sound</Name>
        <PossibleValue>Sound1</PossibleValue>
        <PossibleValue>Sound2</PossibleValue>
      </Feature>
    </ExposedFeature>
</ComponentFeatureSet>
```

Features are then grouped into profiles so a single profile can consist of multiple features, for instance, send an email and a SMS.

For this example the set of available profiles are:

```
<?xml version="1.0" encoding="UTF-8"?>
<EnvironmentFeaturesProfilesSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ComponentFeaturesProfileSet.xsd">
   <Name>UserCellphoneEnvironmentFeaturesProfileSet</Name>
   <ComponentFeaturesProfileSet>
      <Name>UserCellphoneFeaturesProfileSet</Name>
      <ComponentName>UserCellphone</ComponentName>
      <FeaturesProfile>
        <Name>CarEngineStartedProfile</Name>
        <FeatureInstance xsi:type = "CompoundFeatureInstance">
          <FeatureName>SMS</FeatureName>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>From</FeatureName>
            <Value>True Remote Control</Value>
          </FeatureInstance>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>Message</FeatureName>
            <Value>Car Engine has been successfully started</Value>
          </FeatureInstance>
        </FeatureInstance>
      </FeaturesProfile>
      <FeaturesProfile>
        <Name>CarEngineNOTStartedProfile</Name>
```

```
   <FeatureInstance xsi:type = "CompoundFeatureInstance">
      <FeatureName>MessageBox</FeatureName>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
         <FeatureName>Title</FeatureName>
         <Value>WARNING</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
         <FeatureName>Message</FeatureName>
         <Value>Car Engine has NOT been started</Value>
      </FeatureInstance>
   </FeatureInstance>
</FeaturesProfile>
<FeaturesProfile>
   <Name>HomeTemperatureSetProfile</Name>
   <FeatureInstance xsi:type = "CompoundFeatureInstance">
      <FeatureName>SMS</FeatureName>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
         <FeatureName>From</FeatureName>
         <Value>True Remote Control</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
         <FeatureName>Message</FeatureName>
         <Value>The home temperature was set successfully</Value>
      </FeatureInstance>
   </FeatureInstance>
</FeaturesProfile>
<FeaturesProfile>
   <Name>HomeTemperatureNOTSetProfile</Name>
   <FeatureInstance xsi:type = "CompoundFeatureInstance">
      <FeatureName>MessageBox</FeatureName>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
         <FeatureName>Title</FeatureName>
         <Value>WARNING</Value>
      </FeatureInstance>
      <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
```

```
            <FeatureName>Message</FeatureName>
            <Value>The home temperature was NOT set successfully</Value>
          </FeatureInstance>
        </FeatureInstance>
      </FeaturesProfile>
      <FeaturesProfile>
        <Name>HomeAlarmEngagedProfile</Name>
        <FeatureInstance xsi:type = "CompoundFeatureInstance">
          <FeatureName>SMS</FeatureName>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>From</FeatureName>
            <Value>True Remote Control</Value>
          </FeatureInstance>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>Message</FeatureName>
            <Value>The home alarm has been engaged successfully</Value>
          </FeatureInstance>
        </FeatureInstance>
      </FeaturesProfile>
      <FeaturesProfile>
        <Name>HomeAlarmNOTEngagedProfile</Name>
        <FeatureInstance xsi:type = "CompoundFeatureInstance">
          <FeatureName>MessageBox</FeatureName>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>Title</FeatureName>
            <Value>WARNING</Value>
          </FeatureInstance>
          <FeatureInstance xsi:type= "SingleValueEnumeratedFeatureInstance">
            <FeatureName>Message</FeatureName>
            <Value>The home alarm was NOT engaged successfully</Value>
          </FeatureInstance>
        </FeatureInstance>
      </FeaturesProfile>
    </ComponentFeaturesProfileSet>
  </EnvironmentFeaturesProfilesSet>
```

Finally, the *feature profile selector* stores what to do depending on the success or failure and the type of action triggered by the True Remote Control Application (TRCA).

The selector stores a set of profiles and the conditions that must be true for each profile to be selected: in this case study the selector is pretty straight forward, there is a profile for each successful action triggered by the TRCA, and the corresponding not conditions in case the actions were not successful:

```
<?xml version="1.0" encoding="UTF-8"?>
<FeaturesProfileSelector xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ComponentFeaturesProfileSelector.xsd">
   <Name>UserCellphoneEnvironmentFeaturesProfileSelector</Name>
   <!-- Part One: -->
   <FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
      <ConditionName>CarEngineStartedProfileCondition</ConditionName>
      <FeaturesProfileName>CarEngineStartedProfile</FeaturesProfileName>
      <ReferredEventName>NotifyUserOfCarEngineStartedEvent</ReferredEventName>
      <ConditionExpression>
         NotifyUserOfCarEngineStarted_EnginedStartedSuccessfully.xquery
      </ConditionExpression>
   </FeaturesProfileCondition>
   <FeaturesProfileCondition xsi:type = "FeaturesProfileNotCondition">
      <ConditionName>CarEngineNOTStartedProfileCondition</ConditionName>
      <FeaturesProfileName>CarEngineNOTStartedProfile</FeaturesProfileName>
      <ConditionToNegateName>CarEngineStartedProfileCondition</ConditionToNegateName>
   </FeaturesProfileCondition>
   <!-- Part Two: -->
   <FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
      <ConditionName>HomeTemperatureSetProfileCondition</ConditionName>
      <FeaturesProfileName>HomeTemperatureSetProfile</FeaturesProfileName>
      <ReferredEventName>NotifyUserOfHomeActionEvent</ReferredEventName>
      <ConditionExpression>
         NotifyUserOfHomeActionEvent_TemperatureSet.xquery
      </ConditionExpression>
   </FeaturesProfileCondition>
   <FeaturesProfileCondition xsi:type = "FeaturesProfileNotCondition">
```

```
    <ConditionName>HomeTemperatureNOTSetProfileCondition</ConditionName>
    <FeaturesProfileName>HomeTemperatureNOTSetProfile</FeaturesProfileName>
    <ConditionToNegateName>HomeTemperatureSetProfileCondition</ConditionToNegateName>
  </FeaturesProfileCondition>
  <FeaturesProfileCondition xsi:type = "FeaturesProfileIfCondition">
    <ConditionName>HomeAlarmEngagedProfileCondition</ConditionName>
    <FeaturesProfileName>HomeAlarmEngagedProfile</FeaturesProfileName>
    <ReferredEventName>NotifyUserOfHomeActionEvent</ReferredEventName>
    <ConditionExpression>
      NotifyUserOfHomeActionEvent_AlarmEngaged.xquery
    </ConditionExpression>
  </FeaturesProfileCondition>
  <FeaturesProfileCondition xsi:type = "FeaturesProfileNotCondition">
    <ConditionName>HomeAlarmNOTEngagedProfileCondition</ConditionName>
    <FeaturesProfileName>HomeAlarmNOTEngagedProfile</FeaturesProfileName>
    <ConditionToNegateName>HomeAlarmEngagedProfileCondition</ConditionToNegateName>
  </FeaturesProfileCondition>
</FeaturesProfileSelector>
```

The three conditions are specified as follows:

- NotifyUserOfCarEngineStarted_EnginedStartedSuccessfully.xquery:

```
(:Tell if the car has been started successfully:)
let $doc := fn:doc("NotifyUserOfCarEngineStartedEvent.xml") (: Document with the value of the
current instance of the event :)
let $event := $doc/Event
let $event_attribute := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                SimpleEventAttributeSchemaName = "Result"]
let $res := $event_attribute/Result
return $res
```

- NotifyUserOfHomeActionEvent_TemperatureSet.xquery

```
(:Tell if the temperature was set successfully:)
let $doc := fn:doc("NotifyUserOfHomeActionEvent.xml") (: Document with the value of the current
instance of the event :)
let $event := $doc/Event
let $event_attr_action := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
```

```
                                    SimpleEventAttributeSchemaName = "Action"]
let $action := $event_attr_action/Value
let $event_attr_res := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                    SimpleEventAttributeSchemaName = "Result"]
let $res := $event_attr_res/Value

return ($action = xs:string("Temperature"))
      and
      ($res = true())
```

- NotifyUserOfHomeActionEvent_AlarmEngaged.xquery:

```
(:Tell if the alarm was engaged successfully:)
let $doc := fn:doc("NotifyUserOfHomeActionEvent.xml") (: Document with the value of the current
instance of the event :)
let $event := $doc/Event
let $event_attr_action := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                    SimpleEventAttributeSchemaName = "Action"]
let $action := $event_attr_action/Value
let $event_attr_value := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                    SimpleEventAttributeSchemaName = "Value"]
let $value := $event_attr_value/Value
let $event_attr_res := $event/EventAttribute[@xsi:type = "SimpleEventAttribute" and
                                    SimpleEventAttributeSchemaName = "Result"]
let $res := $event_attr_res/Value

return ($action = xs:string("Alarm"))
      and
     ($value = xs:string("on"))
     and
     ($res = true())
```

Chapter 5
Conclusion and Future Work

This document has defined a metamodel capable of modeling context awareness and behavior adaptation using the publish–subscribe principles of distributed event-based systems. The metamodel includes all elements needed to model scenarios where context plays a primary role. It also allows for the use of event-based system features such as event filtering, event enrichment, and event transformations.

Even though the concept of such a metamodel has been previously suggested in the literature, the rough idea was taken and matured to a point where modeling of complex cases is possible. In addition, by allowing context elements to depend on events and vice versa; and by defining context as a generic concept, the metamodel gains the flexibility of not being domain specific. It also provides options on how to model the flow of context-related information. Finally, it provides mechanisms, such as feature profiles selectors and event propagation, to model adaptation.

The main contribution of this metamodel is to open the possibility of making the design, and ultimately the development of context-aware applications, easier: if the basic concepts are implemented on the operating system level of mobile devices, and a GUI or an easy-to-use API is provided; as well as having a DEBS middle-layer on the cloud that could be accessible by the devices, the development of context-aware applications on mobile devices would be simplified immensely. If this point is reached, there is the potential to provide a platform for regular developers to include context awareness into their applications, or inspire them to create new context-aware applications altogether without the difficulty that this type of application presents today.

Finally, providing tools to facilitate the design and implementation of context-aware applications may become even more important as new concepts such as *mobile cloud computing* emerge. These concepts reduce the limitation of mobile devices, but may increase the complexity of designing and implementing applications. So if the design of context awareness can be simplified and its implementation provided by external tools like the ones suggested in this chapter, the design and development processes can concentrate on domain-specific functionality.

In addition to the long term plans for implementation, more immediate suggestions for future work are provided. First, besides features, and events, other mechanisms to trigger the behavior adaptation of a component can be defined. During the design of the case studies using the metamodel, translating

actions or instructions to features proved difficult and a little bit unnatural. Introducing the concept of actions can be a good complement to features. Features work well with simple adaptations, such as how a phone call is received, but in the future, differentiating between features and actions may prove useful in the quest of facilitating the design of context aware applications.

Second, another potential improvement is to introduce additional DEBS concepts into the metamodel such as *access roles* to control the propagation of context-related information. *Access roles* restrict the power of certain components to subscribe to particular events [4]. However, *environments* can provide such functionality for now. Finally, the concept of *time-to-live*, can also be introduced as a mechanism to enhance security and privacy of context data.

Bibliography

[1] A. K. Dey. Understanding and Using Context. *Personal and ubiquitous computing*, vol. 5.1, pages 4-7, 2001.

[2] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85- 90. IEEE Computer Society Press. Santa Cruz, California, Dec 1994.

[3] P. J. Brown, J. D. Bovey, X. Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, vol: 4.5. Oct 1997.

[4] R. Blanco, P. S. C. Alencar. Towards Modularization and Composition in Distributed Event-Based Systems Technical report, David R. Cheriton School of Computer Science, University of Waterloo, CS-2009-08.

[5] R. Blanco. *Process Models for Distributed Event-Based Systems*. PhD thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2010.

[6] J. C. Georgas, R. N. Taylor. Policy-Based Architectural Adaptation Management: Robotics Domain Case Studies. In *Software Engineering for Self-Adaptive Systems*, pages: 89-108, Springer Berlin Heidelberg, 2009.

[7] Adaptive Computing Concept - Adaptive Computing Controller in SAP NetWeaver. Aug 2012. Retrieved from: http://scn.sap.com/docs/DOC-8725 and: https://help.sap.com/saphelp_nwpi71/helpdata/en/46/0714a4e3f34f08e10000000a114a6b/frameset.htm. Accessed on: Dec 2014

[8] K. Pocek, R. Tessier, and A. DeHon. Birth and Adolescence of Reconfigurable Computing: A Survey of the First 20 Years of Field-Programmable Custom Computing Machines. In *Highlights of the First Twenty Years of the IEEE International Symposium on Field-Programmable Custom Computing Machines*. 2013

[9] J. Hoey, C. Boutilier, P. Poupart, P. Olivier, A. Monk, and A. Mihailidis. People, sensors, decisions: Customizable and adaptive technologies for assistance in healthcare. *ACM Transactions on Interactive Intelligent Systems (TiiS) - Special issue on highlights of the decade in interactive intelligent systems archive*. vol: 2.4, Dec 2012.

[10] M. Weiser. The Computer for the Twenty-First Century. *Scientific American,* vol:265.3, pages: 94–104, 1991.

[11] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. *IEEE Communication Surveys and Tutorials*, vol. 16.1, 2014

[12] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, vol: 10. 7, pages 1497-1516, Sep 2012.

[13] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communication Surveys and Tutorials*, vol. 16.1, 2014

[14] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier. The internet of things for ambient assisted living. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages: 804-809. 2010.

[15] P. Hu, J. Indulska, and R. Robinson. An autonomic context management system for pervasive computing. In *Pervasive Computing and Communications, 2008 Sixth Annual IEEE International Conference*, pages : 213 –223, Mar 2008.
[16] D. Zhang, H. Huang, C. Lai, X. Liang, Q. Zou, M. Guo. Survey on context-awareness in ubiquitous media. *Multimedia Tools and Applications,* vol: 67.1, pages 179-211. 2013.
[17] P. Nurmi and P. Floréen. Reasoning in Context-Aware Systems. Position paper, Helsinki Institute for Information Technology (HIIT) - Basic Research Unit (BRU), Dec 2004.
[18] E. S. Barrenechea, P. S. C. Alencar, R. Blanco, D. Cowan. Context-Awareness and Adaptation in Distributed Event-Based Systems. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, CS-2011-14.
[19] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, vol. 35.2, pages: 114-131, 2003.
[20] E. S. Barrenechea. A Context-Aware Publish–Subscribe Scheme. *29th ACM SYMPOSIUM ON APPLIED COMPUTING,* Gyeongju, Korea, Mar 2014.
[21] E.Castillejo, A. Almeida, D. López-de-Ipiña. Modelling users, context and devices for adaptive user interface systems. *International Journal of Pervasive Computing and Communications,* vol. 10.1, pages 69-91, Emerald Group Publishing Limited, UK, Apr 2014.
[22] G. M. Voelker, B. N. Bershad. Mobisaic: An information system for a mobile wireless computing environment. *Mobile Computing,* pages: 375–395, 1996.
[23] WWW Consortium. User agent profile (uaprof). 2001. Retrieved from: http://www.w3.org/2002/02/DIWS/presentations/nilsson/nilsson.pdf?q=uaprof. Accessed on: Dec 2014.
[24] WWW Consortium. Composite capability/preference profiles: structure and vocabularies 2.0—W3C working draft. 2007. Retrieved from: http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430/. Accessed on Dec. 2014.
[25] K. Cheverst, K. Mitchell and N. Davies. Design of an object model for a context sensitive tourist GUIDE. *Computers and Graphics,* vol. 23.12, pages: 883-891, 1999
[26] A. Gomez-Perez, O. Corcho, and M. Fernandez-Lopez. Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Second Printing). Springer, July 2004
[27] WWW Consortium Web Ontology Language (OWL). Retrieved from: http://www.w3.org/TR/owl-features/. Accessed on: Dec. 204.
[28] H. Chen, T. Finin, A. Joshi. An intelligent broker architecture for context-aware systems. *Adjunct proceedings of Ubicomp.*, vol. 3. 2003.
[29] H. Chen, T. Finin, A. Joshi. Using OWL in a pervasive computing broker. *DTIC Document*. 2005.