1-1-2012

Adopt: A Context-Aware Domain Ontology-Based Framework For Public Transportation

Petar Kramaric
Ryerson University

ADOPT: A CONTEXT-AWARE DOMAIN ONTOLOGY-BASED FRAMEWORK FOR PUBLIC TRANSPORTATION

By

Petar Kramaric

B.Sc, Ryerson University, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2012

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 authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

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

Abstract

ADOPT: A CONTEXT-AWARE DOMAIN ONTOLOGY-BASED FRAMEWORK FOR PUBLIC TRANSPORTATION

Petar Kramaric
MSc, Computer Science, Ryerson University, 2012

Understanding dynamic environments is a problem that many developers face when developing mobile applications for a particular domain. Dynamic environments can contain large sets of objects, people, and places that have varying characteristics and attributes that are constantly changing. Therefore, understanding all of the domain concepts and their relationships is a challenge for many application developers. To solve this problem an ontology based framework, ADOPT (context-**A**ware **D**omain **O**ntology-based framework for **P**ublic **T**ransportation) was constructed to provide application developers with the necessary tools and concepts that will allow them to quickly create and personalize mobile applications for the public transportation domain even if they have a limited knowledge of the domain. The goal of ADOPT is to provide application developers with a better understanding of their desired domain in order to personalize an application to meet the travel needs of various different types of passengers.

Acknowledgements

I would like to thank my supervisor Dr. Hossein Rahnama, without whom this thesis could not have been possible. He has devoted countless hours mentoring and guiding me through this process without any hesitation. He has provided me with the foundations needed to achieve my goals and helped me solve problems that I thought were unsolvable.

To my co-supervisor, Dr. Alireza Sadeghian, to whom I own many thanks. Without his insights and encouragement I would never have embarked on this journey three years ago. His willingness to guide a young undergraduate will never be forgotten. I would like to thank him for his patience and willingness to let me explore my ideas and never once hold me back.

I would also, like to thank my parents and brother for encouraging me to achieve anything that I set my mind to. Without their love and support I would not be where I am today. They have supported me throughout all of my life's challenges without a second thought and for that I am truly thankful.

Finally, I would like to thank Karen D'Souza. There are no words to explain how much she has supported me over the past several years. Every time I wanted to turn my computer off and do something else, she has been there to turn it back on and encourage me to continue. I can honestly say that I am truly lucky to have you in my life.

Dedication

This thesis is dedicated to my grandparents who taught me that no matter what life throws at you, if you show kindness, perseverance, and determination you will be able to get through anything.

Table of Contents

List of Figures

List of Tables

List of Expressions

Chapter 1. Introduction

1.1 Motivation

The seamless integration of computer systems into the objects and places that people encounter on a daily basis is slowly becoming a reality. This integration is often achieved through sensor embedded devices that are incorporated into the environmental spaces that people navigate every day such as their homes, workplaces, universities, and vehicles [1]. Through this integration, computational systems are able to obtain an extraordinary amount of data that pertains to our lives in an effort to make our daily tasks more efficient. These sensor-based spaces are able to collect information that can characterize our habits, routines, and schedules in order to personalize a person's experience within a designated space.

In recent years, with the advent of powerful mobile phones, *smart spaces* have become increasingly dynamic entities that are no longer constrained to a physical location. Mobile phones have given computer systems the ability to conceptually construct *smart spaces* anywhere with only limited infrastructure needed. They can provide computational systems with user-centric information such as a person's location, profile, calendar information, contacts, and any other data pertaining to the user. Despite all of this available data, most computational systems still require the user to search for data that is relevant to them at a particular time. They require users to search for things such as near-by restaurants, directions from one location to another, and schedules for upcoming buses and trains, instead of providing this data automatically based on the environment variables that surround the user. Providing data automatically becomes even more problematic in very dynamic environments which consist of thousands of different types of object and people constantly leaving and entering the designated *smart space*. Therefore there is a need to create computational systems that can fully understand

and process all of the various environment variables within a particular domain. By detecting changes to these environment variables, computational systems have the ability to inform people with information that is relevant to them at a particular period of time.

Designing such computational systems for dynamic spaces can, however, become an ominous task for developers, especially if they are not considered experts within a particular domain. For purpose of this document domain experts also known, as domain analysts, are people who have specific knowledge about a particular discipline and can create a knowledge base that consists of a domain's attributes, objects, and characteristics [2]. To create functional applications, developers must understand all of the environment factors that can be used to determine specific information within each *smart space*, as those environment factors will be used to determine what actions can be taken by the application to enhance the experience of people within the space. Unfortunately, developers do not always know or fully understand all of the environment variables available to them to within a particular space. This is especially concerning as spaces slowly become more and more dynamic therefore increasing the strain placed on developers as they must understand all the changing environment variables within each space. Hence, there is a need to create computational systems that developers, who are not considered experts, can use to obtain the necessary knowledge needed to create their own applications. These applications can then be used to enhance the user experience of people located within a particular smart space.

One such environment that encompasses many of the dynamic properties discussed above is the public transportation domain. The public transportation domain is comprised of thousands of different types of users, equipment, objects, and vehicles that are constantly changing within the domain. This is especially evident with travelling passengers as they have varying characteristics, attributes, ambitions, and objectives within their time of travel [3]; hence creating

applications that are personalized to each type of user in a domain can be an overwhelming task for developers of public transportation applications.

Figure 1.1 - The public transportation domain consists of people always on the move with various needs

Additionally, in recent years there has been a steady increase in the amount of passengers that travel using various public transportation methods in Canada [4]. This steady increase can be attributed to various reasons that include saving time and money as well as reducing the amount of vehicle emissions to better protect the environment [5]. At the same time, there has been numerous government based initiatives that encourage the creation and expansion of public transportation systems through monetary measures [4], [6]. With the overall increase in public transportation ridership, there is a need for transportation operators to improve the travel experience of, not only new passenger, but also passengers who have an elevated knowledge of their daily commutes. As public transportation systems are comprised of thousands of different

types of users, improving the transportation experience of every user on an individual basis can create an extraordinary drain on the people operating public transportation systems. This is especially evident when problems arise that are out of the control of transportation operators which includes issues that occur on a daily basis such as breakdowns, delays, malfunctions and system complexity that can cause problems to passengers of all forms. Issues such as delays are especially worrisome because they are often provoked by unexpected factors such as roadside accidents, power outages, weather conditions, and traffic congestion which are often unknown to passengers. Therefore, there exists a need for transportation operators to programmatically obtain all of the environment variables from embedded sensors that can be found within the public transportation domain. Using the data retrieved from these sensors, operators can better provide information about actions such as vehicle delays, passenger detour options, vehicle arrival time expectations, and causes of any machine malfunctions to their passengers. In order to provide better information to passengers, application developers must understand how all of the problems and issues that arise in the public transportation domain can directly affect a passenger's travel needs. These sorts of queries are often impossible to characterize by non-expert developer as no reference point is provided. Additionally, improving the travel experience also includes providing passengers with various entertainment options, points of interests, and social interaction to keep them busy as their travels might be long and tedious [7].

With the rapid expansion of powerful mobile devices, the task of enhancing a passenger's user experience is becoming less burdensome to transportation operators. To achieve the goal of a *smart* public transportation space, transportation operators have the ability to obtain specific information either directly from a passenger's mobile device or through the sensors embedded within stations, vehicles, environment, and other various objects found within the public

transportation domain. Using this sensor-based information transportation operators have the ability to personalize the information that each passenger receives on their mobile device. This information can provide vital data about any possible delays, malfunctions, breakdowns, etc. that impact the travel patterns of passengers as well as inform them of alternative measures that might improve their travel. However, understanding the needs and wants of thousands of passengers in large transportation systems can become a cumbersome task. Therefore, there is a need to create intelligent automatic systems that define all of the various elements found within the public transportation domain and how they are related to each other. By using such systems, application developers have the ability to better understand how various concepts are associated to each other and therefore create mobile applications that can better inform passengers of information that is relevant to them at a particular time.

1.2 Problem Description

Understanding the public transportation domain is a task that can be overwhelming to developers who are not considered experts within the domain. With the large number of sensor-based information available, the initial construction of a public transportation mobile application can be delayed due to the amount of research time needed to obtain all of the information needed to create a dynamic application. Therefore, the need for a computational system that defines all of the available environment information and their relationships is evident. With this computational system in place, application developers would be able to eliminate or at least reduce the amount of research needed to build a fully functioning mobile application for the public transportation domain.

In addition, to create a developer-friendly environment for creating public transportation applications there is a need to make these application dynamic and personalized to the individual

user of the application in order to provide more relevant information. Currently there are numerous mobile applications that inform passengers of various information that is provided to them by their public transportation operator of choice [8] [9] [10]. However, this information is most often provided in a static format such as vehicle schedules, route maps, trip planners, and near-by points of interest [11]. Although this information provides users with some valuable information that can be used to enhance their travel experience, it rarely personalizes the data presented based on the identity of the passenger using the application and what their objectives within the public transportation system are. Through the use of an intelligent public transportation computational system, application developers, who are not considered experts, have the ability to understand how specific environment variables can directly affect a passenger's travel; hence they would have the ability to customize/personalize the application experience for a user based on these variables obtained from the environment.

Currently there are applications that use various vehicle-based environment variables in order to inform the passengers of a more accurate time that their vehicle will be arriving at their desired destination [12], [13]. However, these applications do not provide passengers with any additional features during the delay of the vehicle that can improve their travel experience such as their surrounding points of interest. Therefore, there is a lack of mobile applications that retrieve a combination of both static and dynamic information simultaneously in order to provide the passenger with information that is relevant to them at a particular time.

There is also a lack of mobile applications that present relevant travel information in a user-specific format that is most suitable to the user of the application. This is especially evident with passengers that have a special need or a disability that might impede them from using their mobile phone devices in the same way that a general passenger would. People with special needs

and disabilities should not be exempt from using an application because it does not suit their needs, instead application developers should consider their limitations and restrictions when designing an application and incorporate the necessary adjustments in order to maximize the number of people that can functionally use the application. This is often the case with people who have sight and cognitive disabilities as well as elderly passengers that use public transportation methods as a vital avenue for getting from one place to another due to their lack of ability in operating an automobile [14]. Yet, many mobile applications provide these passengers with only limited accessibility features, if any at all, making these public transportation applications unusable by passenger who perhaps need their relevant transportation information most. The reason for this is that application developers often do not understand how a passenger's accessibility needs are affected within the public transportation domain. By developing applications through a computational system that defines the necessary information needed to determine if a user has a disability, application developers have the ability to personalize the experience of those user's within the mobile application. Public transportation systems also contain passengers that speak hundreds of different languages, which creates a need to alter the represented information on a mobile device in order to meet the needs of diverse populations. Hence understanding the user of the application is vital to providing them with the necessary information in a format that most suits the passenger.

Through these examples it is evident that there is a need to provide a computational system that defines all of the relevant information found within a public transportation domain. Using such a system, application developer will have the ability to personalize specific information based on who the user is and what environment variables can be obtained around them. However, currently there is a disjoint from making information personalized and flexible due to a lack of

developer understanding of the public transportation domain. Therefore, current public transportation applications are generally directed towards the general population and lack any form of personalization based on who is using the application. Creating a system that can obtain information from both the passenger and the environment could be used to enhance the experience of a passenger [15]. This is especially true when the passenger is a frequent user of a public transportation system and their travel needs and patterns can be observed and collected over an extended period of time. Therefore, there is an eminent need to create a computational system that informs mobile applications of any changing information that could result in a more personalized experience for the user. This system needs to also be able to reconfigure various software-based components of an application in order to suit a passenger's growing needs automatically, rather than having the passenger search for the necessary information manually.

1.3 Objectives

Based on the problems described in the previous section, the objective of the presented thesis is to describe a framework that provides application developers with the necessary tools needed to create dynamic personalized public transportation mobile applications. The objective of the proposed framework known as ADOPT, context-**A**ware **D**omain **O**ntology-based framework for **P**ublic **T**ransportation, is to provide application developers with the relevant relationships between all of the concepts and objects found within the public transportation domain. Using these concepts and relationships, application developers have the ability to define how changes to public transportation based objects can directly affect the functionality of their ADOPT-based mobile applications. Based on these changes, ADOPT-based mobile applications have the ability to reconfigure various software components based on the needs of the passenger that is affected by these changes. ADOPT uses an ontological structure in order to define the various user and

environment variables that encompass all of the concepts found within the public transportation domain. It was built as a public transportation domain ontology that integrates with the MUSIC context ontology presented by Reichle et al. [16]. Therefore, another objective of the proposed ADOPT framework is to allow application developers to create multi-user experiences that automatically reconfigure software components based on both the individual user's attributes and characteristics as well as the environment variables obtained from embedded device sensors.

The reconfiguration aspect of ADOPT is critical to this framework as there is a need to personalize each software component of the application based on the needs of the passenger. By creating mobile applications that reconfigure based on what is occurring within a passenger's environment, both transportation operators as well as third-party application developers have the opportunity to improve applications by providing dynamic content in a personalize fashion to each individual user. The objective of this feature is to enhance a passenger's experience while traveling through the public transportation domain. The focus of ADOPT was, however, not to introduce all the possible reconfiguration methods that application developers can use to instantiate reconfigurations, but rather build a framework that allows application developers to incorporate their own sensors and define how the retrieved sensor information should be used to reconfigure their target applications. In addition, ADOPT, provides a blueprint that application developers can use in order to quickly and efficiently build mobile application for the public transportation domain even if the developer is not considered an expert.

To demonstrate the advantages of ADOPT, an objective of the ontology based solution was to reduce the amount of time it took for software components to reconfigure once new sensor information was obtained from the embedded environment and device sensors. Additionally,

ADOPT was built as a pluggable ontology that allows application developers to build their own sensors that can detect various environment variables and include them into the system in order to expand the system's usability. This allows developers to use pre-built sensor retrieval methods that allow an application to register to incoming sensor information about delays, promotions, route information, and any additional information deemed suitable by the application developers and/or transportation operators.

In addition to the proposed contribution, the author of this thesis was a contributing member of the overall development of the MUSIC platform [17] which was designed by various research and industry partners. The MUSIC platform was used as the basis for the thesis presented in this document.

1.4 Thesis Outline

This chapter provided the introduction to the proposed thesis and outlines the motivations, presents the problems currently facing dynamic public transportation domains, and provides the contributions that will solve the described issues. Chapter 2 presents all of the background information and related work needed to fully understand the concepts presented in this thesis. It will focus on the characteristics, attributes, and concepts that pertain to dynamic spaces. The concept of context-aware computing and its association to dynamic spaces and objects will be introduced and examined in this chapter. Sections 2.2, 2.3, 2.4, and 2.5 and will focus on the various principles and characteristics that can be associated to context-aware computing in order to provide a more thorough understanding of dynamic spaces. Section 2.6 will focus on linking the concepts discussed in the previous sections to the public transportation domain as the concept of Intelligent Transportation Systems (ITS) will be examined in detail.

Chapter 3 introduces the proposed ADOPT solution for a public transportation domain, and in section 3.4, examines all of the concepts and models that were implemented into the public transportation ontology. Section 3.5 provides the system architecture of how the ontology can be used to create fully functional mobile applications designed for public transportation. The goal of this section is to introduce a solution that will incorporate the context-aware principles discussed in Chapter 2 into a dynamic space such as that of public transportation.

Chapter 4 provides all of the quantitative and qualitative results that prove that the ADOPT solution, described in Chapter 3, does in fact provide an improvement over linear key-value based systems. This chapter, also, introduces a case study of a mobile application that was implemented through both linear key-value models and the ADOPT framework, in an attempt to personalize a metro passenger's travel experience by reconfiguring various software components based on changing sensor information. Chapter 4, also, demonstrates through quantitative results, that the ADOPT framework can be used to reconfigure system components faster than standard linear key-value solutions, therefore providing a more optimal solution for context-aware public transportation mobile applications. Qualitative results in this chapter will discuss how research time can be reduced through the use of ADOPT as all the necessary public transportation concepts are already built into the framework.

Finally, Chapter 5, provides a summary of the presented work in this thesis, it discusses some limitations to the current ADOPT framework as well as propose future work that can further enhance the overall system.

Chapter 2. Background and Related Work

Public transportation systems provide an ideal environment to demonstrate how sensor-based information can be highly dynamic and ever-changing. Public transportation systems consist of thousands of diverse people who all have different objectives, emotional states, identities and are constantly moving from place to place. This makes it an ideal candidate for the integration of the *context-aware* principles discussed in this chapter which aims to provide a thorough review on the concepts, principles and foundations of context-aware computing. These concepts and principles include *context and mobile sensing*; the collection of data from physical, virtual, and logical sensors, *context abstraction*; the transformation of raw data into high level data an application can process, *context modeling and synthesis*; the formal representation of data within a model and its reasoning techniques, and *context distribution*; the collection of context from various devices and its distribution to all other devices within the system. In order to demonstrate these four concepts a review of the Intelligent Transportation Systems (ITS) literature and the challenges facing both research and the industry will be discussed.

2.1 Context-Aware Computing

The first notions on context-aware computing were introduced by Weiser [18] who described a new form of computing, known as ubiquitous computing, that would vanish into the background by seamlessly integrating into the devices that people use every day. One of the fields that emerged from this vision was context-aware computing, which details the need to create computational systems that can adapt and reconfigure their behavior, without interaction from the user, in order to increase productivity, usability, and effectiveness of a system [19]. Theses adaptations/reconfigurations in context-aware computing are triggered by changes to context variables defined within a system.

The term *context* was first conceived by Schilit and Theimer [20] who originally equated it to any information that can be used to characterize how a mobile user is situated in a particular environment or scenario. Schilit and Theimer further elaborated on their definition by identifying context as the locations and identities of nearby people and objects. As the term context matured, alterations to the original definition were suggested by other researchers such as Ryan, et al. [21] who claimed that a user's location and identity did not provide sufficient information for a system to fully characterize a user's environment. Therefore, they added time to the original definition. Time is a contextual characteristic that allows a system to understand a user's history and/or behavioral patterns which can be used to obtain a more precise understanding of what a user's objectives are within a specific scenario/domain.

Dey et al. [22] took these ideas a step further and defined context as "Any information that can be used to characterize the situation of entities (i.e., a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical object". Although this definition seems quite broad it entitles all the necessary information needed to define context. In essence, context can be described as any relevant information used to characterize a specific scenario. By using contextual information in a particular scenario, a system has the ability to provide relevant information and/or services based on the information collected [23]. This can be seen in early projects such as Stick-e Notes [24] where based on a user's location, a set of user created notes were made available on their mobile device. By detecting changes to the contextual information, in this case location changes, applications are able to initiate the desired reconfigurations defined within the system architecture. In the Stick-e Notes project, as the user changes their location, their mobile device

automatically reconfigures by changing the displayed notes on the user's screen to reflect the notes of the new location. If however, in a certain location no notes are available then nothing is displayed to that user. Since context-aware systems need to reconfigure automatically there is a need to create systems that respond in real-time, are transparent and intuitive, and require users to perform as little interaction as possible with a device [23].

To better understand the concept of context-awareness, a scenario of a context-based restaurant finder can be examined. In such a scenario relevant contextual information could include a user's dietary needs, meal price range, and/or physical distance to a restaurant [25]. Based on these contexts, an application can display a list of restaurants that will best suit the user's current needs. When any of the defined context variables change, a corresponding change to the result set of restaurants the application displays is made. For example, if a user has indicated that their dietary needs are vegan, the result set of displayed restaurants might be significantly different from a user who has not indicated such a need. Based on all the contexts defined, the goal of a restaurant finder system would be to provide a relevant set of restaurants based on a user's needs by detecting them automatically through the use of embedded sensors. This increases the usability of a mobile application as it eliminates the need for users to persistently interact with the application by manually searching for desired information. When deciding which contextual information is central to a system developers need to understand the domain which their system will be based within as the inclusion of irrelevant context variables can lower productivity and slow their system down. Using relevant context information allows the system to reconfigure only when a need is detected, creating a more efficient and useable the system to its users.

Context-aware systems are made up of four main principles; sensing, abstraction, modeling and synthesis, and distribution. Section 2.2 will discuss the challenges and issues of context and

mobile sensing within a context-aware system. Section 2.3 describes the importance of abstracting low-level context data into high-level situational data. Section 2.4 defines how sensed and abstracted data can be modeled and processed by a computer system, and section 2.5 will conclude with a discussion on the challenges of distributing contextual information throughout a system. These relationships between these principles can be observed in Figure 2.1.

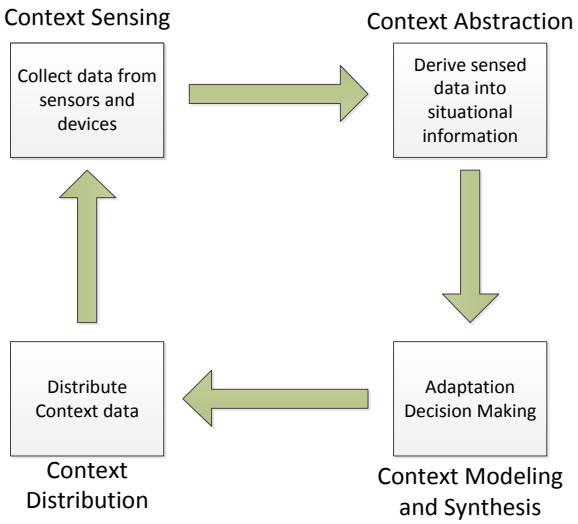

Figure 2.1 - Fundamental concepts of context-aware computing

2.2 Context and Mobile Sensing

The first core function of context-aware systems is the collection and gathering of sensor data in order to determine the contextual information needed for system reconfiguration. This process is known as context and mobile sensing. According to Indulska and Sutton [26], context and mobile sensing can collect data from any of the following sources:

- *Physical Sensors*; hardware sensors used for detecting physical data. Some examples include using accelerometers for movements, microphones for audio, electronic thermometers for temperature, etc.

- *Virtual Sensors*; software services that can detect information stored in databases and/or other software applications. Some examples of virtual sensors include collecting location information based on a user's mobile device calendar which can indicate where a user should currently be located as well as retrieving information about a user's friends and family through a contact list found on their mobile device.
- *Logical Sensors*; the combination of physical and logical sensors. For example a logical sensor can be used to get a person's location based on which desktop that person is logged into (physical) and a database of where that desktop is physically located (virtual) [26].

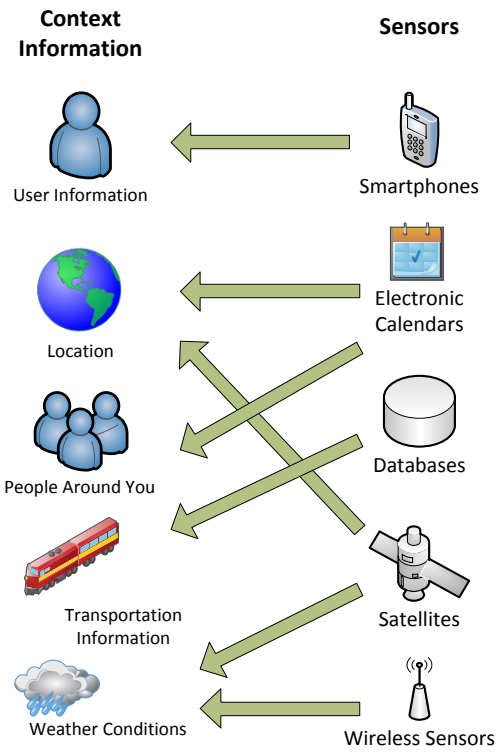

Figure 2.2 - Examples of different types of context information determined by various sensors

Figure 2.2 demonstrates that contextual information can be sensed from a multitude of physical, virtual, and logical sensors. Due to their dynamic nature, sensors offer an ideal way of detecting

and collecting raw context information, however there exist variables that at this present time cannot be determined through the use of sensors. In order to collect this non-sensor based data, a context-aware system needs to have the ability to process user-supplied data which the user can manually enter through a system device [27]. An example of this data is often seen around a user's profile and can include information such as dietary needs, religious beliefs, date of birth, etc. which sensors currently cannot detect. Regardless of the way that data is collected, context information can be classified as either static or dynamic. Static context is the data that is permanent and does not change or seldomly changes, while dynamic context refers to data that changes frequently. Examples of static and dynamic context information can be seen in Table 2.1.

Table 2.1 - Examples of static and dynamic context information

Context Type	Examples
Static	Date of Birth, Religious Beliefs, Eye Colour, etc.
Dynamic	Location, Mobile Device Bandwidth, Time of Day, Temperature, etc.

Knowing whether a context is static or dynamic allows context-aware systems to determine the appropriate time for collecting contextual data. In context-aware systems there are two main ways for collecting dynamic data; continuous or event-based [28]. Continuous context sensing involves the need for a persistant connection with sensors where data can be obtained at specific intervals that are predefined by a system expert in order to reduce overhead. This is espacially useful when data is provided through software services and agents using conntection-oriented protocols. Event-based context data, on the other hand, does not need to be constantly active, but

rather activated and deactived based on pre-defined events. An example of event-based sensing can be seen when determing a user's location based on Radio-Frequency IDentification (RFID) tags. When a user enters a room and scans their RFID tag (often found in key cards), a system event is triggered that indicates that a particular user has changed locations. Continous sensing, on the other hand, would notify the system at specific time intervals even if the user's location has not changed. An advantage of event-based sensing is that it does not rely on the current system states but rather depends on the changes to those states.

When collecting sensor data, either continously or through events, the notion of imperfect data is a challenge that is present within all context-aware systems as sensors can become unreliable. Accroding to Hendrickson and Indulska [27] there are four major categories that imperfect data can be classified as, they are as follows:

- *Erroneous Data*; occurs when there is a discrepancy between data received from a sensor and the actual data. Erroneous data can occur when sensors are programmed incorrectly resulting in incorrect device data.
- *Ambiguous Data*; occurs when multiple sensors contradict each other by providing opposing data. This can be seen when calculating the speed of the same vehicle using different devices such as speedometers and radar guns.
- *Unknown data*; occurs when data cannot be retrieved from a sensors. This is often the case when a mobile device loses connectivity to the server and cannot report specific data.
- *Imprecise data*; occurs when data is received that, although correct, is not specific enough for the system. This occurs when a system is trying to determine a user's

location based on mobile cellular triangulation. Although the system can determine the limited range of where the user is, their exact location is difficult to calculate.

Despite allowing context-aware systems to process data into context, a single sensor cannot always fully characterize a context. There are cases where the use of multiple sensors is needed in order to define a single context; this is known as *context fusion*. According to Bernardos et al. [29], *context fusion* is the process of collecting and combining data from multiple heterogeneous sensors in order to form more complex and reliable context data. Combining multiple sensors aids the context-aware system to further understand the environment that a user is currently in as more data is available to make various assumptions. This can be seen in a context-aware system that needs to have knowledge of outdoor weather conditions. It is not enough to simply detect and collect temperature readings from a single device to determine the state the weather is currently in. The system needs to also obtain data from sensors that detect precipitation, air pressure, wind speed, humidity, etc. By combining these readings, the context of "how the weather is outside" can be determined more reliably and effectively than simply using the temperature sensor. Then based on the weather conditions outside, the context-aware system can reconfigure its behaviour accordingly.

2.3 Context Abstraction

After a context-aware system has collected all the necessary data from the sensors it needs to be able to understand the data by converting it into contextual information that the system can process. This is done through context abstraction. Context abstraction interprets the raw data received from the sensors into something more meaningful to the system. Raw data, often referred to as low-level data, consists of quantitative data that is received directly from a sensor such as the precise temperature reading in Degrees Celsius or physical locations of objects using

geographic coordinates [30]. Context-aware systems can manipulate this quantitative data into context variables that the system can process. For example the temperature reading of a room at a raw data value of 25.2 Degrees Celsius, without any interpretation, is meaningless to a context-aware system without the data being abstracted. Therefore, the raw data needs to be processed using intelligent interpretation to form what is known as high-level abstracted data [31]. Using the same example of temperature, by preforming high-level abstraction a system can determine if the temperature reading indicates that the room is "too hot", "too cold", "just right", etc. which can then be used to determine if system reconfiguration should occur. Another form of abstraction, according to Dey [32], is known as *situational abstraction* which combines multiple high-level abstracted contexts to get a better understanding of what is currently happening within the system domain. Examples of *situational abstraction* include determining, based on the defined contexts, whether a user is "in a meeting", "eating lunch", etc.

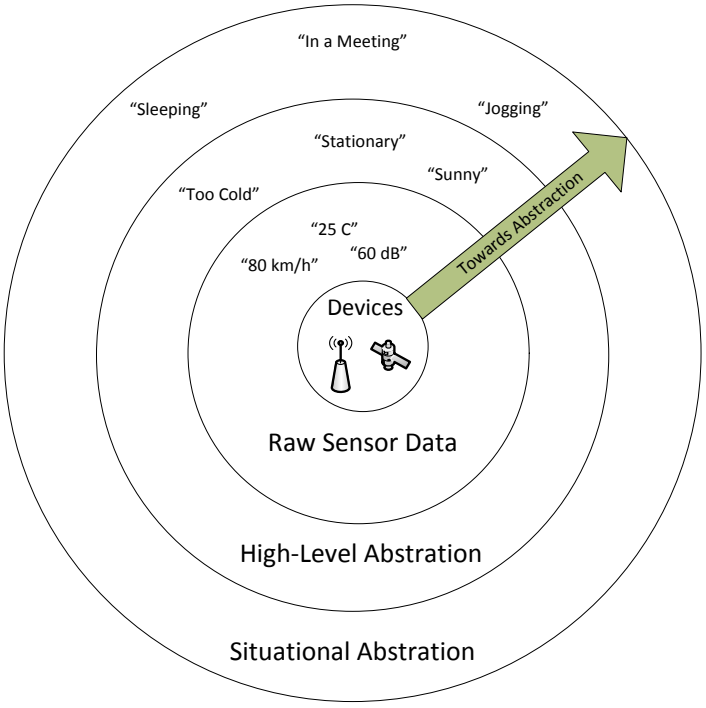

Figure 2.3 - Context abstraction levels

By determining situational contexts, computational systems are able to get a better understand of what states its entities, such as users, are, therefore providing a better understanding of which software component might need to be reconfigured. Figure 2.3 demonstrates how raw data, high-level data, and *situationally abstracted* data are associated to each other, in this case, the higher the level the more abstracted the context is.

In order to create high-level and *situationally abstracted* contexts, the use of widgets, interpreters, and aggregators was proposed as part of the Context Toolkit architecture [22]. The goal of widgets, interpreters, and aggregators is to transform the data received from sensors into contextual information defined by the computational system. Context widgets provide the link to the encapsulated raw sensor data that can be reused by any application across any domain. For example a *presence* widget [22] can be used to determine who is currently present in a particular room based on the RFID tags that are scanned whenever someone enters and leaves a room. N general the advantages of widgets according to Dey et al. [22] are as follows:

- *Hiding Complexity*; manipulate the sensor response data into data an application can use
- *Abstract Context Information*; to suit the needs of an application
- *Provide reusability*; can be used by various applications at the same time

Interpreters, on the other hand, are responsible for increasing the abstraction levels of particular context information. According to Dey et al [22], raising the abstraction levels of a context variable can come in two forms:

- *Simple*; raising the abstraction of context data retrieved from a single widget, such as converting geo-coordinates into city names using a geographic information database.

- *Complex*; raising the abstraction of context data retrieved from a combination of widgets, such as determining if a meeting is occurring by using a presence widget, and a sound level widget [22]. *Situational abstraction* is usually determined through *complex* abstraction as multiple widgets are often needed to define a particular situation.

Finally, aggregators are used for collecting related contextual information relevant to an application. They deliver specific context information from the widgets to the application itself in order to reduce the complexity that an application may face when collecting context from multiple widgets. Aggregators allow the application to retrieve information about a specific entity from one source rather than having to retrieve it from multiple sources.

According to Schmidt et al. [33], an alternate way to abstract raw sensor data into complex situational data is through the use of *cues*, *contexts*, and *scripts* which are particularly useful when dealing with sensor fusion principles discussed in section 2.2. *Cues* take the input received from the sensors and transform it into a symbolic output based on a pre-defined set of possible values, such as "stationary" or "walking" which can be used for determining user movement. Every *cue* is assigned to a single sensor; however multiple *cues* can be dependent on a single sensor [33]. For example there could exist two *cues* that are created based on a single accelerometer sensor; one cue can be used to determine if a user is "stationary" or "moving" while another *cue*, based on the same sensor, can determine the orientation of a mobile phone as either "landscape" or "portrait".

By combining the data received from multiple cues, further abstraction is performed to create contexts that can be used to define a particular situation [33]. Table 2.2 displays some examples of cues that can be transformed, using intelligence methods, to create contexts.

Table 2.2 - Examples of cues and contexts

Contexts	Cues
Sleeping	Artificial light, stationary or moving, room noise, room location
Jogging [33]	Natural light (cloudy or sunny), walking or running, dry or raining, high pulse

Finally the scripting layer determines whether a context is being entered, left, or neither in order to determine when system reconfiguration should occur [33]. When retrieved data from a cue results in a change to a context, the new context information is considered to be the entering context, while old context is considered to be the *leaving* context. For example, when a user movement context changes from "moving" to "standing", the entering context is "standing" and the leaving context is "moving". The goal of using these levels of abstraction is to allow an application developer to fully characterize any situation relevant to an application. The number of cues and contexts that one creates are usually in direct correlation with how complex a context-aware system is. In most cases, the more complex a scenario is, the more sensors, cues, and contexts are built into the system.

Context abstraction models affect the reliability of every context-aware system as they can reduce the level of complexity when dealing with raw sensor data. Abstraction allows the application developers the opportunity to use the data obtained from sensors in order to create contextual variables which can be used to reconfigure an application. Without abstraction, it is difficult for a system to determine what the user is doing and whether or not the system should behave differently in a particular situation. Whether it is through the use of cues, interpreters, or

any other mechanisms, abstraction allows an application to process the situation that a user is currently in.

2.4 Context Modeling and Synthesis in Dynamic Spaces

Along with the ability to sense and collect contextual data, context-aware systems need to have the capability of storing sensed and abstracted data in a machine processable form in order for system reconfiguration to successfully occur [19]. By having the ability to process the sensed data and convert it into data the system can understand, abstraction is allowed to occur. This is a critical aspect of any context-aware system and is known as context modeling. Context modeling addresses the need of demonstrating how various context variables interact with each other in order to validate that a particular situation is occurring. For contextual representation to be modeled, a data model should be conceived for storing information about the context of a particular event [34]. Due to the highly dynamic nature of context-aware systems, a good context modeling schema should reduce complexity, improve the maintainability, and provide reusable components [35]. A summary of the various context modeling techniques used in context-aware systems can be seen in Figure 2.4.

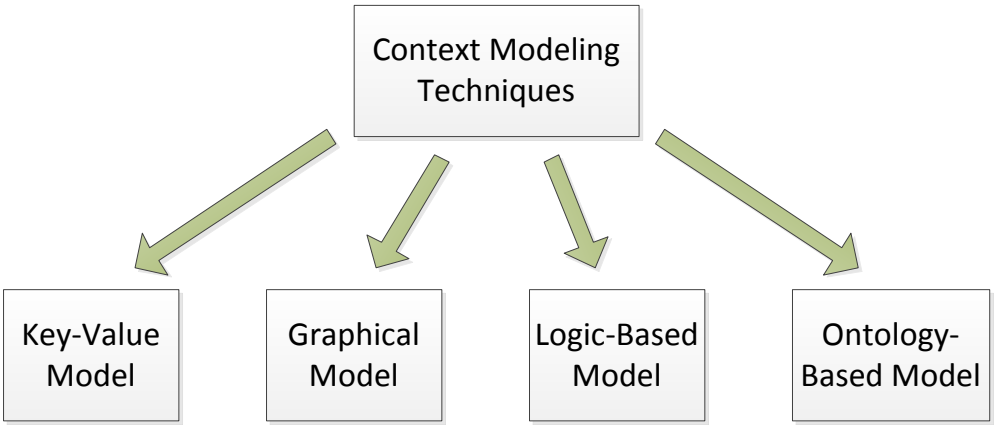

Figure 2.4 - Context modeling techniques

Along with representing the contextual data within a model structure, context-aware systems need to be able to deduce the contextual information found within a model to allow the system to process which situation is occurring. By determining which situation is occurring, the system can decide software component should be reconfigured; according to Zhang et al. [36] this is known as context synthesis/reasoning.

2.4.1 Key-Value Modeling

A data structure that can be used to model contextual information is a *Key-Value Model* (KVM). In order to reduce the complexity of contextual information KVMs use pairs of keywords and values for each system defined context variable. However, as the complexity of context-aware systems increases, the simplistic nature of KVMs make this form of modeling difficult to integrate efficient retrieval algorithms used for determining context information [37]. A simple key-value modeling example can be found in the Active Map Service (AMS) [20] project where objects in the computational system are located based on key-value queries for specific attributes. An example of a key-value tuple can be seen in Expression 2.1, which informs the system that the context entity person has a "role" of a "passenger". This information can then be used to obtain information about a specific passenger. The *Key-Value Model* is also the primary model used in the Context Toolkit [22].

$$person(key = \text{“}role\text{”}, value = \text{“}passenger\text{”})$$

Expression 2.1 - Example of key-value pair

Several other examples of key-value modeling can be seen in Figure 2.5, where specific contextual information is paired with data retrieved from various sensors. In KVMs, the system

developer is responsible for determining which contexts are available to be used by an application of that system.

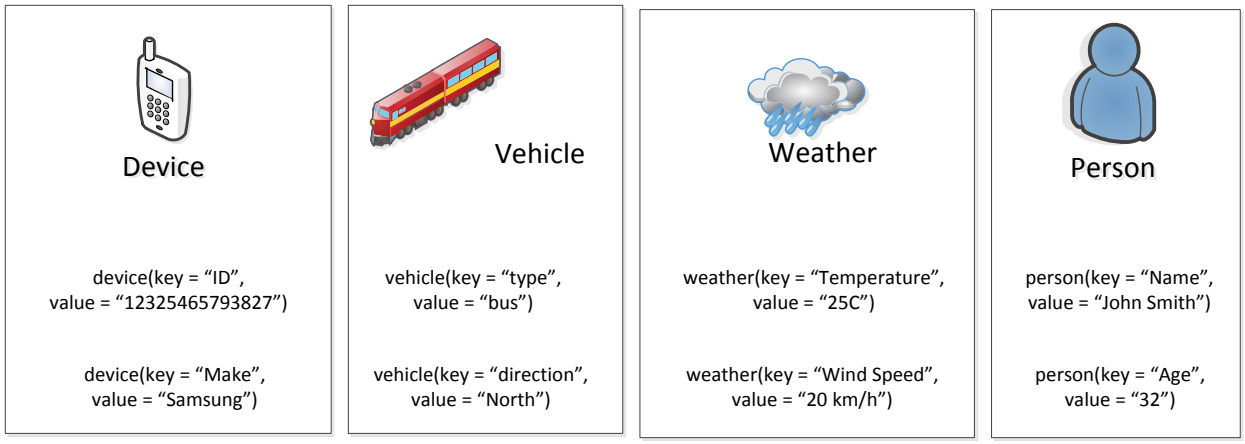

Figure 2.5 - Various examples of key-value pairings

2.4.2 Graphical Modeling

One of the more popular modeling techniques used for general purpose modeling is the Unified Modeling Language (UML) which can be adapted to model context through its graphical components [37]. Modeling techniques that involve graphical components such as UML diagrams, Entity-Relationships (ER) diagrams, or conceptual graphs are known as *Graphical Models*. One of the main benefits of graphical models is their ability to make information very readable and understandable to the application developer [38]. One of the examples of a *Graphical Model* was developed by Hendrickson et al. [39] who created an extension of the Object-Role Model (ORM) [40] that revolves around the notion of a *fact*. In ORMs, *facts* are used to model domains by identifying all the necessary entity types such as the people and/or objects that have functioning roles within the domain [37]. An example of a simple ORM fact can be seen in Figure 2.6 where two entities, person and device, are associated to each other using the relationship “owned by”, suggesting that every device with an identifier (ID) in this

domain is owned by a specific person. Using ORM facts, such as the one in Figure 2.6, can help developers better understand all of the contextual information found within a domain and the scenarios that an entity or context can be a part of. One drawback to the graphical model approach is the difficulty in characterizing incomplete and ambiguous data obtained from sensors [37]. Although contextual information can be characterized using the graphical models, they are limited in the way they represent inaccurate data obtained from sensors therefore that this information can often be left out of graphical models.

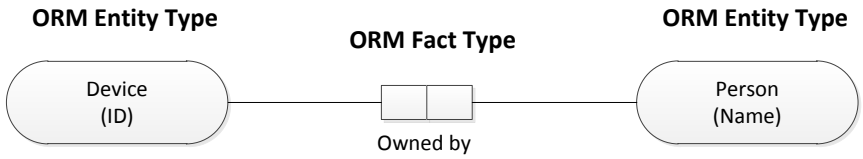

Figure 2.6 - Example of simple ORM fact type

2.4.3 Logic-Based Modeling

Similar to graphical models, *Logic-Based Models* (LBMs) also uses facts in addition to expressions and rules to model context variables. *Logic-Based Models*, unlike *Graphical Models* however, provide a high level of formality while lowering readability and understandability in a mutual trade-off [37]. Formality is critical to functionality of LBMs as it allows them to determine exactly which information is needed to define a particular context. As facts are the fundamental piece of information in LBMs, they can be used to define the various different types of sensed and non-sensed context variables [41]. An example of *Logic-Based Models* can be seen in Ranganathan and Cambell's [41] first order predicates which are defined in Subject Verb Objects (SVO) form, as presented in Expression 2.2.

$$Context - Type\ (< Subject >, < Verb >, < Object >)$$

Expression 2.2 - Structure of Subject Verb Object predicates

Based on the defined SVOs, computational systems have the ability to determine all the context information found in the domain. Several examples of Logic-based predicates can, also, be seen in Table 2.3.

Table 2.3 - Examples of logic-based first order predicates

Predicates	Description
Location(Chris, Entering, Room 3231)	Defines a Location predicate that consists of user "Chris" entering room 3231.
Temperature(Room 3231, "=", 23C)	Defines a Temperature Predicate where the temperature of room 3231 is equal to 23 Degrees Celsius.

2.4.4 Ontology-Based Modeling

Another technique for modeling contextual information is through the use of ontologies. *Ontology-Based Modeling* is used to describe a particular domain by forming relationships between the various objects found within a specific domain. According to Kabilan [42] ontologies are defined as real-world concepts that are organized hierarchically and modeled using classes, characteristics, relationships, and properties. Ontologies, according to Kabilan [42], can be classified as any of the following ontology types:

- *Top Level Ontologies*; general concepts found in any domain such as time, space, and matter.
- *Domain Ontologies*; concepts that can be found within a particular domain.

- *Application Ontologies*; concepts that are contingent on both a particular task as well as a specific domain.

By separating top level ontologies and domain specific ontologies there is an increase in reusability as the same top level ontology concepts can be associated to any number of domain specific ontologies. In many context-aware systems, top level ontologies are comprised of core contextual concepts such as location, person, or device that can be associated to any domain specific ontology. This can be seen in the comparison of Figure 2.7 and Figure 2.8. Figure 2.7 depicts a sample ontology of a real estate domain which consists of two type of users; realtors and guests, two location where either person can be located; indoor or outdoor, along with what the realtor is currently doing; in this case selling a house.

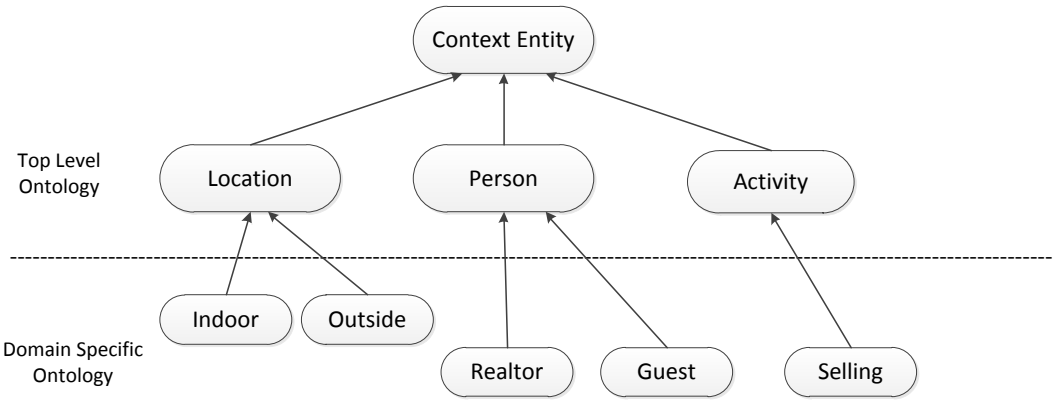

Figure 2.7 - Example of a real estate ontology that can determine where a specific person is located

Figure 2.8 uses the same top level ontology but with a different domain specific ontology, in this case an airport tracking ontology. The ontology presented in Figure 2.8 describes a scenario where an employee or traveler is located in the plane or a terminal while in the process of waiting for their flight or currently flying on their desired flight.

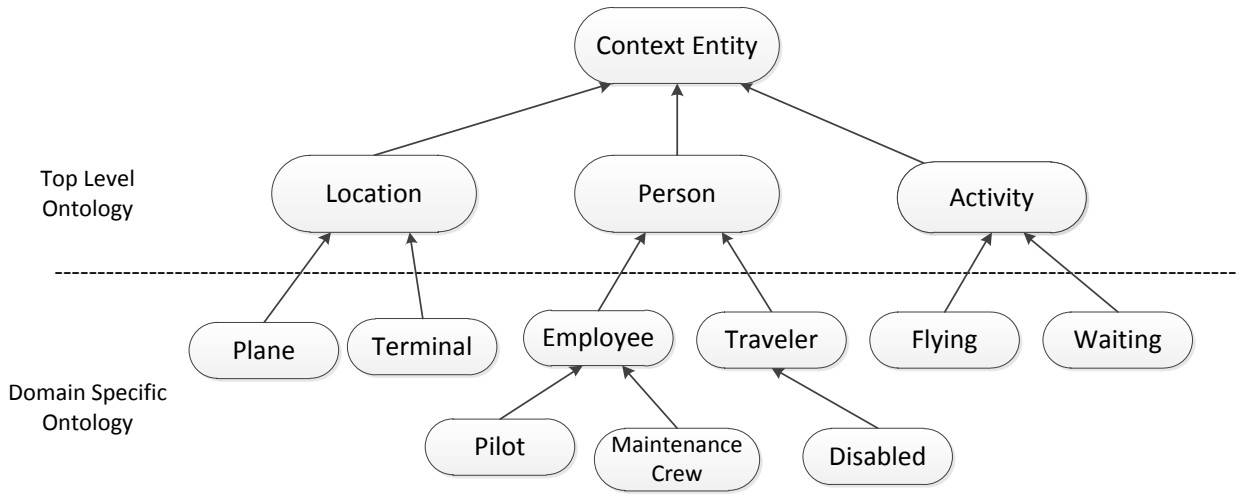

Figure 2.8 - Example of an airport ontology that can determine where a specific person is located

By separating the top level ontology from the domain specific ontologies, it allows various unique systems to use the same defined contextual information throughout various domains. An example of this can be seen in Figure 2.7 and Figure 2.8 where the location context can be derived differently, such as outdoor/indoor or plane/terminal, based on the domain that is associated to it. This is an advantage to an application developer because they do not need to worry about which information is used to form the location context, only that the location context should be implemented.

Application ontologies, also, add the ability to define when a particular task or scenario has occurred such as the contexts that surround a navigational application in a public transportation domain. One of the benefits of using an ontology based modeling schema is that knowledge representation languages such as the Web Ontology Language (OWL) and Resource Description Framework (RDF) can provide context interpretability to context-aware systems. Another benefit is the ability to express and process, through OWL and RDF, a complete knowledge-base about the contexts in the domain, their characteristics, as well their relationships [43].

An example of an *Ontology-Based Model* developed using a top level ontology with domain specific ontologies is CONtext ONtology (CONON) [44]. CONON was comprised of a top level ontology that consists of the ontology concepts location, user, activity, and computational entity, which can be used for retrieving contextual information. These concepts, according to Wang et al. [44], are the fundamental contexts needed for capturing data about a particular situation. The top level ontology concepts; location, user, activity, and computational entity can, then, be used to connect to any of the domain specific concepts in order to create more abstracted contextual information. Other well-known Ontology-Based Context Models include COntext BRoker Architecture (COBRA) [45] and the Service-Oriented Context-Aware Middleware (SOCAM) [46].

2.5 Context Distribution

The final principle of context-aware systems is the way contextual information from sensors is distributed over an environment in order to allow the contextual information to be available throughout the context-aware system. Distribution systems need to have the ability to resolve any issues that might arise when devices fail to retrieve sensor information. For example, if one particular sensor becomes unavailable, the system should still be able to retrieve the last known contextual information from that sensor or obtain the same sensor information from another sensor available within the environment. There are two main solution categories proposed in the literature that are used to handle context distribution; through a central server or through a peer-to-peer environment [47].

According to Kirsch-Pinheiro et al. [47] using a centralized approach for retrieving contextual information requires the need of a central entity that manages the contextual information available in a system. The central entity needs to be able to handle any requests that applications

make in order to retrieve specific contextual information. An example of a centralized approach can be seen in the Pervasive, Autonomic, Context-aware Environments (PACE) project [48]. In PACE, contextual information is stored and managed through the use of distributed context repositories [40].

The benefit of such an approach is that individual context-aware applications are not directly linked to a single context repository but can rather discover them through their catalog names. An advantage of this approach is seen when a context repository becomes unavailable, in such a case applications need to have the ability to discover other repositories with different catalog names in order to get the desired contextual information. Like many server based topologies, one of the drawbacks to a centralized approach is their susceptibility to malicious attacks on the central server which can have an effect on the robustness of the system [49]. If the central server was to loose connectivity or become unavailable then the applications in all likelihood would not be able function properly when an application component needs to be reconfigured.

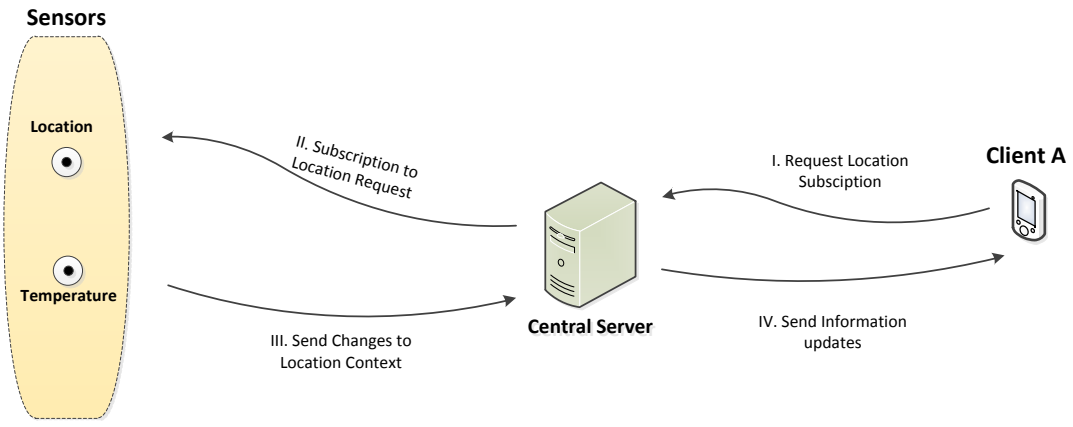

Figure 2.9 - Distribution through a centralized server approach

Figure 2.9 depicts the process of a client, in this case Client A, subscribing to a location sensor that is accessed only by the central server. In such an approach the central server handles all of

the subscriptions made by the various clients and notifies them when a change in context occurs. Once the subscription is made any changes to the data retrieved from the sensor will be routed to all subscripted clients by the central server.

Unlike centralized approaches, the peer-to-peer model does not store context information in repositories or on servers; instead it is distributed throughout the devices found within the system environment. Although the cost of communication is higher than that of a centralized approach, robustness of the system is improved as a single failure in the network will not disrupt the context retrieval of all other nodes. Without the need to centralize information, adopting peer-to-peer architecture appears to also be a good technique for mobile context-aware systems as they are highly dynamic in nature [47]. An example of a peer-to-peer approach to context distribution can be seen in the Peer-to-peer Context Sharing Model (PCSM) [49]. In PCSM, context information is storing locally on each user's mobile device for the purpose of context sharing and fusion. Each device, within the domain, contains a *Registration Center* which stores all the context references that are used to route context information to other nodes. Once a new device enters the system it is able to retrieve all of the references to existing context information by sending *Context Sharing Messages (CSM)*. CSMs are used as broadcasting messages that notify other devices that a new device has entered the system and that contextual information needs to be delivered to it. As more contextual information becomes available, it is routed to all the other devices within the system. Figure 2.10 demonstrates the process that peer-to-peer connected nodes perform in order to successfully subscribe to sensors already accessed by other nodes in the system. In Figure 2.10, Node B subscribes to sensors accessed by Nodes A and C in order to receive updated location and temperature data from the environment sensors. The subscriptions in this model are processed on the device level rather than the server.

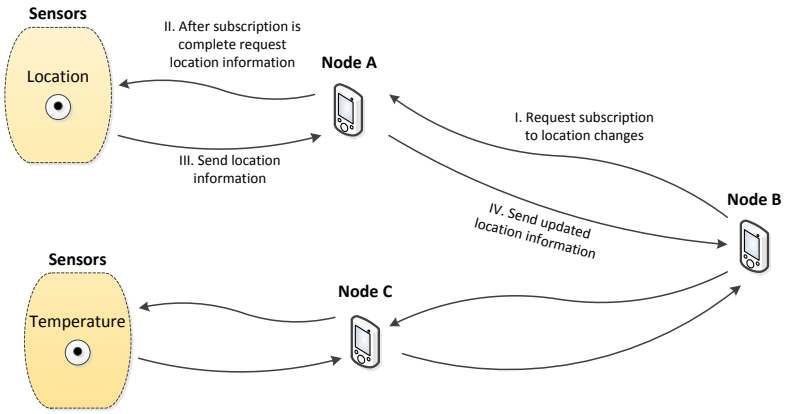

Figure 2.10 - Example process of steps for peer-to-peer context distribution

One of the problems that arise from peer-to-peer context distribution is the amount of communication that needs to occur between system nodes. In order to reduce the overhead, several context grouping techniques [47] and [50] have been proposed. Context grouping allows nodes, such as mobile devices, with common contextual characteristics to be separated into distinct groups [50]. Examples of such characteristics and the explanation to why grouping can occur can be seen in Table 2.4. The process of sharing contextual information is made easier through context grouping as all common information can be found within a specific group of nodes rather than sending request messages to all the devices within the network. This is especially true in domains that consist of thousands of users such as public transportation.

Table 2.4 - Examples of common characteristics for context grouping

Characteristics	Description
Location	Group all devices that are located in the same proximity.
Profession	Group all devices whose users have the same profession such as an accountant or maintenance crew.
Interest	Group all devices whose users have similar interests such as playing basketball or stamp collecting.

An example of context grouping can be seen in Figure 2.11 where information about a user's travel plans can be grouped based on their preferred form of travel as well as their method of transportation. In Figure 2.11, based on a user's travel "interest", in this case "airplane" and "Florida", a system can group users' mobile devices that have the same interest in order to share other common contextual information. This creates a much more cost efficient solution to peer-to-peer distribution as context discovery is limited to other devices with the same interest.

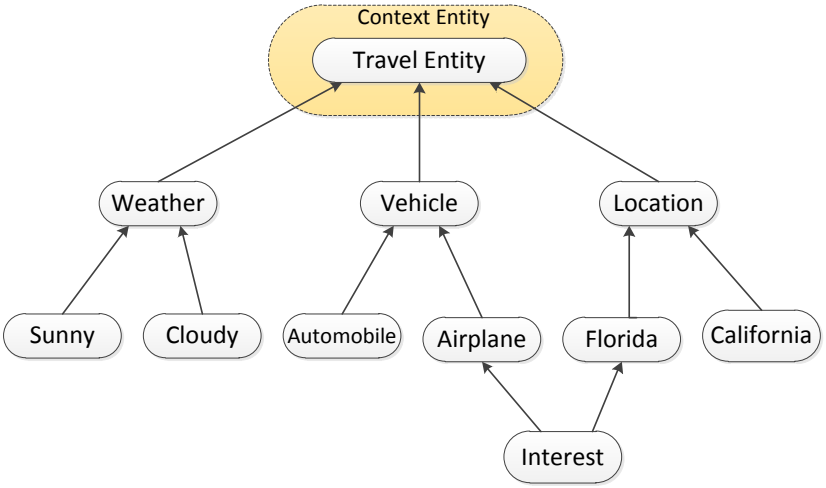

Figure 2.11 - Example of a travel ontology for context grouping through the *Interest* concepts

One of the functional requirements of any context distribution system is the ability to share context information between the devices found within the system. Context sharing, especially in mobile environments, enhances both node communication and social interaction between the system users [51]. It allows system applications to minimize the direct communication with sensors and therefore decreases their dependencies. If one application or device has the ability to retrieve context data from a sensor then all the other applications and devices should have this ability as well, regardless of whether a particular sensor becomes unavailable. This feature should be present in both centralized and peer-to-peer context distribution solutions.

2.6 Contextual Information and Intelligent Transportation Systems (ITS)

In order to improve the travel needs of passengers, transportation and commercial agencies have attempted to integrate various Information and Communication Technologies (ICT), which resulted in a new computing field known as Intelligent Transportation Systems (ITS). More specifically, ITS according to Shank and Roberts [52], refers to the "application of advanced sensor, computer, electronics, and communication technologies and management strategies in an integrated manner to improve the safety and efficiency of the transportation systems". Although this statement exemplifies a broad definition of ITS, its meaning can be simplified dividing it into numerous subsections that deal with specific needs, they are as follows:

- *Advanced Public Transportation Systems (APTS)*: Refers to the various features of public transportation such as improving the quality of service, operational efficiency, reducing travel times, as well as operational and maintenance costs [53].
- *Advanced Traffic Management Systems (ATMS):* Refers to the gathering and combining real-time data in regards to the flow of vehicle traffic, motorist information, incident managements, etc. [54].
- *Advanced Traveler Information Systems (ATIS):* Refers to the enhancing the "planning, perception, analysis, and decision making" [55] in order to improve travelers' needs.
- *Vehicle-to-Vehicle communication (V2V):* Refers to the communication between vehicles in order to improve safety [56] and make travel more efficient.

According to Foth and Schroeter [57], public transportation systems are highly dynamic environments that are made up of diverse socio-economic passengers that can use public

transportation for extended periods of times on a frequent basis. Furthermore, transportation systems are used every day by passengers who consider it part of their daily lives, this according to the original Weiserian theory [18], make it an ideal circumstance for integrating ubiquitous computing [58]. Since passengers often sit on public transportation vehicles with little interaction, there is a need to integrate context-aware systems and applications that will inform passengers about specific transfer information, nearby retail and commercial opportunities, as well as the ability to communicate with the people around them. With a need to improve such functionalities along with their highly dynamic nature, it is evident that ITS, and more concretely APTSs, make ideal candidates for ubiquitous technology integration. By adapting ubiquitous computing concepts in public transportation, the goal of enhancing user travel becomes tangible. Figure 2.12 demonstrates how ITS is affected by integrating the various context-awareness principles discussed in earlier section of this chapter.

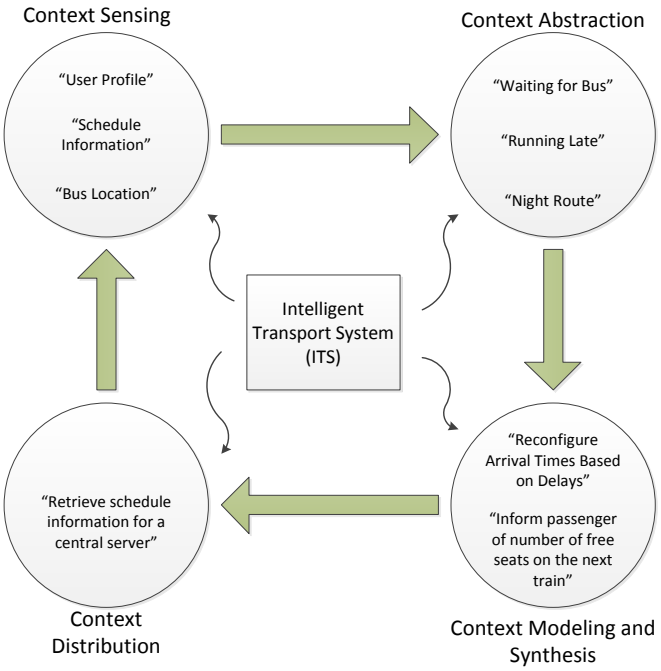

Figure 2.12 - Examples of context aware concepts within ITS

There are many issues and challenges that arise in ITS and in particular the public transportation sector. According to Kjeldshov et al. [59] developers are faced with numerous challenges when designing public transport systems, in particular these systems are both complex and inflexible by nature. They are often complex as they are constructed based on hundreds of routes with various paths of getting from one stop/station to another which can be overwhelming to new passengers. Public transportation systems are also inflexible as vehicle routes rarely change; they follow a specific sequence of stops and therefore might not suit all passengers. Public transportation vehicles are also often delayed by various factors including; traffic congestion, weather conditions, and road construction that makes timing trips difficult and inaccurate. All of the aforementioned items can be considered contextual information in a public transportation system as all of those variables can directly affect the functionality of the vehicles within the domain. Therefore any intelligent system created must be aware of these concerns and adapt accordingly.

As discussed, public transportation systems are a perfect candidate for context-aware integration and when integrated with ITS solutions form what is known as *Ubiquitous Transportation Systems (UTS)*. According to Lee et al. [60] UTS are the "transparent service systems with the omnipresent and transcendent transportation intelligence in all transportation related objects (Smart Objects), which makes the transportation more safe and efficient" and are comprised of four main properties:

- *Anything*; describes any entity found in the system such as vehicles, stop, stations, device, etc.
- *Anybody*; describes any type of person found in the environment such as driver, traveler, pedestrian, etc.

- *Anytime*; describes any time period when travel is happening, about to happen, or has already happened.
- *Anywhere*; describes any location in the transportation environment such on the road, underground, in a station, etc.

In order to implement effective UTS solutions in the transportation sector, information should be provided in real-time through the use of handheld devices that allow passengers to access their desired travel information at any time. With the advent of smart mobile devices and high-bandwidth wireless infrastructure, users can have the ability to access network resources through the Internet and can therefore retrieve real-time information provided by transportation operators. Some of the resources that real-time systems can provide include; schedule timetables, traffic patterns, maps, vehicle wait times, etc. By integrating the UTS with handheld mobile devices the reality of context-aware transport systems becomes more tangible.

One of the original context-aware systems developed to improve user experience in public transportation was known as *Bus Catcher* [61]. *Bus Catcher* was an application designed to provide users with individual traveler information based on a specific traveler's need. The goal of *Bus Catcher* was to provide users with the ability to view specific bus locations in the user's proximity as well as provide accurate updated timetable information based on the time of day it is and the varying weather conditions. *Bus Catcher* was designed as a thin client that was developed on the Compaq Ipaq Personal Digital Assistant (PDA) that would connect to a central server to retrieve the appropriate information. In order to maximize the efficiency of the application most the processing, such as updating timetables, getting weather conditions, and nearby vehicle information, was performed on the central server. One of the original advantages of the *Bus Catcher* system was that it was developed to process real-time information so that

schedule timetables and specific route information were updated as soon as the changes were detected. The biggest drawback to this system was that it did not take the individual user's profile into account as personal user traits such as a possible disability or basic user preferences were not considered. Missing some of the personal features could have resulted in the alienation of a section of the public transportation community that might require assistance most and therefore suggests that *Bus Catcher* was not as flexible and convenient a solution for all the passengers.

Another solution designed to improve public transportation systems was the *ScudWare* project [62]. *ScudWare* allows users of any Smart Vehicle Space (SVS) to use their handheld mobile device to activate and use services defined within the computational system. *ScudWare* does not directly indicate which services are available in SVSs but indicates that it is up to the system creators to create their desired services. In *ScudWare* the SVS is comprised of four main components, they are as follows:

- *Auto-Appreciating Context System*; Senses the data from the sensors in the system such as cameras, sound receivers, and other wireless sensors. This functionality is very similar to context and mobile sensing described in section 2.2.
- *Auto Controlling System*; Controls the components of the system that deal with communication, navigation, vehicle control, and security.
- *Smart Reasoning System*; Uses the context data and ontology-based modeling in order to make appropriate decisions and adaptations. This is the context modeling aspect, discussed in section 2.4, of the system.

- *Centralized Processing System*; Handles the communication between system components.

Although *ScudWare* was originally designed to function in an automobile setting rather than a public transportation one, much of the same principles apply to both such as an environment that is dynamic changing on a constant basis. Using the *Ontology-Based Modeling* technique, *ScudWare* defines three main context categories:

- *Vehicle Context*; status of the vehicle elements such as temperature, speed, lighting, etc.
- *Environment Context*; status of environment elements such as the weather conditions, and traffic congestion.
- *Driver Context*; driver attributes such as tiredness, ability to drive, and alcohol density.

Using these contexts *ScudWare*, uses an ontology in order to define and process the various context variables discussed above. *Scudware* defines context through the following format; C=(S, P, O) where S demotes the context is question, P denotes the attribute in question, and O denotes the value of the attribute. An example of defining a driver who has a blood alcohol level of over 0.08 percent and therefore is perceived to be drunk is represented by Expression 2.3.

$$(\text{Driver}, \text{Property}, \text{Alcohol}) \wedge (\text{Alcohol} >= 0.08\%) \rightarrow (\text{Driver}, \text{Status}, \text{Drunk})$$

Expression 2.3 - Example of context definition using ScudWare [62]

Both *Bus Catcher* and *ScudWare* provide examples of how context-awareness can be designed for public transportation system. Both techniques, however, lack the ability to fully characterize a transportation setting as they are designed for specific purposes and scenarios and limit

application developers from using the same system to create various different scenarios and applications within the public transportation domain.

Chapter 3. Methodology

In order to successfully implement the context-aware principles discussed in Chapter 2 into a dynamic context environment such as public transportation, a comprehensive system that allows for rapid system reconfiguration and adaptation needs to be created. This chapter examines the requirements needed for creating such a system, describes a framework that meets these requirements, and introduces a domain ontology that enhances the integration of context-aware principles into a dynamic settings such as the public transportation. Using this domain ontology, application developers will have a better understanding of the public transportation domain and therefore be able to decrease the amount of time needed to create context-aware mobile applications.

3.1 Requirements of Context-Aware Systems

When deciding which context framework is most suitable for a rapidly changing domain there are three main features/properties that need to be examined. The first feature examines a system's ability to obtain relevant contextual information within a particular setting and processes that information to determine when and how component reconfiguration should occur. This feature, according to Ruiz and Sanchez [63], is classified under the context sensing and synthesis principle discussed in section 2.4. Under this principle a framework must be able to include various reusable sensors that can be used by a multitude of applications, model when system/component reconfigure should take place using at least one of the modeling techniques described in section 2.4, and allow for dynamic activation and deactivation of sensors depending on the needs of the application [63].

Secondly, context-aware systems should have the ability to automatically make decisions in a timely manner regardless of the system's size [63]. This feature, also known as multi-dimensional decision making, enhances context-awareness by allowing any device registered to the system to reconfigure based on the changing contextual information retrieved from sensors within the domain. Multi-dimensional decision making is aided by the abstraction of retrieved sensor data into information that the system can process and use to determine which variant of a component needs to be reconfigured. As a result of this feature, a prerequisite of context-aware systems is that they must be able to register/unregister various components such as sensors and devices at any given time depending on the contextual needs of the system [63]. As not all contextual changes result in component reconfiguration, there is, also, a need for context-aware systems to register and unregister to individual components rather than the system as a whole.

The third major feature of a context framework is that there needs to be an emphasis on the reusability of system components in order to allow numerous domain-based applications to be developed rapidly using the framework [64]. This indicates that developers should not be recreating data retrieval methods for sensors found in every domain but rather reuse prebuilt context sensor widgets, described in section 2.3, that retrieve and abstract data directly from sensors. To reduce domain dependencies, context sensor widgets should be implemented independently of each other, therefore allowing devices to register to each widget based on the needs of each application.

Considering these features and requirements, a middleware solution known as MUSIC [16] (self-adapting applications for **M**obile **US**ers **I**n ubiquitous **C**omputing environments) was chosen as the basis for implementing the context-aware principles into the public transportation domain. This middleware was chosen, in part, due to the separation of the fundamental context-aware

principles that include sensing, abstraction, reconfiguration, and distribution from the application logic itself [65]. By separating these principles each component is implemented individually and is therefore decoupled from all other principles thereby enhancing a system's reusability. Additionally, each context-aware principle was implemented modularly resulting in further separation, allowing middleware developers to continuously implement new components such as context sensor widgets without having to directly affect any of the other components within the system. By separating all the system in this manner, application developers can register their applications to the pre-built components without needing to create their own.

The second major reason this framework was chosen was its ability to automate the multi-dimensional decision making process. To accomplish this, the framework was designed using the Model-Driven Development (MDD) approach in order to organize and manage developer created domain models that define when and how reconfiguration can occur [66]. The middleware uses a combination of user-defined ontologies and UML diagrams to model the appropriate domain and its reconfiguration properties. The MDD approach, also, improves flexibility as it allows developers to decide which modelling technique, such as OWL based ontologies and UML diagrams they prefer using to define under which specific conditions system reconfiguration should occur.

3.2 Ontology Structure

In order to enhance the scalability and adaptability of ADOPT, the system ontology is designed as a three-level hierarchy consisting of a top-level ontology, domain-specific ontologies, and application ontologies [65]. This three-tiered model can be observed in Figure 3.1.

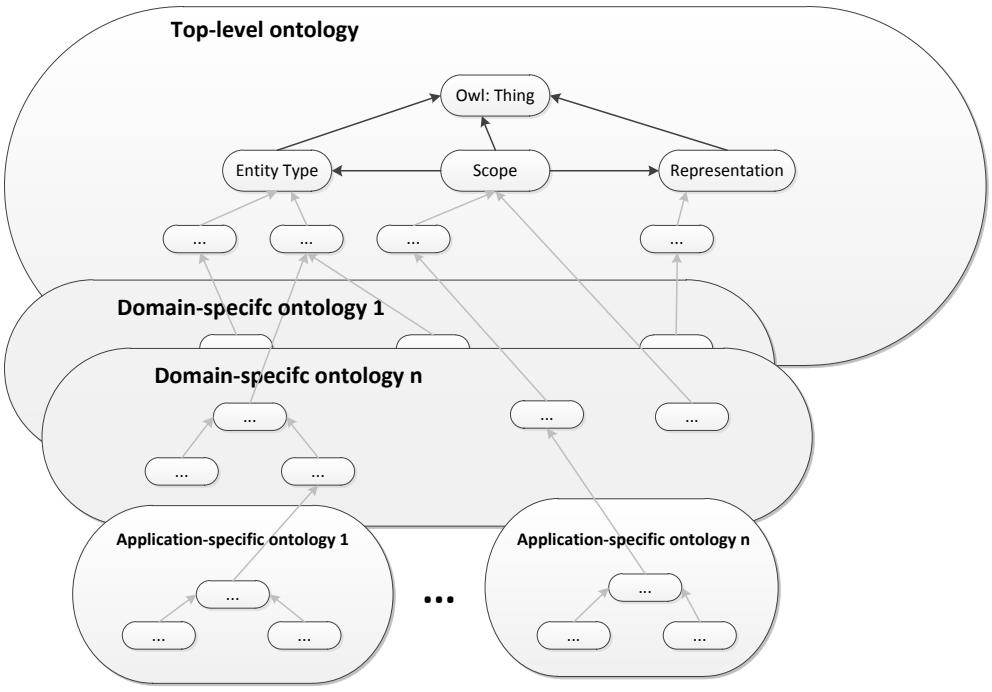

Figure 3.1 - Architecture of the system ontologies [65]

The top-level ontology, which will be further discussed in section 3.3, is comprised of universal concepts that can be used to create contextual information within any domain. The domain-specific ontologies, as shown in Figure 3.1, are developed as extensible parts of the top-level ontology and are used to provide a set of concepts that can be found within a specific domain that represents a part of the world [65]. Once a domain-specific ontology is developed, the relationships between itself and the various concepts of the top-level ontology need to be established. Through these relationships, the creation of contextual information is made possible; this will be further examined in section 3.4.2. The application ontology layer, much like the domain-specific ontology layer, is also built as an extensible part of the overall ontology and defines the concepts relevant to a particular application or scenario [65]. This ontological structure allows for an arbitrary number of domain ontologies to be built using the same top-level concepts. By creating a three-tiered hieratical ontology system, the system attempts to reduce the amount of overhead that developers need to perform when developing their context-

aware applications [65]. Therefore, application developers can reduce the amount of research needed to understand the public transportation domain as most the relevant concepts and relationships are defined within the ADOPT domain ontology.

The advantage of the three-tiered framework is that each tier can be developed independently of the others as each tier has a unique functionality. By creating a model that consists of unique independently functioning systems, the framework discussed in this chapter is considered a System of Systems (SoS) [67]. The SoS architectural approach allows application developers to use ADOPT to incorporate independent application ontologies using the concepts defined in the domain-specific ontology. This feature allows the proposed ADOPT system can be reused by any application developer who intends on creating functional public transportation applications. Therefore, the proposed ADOPT framework discussed in this chapter is built using a top-to-bottom architecture with the top-level ontology consisting of basic elements that can be applied in any domain. The domain-specific ontologies are then built as sub-systems of the top-level ontology, with application-based ontologies built as further sub-systems of each domain-specific ontology. This structure allows application developers to incorporate specific application ontologies into the framework in order to reuse the already created software components such as context widgets that inform applications of changing sensor information [68].

3.3 Top-Level Ontology

As the framework supports a wide variety of applications designed for various domains, there is a need to establish reusable components that can be accessed across any domain [65]. To satisfy this need, a top-level ontology was used to define the reusable components found within all domains. According to Reichle [65], the top-level ontology designed in this framework was motivated by the Aspect Scale Context (ASC) model described by Strang et al. [43]. The ASC

model is comprised of *aspects* which consist of information used to characterize specific data within a domain and *scales* that are used to describe the format that an *aspect* can be represented in. By combining *aspects* and *scales*, the ASC model is able to provide contextual information to applications of a particular domain [65]. One example of contextual information that can be categorized within the ASC model is location; where *aspects* are defined as geographic coordinates used to describe the location of a particular item and *scales* are represented through latitude and longitude measures used to describe an item's geographic location coordinates.

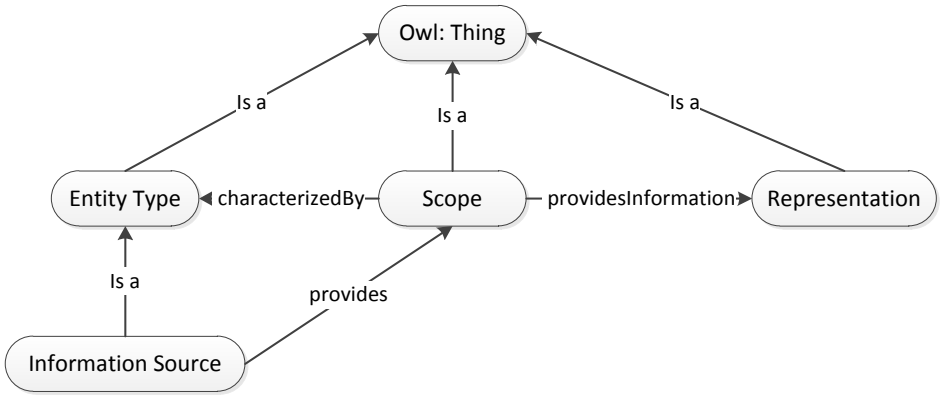

Figure 3.2 - Top-level ontology concepts [65]

Using the ASC model as the basis for the top-level ontology; the top-level ontology is comprised of three core concepts; *entity types*, *scopes*, and *representations* [65], that are used to define all context information found within a domain. The ontology representation of the concepts can be seen in Figure 3.2.

The entity type concepts in the top-level ontology represent all of the real and/or logical entities (objects) found within a particular domain. They are comprised of information sources such as the physical, virtual, and logical sensors discussed in section 2.2, from which raw sensor information can be obtained from. An example of an entity type is a person that either affects or is affected by any changes to sensor data within a domain, such as a *student*, or *professor* that

can be found within a *university* domain. These entities can also be grouped together based on similar attributes, suggesting that the concepts of *student* and *professor* can be grouped together under a concept *person*, as seen in Figure 3.3. When creating a domain-specific ontology that uses this top-level ontology approach, it is important to include all relevant entity types as they will later (section 3.4.2) be used to define the contextual information found within the targeted domain.

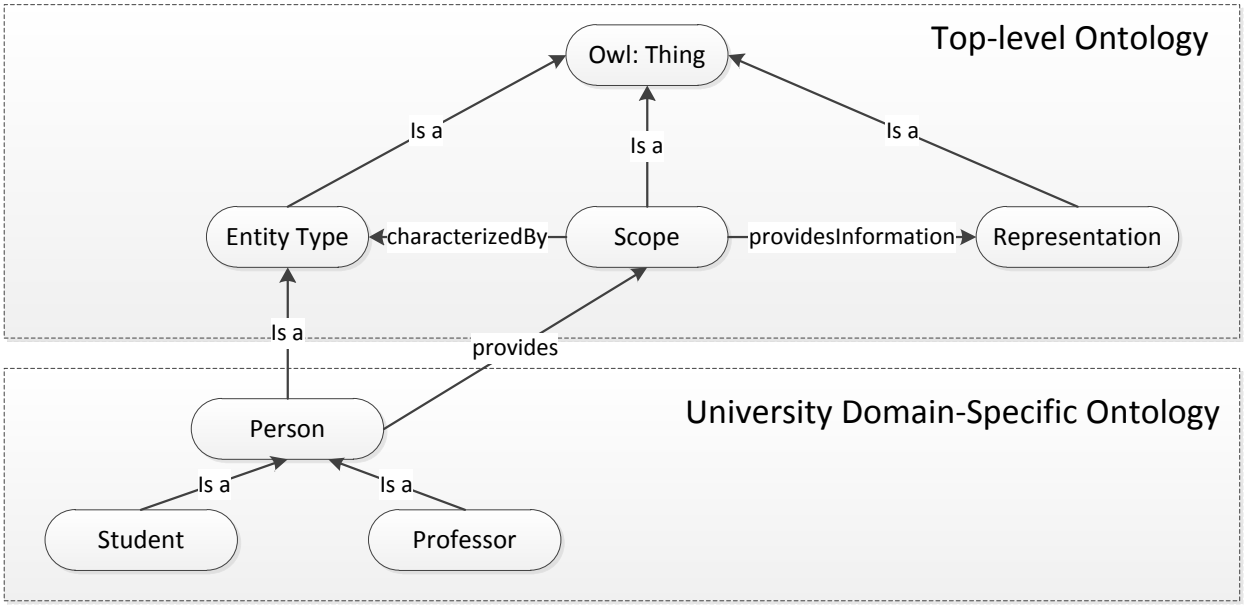

Figure 3.3 - Example of incorporating a person entity type within the top-level ontology

The *scope* concept is used to define the functional context variables that can be obtained from the set of defined entity types [65]. Each scope defines the information needed to determine context information that is associated to at least one entity type. For example, the *location* scope can be used to define the location of a *person* entity type, which can provide information about where a professor is currently lecturing. Using the top-level ontology, an unlimited number of scopes can be provided based on the defined set of entity types [65]. The information provided by a scope is structured using the *representation* concept of the top-level ontology, which defines the internal

structure or format of the information provided by the scope [65]. An individual scope can be represented by any number of representations needed for that particular domain. For example, the location of a person can be obtained through the *location* scope that can use numerous representations such as geo-coordinates, street addresses, postal codes, cities, etc. The representation *LocationGeo* can be used to define the structure of the *location* scope by defining the latitude and longitude locations of a physical object, meanwhile a *LocationAddress* representation can define the location scope by combining strings such as the street name, unit number, city, and country. This example is demonstrated in Figure 3.4.

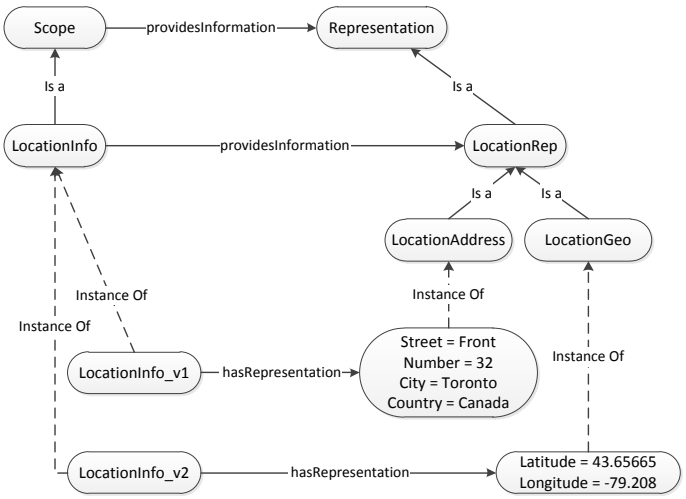

Figure 3.4 - Example of multiple representations for location scope [65]

The representation concept can be subclassed into two main categories; *composite* representations and *basic* representations. Basic representations are used to define a single piece of information that can be obtained from an individual sensor. For example, temperature information (scope) can be classified under a basic representation as it can be obtained from a single temperature sensor and can be categorized by individual measures such as Celsius or Fahrenheit. Since the Celsius representation yields a different measure than that of a Fahrenheit representation, two separate basic representations should be created for the *temperature* scope.

By creating different representations for an individual scope it allows application developers to choose which measure or representation they are more comfortable developing with. Composite representations, on the other hand, are formed using multiple representations associated to multiple scopes. For example, a weather scope cannot be defined through a single scope as weather is often affected by multiple factors. Therefore, the weather scope might provide information that combines the scopes for temperature, humidity, precipitation, air pressure, etc. which consist of their own basic representations. The weather composite representation, therefore, can combine these and other basic representations in order to structure the data in a particular format that the system is able to understand and process for component reconfiguration.

3.4 Domain-Specific Ontology Designed For Public Transportation

The domain-specific ontology discussed in this section will provide developers with the fundamental concepts and properties needed to create specific applications designed for the public transportation domain. This section informs application developers of which entity types may be incorporated into their public transportation applications. Using the entity types described in this section, the process of establishing relevant contextual information, described through scopes, will be defined. In particular, understanding why some scopes were included into the domain-specific ontology while others were omitted is an essential concept to understanding how a public transportation domain-specific ontology functions. In addition, some proposed advantages of ADOPT model are examined when compared to linear key-value models. The linear key-value model was chosen for comparison as it provided a good initial barometer for determining if the ADOPT framework provides an improvement for software reconfiguration. If the ADOPT framework provides an improvement over the simplistic key-

value model it would provide a good stepping stone to move on to more complex modeling techniques.

The proposed public transportation ontology allows application developers to create fully functioning applications that can be designed to provide static information or more complex user personalized applications that reconfigure based on the needs of thousands of users. However, the goal of ADOPT is primarily not to create a system that will contain all the information needed to create all possible public transportation applications, but rather to provide application developers with the necessary tools and requirements needed to develop pluggable context-aware application ontologies designed for the public transportation domain. It is the responsibility of each application developer to create their own application-specific ontology using the ADOPT ontology, which will result in a more complete context-aware system. This section is divided into three main subsections; entity types, scopes, and representations that detail the connection of the context-aware principles within the ADOPT framework.

3.4.1 Entity Types

The first step to creating functional entity types found within a public transportation domain was to determine which users and objects directly affect the functionality of a public transportation system. This step allows application developers to gain a better understanding of their desired domain and therefore understand which application users and objects are directly affected by changes to contextual information. If application developers fully understand their domain then they will be able to decide which sensor data their application needs to obtain and how each entity is affected by the changing sensor data. Therefore, creating a concrete set of entity types was the initial task of developing the fundamental aspects of a context-aware public transportation system.

The initial set of entity types in the proposed ADOPT ontology was quite extensive with over seventy unique concepts being defined. With this large set of concepts subclassed directly from the entity type concept, the readability of the system was jeopardized as the sheer number of concepts can be overwhelming especially when related concepts are not grouped together. In order to improve this, five categories of entity types were implemented into the proposed public transportation ontology. The five categories, also seen in Figure 3.5, are as follows; *person* [69] [70], *place* [71] [1], *vehicle* [70], *device* [71], and *resource*. The goal of these categories was to establish all of the logical and real-world objects that can be used to construct contextual information within the proposed public transportation domain.

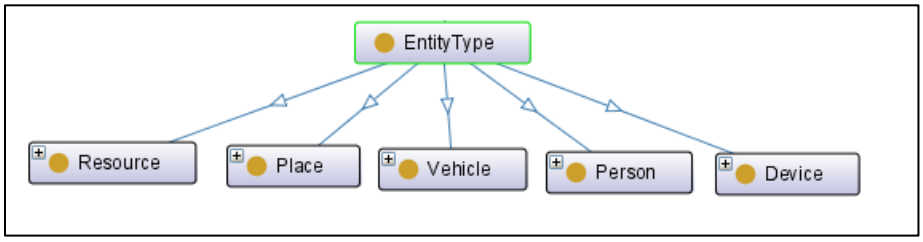

Figure 3.5 - Public transportation domain entity type subclasses

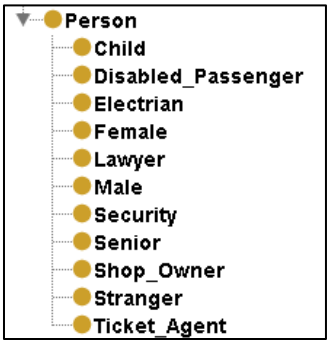

Figure 3.6 - Sample list of person entities before filtering out occupation-based persons

The first step to maturing the entity type concept was to introduce all of the various types of people found within the public transportation domain and include them within the *person* entity type. A sample list of the initial person entities can be seen in Figure 3.6. Once the complete list

of *person* entities was established it was important to discard any *occupation-based persons* from the initial result set. Occupation-based persons describe people within the domain that are associated with a particular occupation title and are only affected by contextual changes on an application or scenario basis. Occupation-based persons are people who are not found in the majority of applications designed for the public transportation domain. Removing occupation-based persons from the ADOPT ontology reduces the overall size of the ontology and therefore improves its readability without losing any functionality. In ADOPT, occupation-based person entities should be migrated to the application ontology layer as their needs are specific to particular scenarios and applications. For example, one such occupation-based person, as seen in Figure 3.6, is the *lawyer person* that describes a passenger who travels using public transportation and has an occupation of a lawyer. If the *lawyer person* was added to the domain ontology then every application built using that domain ontology would include the *lawyer person* entity. As there is only a limited amount of scenarios where a lawyer passenger entity is needed, it would be more efficient for the lawyer entity to be implemented directly within the application ontology.

Using the ADOPT model, application developers can dynamically add additional application ontologies to the domain ontology based on new application requirements they might have. Linear key-value models, on the other hand, do not allow for the addition of new application ontologies once the original model has been constructed because for each new application a new rule database must be created. Therefore, an application developer would need to construct new if-else statements to determine when system reconfiguration should take place resulting in a lack of reusable components for applications with similar capabilities.

The proposed ADOPT solution also provided developer with the ability to group entities with similar attributes under a single individual entity. By grouping entities, ADOPT further eliminates occupation-based persons by grouping related occupations under a single more generic entity type. This feature is evident in the example of the *maintenance person* entity. Instead of adding *person* entities such as an *electrician*, a *network administrator*, or a *cleaner* to the domain ontology these entities can be grouped under the more generic entity *maintenance*, as seen in Figure 3.7. If an application developer creates a specific application or scenario for one of these *maintenance* entities, such as an *electrician*, they can simply subclass the *maintenance* entity into their own application ontology and implement their own *electrician* entity.

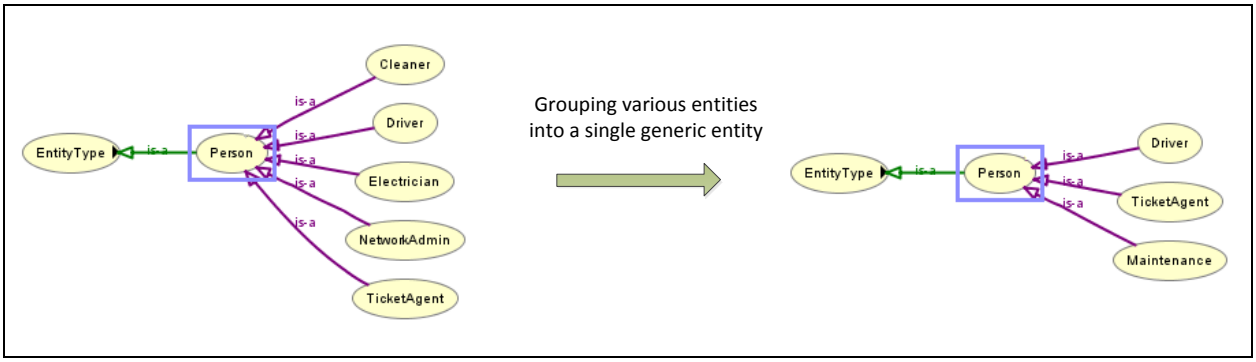

Figure 3.7 - Sample of grouping maintenance employee entities

Furthermore, any *domain essential entities* need to be incorporated into the domain ontology. Domain essential entities are entity types within a domain that directly affect the status of other entities. For example, the maintenance entity directly affects the status of vehicles, elevators, escalators, etc. as a malfunction or breakdown of such elements will directly result in changing information received from sensors that can affect a subset of people using the public transportation domain. For this reason, creating a *maintenance* entity within the *person* entity optimizes ADOPT by providing more relevant entities that an application developer could

incorporate, customize and/or reuse rather than manually implementing them into their own ontology. Domain essential entities differ from occupation-based entities as they include only the entities that can directly cause contextual changes to other entities. For example a *lawyer* passenger occupation-based entity does not directly affect the functionality of public transportation system, while a maintenance crew member can directly affect the public transportation system as the breakdown of vehicles can cause changes to contexts that affects a subset of system users. The rest of the *person* entity type in ADOPT is subcategorized into the following entities:

- *Passenger*; reflects all the different types of passengers found in the public transportation domain regardless of their occupation.
- *Maintenance*; reflects the various types of maintenance crew employees available.
- *Driver*; describes all the employee drivers, regardless of which transport they operate.
- *TicketAgent*; describes the people that collect and/or sell tickets/passes within the domain.

Although this can be considered a generic enough solution for the *person* entity, the *passenger* entity can still be further subcategorized in order to give application developers more granularities on the various types of passengers they can incorporate into their applications. Once again this needs to be done in a generic way that allows application developers to further subclass the passengers if needed. Therefore, the *passenger* entity is subclassed into the following entities; *general*, *senior*, *child*, and *disabled*. Each defined *passenger* subclass has a specific set of characteristics, attributes, and needs that distinguish them from the others as the needs of *disabled* passengers are very much different than those of a *general* passenger. These

needs should to be implemented by the application developers for each transportation system which will result in different reconfigurable components based on what that passenger's needs are. This is another important advantage of the ADOPT model compared to linear key-value models, as it allows for the customization of entities based on the needs of the application. If an application developer desires to implement their own *passenger* entity to include persons such as a *bicyclist*, *lawyer*, and/or *tourist* they can do so by extending the domain-specific ontology and incorporating the new passenger entities into their own application ontology. An example of incorporating application-specific *passenger* entities can be seen in Figure 3.8 where the application-based passenger entities; *bicyclist*, *lawyer* and *tourist* were directly subclassed from the domain ontology's *general* passenger entity.

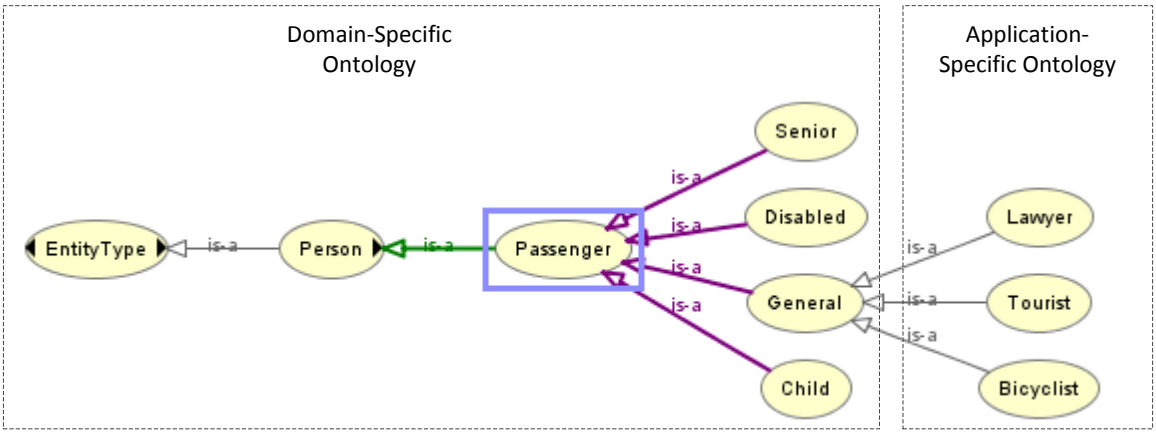

Figure 3.8 - Applying application specific entities to the ADOPT ontology

Using the proposed ADOPT ontology, developers can also distinguish between *general* lawyers and those who are considered *disabled* and therefore require special needs within the public transportation system; such as the locations of elevators within a station, or to inform them of any accessibility limitations that approaching vehicles might consist of. The complete *person* entity in ADOPT can be seen in Figure 3.9.

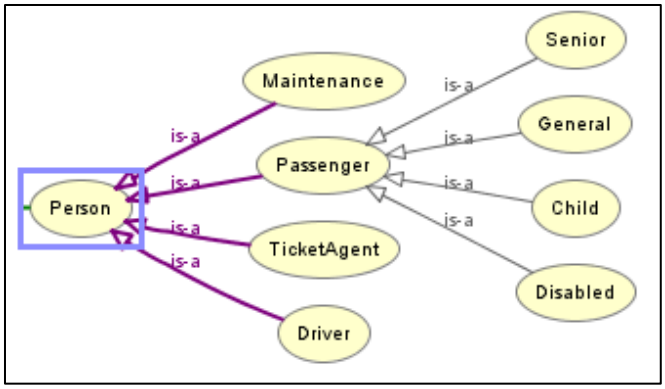

Figure 3.9 - ADOPT person entity type

In order to define all of the physical locations found within the public transportation domain, high-level abstracted data, discussed in section 2.3, was used to produce the *place* entities found within ADOPT. All high-level physical locations found within the *place* entity type such as the entities *station*, *stop*, *ticket booth*, *elevator*, etc. can be obtained by abstracting low-level measures such as geographic coordinates, street addresses, and/or a 3-dimensional Cartesian coordinate system. The *place* entity can further be divided into two subclasses; *indoor* and *outdoor* [1]. The *indoor place* entities describes the physical locations found inside public transportation structures such as stations, while the *outdoor* place entity describes particular locations of the domain found outdoors. Table 3.1 demonstrates the two subclasses of the *place* entity with some examples found within each subclass. The goal of the *place* entity type is to define the various points of interest (POI) within a public transportation domain that people designated within the *person* entity type come across during their travel.

Table 3.1 - Examples of *indoor* and *outdoor place* entities

***Place* Entity Subclass**	**Examples**
Indoor	Shop, Elevator, Escalator, Washroom, etc.
Outdoor	Stop Shelter, Taxi Lot, Parking Lot, etc.

It is important to note that an *indoor place* entity is not necessarily disjoint from an *outdoor place* as places such as stairwells and ramps can be located both indoors as well as outdoors. Although some *place* entities can be considered objects or structures rather than physical locations they can still be described using location variables such as geographic coordinates or a Cartesian coordinate system, therefore they are be subclassed under the *place* entity.

The proposed *vehicle* entity type defines the various vehicles found within the public transportation domain. It includes entities such as a *bus*, *streetcar*, *subway*, *train*, *ferry*, etc. The *vehicle* entity type represents any physical machine that transports a passenger from one location to another based on some form of payment by the user. The payment principle is an important concept in the proposed *vehicle* entity type as concepts such as *escalators* and *elevators* can also be used to move a passenger from one location to another however they are used for very short distances and do not require a passenger to pay a transportation fee. If an application developer desires to implement a system where passengers pay for using elevators and escalators, then they can further subclass the *vehicle* entity within their own application ontology to include the necessary entities.

The fourth entity class of the designed public transportation domain is the *device* entity. The *device* entity type consists of the various devices that a person (found within the *person* entity) can operate in order to get specific information that is relevant to their objectives within the public transportation domain. For example, if there is a delay on a particular route, a passenger who is waiting for a vehicle on that route needs to be informed of the delay. Although this information can be announced, through intercoms, to passengers waiting at various stations, there could exist a subset of smaller based stops along the route that do not have access to this particular type of information unless it is sent directly to a user's handheld mobile device.

Therefore, the goal of the *device* entity is to describe all of the devices that can inform passengers and staff of information that is relevant to their specific tasks during any period of time. These devices include entities such a *phone*, *computer*, *tablet*, and any other portable devices used to execute particular mobile and system applications. These devices are convenient not only to public transportation passengers but to staff such as the maintenance crew who can use PDAs to assist them in repairing malfunctioning escalators or security staff who can use them to locate any distressed passengers. Additional devices that can execute mobile and system applications should be added to the developer's application ontology. A sample subset of the *device* entity can be seen in Figure 3.10.

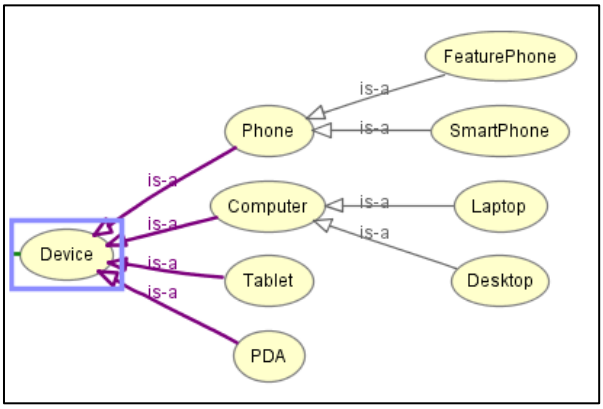

Figure 3.10 - Sample subclasses of Device Entity

The final entity type used in the ADOPT ontology is the *resource* entity. The *resource* entity describes all the usable parts that can be accessed by people within the public transportation domain, such as hardware components found on an external system device, and physical characteristics of various objects found within vehicles and locations described within the *place* entity. The data provided about all of the entities within the resource entity can be obtained through the use of embedded sensors that are found throughout the public transportation domain. Therefore, the *resource* entity is subclassed into the following three main subclasses;

- *Device resource*; any hardware resource that can be obtained from a handheld device.
- *Vehicle resource*; any physical characteristic that is associated to a vehicle within the domain.
- *Place resource*; any physical object that can be found to within a *place* entity.

The *device resource* entity is used to describe all of the resources, characteristics, and features of handheld devices which can be associated to any of the entities found within the *device* entity type discussed earlier in this section. Device resources equate to physical hardware components built into handheld devices such as sensors and manufactured chipset. Some examples of a *device resource* include a device's *battery*, *memory*, *CPU*, *Bluetooth*, etc. which can all be found within existing smart mobile phones, which is represented by the *smartphone device* entity. A sample set of entity types found within the device resource entity can be seen in Figure 3.11.

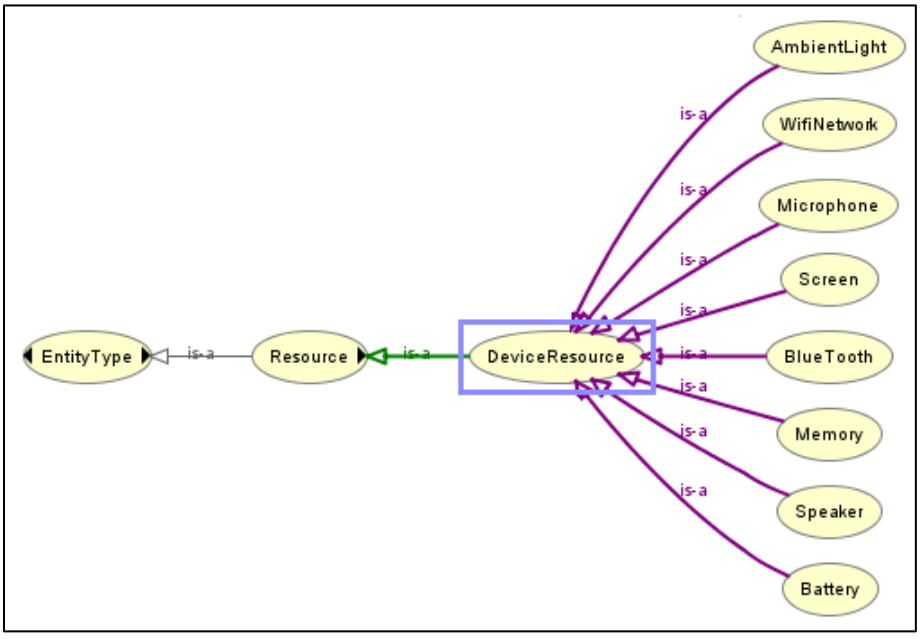

Figure 3.11 - Sample of device resources entity type

The *vehicle resource* entity contains all the resources found physically on public transportation vehicles. Using the sensors that relate to these resources, transportation operators are able to obtain information about the current conditions of objects within a vehicle as well as the vehicle condition itself. Some of the *vehicle* resources include entities such the *Engine* and *GasTank* that directly relates to a vehicle's physical engine and gas tank that may be present on a vehicle. A sample list *vehicle* resources found in the proposed ontology can be seen in Figure 3.12.

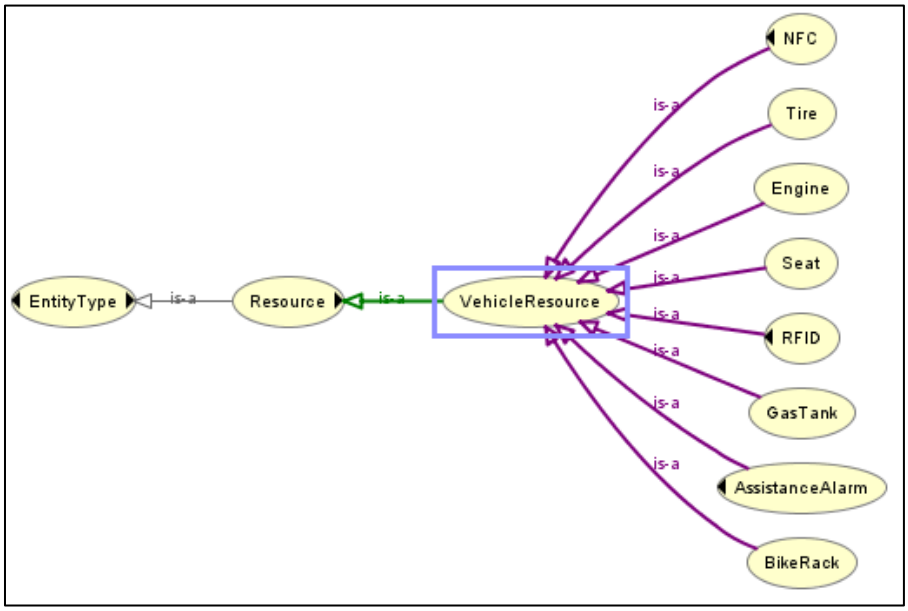

Figure 3.12 - Sample of *vehicle resource* entity type

Finally, the *place resource* describes all the resources found within the public transportation domain that can be observed within a physical locations described by the *place* entity. Each resource in this entity can be directly associated to at least one *place* entity; therefore, *place resource* entities need to be grouped according to their corresponding *place* entities. For example a station's emergency assistance alarms and Near Field Communication (NFC) payment sensors are grouped under the *StationPlaceResource* as these resources can be found within a station's physical location. Therefore, each *place* entity will have a corresponding set of *place resource*

entities grouped together. The goal of this feature is to improve the readability of the *place resource* entity as this concept consists of more than thirty entities. Samples of the *place resource* entity can be seen in Figure 3.13.

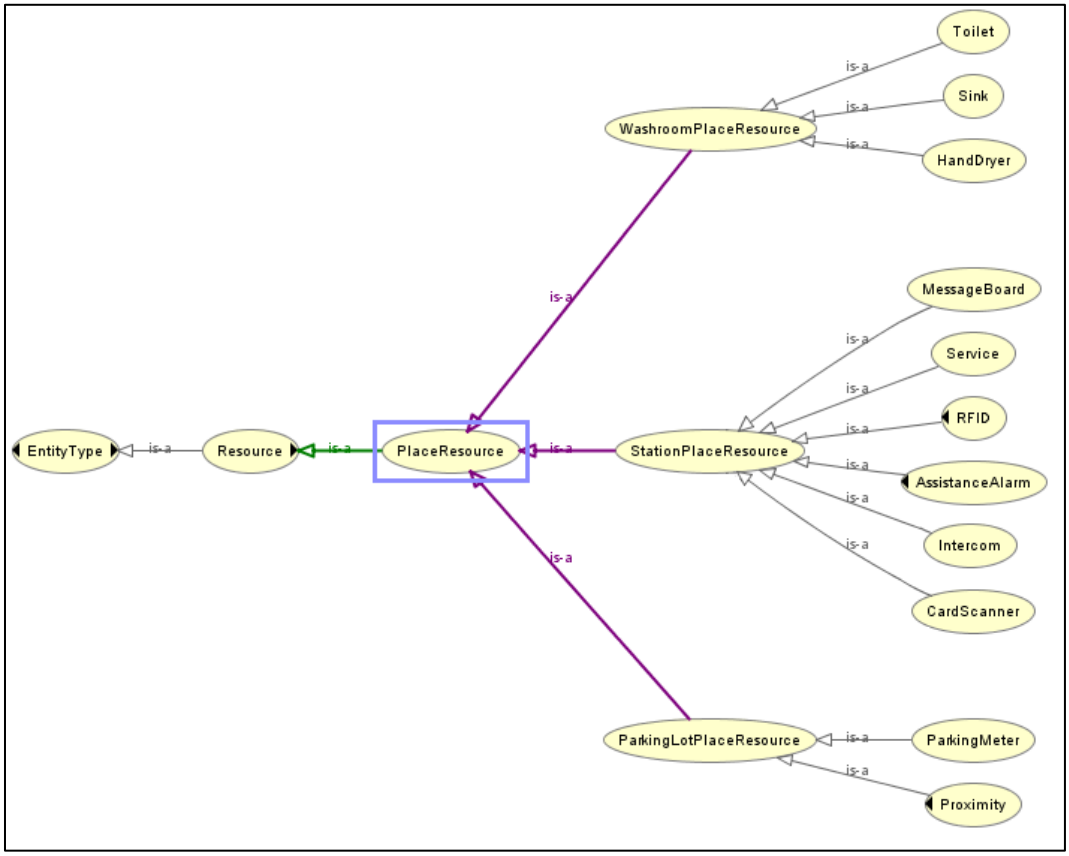

Figure 3.13 - Sample of *place* resources

Similar to the *place* entity, resources are not disjoint, which implies that a resource associated to a *vehicle* resource can also be found within the *place* or *device* resources as well. One example is the *NFC* payment resource, which allows a user's device to transmit payments from their handheld device to transportation operator devices by being in close proximity to each other. The *NFC* payment resource can be located anywhere inside a public transportation domain including a device, a vehicle or place such as a station.

The role of *resource* entity does not, however, directly inform developers which specific contextual information can be obtained from the various resources but rather it inform them of which data, in regards to the objects and people, can provide sensor data used to create contextual information. For example, the *screen device* resource, as seen in Figure 3.11, does not inform developers which specific contextual information it provides but rather it allows the application developers to develop contextual information such as a device's screen resolution which can be obtained from sensors that retrieve data about a *screen device* resource.

3.4.2 Scopes

Once all of the entity types of the public transportation domain had been established, domain-specific contextual information needed to be established. Since the goal of the entity types was to establish all domain specific objects and people that can be described by sensor information, it was critical that scopes be associated directly to at least one entity type. This association is created using the object property *characterizedBy*, as seen earlier in Figure 3.2. If a scope cannot be directly associated to at least one domain-specific entity type than it should be removed from ADOPT ontology.

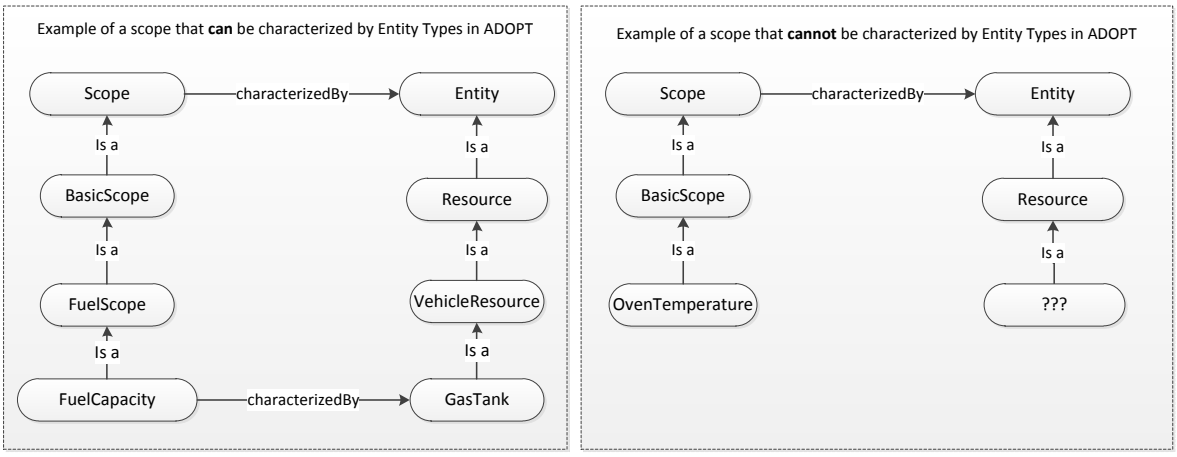

Figure 3.14 - Example of scopes that can/cannot be characterized by entity types within ADOPT

One example of scope that is directly associated to an entity type is the *FuelCapacity* scope which indicates the amount of fuel remaining in a vehicle's gas tank. As Figure 3.14, demonstrates the *FuelCapacity* scope is directly associated to the *GasTank* entity type through the *characterizedBy* object property. Meanwhile, as Figure 3.14 also demonstrates, a scope such as *OvenTemperature* that describes the temperature of a kitchen oven should not be included within the ADOPT ontology as it cannot be associated to any particular entity found within the ADOPT entity types. This approach allows developers to implement only the relevant scopes (context variables) that directly affect people and objects within the public transportation domain. Within ADOPT, scopes can be divided into two subclasses; *basic* and *composite*. Basic scopes are used to characterize the contextual information that can be described through a single measure, such as representing the location of a user through the *city* scope which indicates which city a user is currently located in.

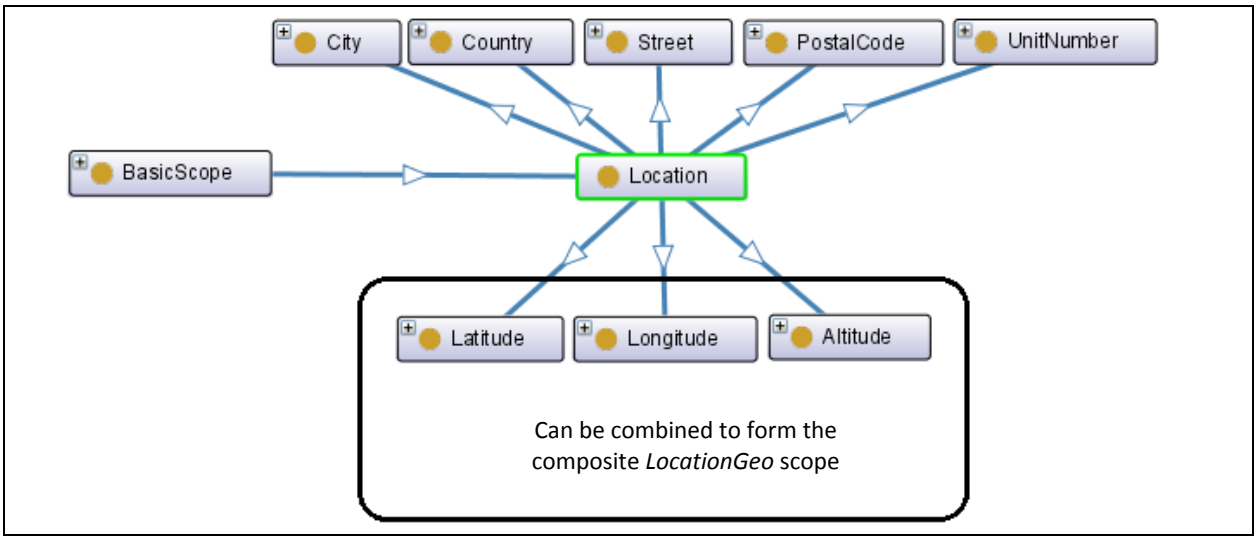

Figure 3.15 - Example of location basic scopes

Basic location scopes, within ADOPT, include concepts such as *street*, *country*, *postalcode*, etc. which are all be grouped under the basic scope *Location* as seen in Figure 3.15. Composite

scopes, on the other hand, combine multiple basic scopes in order to represent specific contextual information such as the geographic location of a user by combing the *latitude* and *longitude* basic scopes into a single *LocationGeo* composite scope. Dividing the *Location* scope into multiple smaller basic scopes, allows application developers to incorporate only the specific location information that is needed within their application. This is an important characteristic of the proposed ontology as not all applications need to know the exact location of a user/object; instead they might only need one or two of the variables seen in Figure 3.15. Another example of contextual information that can be associated to any entity type is the *DateTime* scope which is comprised of basic time elements; *second*, *minute*, *hour*, *day*, *month*, and *year*. The goal of this *basic s*cope is to accurately portray time within the public transportation domain. Using the *DateTime* scope, temporal context information such as when a specific vehicle will be arriving at a passenger's desired location can be obtained. By dividing the *DateTime* scope into various time elements, it allows application developers to use granular time components to display contextual information in units such as seconds or more complexly as a combination of all six elements found within the *DateTime* scope. Through the *Location* and *DateTime* scopes, an advantage of ADOPT over linear key-model models is evident; in a linear models the time and location is always represented in the same fashion as defined by the system such a timestamp for time and geo-coordinates for location. Using ADOPT, each application developer can customize their desired contextual information in such a way that suits their implementation strategy most such as defining time using an hour metric, a timestamp, or epoch time depending on the needs of the application.

Once all system-wide basic scopes are established, specific entity type scopes needed to be created. Entity type scopes describe the contextual information for a particular entity type

described in section 3.4.1. An example of an entity type scope is the *Network* scope which describes the contextual information about the cellular network of an application user's mobile device. The *Network* scope is characterized by the *CellularNetwork* entity type which is subclassed from the *device resource* entity type. The following are examples of basic scopes that can be subclassed from the *Network* scope:

- *NetworkStatus*: The current status of a device's network connection.
- *NetworkStrength*: The current signal strength of a device's network connection.
- *NetworkStrengthTrend*: Describes whether the signal strength is increasing, decreasing or remaining steady.
- *NetworkType*: Describes the current network method used for radio signal communication such as CDMA, GSM, etc.

Allowing a system to obtain the cellular network information about a user's handheld device is a valuable context variable for applications that handle extensive network communication. For example, if a user is trying to view a vehicle navigational video on their mobile device which has at a strong network signal through a 3G connection (HSPA+), a mobile application has the ability to stream a video explaining the route on the application user's mobile device. If, however, in the middle of the stream the mobile device's network signal drops significantly (based on the *NetworkStrength* scope) the application needs to be able to reconfigure and display a textually based variant of the application in order to reduce the amount of information downloaded onto the device so that the user does not have to wait for the video to buffer. This example is summarized through Figure 3.16.

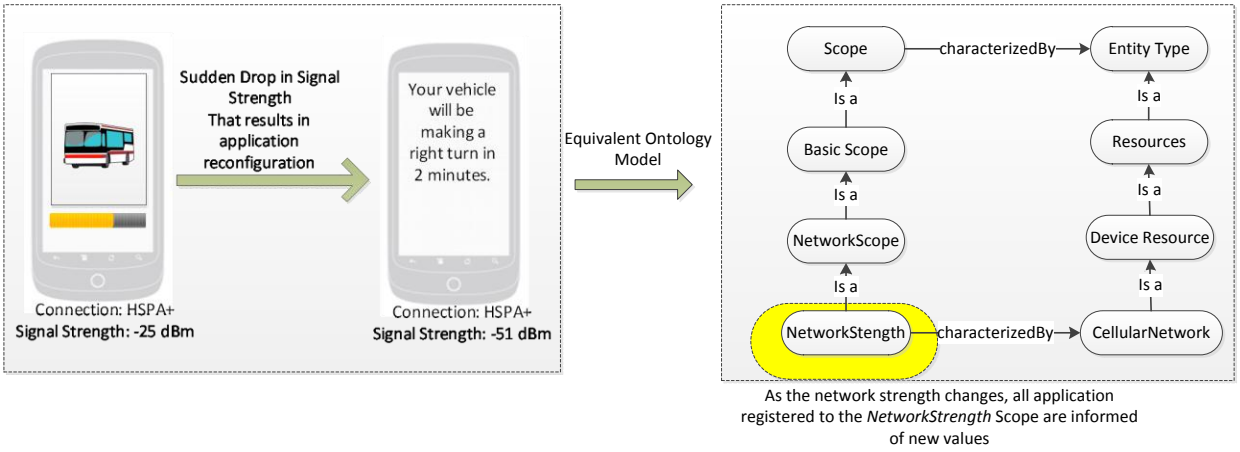

Figure 3.16 - Example of application reconfiguration based on changes to *NetworkStrength* scope

In addition to the *Network* scope, various other device resource entities such as the *battery*, *CPU*, *screen*, etc. need to be incorporated as basic scopes to allow application developers to obtain the available device-specific information. All scopes that correspond to the entity types found within the resource entity type should then be grouped together in order to improve the readability of the ADOPT. Therefore, each resource subclass should also be subclassed within the basic scope as seen in Figure 3.17.

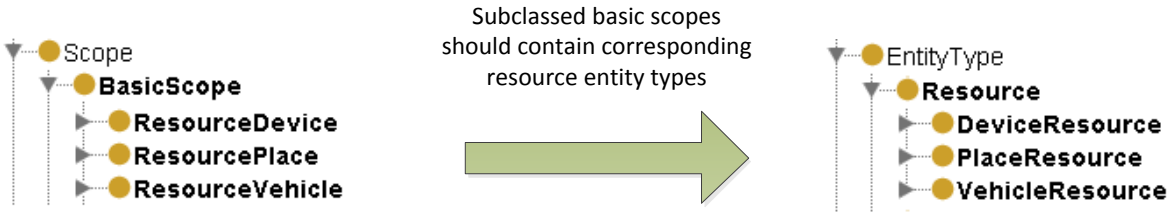

Figure 3.17 - Sample basic scopes with corresponding entity types

The second type of scope, composite, is comprised of contextual information that combines multiple basic scope elements in order to provide high-level abstracted contextual information, as described in section 2.3. One example of a composite scope is *LocationGeo* which combines the three elements of the basic scope *Location*; *latitude*, *longitude*, and *altitude*, in order to provide a comprehensive physical location of a person or object. Using this *LocationGeo*

composite scope, ADOPT can associate the latitude and longitude measures for a particular station as such associating the station “Bercy” to a location with a latitude of “48.843932” degrees and a longitude of “2.38266” degrees.

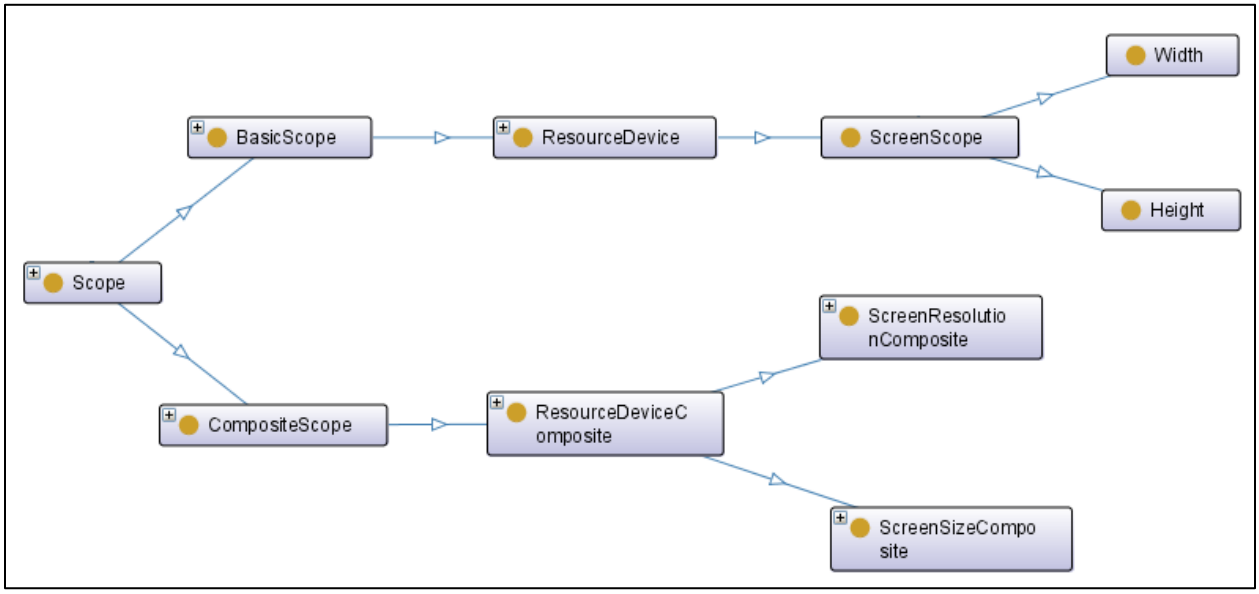

Figure 3.18 - Example of basic and composite scopes for a screen entity type

Another example of how to incorporate multiple basic scopes into an individual composite scope can be seen in Figure 3.18 where the contextual information about a device’s screen can be constructed using the basic scopes; *width* and *height*. By combining the *width* and *height* scopes, ADOPT has the ability to construct contextual information such as a device’s screen size and resolution using sensors that detect a device’s width and height. The scopes that represent screen resolution, *ScreenResolutionComposite*, and screen size, *ScreenSizeComposite*, are considered composite scopes as they are constructed using a combination of basic scopes, which in this case would be the width and height. The difference between the two composite scopes can be seen in the representation of each, as the *ScreenResolutionComposite* is characterized using an integer representation that describes the number of pixels for a screen’s width and height, while the

ScreenSizeComposite scope is constructed using a float representation that describes the physical width and height using inches as its measure.

By deriving composite scopes directly from basic ones, it allows application developers to build additional domain-specific context variables that can be reused by multiple applications within the domain, thereby increasing the reusability of the ADOPT ontology. This feature allows application developers to reuse already constructed composite scopes by incorporating them into their application ontologies. If, on the other hand, a scope is missing from the ADOPT ontology and a developer implements it, it can be extended into the ADOPT repository so that it can be used by other developers. This feature presents another advantage of ADOPT over linear key-value models which have difficulties in integrating dynamic components into their system. One of the goals of ADOPT was to improve the integration process of newly developed components and to allow any new developer to integrate these newly added components into their applications as seamlessly as possible.

Finally, composite scopes also provide the domain specific ontology with the ability to abstract sensor data into high-level contextual information through sensor fusion, discussed in section 2.2. An example of sensor fusion within ADOPT is seen through the *LatenessComposite* scope whose goal is to determine whether or not a particular public transportation passenger is going to be late for an event by combining various scopes obtained from multiple sensors. In this example, the composite scopes that are combined are the *LocationComposite*, *DelayComposite*, and the *SpeedComposite* which together form the *LatenessComposite* scope.

The *LatenessComposite* is constructed using a *Location* scope (either *LocationGeo* or *LocationAddress*) which determines both the passenger's and event's location, either as a geo-

coordinate or address obtained through the device GPS sensors. The *DelayComposite* scope determines if the passenger's transit vehicle will encounter any delays on its travel by accessing the composite scopes that describe the weather conditions, traffic congestion, and slowdowns due to power outages. The *SpeedComposite* scope indicated the speed the vehicle is currently travelling at. By combining these three composite scopes into the *LatenessInfo* scope which is represented using the composite representation *LatenessRepresentation* as well as applying additional application logic that determines how speed and delays directly affect a vehicle's expected time of arrival (ETA), a person's lateness, in terms of time, can be determined. This example's breakdown can be further seen in Figure 3.19.

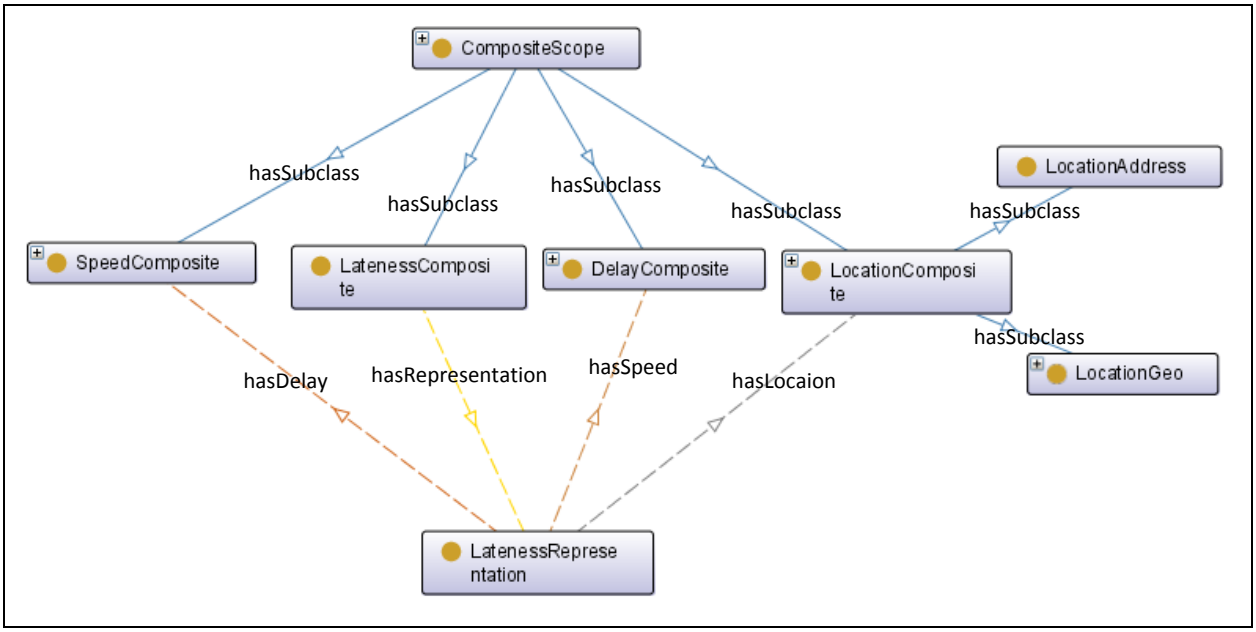

Figure 3.19 - Example of using multiple composite scopes to create a *LatenessComposite* scope

3.4.3 Representations

When all of the contextual information was defined and the necessary scopes for a domain were created, the structure and format of each scope's provided information needed to be established, this step was achieved through the top-level *representation* concept. In order to accurately reflect

the created scopes, representations are divided into two subsections; *basic* and *composite* [65]. Basic representations are closely coupled with basic scopes and are therefore used to accurately portray context information that is obtained through an individual measure. For example, an *altitude* basic scope can be represented as a float value using the "meter" or "feet" unit base. Therefore, the *altitude* of a person or vehicle can be retrieved in the form; "100 meters" or "328 feet", depending on the preference of the application developer and their application ontology. However, representing a scope that defines a particular date (*Date* scope) cannot be achieved through a basic representations as a date is constructed using a combination of numerous basic scopes such as *day*, *month*, and *year*. Therefore, basic representations cannot be derived from other basic representations as they are the base unit of all representations [65].

One problem that arises from basic representations is the need to accurately portray units of measure in contextual information. Having contextual information represented as a value without a unit can be misleading to the application developer. For example, if a sensor was to report that the altitude of a person as "100" without a unit the application developer would have no knowledge to determine if this value was representing the altitude measured in "feet", "meters", or through some other measure. Therefore, there was a need to integrate an additional units ontology in order to portray contextual information in a more meaningful way [65]. To solve this problem the SWEET Unit Ontology [72] was incorporated into the top-level ontology [65]. The SWEET Units Ontology is comprised of *BaseUnits*, which represent the basic units of measure according to the International System of Units (SI), and *DerivedUnits*, which represent units of measure that can be calculated using units from the *BaseUnits* concept. The *DerivedUnits* concept of the SWEET Unit Ontology demonstrates the advantage of this particular ontology as all units can be derived directly from the instances of the *BaseUnits*. For example, a *second* is an

example of a *BaseUnit* as it is the basic unit of measure for time according to the SI. Using the basic *second* unit, developers have the ability to derive additional units such an *hour* unit which can be obtained by dividing the second base unit by sixty. Therefore using the SWEETS Unit Ontology developers have the ability to construct any desired unit of measure by deriving it from a base unit found in the *BaseUnits* concept.

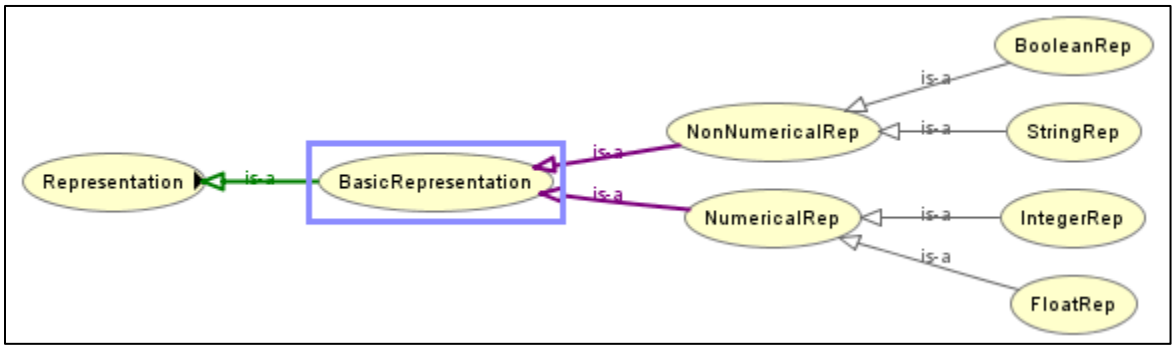

Figure 3.20 - Ontological structure of basic representations

According to Reichle [65], basic representations, as seen in Figure 3.20, can be subclassed as either numerical, which is structured by integers, floats, etc., or non-numerical data structured through strings, Boolean values, etc. When all basic scopes have been defined, ADOPT needs to be able to associate a basic representation to each basic scope. This association is defined using the *hasRepresentation* object property, as seen earlier in Figure 3.2. The *hasRepresentation* property allows a scope to be associated to at least one representation regardless of whether that representation is a composite or basic one. The *hasRepresentation* object property links instances of the scope concept to instances of the representation concept as seen in Figure 3.21. In order to define a *hasRepresentation* property on a particular scope, a *class restriction* needs to be implemented. The simplest way to implement a *class restriction* on a particular scope is to associate one of the basic representation subclasses, such as *StringRep*, *FloatRep*, *IntegerRep*, etc., to that particular scope.

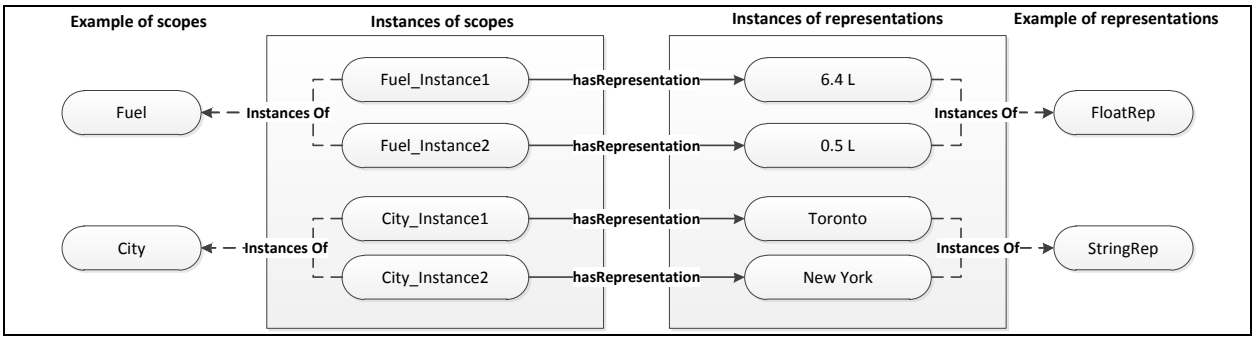

Figure 3.21 - Example of associations between basic scopes and basic representations

For example, in ADOPT the basic *city* scope can be defined as a string value representing the name of the city. Expression 3.1 demonstrates a class restriction implemented for the city basic scope.

$$hasRepresentation\ only\ StringRep$$

Expression 3.1 - Implementation format of a class restriction

Expression 3.1 implies that the location information of the city basic scope can only be represented using an individual string value (*StringRep*) such as "Toronto", "New York", "Paris", etc. as seen in Figure 3.21. The same structure can be followed for other non-numerical basic representations such as the *BooleanRep* representation. A basic Boolean representation can be used to define whether or not an assistance alarm is currently activated, as seen in Expression 3.2, with a result yielding true if the alarm is activate and false if it is not.

$$hasRepresentation\ only\ BooleanRep$$

Expression 3.2 - Example basic Boolean representation

Basic representations can also associate numerical representations to the corresponding scopes, with a unit value, defined within the SWEET ontology, distinguishing what each numerical representation indicates. One example includes the *BatteryLevel* scope which is used to obtain

information about the current battery level of a device. A device's remaining battery life can be represented using multiple formats such as a percentage or time based measure, as seen in Figure 3.22. By having a system that allows for multiple representations, application developers can choose which structure or format they feel most comfortable working with.

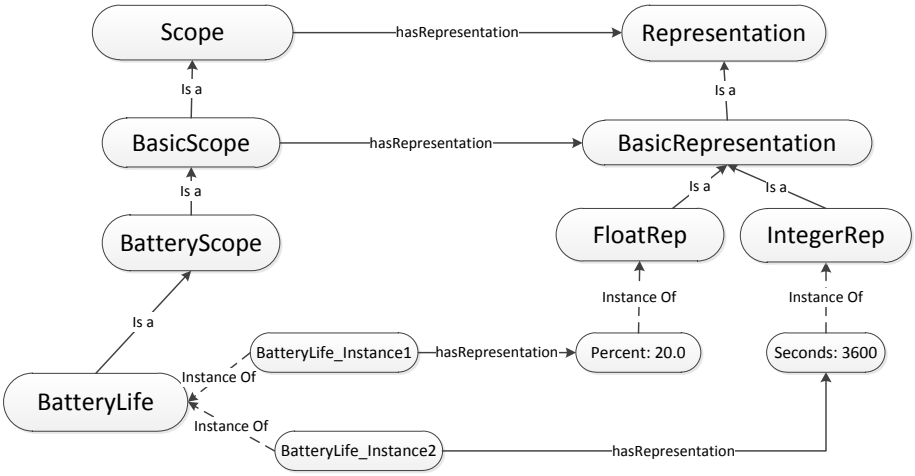

Figure 3.22 - Example of BatteryLife scope with multiple representations

There are two ways to associate multiple representations, as seen in Figure 3.22, to a single scope in the ADOPT ontology, either by implementing them directly into the *hasRepresentation* class restriction or by defining a new scope for each representation such as *BatteryLifePercentage* and *BatteryLifeTime* with each having its own representation restriction. As discussed earlier in this section, ADOPT allows application developers to give their representations a unit value in order to allow application developers to choose which format they want to process the desired contextual information. In order to add a unit measure to the class restriction, the object property *hasUnit* must be added to the restriction. An example of the *BatteryLifePercentage* basic representation with a *hasUnit* restriction is defined in Expression 3.3.

$$hasRepresentation\ only\ (FloatRep\ and\ (hasUnit\ value\ percent))$$

Expression 3.3 - Example of restriction with *hasUnit* property

Expression 3.3 indicates that all formatted contextual information for the *BatteryLifePercentage* scope must be formatted as a percentage. If a developer was to add another basic representation such as an integer value that will represent the amount of time a battery has remaining in terms of seconds it can be added to the restriction through the ***or*** operator demonstrated in Expression 3.4.

$$hasRepresentation\ only\ ((FloatRep\ and\ (hasUnit\ value\ percent))or\ (IntegerRep\ and\ (hasUnit\ value\ second)))$$

Expression 3.4 - Example of a restriction that allows for multiple units to be declared

Finally, representations can also contain enumerations, meaning that the formatted representation can be a specific value obtained from a pre-defined set of values. An example of such representations can be seen in the basic representation for the *BatteryChargeStatus* scope which is used to provide contextual information that describes whether or not the battery is currently being charged. The system handles the proposed enumeration restriction by implementing the *hasSimpleValue* object property for the desired scope. Expression 3.5 demonstrates a representation indicating the status of whether a device is currently charging using a basic StringRep representation whose value can only be *charging*, *discharging*, or *unknown*.

$$hasRepresentation\ only\ (StringRep\ and (hasSimpleValue\ only\ \{"Charging","Discharging","Unknown"\}))$$

Expression 3.5 - Example of declaring contextual information through enumerations

The second type of representation used in ADOPT is the composite representation, which is used to structure the composite scopes discussed in section 3.4.1. Since composite scopes cannot be represented through individual basic representations, ADOPT needs to have the ability to

combine multiple basic representations in order to structure more complex data. One example of a composite scope is the *LocationAddress,* which is used to obtain the address of a passenger/object. Although the system can retrieve all the individual components of the *Location* scope such as a *city*, *country*, *street*, etc. using basic representations, the application developers would need to retrieve too many individual components in order to determine a particular address. Therefore, using a composite representation allows application developers to retrieve all the necessary address information through a single retrieval method.

Through composite representations, application developers have the ability to choose which components of the *Location* scope are needed to determine the address within their application. To accomplish this, composite representations group basic scopes through their *object property restrictions*, which are used to create relationships between the instances of various scopes found in the ontology [73]. One example of this can be seen in the composite representation *AddressRepresentation* which has an object property restriction called *hasCity* that indicates that instances of the concept *AddressRepresentation* must include instances of the *city* basic scope. In order to link instances of *AddressRepresentation* and *city*, the *hasCity* restriction must implement the OWL properties *Domain* and *Range*. The *Domain* and *Range* of an *object property restriction* links instances of a *Domain* class to the instances of a *Range* class [73]. In the *hasCity* restriction example, the *Domain* is defined as the *AddressRepresentation* and the *Range* is defined as the *city* basic scope. This example is further expanded in Figure 3.23 as the composite representation *AddressRepresentation* is associated to five object property restrictions; *hasCity*, *hasCounty*, *hasPostalCode*, *hasNumber*, and *hasStreet* which are directly associated to the basic *Location* scope's *city*, *country*, *postalcode*, *unitnumber* , and *street* subclasses. Through this method, an application which desires to retrieve contextual information about an address does so

through the *LocationAddress* composite scope which returns an *AddressRepresentation* representation.

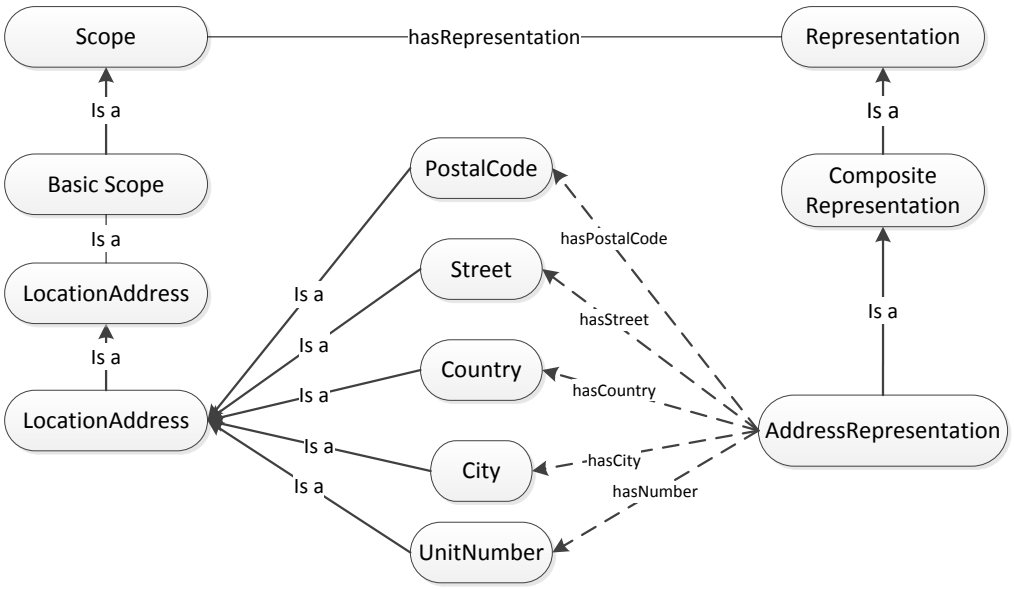

Figure 3.23 - *AddressRepresentation* example using object property restrictions

3.5 System Architecture Model

In order to allow ADOPT based mobile applications to reconfigure software components based on changes to the various defined scopes, the architecture of the system needs to be examined. The architecture of ADOPT was built on top of earlier work presented in MUSIC [17] model and is comprised of three main layers:

- *Ontology Layer*: comprised of all entity types, scopes, and representations that are created by the application developer for their desired mobile application.
- *Context Layer*: describes all contextual variables that will be implemented in the desired application and is comprised of the application ontology and context model modules. The

context model defines all the contextual variables used for software component reconfiguration [74] within the desired public transportation application.

- *Logic Layer:* describes all the software components that can be reconfigured within the desired application and the variants of each component [75] (component plan module), as well as determine which variant of a software component should be activated based on changing contextual (utility function [76]).

An application developers initial step to initiating the ADOPT architecture is to derive all possible entity types and scopes from the domain ontology, described in section 3.4.1 and 3.4.2, to construct an application-based ontology for a mobile application developer's desired scenario. Once the application ontology is constructed, the developer needs to create all the context models that will define the contextual information that will be obtained from various environment and user based sensors. Context models are further explained in section 3.5.1. In order to process the contextual information defined in the contextual models, a component plan needs to be created that will inform the application of which variants each software component can be reconfigured to, this is further explained in section 3.5.2. Finally, once all of the contextual information and component plans have been defined, the application developer needs to create utility functions, described in section 3.5.3, that will notify the application of any reconfigurations that need to occur based on the changing sensor information defined within the contexts models. This entire process is exemplified in Figure 3.24.

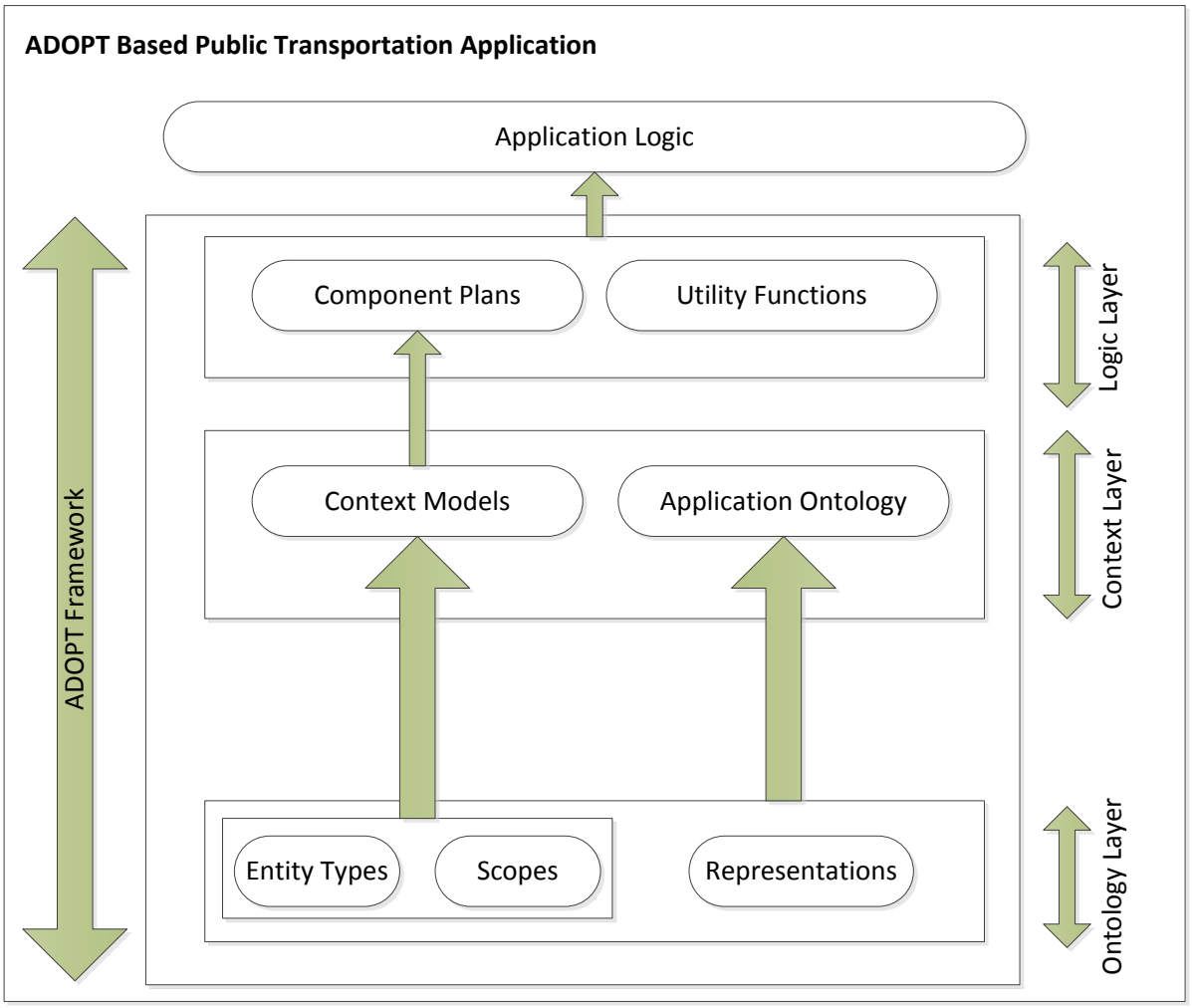

Figure 3.24 - ADOPT Architecture

3.5.1 Context Models

When developing an ADOPT based public transportation application, application developers need to define which scopes and entities their mobile applications will implement in order to reduce the amount of imported ontology code added to the application. To do this, application developers must define which contextual variables will be implemented into their application and associate the necessary basic and composite scopes to them. Figure 3.25 demonstrates a generic context model used for both composite and basic scopes described earlier in this chapter. Using

Figure 3.25 as the basis for defining contextual information an application developer needs to create a context model for every piece of contextual information found within their application.

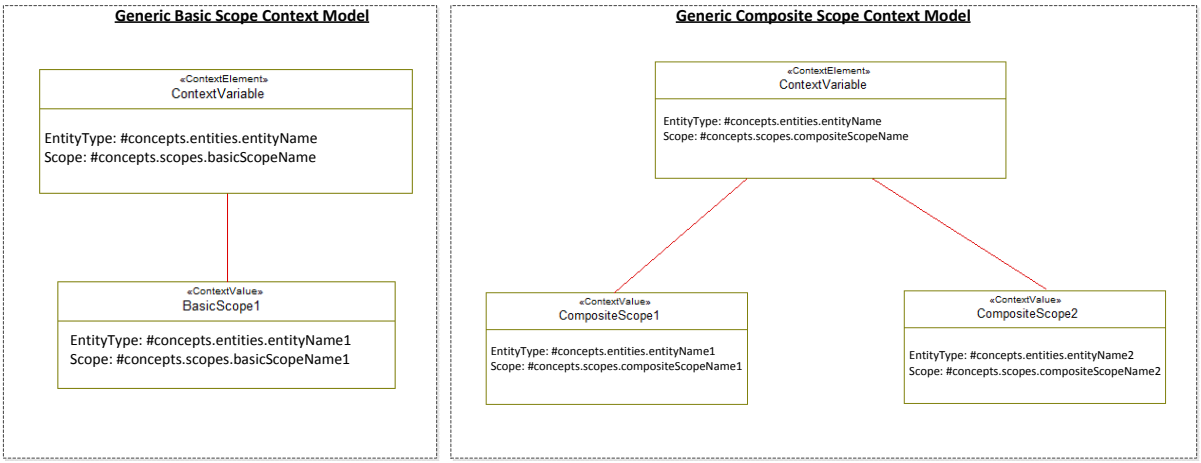

Figure 3.25 - Generic context model for basic and composite scopes

Elements found within the context model are always associated to a particular entity type described in section 3.4.1 and a scope described in section 3.4.2. This association allows application developers to link which context sensor widget, described earlier in section 2.3, their application will register to in order to retrieve changing contextual information. If a developer desires to include their own scope which is not included within ADOPT, then they then will need to build the context widget associated to that new scope themselves. The context model elements are each associated to an entity type and scope using the structure found in Expression 3.6.

$$\#concepts.elementType.elementPath.elementName$$

Expression 3.6 - Generic context model element definition

In Expression 3.6, the *elementType* characteristic describes whether a context model element is an entity type or a scope; while the *elementName* describes which element described in section 3.4.1 or which scope describe in section 3.4.2 the context model element references.

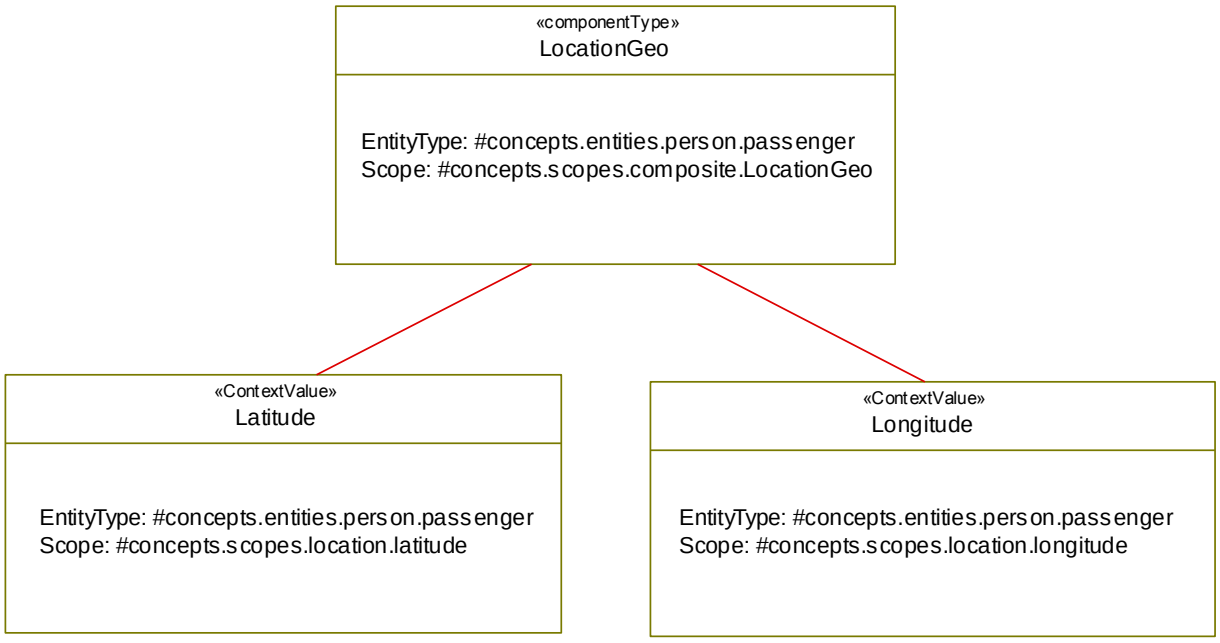

Figure 3.26 - Example of location context model

Figure 3.26 demonstrates an example of a location context model that a developer would instantiate when attempting to obtain a physical location, such as the geographic coordinates, of a passenger entity. If a passenger is detected to have changed their location, defined by the scope name #concepts.scopes.basic.Latitude and #concepts.scopes.basic.Longitude, then applications that have registered to the context varaible LocationGeo by the scope name #concepts.scopes.composite.LocationGeo are informed of any location changes.

3.5.2 Component Plans

Once all the context models have been created, the application developer needs to define all of the components that can be reconfigured within the application. To define the reconfigurations within an application, application developers needs create a component plan that outlines all possible variants for each software component. Variants can be classified as one of two different types; *atomic* and *composite* realizations [75]. *Atomic* realizations are used to represent a variant

that is composed of an individual software module that reconfigures when a contextual change is detected. *Composite* realizations, on the other hand, are composed of multiple atomic realizations suggesting that multiple modules need to be reconfigured when there is a change to contextual data associate to that variant's component type. Figure 3.27 demonstrates a generic example of a component type with multiple variants represented through both *composite* and *atomic* realizations. Figure 3.27, also, demonstrates that a generic component type can be comprised of multiple variants with each variant comprised of unique modules. Each variant of a component becomes activated under specific conditions that must be satisfied based on the processed contextual data retrieved from a sensor. One example of a component type with multiple variants is a component that determines which location sensor data to use in order to retrieve an accurate location of a particular user.

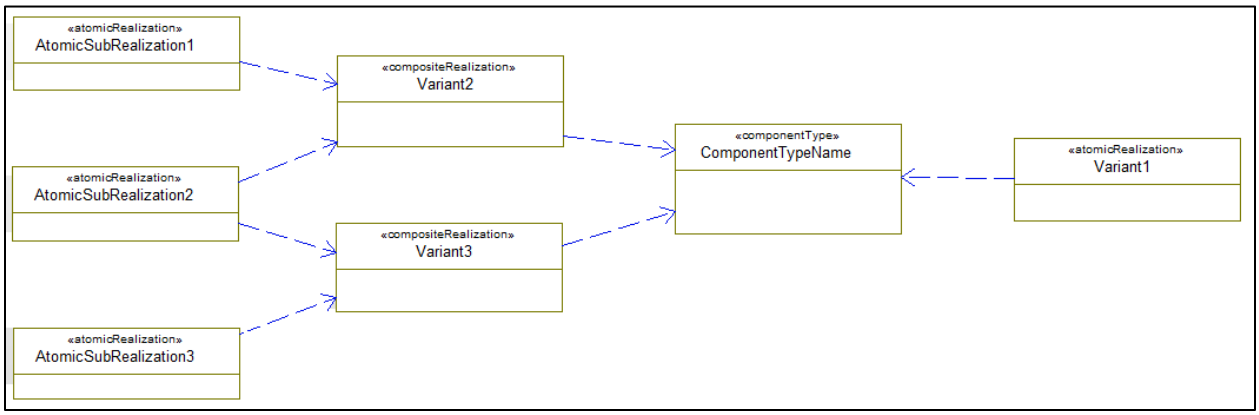

Figure 3.27 - Generic component type definition

For this particular example let's assume that there are only two ways of retrieving location data; through GPS coordinates (more accurate but can only be used outdoors) or by triangulating the user's position through nearby cellular towers and Wi-Fi access points (less accurate but can be used anywhere). Figure 3.28 demonstrates the GPS variant as an atomic realization because it is composed of an individual module, while the triangulation module is described as a composite

realization because is it comprised of multiple modules, in this case cellular towers and Wi-Fi access point modules.

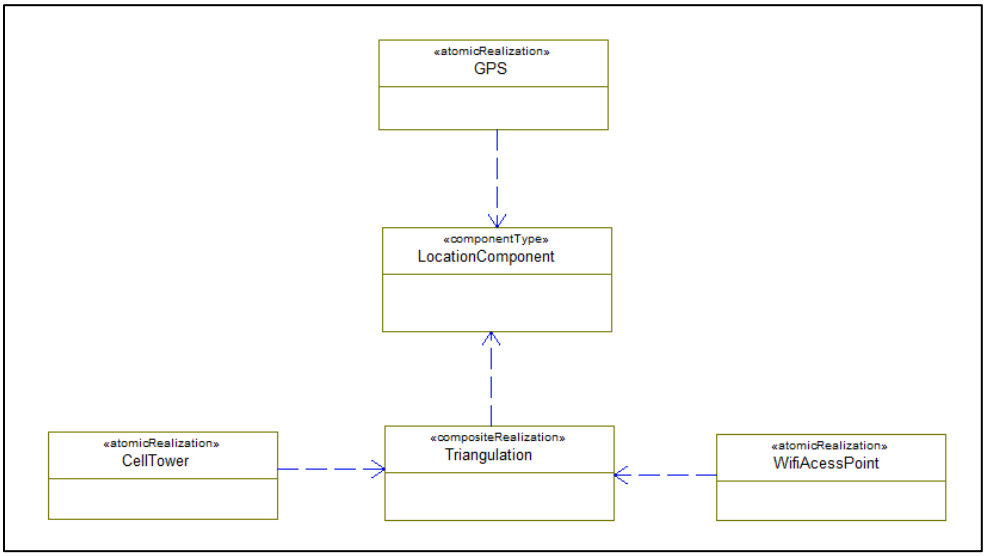

Figure 3.28 - Example of component type definition through *atomic* and *composite* realizations

3.5.3 Utility Functions

The final step of the architecture is to determine when a component's variants need to be reconfigured. In order to successfully achieve this step, all the necessary contextual information that the application will process need to be defined using the context models along with all the possible variants that components can be reconfigured to. In order to successfully determine how system reconfiguration should occur, a *utility function* for each component needs to be created in order to determine when a variant should be activated and deactivated. The utility function processes retrieved contextual information and uses developer-defined probabilistic functions to determine which variant of a particular component needs to be activated, the probabilistic functions provide a threshold value that determines which variant to reconfigure to [76]. Within the utility function each variant is given a unique threshold range from 0 to 1 which will inform the application of which variant to reconfigure to. For example, a utility function can be used to

determine which variant of the location component, seen in Figure 3.28, to reconfigure to. For simplicity, the following example will remove the WifiAccessPoint realization from the location example provided in Figure 3.28 and only allow reconfiguration of the location component to be either the GPS or CellTower variants. In order to determine which of the two variants to activate, the following ADOPT domain ontology scopes will be processed to determine a threshold value that will inform the application of which variant to use:

- *GPSAvailablity*: a Boolean value that indicates whether or not the mobile device can currently obtain a signal from GPS satellites.
- *Accuracy*: the required accuracy in meters of the retrieved location data.
- *BatteryLifeRemaining*: the current battery level, in percentage, in case the application needs to conserve power at a low battery level

```
5  double totalLocationUtility = 0.0;
6  double gpsUtility = 0.0;
7  double cellTowerUtility = 0.0;
8  double maxUtility = 4.0;
9
10 //Determine whether or not the mobile device can connect to a GPS satellite
11 String gps = (String) context.getValue("GPS_AVAILABLE");
12 gpsUtility = (network.equals(Constant.CONNECTED))? 2.0 : 0.0;
13
14 if (gpsUtiltiy == 0.0){
15
16     //If it cannot connect to a GPS satellite check to see if the application can connect to a cell tower
17     String cellTower = (String) context.getValue("CELLTOWER_CONNECTED");
18     cellTowerUtility = (cellTower.equals(Constant.CONNECTED))? 0.5 : 0.0;
19
20     if (cellTowerUtility == 0.0)
21         return 0.0;
22 }
23
24 //Determine the accuracy required for the application
25 int accuracy = (Integer) evalContext.evaluate("ACCURARY", context)).intValue();
26 totalLocationUtility = (accuracy < 50) ? 1.0 : 0.0;
27
28 //Determine the current battery life of the mobile device
29 int batteryLife = (Integer) evalContext.evaluate("BATTERY_LIFE", context)).intValue();
30 totalLocationUtility = (batteryLife > 15) ? 1.0 : 0.0;
31
32 totalLocationUtility += (gpsUtility + cellTowerUtility);
33
34 return new Double (totalLocationUtility/maxUtility);
```

Figure 3.29 - Example of a utility function for the location component

Using these three contextual variables, Figure 3.29 displays a location utility function that will provide threshold values ranging from 0 to 1 that indicate which variant of the location component to use. Table 3.2 provides all the possible variants defined and their associated utility threshold values. Therefore, when a utility function results in one of the defined "Utility Function Outcomes" values, the designated location variant will made available.

Table 3.2 - Example of threshold values that correspond to a particular location variant according to Figure 3.29

Utility Function Outcome	Designated Location Variant
0.0 – 0.125	- Indicates that neither of the variant requirements have been met and that the user should be informed that a location cannot be determined resulting in a **None variant**.
0.125 - 0.75	- Indicates that not enough of the requirements for the GPS variant have been met but the requirements for the **CellTower variant** have been met.
0.75 - 1.0	- Indicates that all the requirements of the **GPS variant** of the location component have been meet.

In addition to Table 3.2, Figure 3.30 demonstrates a graph of all the possible threshold values obtained from Figure 3.29 and which variant each threshold value will represent. Through this graph, it is evident that under a majority (utility outcomes between 0.125 and 0.75) of circumstances the location component will yield the CellTower variant as it requires the least amount of contextual information to function correctly. The GPS variant only covers 25% of the possible outcomes (0.75 - 1.0) as it functions correctly only under specific contextual information conditions, such as the presence of GPS satellites. A small subset of outcomes (0.0 – 0.125) will yield an offline variant of the location component because the retrieved contextual information will not have met either the CellTower and GPS variant criteria. This usually

happens when a mobile device in not in range of any cellular towers and has no direct line of sight to any GPS satellites.

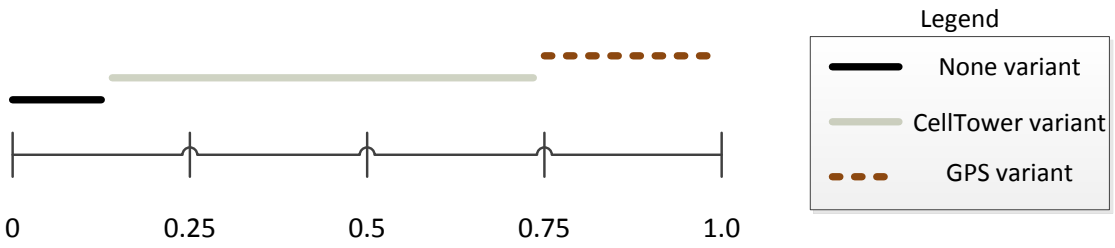

Figure 3.30 - Graph representing all of the threshold values described in Table 3.2

3.6 Summary of Adopted Methodology

The goal of ADOPT was to improve the creation of context-aware applications designed for a public transportation domain. Using the proposed system, application developers will have the ability to efficiently create context-aware applications that retrieve sensor information, abstract that information into data that the application can process, as well as initiate component reconfiguration based on changes obtained from the sensed information. The goal of the ADOPT framework was to allow application developers who are not considered experts in the public transportation domain to quickly understand the contextual information available to them within the domain and, therefore, reduce their development time to successfully implement a functional public transportation mobile application.

In this chapter the proposed ADOPT domain ontology was built as an extensible component of Reichle's top-level ontology [65], and introduced public transportation concepts that directly relate to the top-level ontology's entity types, scopes, and representations. ADOPT details various real-world/logical objects and people, also known as entity types, which directly affect and/or are affected by contextual changes within the public transportation domain. Using these

entity types a concrete set of contextual information, known as scopes, was created which can be used within any public transportation mobile application. Through these scopes, application developers have the ability to directly integrate context sensors without the need to implement them on their own.

ADOPT is comprised of created simple and complex representations and formats that the contextual information can be delivered in when an application requests for contextual information about a particular entity. If additional scopes and representation need to be created, ADOPT provides application developers with the necessary tools to incorporate the conceived concepts. A sample of the final public transportation domain specific ontology can be observed in Figure 3.31. It is important to note that Figure 3.31 only displays a portion of the overall ontology which consists of more one hundred and fifty concepts. In addition to Figure 3.31, there are several figures found throughout this chapter including Figure 3.5, Figure 3.9, Figure 3.10, Figure 3.11, Figure 3.12, Figure 3.13, Figure 3.15, Figure 3.17, Figure 3.18, Figure 3.19, and Figure 3.20 that describe the ADOPT public transportation ontology in greater detail.

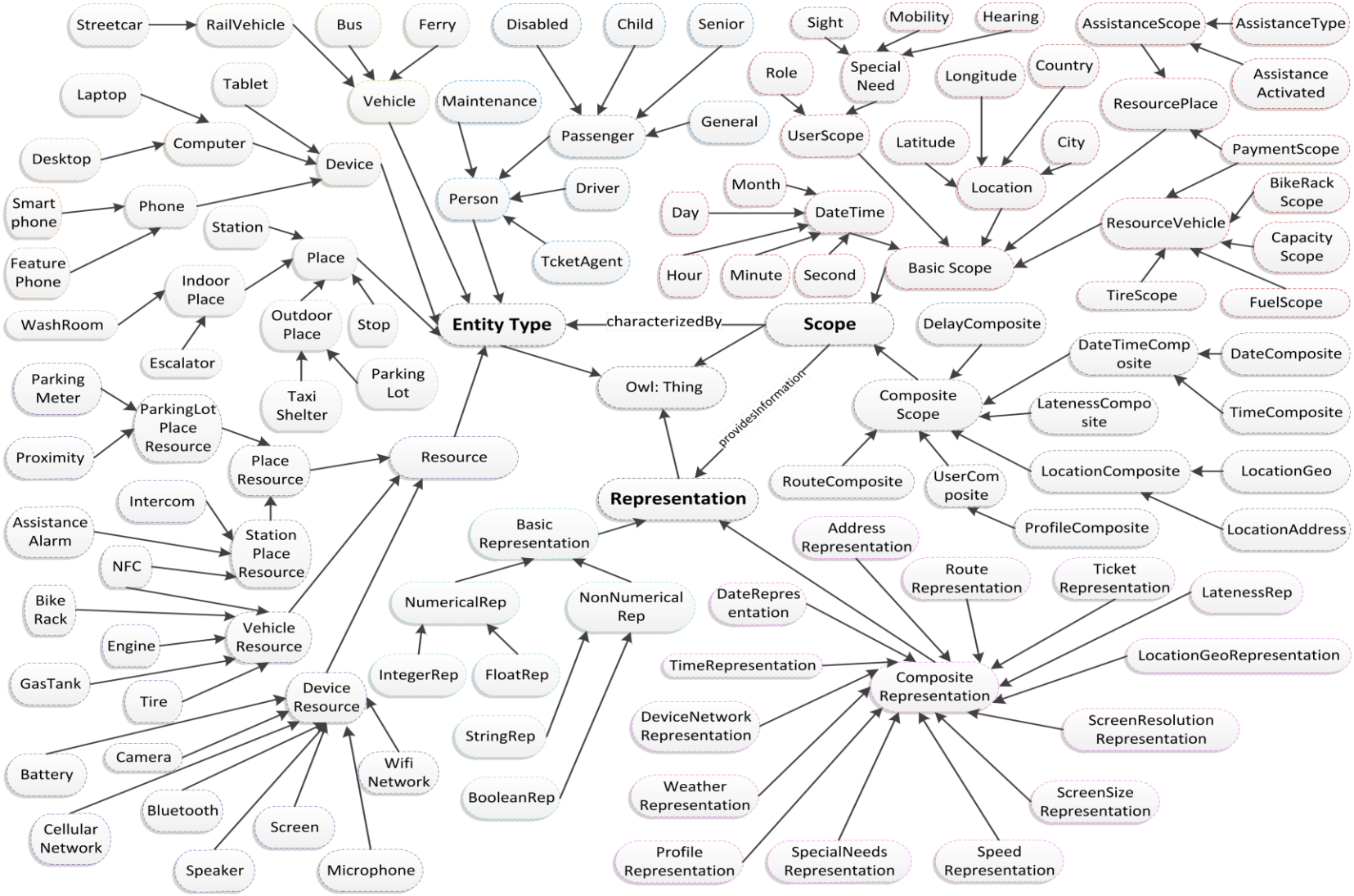

Figure 3.31 - Sample of proposed public transportation ontology

Chapter 4. Results and Analysis

Based on the methodology presented in Chapter 3, the strengths of the ADOPT framework will be examined in this chapter. In particular, a specific case study comparing a linear key-value model to the ADOPT framework will be examined in detail with the results presented. The goal of this chapter is to demonstrate several key advantages through both qualitative and quantitative measures of the ADOPT framework.

4.1 Case-Study: Person with Reduced Mobility (PRM) Application

The Person with Reduced Mobility (PRM) application was a mobile application designed to demonstrate how contextual changes can result in component reconfigurations within a public transportation domain. The objective of the PRM application was to enhance the travel experience of general, elderly, and disabled passengers who travel on line 14 of Paris's RATP Metro system. In order to enhance their travel, the PRM application provides a set of passenger relevant information, also known as *services*, based on the specific station that a passenger is currently located in. As the passenger moves from station to station on line 14, the list of relevant *services* changes based on the station the user is arriving at. Relevant *services* to a passenger can include:

- *Route Map*/*Station Location*: map of the route the passenger is currently traveling on as well as the map of the station that they are currently located in, along with any POIs nearby.
- *Arrival Alarm Notifications*: informs the passenger when they have arrived at their declared destination station.

- *Request Assistance*: informs a safety marshal of the whereabouts of the passenger in the event that they have requested assistance.
- *Station Information*: additional information about the user's current station.

The availability of these, passenger relevant *services,* are determined based on the contextual variables defined by the system. In the PRM application, the following list of context variables was used to determine, not only under which conditions the previously described *services* should become available, but also when specific components such as the Text-To-Speech (TTS) service should be made available to the passenger:

- *Special Need*: indicates whether a passenger has a disability, in the case of the PRM application, disability options included; sight, mobility, both or none.
- *Date of Birth*: indicates the date of birth of the passenger which can be used to determine their age.
- *Location*: detects the station that a passenger is currently located in.
- *Network Signal Strength*: the 3G connection signal strength of the passenger's mobile device.

4.1.1 Special Need/Date of Birth Context Specification

Two contextual variables, disability and age, were used to reconfigure an individual passenger's mobile screen layout in order to provide them with a flexible and more intuitive experience. Due to a lack of sensors that determine a user's disability and age, these variables needed to be pre-filled by the passenger during the first time the mobile application is launched. The context model of the disability and age context variables, as per the specifications described in section 3.5, can be seen in Figure 4.1. The context model, in Figure 4.1, describes the disability attribute

which is constructed following the conventions described in section 3.5. It is defined as a composite scope that is made up of two basic scopes; sight and mobility which are indicated using Boolean values.

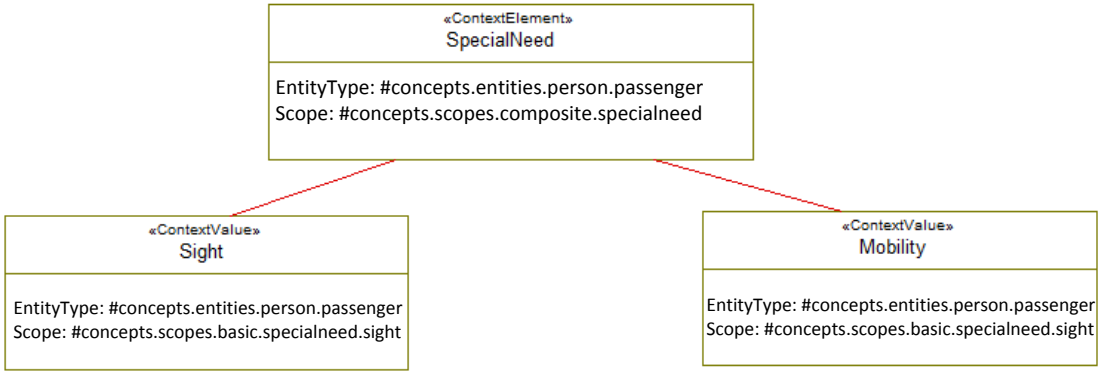

Figure 4.1 - Disability context model for PRM application

Figure 4.2, on the other hand, demonstrates how the date of birth composite contextual variable can be modeled using the individual metrics required for determining a user's age.

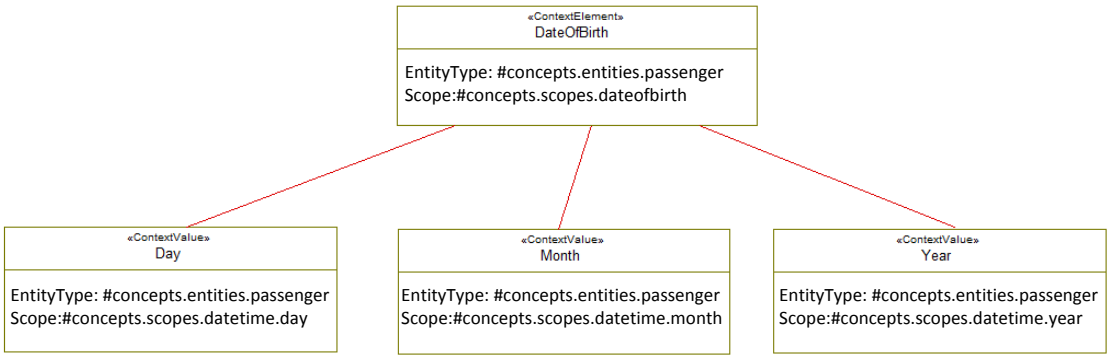

Figure 4.2 - Date of birth context model for PRM application

Using the context models described in Figure 4.1 and Figure 4.2, the PRM application is able to reconfigure components such as the user interface layout based on changes to the contexts

SpecialNeed and DateOfBirth. In the PRM application, if a passenger has indicated that they do not have a disability and their age is determined to be less than 65, then the list of relevant *services* was presented in a grid-like layout, as presented in Figure 4.3, where each service could be activated by the user selecting an icon.

Figure 4.3 - PRM screenshot layout for passenger with no disability

If the passenger has indicated a sight disability then the application needs to reconfigure its layout by removing all the icons and texts, as a person with limited sight would have difficulties viewing the icons and texts, and provide the passenger with a *gesture*-based user interface. The *gesture*-based user interface allows users to activate any *service* by performing a specific touch-and-drag pattern on the mobile device's screen instead of selecting each *service* directly. The distinction of the two methods can be seen in Figure 4.4.

If, on the other hand, if the user has indicated that they have a mobility impairment or that they are older than 65, the layout needs to reconfigure and provide both a grid-like interface along with the gesture-based interface that will allow a passenger to choose between the two methods depending on which is more convenient. In addition to the reconfiguration of the layouts displayed on the screen, if either a sight, or mobility disability, or an age of greater than 65 has been indicated by the user, TTS will be activated thereby converting all information into audible speech that the passenger can better understand. If the user has not indicated a disability and is under the age of 65 then the TTS feature will not be activated.

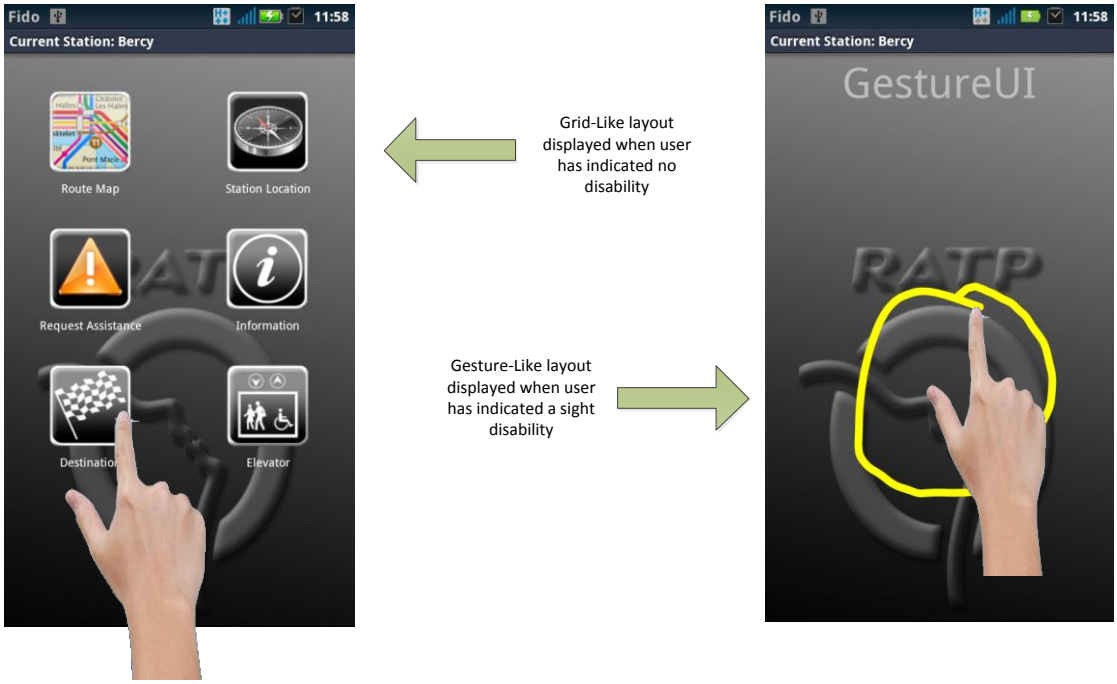

Figure 4.4 - Distinction between activating the destination service through grid and gesture based layouts

4.1.2 Location Context Specification

The location contextual variable in the PRM application reconfigures the availability of the relevant *services* based on the station a passenger is currently located at. There are two ways for the PRM application obtain a location accurate enough to understand which station a passenger

is located at; using the Global Positioning System (GPS) or through cellular tower identification. If a passenger is located outdoors and their mobile device has a direct path to the GPS satellites then the device should use the embedded GPS sensor to retrieve the location of the passenger. If, however, the passenger is located indoors within a station and there is no clear path for a mobile device to connect to GPS location satellites [77], the PRM application determines the station that a passenger is located at through cellular tower identification as each station on line 14 of the Paris Metro has dedicated cellular towers. Therefore once a user's device connects a particular tower, its identification number can be referenced to a database entry to determine which station the passenger is currently located at. The contextual model for the location context of the user can be seen in Figure 4.5.

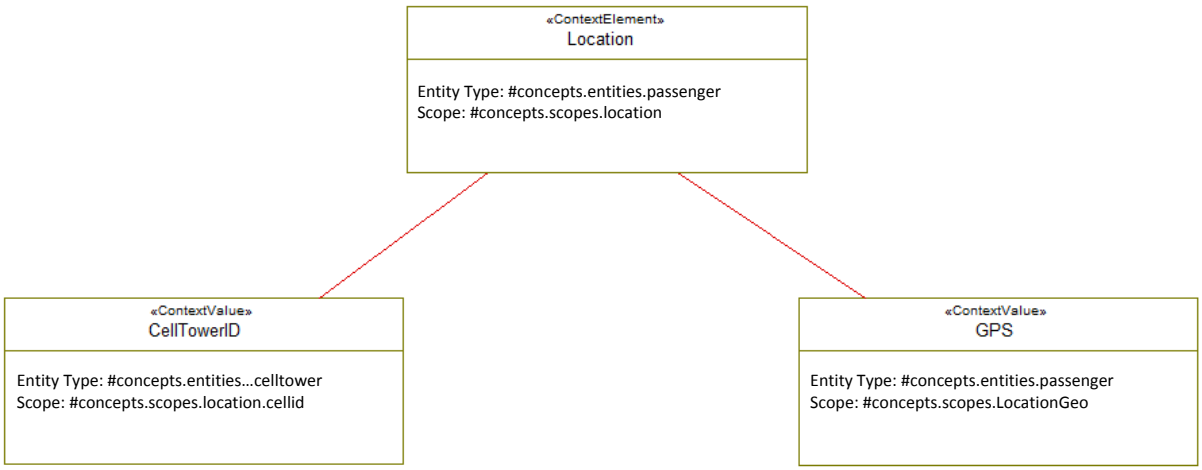

Figure 4.5 - Location context model for the PRM application

An example of using the location context to retrieve relevant *services* can be seen in Figure 4.6 where a passenger's location directly affects the relevant information that is presented to them. In Figure 4.6, when the passenger is detected to be located at "Bercy" station, either by using the GPS sensor or cellular tower identification, relevant *services* such as *Station Location*, *Route*

Map, *Request Assistance*, etc. will be made available to the passenger. When the passenger leaves "Bercy" station and enters "Cour Saint Emilion" station, a different set of relevant services will be made available to that passenger to reflect information found at the new station, in such a case the *Station Location* service will no longer be available on the passenger's mobile device as the system does not contain a station map for "Cour Saint Emilion" station.

Figure 4.6 - Example of service availability at different stations

4.1.3 Network Signal Strength Context Specification

The final context variable used in the PRM application is the cellular network signal strength of the 3G data connection on the user's mobile phone. In order for the PRM application to determine what the cellular signal strength between the passenger's device and the cellular tower are, two key contextual variables (scopes) need to be examined; *NetworkStatus* and *NetworkStrengh*. *NetworkStatus* informs the PRM application of whether or not the mobile device currently has a mobile connection to a cellular tower. If no connection is available then the device will not be able to access any of the *services* provided as no Internet connection is

available to retrieve information *services* from. If there is a connection to a cellular tower, the *NetworkStrengh* variable is used to determine how fast the connection is. Based on the connection speed, the PRM application will need to reconfigure to reflect the fluctuating signal strength. The contextual model of a mobile signal strength can be seen in Figure 4.7.

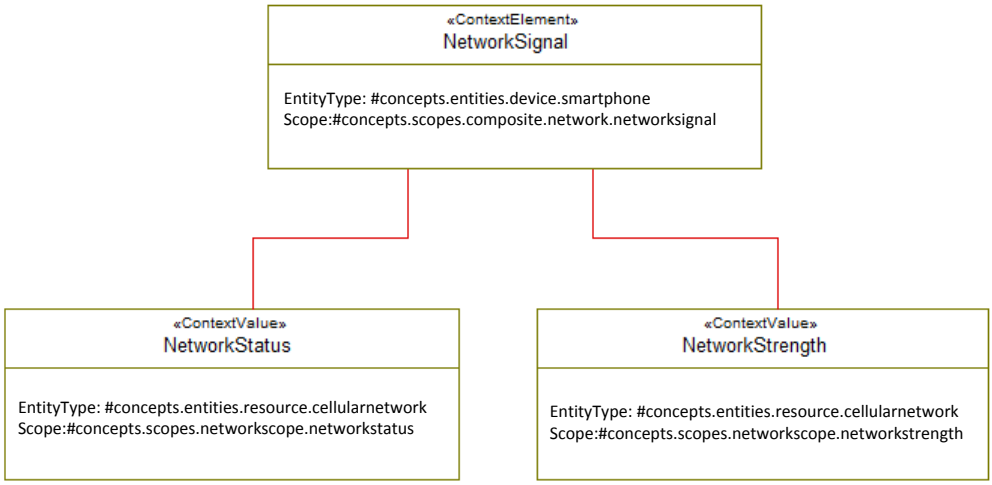

Figure 4.7 - Network signal context model for PRM application

In the PRM application, the signal strength is calculated using decibels (dB) and is divided into four variants based on prior testing of the signal strength variable. For the purpose of the PRM application, Table 4.1 examines the various threshold ranges and their corresponding.

Table 4.1 - Signal strength thresholds

Range (dB)	Value	Explanation
-50+	High	Occurs when the Internet connection to the mobile device is good enough for streaming videos
-50 to -70	Medium	Indicates that the Internet connection is slightly weaker and therefore should display a graphical version of a service without any videos

-70 to -113	Low	Indicates that the Internet connection is of poor quality and downloading data should be kept at a minimum resulting in a text-based version of a service
-113	No Connection	Occurs when there is not sufficient access to the connection resulting in no Internet service to the mobile device

In the PRM application, if the application is receiving "high" signal strength, according to Table 4.1, and a user activates a *service*, that *service* will be presented in its intended form, if however at any point the mobile device starts to receive a "low" signal strength, the *service* will automatically reconfigure to display a text-based variant to reduce the amount of data downloaded from the server.

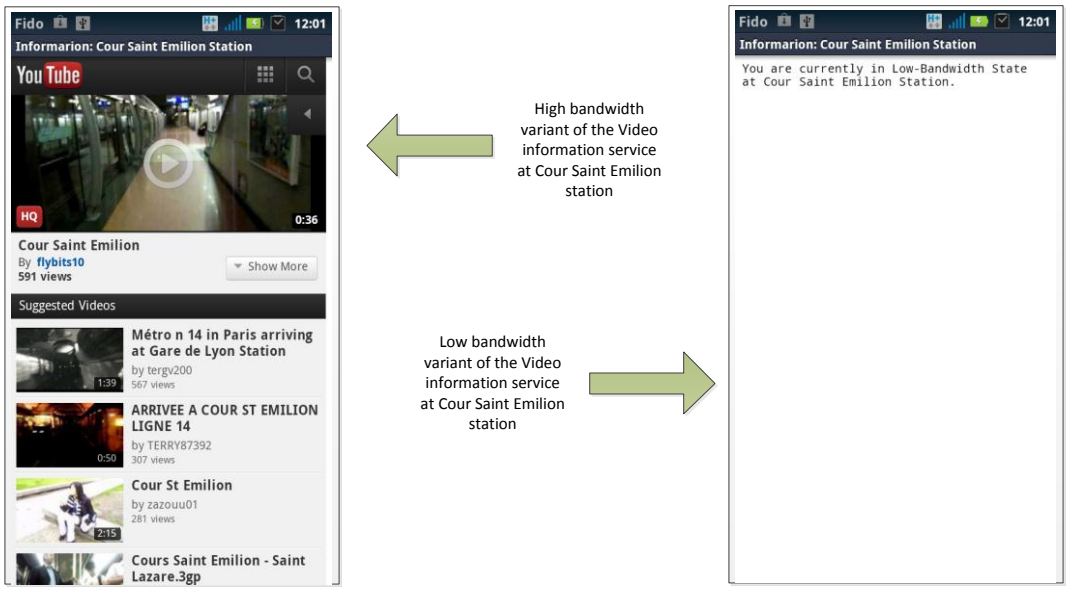

Figure 4.8 - High and low bandwidth layout variants of information service

The same can be applied to the "medium" and "offline" variants. Whenever the signal changes, based on the values of Table 4.1, the corresponding change is made on the mobile device. The

inclusion of the signal strength variable allows mobile devices that receive poor connection to reduce the amount of information displayed so that it can be rendered in a quicker manner. Forcing the passenger to view a video when a low signal connection is present can take an inordinate amount of time. An example of this process can be observed in Figure 4.8.

4.2 Approaches to Application Implementation

Based on the contextual requirements described in section 4.1, the PRM application was built using two different approaches; the linear key-value model and the proposed ADOPT framework. The reason the key-value model was chosen as the comparison model was to provide a benchmark using the simplest context model with the ADOPT solution, if the ADOPT solution demonstrates an advantage over key-value models it can then be compared to more complex context models. The goal of these two approaches was to create a functional mobile application that demonstrates the ability to retrieve contextual information from both the passenger and environment sensors (context and mobile sensing), abstract sensed data into information that can be processed by the system (context abstraction), and determine when reconfiguration should occur (context modeling and synthesis). In order to best implement both approaches on a mobile device, the Android operating system was chosen, in large part, due to the *openness* of its platform [78]. Out of the five current smart mobile platforms; Android, Symbian, Windows Mobile, iOS, and Blackberry, Android provide developers with the ability to access the most system resources without requesting permission from either the user or the device [78]. This implies that Android application developers have the ability to access and manipulate specific system resources such as obtaining the signal strength of a 3G data connection and location through cellular tower identification which is either not possible or requires special permissions on other mobile platforms.

4.2.1 Linear Key-Value Modeling Approach

The first version of the PRM application that was created was one based on a linear programming pattern where the application was created using only Android's mobile Software Development Kit (SDK) without integrating any additional libraries or components. In the linear modeling technique the first step was to create all the user-interfaces (views) that would be associated to the PRM application. This includes both the grid and gesture based user interfaces, discussed in section 4.1.1, that will be displayed to the passenger based on their pre-defined disability and age settings. Once all the views had been created the application needed to handle a user's interaction with the views. Therefore, each view component was associated with listeners that detect when a user has requested to interact with the device. By creating all of the views and listeners, the skeleton of the application was created with only the contextual logic itself missing.

Before the contextual based application logic could be created, the application needed the ability to retrieve information from specific mobile sensors in order to determine if any context variables had been changed. Therefore, context widgets, described in section 2.3, needed to be created for each dynamic context variable discussed in section 4.1 including the user location and network signal strength. These context widgets were created using Android software components known as Services, which are used by the Android platform to perform either long-running operations, network communications, and/or collect specific information that can be broadcast to the application itself. The cellular tower identification context widget retrieved cellular tower information whenever the mobile device changed cellular towers. Once the cellular tower identification could be retrieved from the cellular tower itself, the information needed to be sent to the server, which would corresponded the cellular tower identification

number to the appropriate cellular tower. Once the context widget for a cellular tower obtained the new station location, it would notify the PRM application that there has been a change in location and the PRM application would need to retrieve the relevant services for the new station. The second context widget that needed to be created was the network signal strength widget. This widget retrieved the changing signal strength information every time there was an adjustment to the connection. Once the widget retrieved the information, it would inform the PRM application of the new signal strength. The other two context widgets; disability and age, did not need to communicate directly with any sensors, they only needed to retrieve the disability and age attributes from the server on the application's launch. As these variables are static they only needed to be accessed on the initial run-time of the application.

Based on the four context variable implemented by the PRM application, the linear key-value model programming approach used if-else rules in order to determine whether the application should reconfigure any software components. This suggests that whenever a context widget informed the PRM application of new context data, the application needed to check to see if the new context data is tangibly different than the previously retrieved data. If it is than the PRM application would use if-else rules in order to determine which variant of the component the application should reconfigure to. An example of a key-value pair found within this approach can be seen in Expression 4.1.

$$(key = "Bercy\_\text{CellTowerID}", value = 1722549)$$

Expression 4.1 - Sample of key-value pair for PRM application

In order to fully characterize all of the information retrieved from the sensors, every piece of data provided by the context widgets needed to be characterized using the same key-value format

presented in Expression 4.1. Once all the possible key-values pairs were constructed and a context widget retrieved specific sensor-based information, the application must check all of the key-value pairs for a particular component (in this case location) to find out which key the data obtained by the context widget matches. This process was performed through if, else if, and else statements to check for matches. Once a match was found the application needed to reconfigure to the appropriate match. The steps used for creating the PRM application through a linear modeling approach are summarized in Table 4.2.

Table 4.2 - Summarized process of create a linear PRM application

Step	Summary
1	- Construct all client-based user interfaces including the grid and gesture based components (Create all application views)
2	- Create all action listeners used to respond to the user's action and clicks on each of the component found in each user interface (Create all controllers)
3	- Create dynamic context widgets for retrieving connection signal strength and cellular tower identification through Android Services
4	- Create static context widgets for retrieving context variables such as age and disability that do not change often.
5	- Create all key-value pairs for all the components found within the PRM application.
6	- Create simple if-then rules that determine when application reconfiguration need to occur within the application logic

4.2.2 Proposed Ontology Modeling Approach

Unlike the linear modeling approach, the first step of the ADOPT model approach was to design an application ontology using the ADOPT ontology described in Chapter 3. Based on the

requirements described in section 4.1, there were no significant additions to the domain ontology scopes as all of the required contexts; disability, date of birth, location, and network signal strength were already included in the domain ontology. However, there were several minor additions that needed to be added such as the *CellTower* entity; which was added to the *resource* entity type. The *CellTower* entity needed to be added as there was a need to associate the location of a particular station to a *CellTower* identity. In addition, a *CellID* basic scope was added to the *ResourcePlace* basic scope in order to allow the PRM application to connect to the *CellTower* scope and obtain its identification. Also added to the entity type list was the *RelevantService* entity which was used to describe the relevant *services* that a user can view at each station. Finally, a complete list of *Individuals* was added to ADOPT ontology in order to represent all of the objects within the PRM application's domain [73].

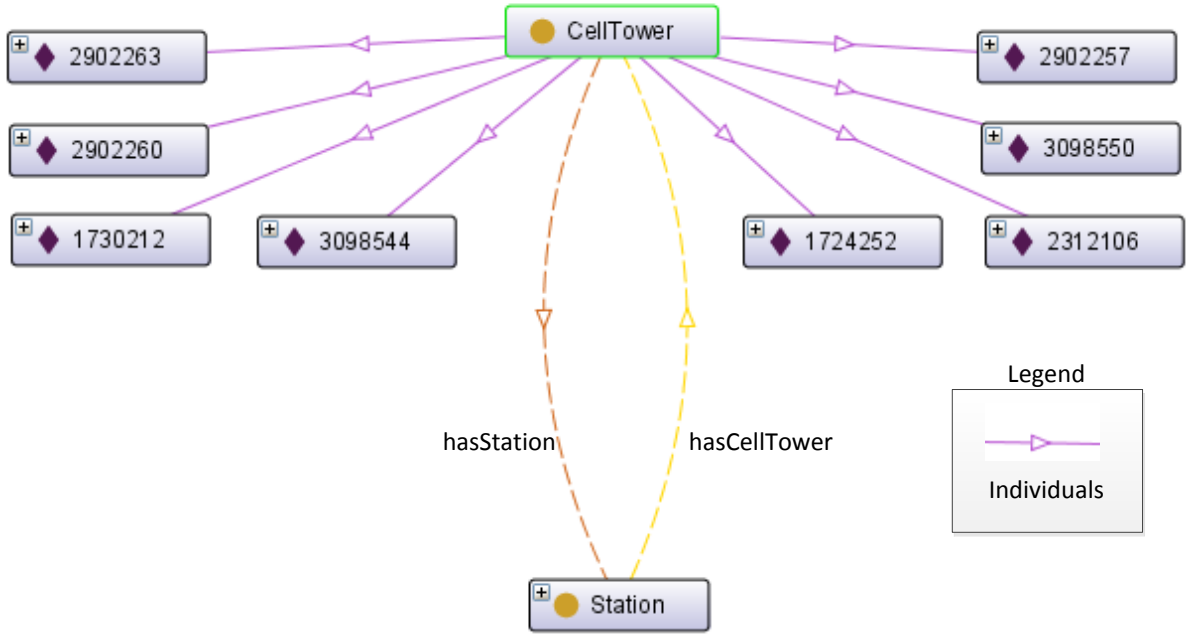

Figure 4.9 - Example of the *hasStation* and *hasCellTower* object properties

The *Individuals* included all instances of stations, cellular towers, and relevant services. In order to link the various individuals to each other, object properties needed to be created for the each entity. Examples of object properties added to the proposed ontology were the *hasCellTower* and *hasStation* properties which link instances of the *Station place* entity type and those of the *CellTower StationPlaceResource* entity. The *hasCellTower* indicates that all *Station* instances in the PRM application domain must be associated to at least one *CellTower* instance while the *hasStation* property represents the inverse of the *hasCellTower* property indicating that all *CellTower* instances must be associated to at least one station. This example can be seen in more detail in Figure 4.9. A sample list of the various individuals and properties that were added to the application ontology can be seen in Table 4.3.

Table 4.3 - Examples of individuals and their properties

Entity Type	Individual	Object Property	Data Property
Station	Bercy	hasCellTower 1730212	
Station	CourSaintEmilion	hasCellTower 3098544	
CellTower	1730212	hasStation Bercy	
CellTower	3098544	hasStation CourSaintEmilion	
RelevantService	RouteMap	hasStation Bercy hasStation CourSaintEmilion	hasServiceName “Route Map”
RelevantService	StationLocation	hasStation Bercy	hasServiceName “Station Location”

Once the ontology for the PRM application was constructed, application logic needed to be created that would allow changes to the scopes (contextual information) to reconfigure specific

components of the application. Within the PRM application there were four main components that were affected by changes to contextual information; user interface, activation of TTS, availability of relevant location based services, and the structure (graphical versus text based) that *services* should be presented in. The user interface, as discussed in section 4.1.1, was comprised of four main variants; *offline, mobility, sight, and standard* which could be reconfigured based on a user's disability and age.

The *offline* variant of the user interface becomes available whenever the mobile device suddenly loses its data connection and the *NetworkStatus* scope returns a "Disconnected" value. The goal of displaying an *offline* variant of the user interface is to display offline content to the user in order to avoid the application from displaying no data as it cannot obtain any of the relevant information *services*. If the application, however, was able to initiate a 3G data connection than one of the user interface variants, based on the disability and age variables, would become available to the user. The architecture for the user interface component variants can be seen in Figure 4.10.

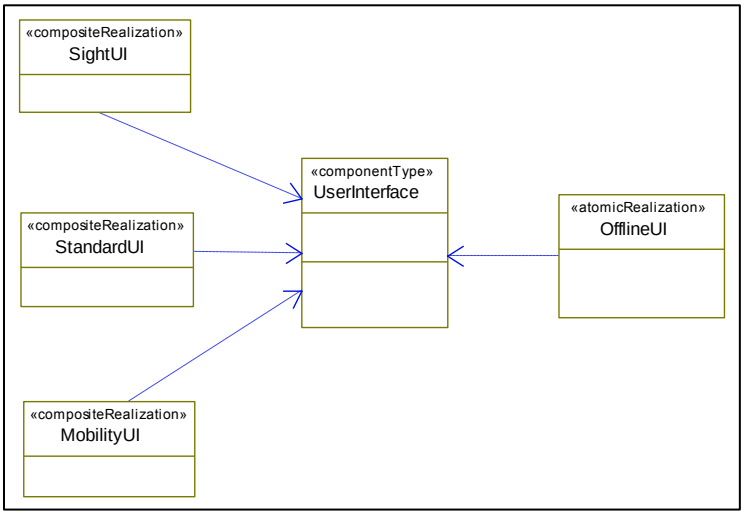

Figure 4.10 - PRM user interface variants

If a mobile device suddenly loses all network connectivity and the network status, as dictated by the *NetworkStatus* scope defined in ADOPT, is set to “Disconnected”, then the PRM application would need to reconfigure in order to display the corresponding variant of the user interface, in this case the *OfflineUI*. Figure 4.11 demonstrates that the *OfflineUI* variant should be made available only when the *NetworkStatus* scope is determined to be “Disconnected”.

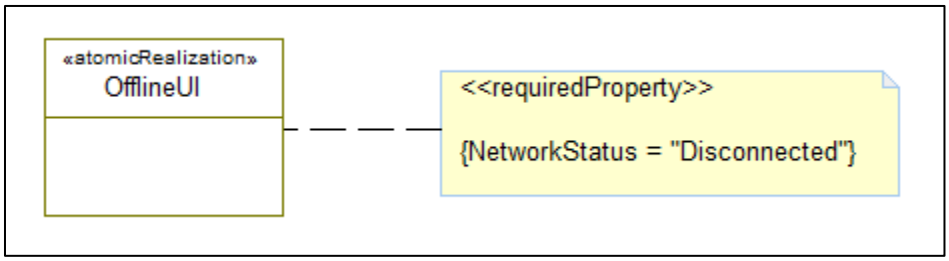

Figure 4.11 - Example of contextual information associated to particular atomic realization

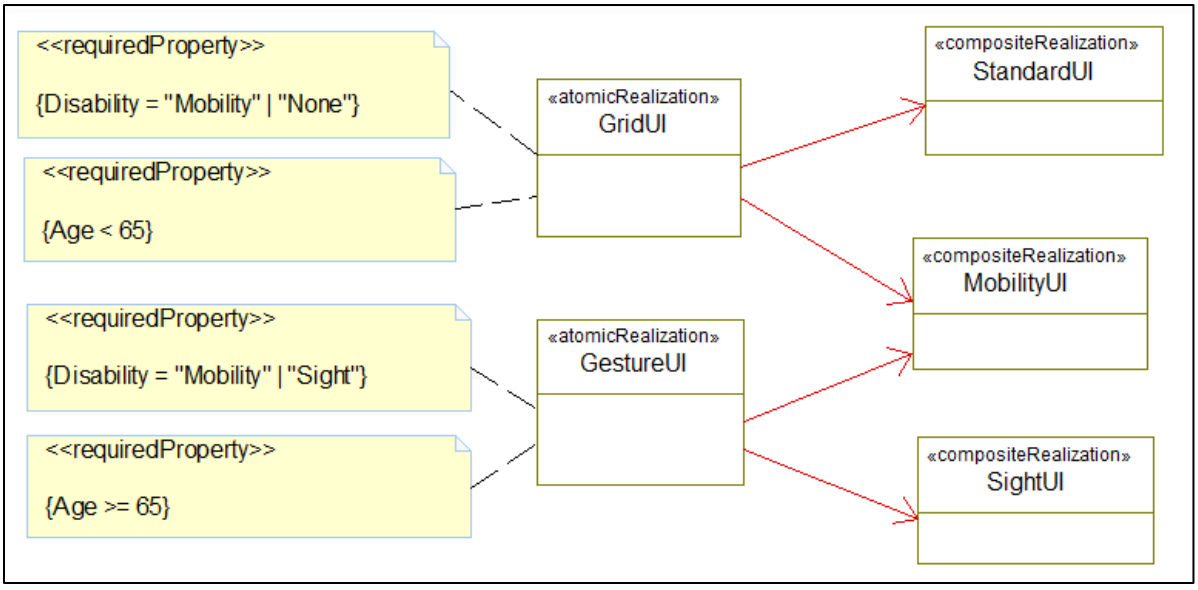

Figure 4.12 - Example of the PRM user interface composite realization

Within the PRM application, the *MobilityUI* composite realization is comprised of the atomic realizations; *GridUI* and *GestureUI* as seen in Figure 4.12. In this example, the *MobilityUI* user interface variant is comprised of both the grid and gesture based layout (described through

atomic realizations in Figure 4.12) which were discussed in section 4.1.1. Based on the requiredProperty constraints, as seen in Figure 4.11 and Figure 4.12, the PRM application is able to determine which user interface variant to display to the user based on which requiredProperty values are retrieved from the context widget. For example, if the disability context widget retrieves a sight disability then the *SightUI* composite realization user interface variant should be displayed to the user as the *GestureUI* component is the only atomic realization that has a disability requiredProperty of "Sight". Figure 4.13 displays composite realizations for the location, signal strength, and TTS components found within the PRM application.

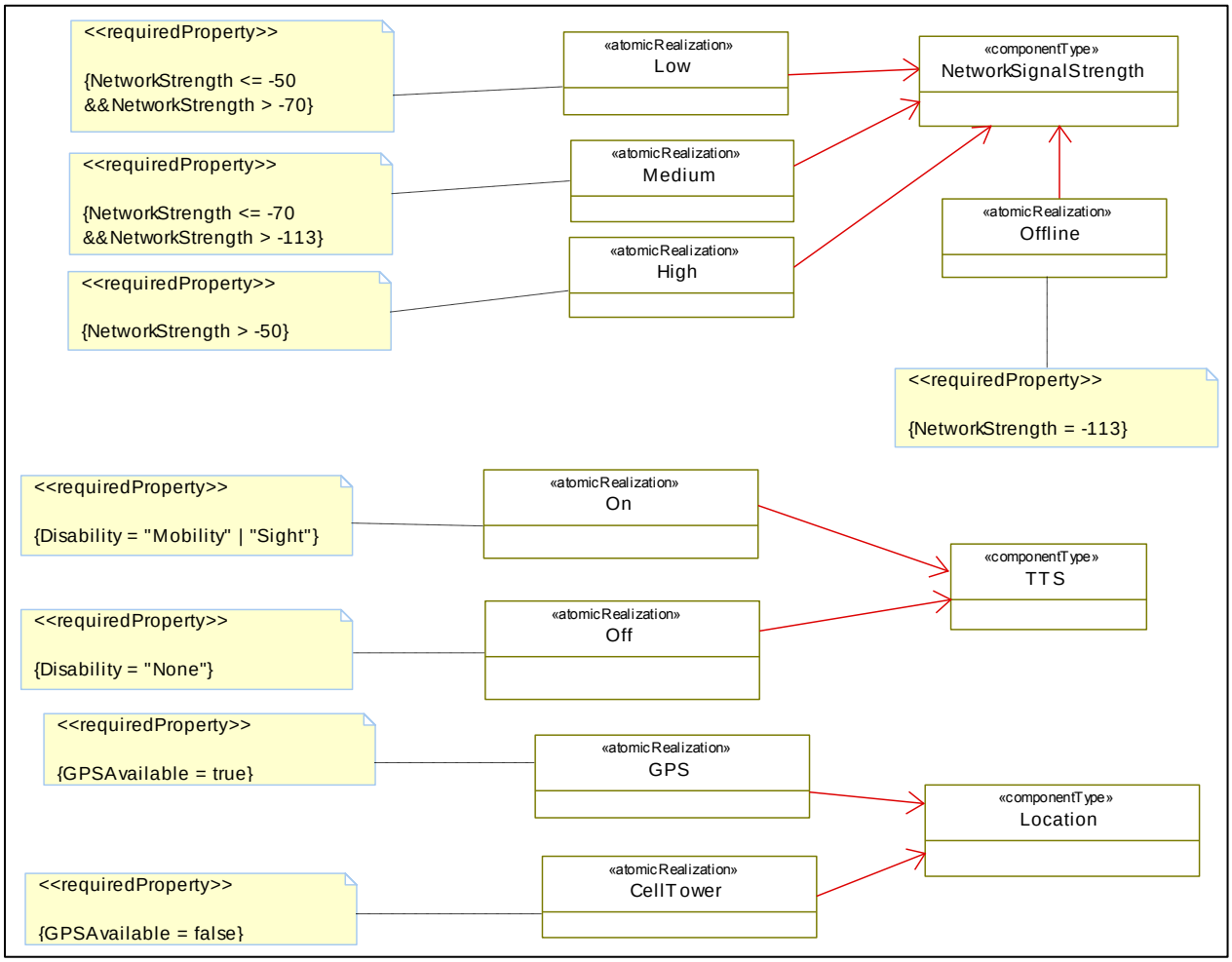

Figure 4.13 - PRM component realizations

Once all the atomic and composite realizations have been created and associated to each variant of a component, the rules for when variants should reconfigure needed to be created. As discussed in section 3.5 this process is achieved through the *utility function*. Once it is determined that a new variant should become available, the PRM application is notified and the application needs to deactivate any previously activated variants and activate the new variant determined by the utility function. A PRM utility function for the user interface component can be seen in Figure 4.14, where based on disability and age contextual information obtained from the corresponding context widget, a specific user interface variant should be displayed to the user.

```
5   double totalUtility = 0.0;
6   double networkUtility = 0.0;
7   double ageUtility = 0.0;
8   double disabilityUtility = 0.0;
9   double maxUtility = 6.0;
10
11  //Determine whether mobile device currently has network connection
12  String network = (String) context.getValue("NETWORKSTATUS");
13  networkUtility = (network.equals(Constant.CONNECTED))? 1.0 : 0.0;
14  if (networkUtility == 0.0)
15      return networkUtility;
16
17  //Determine the age of the mobile device user
18  int age = (Integer) evalContext.evaluate("AGE", context)).intValue();
19  ageUtility = (age >= 65) ? 1.0 : 0.0;
20
21  //Determine the disability of the mobile device user
22  boolean mobilityUtility = ((Boolean) evalContext.evaluate("MOBILITY_DISABILITY", context)).booleanValue();
23  disabilityUtility += (mobilityUtility)? 1.0 : 0.0;
24
25  boolean sightUtility = ((Boolean) evalContext.evaluate("SIGHT_DISABILITY", context)).booleanValue();
26  disabilityUtility += (sightUtility)? 3.0 : 0.0;
27
28  totalUtility = (networkUtility + ageUtility + disabilityUtility) / maxUtility;
29
30  return totalUtility;
```

Figure 4.14 - Sample utility function for determining user interface variants

Using the logic described in Figure 4.14, the utility function returns a value that directly corresponds to a particular user interface variant that an application should reconfigure to. Table 4.4 describes the user interface utility outcome values based on Figure 4.14 and what they indicate to the PRM application.

Table 4.4 - Utility function values and their corresponding user interface variants

Utility Function Outcome	User Interface Variants
0.0	- Indicates that there is no internet connectivity therefore the **OfflineUI** variant of the user interface should be displayed (see lines 12-15 of Figure 4.14).
0.01 – 0.167	- Indicates that the user has no disability and is under the age of 65 therefore the **StandardUI** variant should be displayed to the user.
0.167 – 0.5	- Indicates that the user either has a mobility disability and/or is over the age of 65 (see lines 22-23 of Figure 4.14) and therefore the application should display the **MobilityUI** variant.
0.5 – 1.0	- Indicates that the user has a sight disability (see lines 25-26 of Figure 4.14) and therefore the **SightUI** variant should be displayed.

In addition to Table 4.4, Figure 4.15 demonstrates a graph of all the possible user interface threshold values obtained from Figure 4.14 and which variant each threshold value will represent

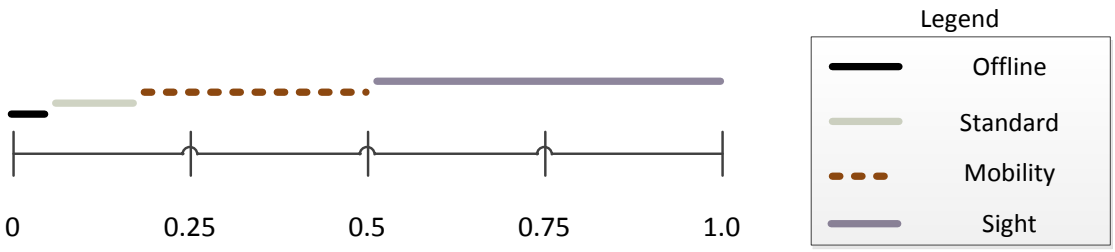

Figure 4.15 - Graph depicting all of the threshold values for the user interface utility

The signal strength utility functions much the same way as the user interface utility, the first action the utility function takes is to determine whether or not the mobile device has a network connection to the Internet.

```
double totalSignalStrenthUtility = 0.0;
double connectionAvailableUtility = 0.0;
double maxUtility = 4.0;

//Determine whether mobile device currently has network connection
String network = (String) context.getValue("NETWORKSTATUS");
connectionAvailableUtility = (network.equals(Constant.CONNECTED))? 1.0 : 0.0;
if (connectionAvailableUtility == 0.0)
    return connectionAvailableUtility;

totalSignalStrenthUtility = connectionAvailableUtility;

//Determine the network signal strength of the Internet connection
int signalStrengthUtility = evalContext.evaluate("SIGNALSTRENGTH", context)).intValue();

//Check to see which range the signal strength matches
if (signalStrengthUtiltiy >= Constant.HIGH_SIGNAL_STRENGTH)
    totalSignalStrenthUtility+=3.0;
else if (signalStrengthUtiltiy > Constant.MEDIUM_SIGNAL_STRENGTH && signalStrengthUtiltiy < Constant.HIGH_SIGNAL_STRENGTH)
    totalSignalStrenthUtility+=2.0;
else if (signalStrengthUtiltiy > Constant.LOW_SIGNAL_STRENGTH && signalStrengthUtiltiy < Constant.MEDIUM_SIGNAL_STRENGTH)
    totalSignalStrenthUtility+=1.0;
else
    totalSignalStrenthUtility+=0.0;

return new Double (totalSignalStrenthUtility/maxUtility);
```

Figure 4.16 - Utility function for signal strength

If it doesn't, it returns the utility value corresponding to the "No Connection" as seen earlier in Table 4.1. If the utility function can obtain a network connection it then needs to determine which range the received signal strength is associated with. The utility function for signal strength is displayed in Figure 4.16. Using the signal strength utility, in Figure 4.16, Table 4.5 displays all the corresponding utility values and what each threshold value reconfigures to. Figure 4.17 demonstrates the signal strength variant with the corresponding threshold values.

Table 4.5 - Signal strength threshold values used for reconfiguration

Utility Function Outcome	Signal Strength Variants
0	- Indicates that there is no internet connectivity therefore the **Offline** variant of a service should be displayed.
0.01-0.5	- Indicates that the signal strength is very weak and the service should only be displayed in a text-based therefore the **Low** variant is displayed.
0.5-0.75	- Indicates that the signal strength is not strong enough to display video based services therefore the service needs to reconfigure to a **Medium** variant with consists of images and text rather than

	videos.
0.76-1.0	- Indicates that the signal strength is strong enough to reconfigure to the **High** variant of the signal strength allowing the service to be displayed in its intended form.

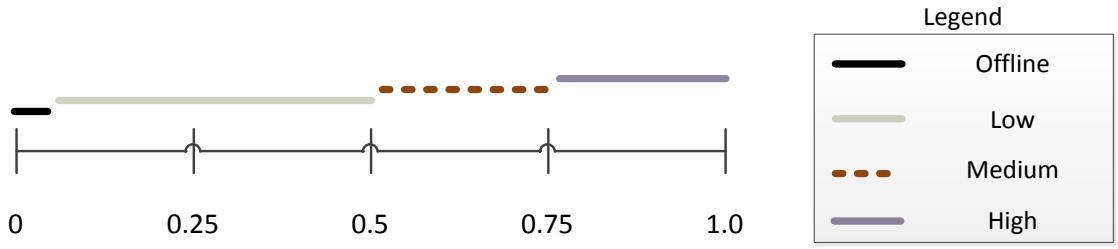

Figure 4.17 - Graph depicting all of the threshold values for the signal strength utility

As an application receives changing utility function values it will automatically activate and deactivate particular components or variants of the application itself. Using ADOPT, the PRM application has a total of four utility functions that the application was registered to. These include the following:

- *User Interface Utility*: indicates which user interface (grid, gesture, or both) should be displayed to the user.
- *TTS Utility*: indicates where or not TTS should be activated for the user.
- *Bandwidth Utility*: indicates where information services should be displayed in a video-based, graphical-based, text-based, or offline format.
- *Location Utility*: indicates which location retriever should be used to obtain a passenger's location, explained in detail in section 3.5 Figure 3.29 and Table 3.2.

These utility functions help the application determine all the contextual reconfigurations that can occur within the PRM application. The entire application model can be seen in Figure 4.18 where all the utility functions and component types are displayed. Each utility function is also associated to various property types that define contextual information that each utility function is allowed to access. This simplified model demonstrates the core functions that all ADOPT-based mobile applications must be built on.

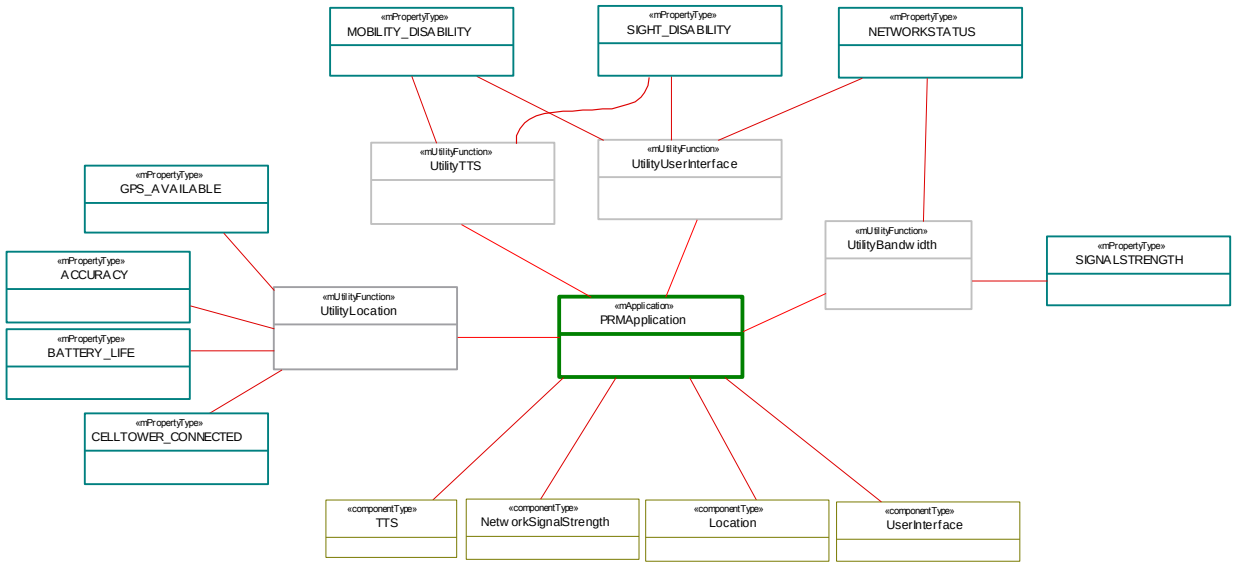

Figure 4.18 - Person with Reduced Mobility Application Architecture

When all of the contextual logic is implemented, the next step of the proposed ontology approach was to create all of the platform-specific code for all of the variants to be fully functional. Similarly to the linear modeling approach all of the variants included in the PRM application were constructed using the Android mobile platform.

To further demonstrate the flexibility of the proposed ADOPT approach, the same process used for the Person with Reduced Mobility mobile application was used to create another travel-based mobile application, GO Mobile [8], for Ontario's GO Transit. Much like the PRM application,

GO Mobile main's contextual variable is the location of the user which is used to inform passengers of nearby stations as well as valuable data about upcoming stations during their travels. The GO Mobile application also accesses the accessibility of the user of the device and renders the appropriate layout of the application to the user in order to suit the passenger's needs. The goal of the application is to improve the travel needs of passengers that travel on GO Transit by offering them information such as upcoming schedules, maps of routes, arrival alerts, and departure information based on your favourite routes. Figure 4.19 displays several screenshots of the GO Mobile application.

Figure 4.19 - Screenshots of GO Mobile

4.3 Results of Linear and Proposed Ontology Modeling Approaches

In this section the results of the linear key-value and ADOPT approaches will be discussed in detail with the advantages of the ADOPT model described in detail. The quantitative results presented in this section will be examined in order to demonstrate the strengths of ADOPT in a context-aware public transportation domain. The PRM case study discussed earlier in this chapter is used as the basis for the results examined in this section.

4.3.1 Testing Environment

In order to obtain the results described in this chapter, a test environment needed to be created to successfully retrieve the necessary experimental data. For the purpose of this study, all data was simulated at the Digital Media Zone (DMZ) laboratory located at Ryerson University. Since a subset of the data retrieved involved a user to move from one station to another, the DMZ was divided into multiple subsections with each subsection representing a unique station on Paris Metro's line 14. Therefore moving from one subsection to another simulated a passenger traveling from one station to another. Both versions, linear key-value and ADOPT, of the PRM application were installed on the Motorola Atrix device, which was used to obtain all of the charted information displayed in this section. The Motorola Atrix used for each test was using Android's Gingerbread (2.3.6) operating system and contained a Dual-core 1 GHz Cortex-A9 processor with 1 GB of RAM.

As the results were obtained from a physical device, the availability of contextual information such as the 3G data connection was dependant entirely on the device itself. All signal strength data was obtained only when the device was connected through a 3G connection, if the device automatically switched over to a 2G or EGDE data connection the experiment was stopped until a 3G data connection was re-established. This was done in order to obtain the most accurate results possible. In order to reduce the factor of data connection speed, all results were calculated from the time that a contextual change occurred to the time that a new variant was reconfigured with all server data transfer time being removed from the results in order to increase the validity of the experiments. Additionally, the only context that was simulated was the location of each station on Paris' line 14 as discussed earlier.

4.3.2 Case Study Results

The goal of calculating quantitative measures is to demonstrate that ADOPT provides a performance improvement of the PRM application over the linear key-value approach. As this study is focused on the topic of context-awareness, all quantitative results were gathered from the time that a contextual change was detected by the context widget to the time that a component was successfully reconfigured. One of the goals of ADOPT was to improve the process of reconfiguring components based on contextual changes retrieved from the sensors within the environment. Therefore, all quantitative results were gathered using the four main case scenarios as described in Table 4.6.

Table 4.6 - Four specific PRM scenarios for which quantitative measure are calculated

Use Case Scenario	Contextual Information	Scenario Example
1	User's Disability and Age	Once the passenger initializes the application, the application should obtain a user's disability and age from a user profile and render the appreciate user interface.
2	User's Location	As a user travels on Paris Metro's Line 14, based on the station they are currently located at, a unique set of relevant information is displayed to the user.
3	Activation/Deactivation of TTS	If the disability of a user is detected as sight or mobility, the Text-To-Speech service should be activated on an application's launch.
4	Signal Strength of 3G Network Connection	When a user activates an information service at a station, based on the 3G signal strength either a high, medium, low, or offline variant of the service is displayed to the user.

In order to demonstrate a specific advantage of the ADOPT approach the quantitative measures that were calculated for the four scenarios described in Table 4.6 were done so using the time metric which indicated the amount of time in milliseconds it took to successfully reconfigure each application component after a contextual change has been detected. For these experiments, the PRM application was initiated only once and each reconfiguration occurred during the application's initial lifecycle. The first measure calculated was the time it took for the PRM application to reconfigure based on the retrieved disability and age by rendering the proper user interface, as described in section 4.1.1. In addition, all experiments defined the range from the minimum time value to the maximum value in order to determine the variation in the number of outcomes. The range of each experiment was calculated using the formula defined in Expression 4.2.

$$Range = Maximum\,Time\,Value - Minimum\,Time\,Value$$

Expression 4.2 - Range calculation

Table 4.7 demonstrates the average time of fifty reconfiguration attempts to deduce a passenger's disability (sight, mobility, both, or none) and age attributes in order to display the appropriate user interface.

Table 4.7 - Average time to initialize the user interface (milliseconds)

User Interface	Linear Key-Value Approach	Linear Key-Value Range	ADOPT	ADOPT Range
Standard	100.12	77	91.18	76
Mobility	139.28	162	112.12	105
Sight	113.68	115	59.32	89

By examining Table 4.7, it is evident that the ADOPT approach provides faster reconfiguration of the appropriate user interface variant. ADOPT reduces the time it takes to configure the user interface by over 8.93% for the standard user interface, 19.50% for the mobility user interface, and 47.82% for the sight user interface, which an average improvement of 25.42% or 30.15 milliseconds for all three user interfaces. Figure 4.20, Figure 4.21, and Figure 4.22 demonstrates the fifty reconfiguration time entries tabulated to obtain the averages displayed in Table 4.7.

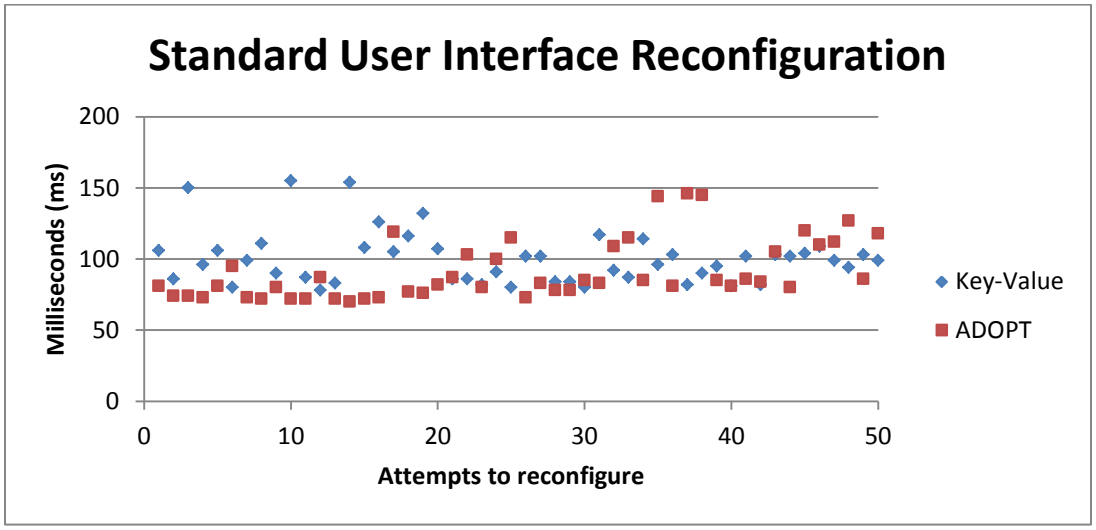

Figure 4.20 - Times in milliseconds of all standard user interface initializations

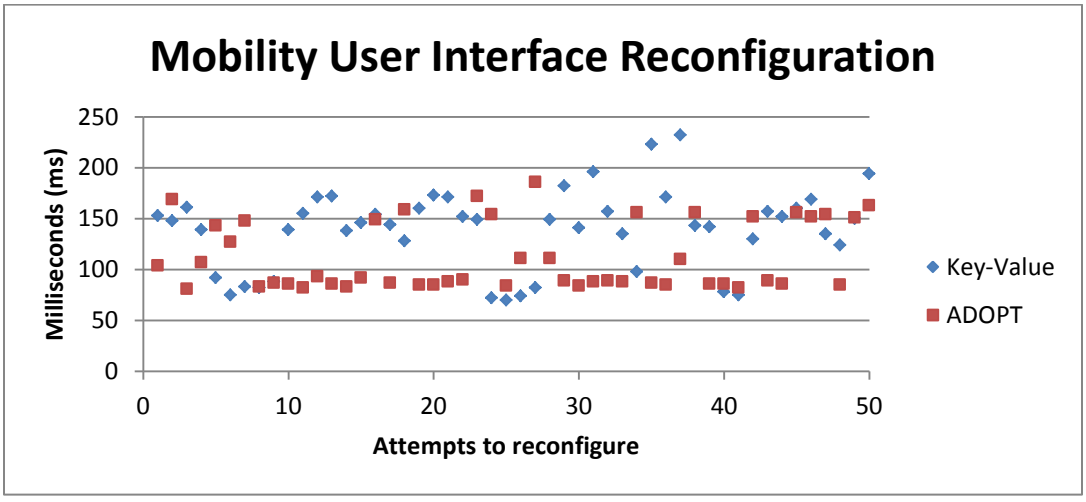

Figure 4.21 - Times in milliseconds of all mobility user interface initializations

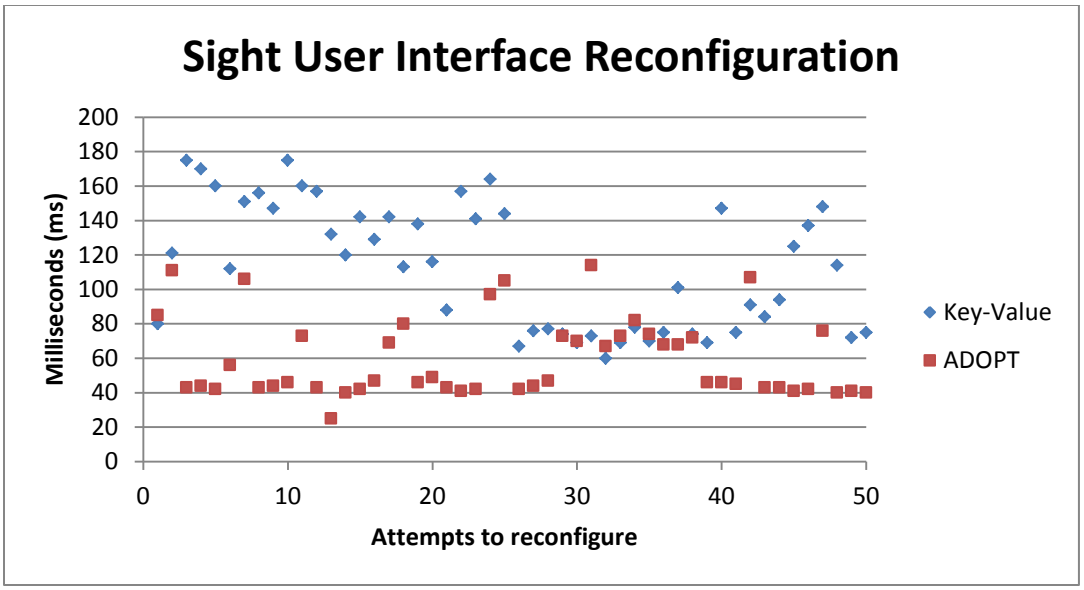

Figure 4.22 - Times in milliseconds of all sight user interface initializations

The second measure where time was used was the reconfiguration of relevant information services based on the station that a user is currently located at. The list of relevant *services* is unique to each station on line 14 of the Paris Metro, therefore as soon as the application detects a change in cellular tower identification or GPS coordinates, the new station's list of *services* needed to be updated in a timely manner as the Paris Metro train was recorded to only remain at each station for an average of 30 seconds. Table 4.8 demonstrates the duration it took for the PRM application to retrieve a station's *service* set once it has detected that the user is inside a new station.

Table 4.8 - Average time to retrieve and display the relevant service at various Paris metro stations (milliseconds)

Linear Key-Value	Linear Key-Value Range	ADOPT	ADOPT Range	Percentage Difference
29.1	48	25.34	27	+12.92%

Table 4.8 demonstrates that the average of fifty location-based changes and provides an advantage by an average of over 3.76 milliseconds that it takes for the ADOPT approach to retrieve and render the relevant services after there has been a change detected to the user's location.

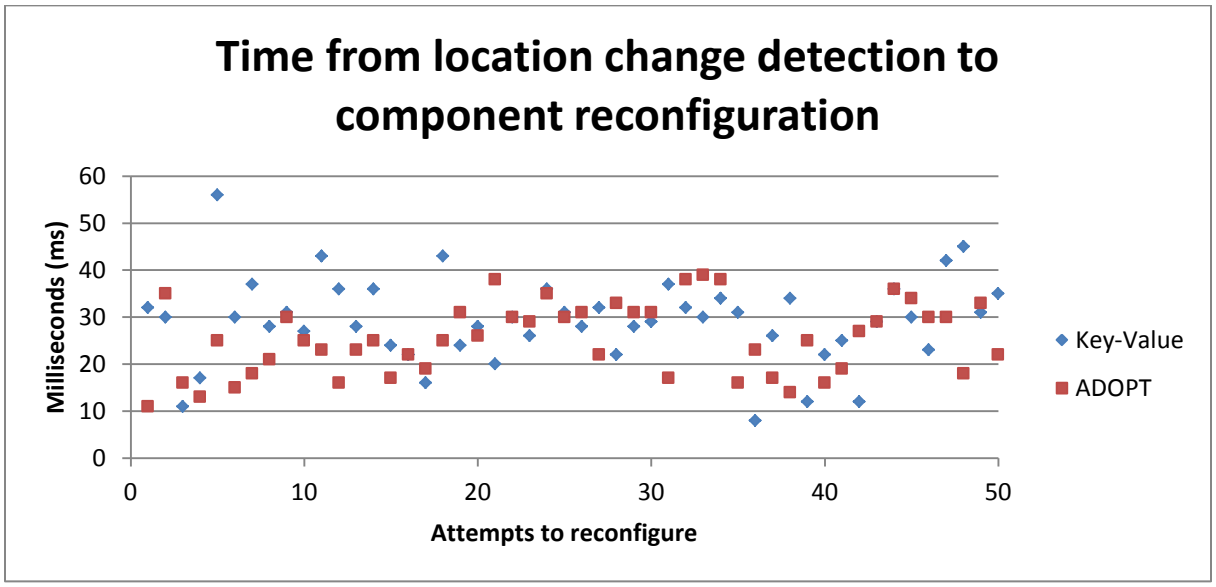

Figure 4.23 - Time taken in milliseconds for the location component to reconfigure

Figure 4.23 demonstrates fifty different attempts for the mobile device to obtain new cellular tower identification or GPS coordinates to the time that it took to render the relevant *services*. Table 4.9 demonstrates the time that it took for both approaches of the PRM application to detect a disability and activate/deactivate the Text-To-Speech (TTS) component on an Android based mobile device. As can be observed from Table 4.9, there is an average reduction of 69.22% in the amount of time it took for the proposed ontology approach to determine if either the sight or mobility disability was present and therefore activate the TTS component.

Table 4.9 - Average time for TTS activation/deactivation (milliseconds)

TTS	Linear Key-Value	Linear Key-Value Range	ADOPT	ADOPT Range
Activated	45.75	62	14.08	28
Deactivated	Negligible	Negligible	Negligible	Negligible

The deactivation of the TTS component took under a millisecond for both approaches as the only time the TTS service should be disabled is when the PRM application is turned off. During this time there is no logic that determines if the TTS component should be deactivated, it is simply turned off because the application is no longer in use.. Figure 4.24 further demonstrates the advantage of ADOPT as it displays all seventy five attempts at activating the TTS component.

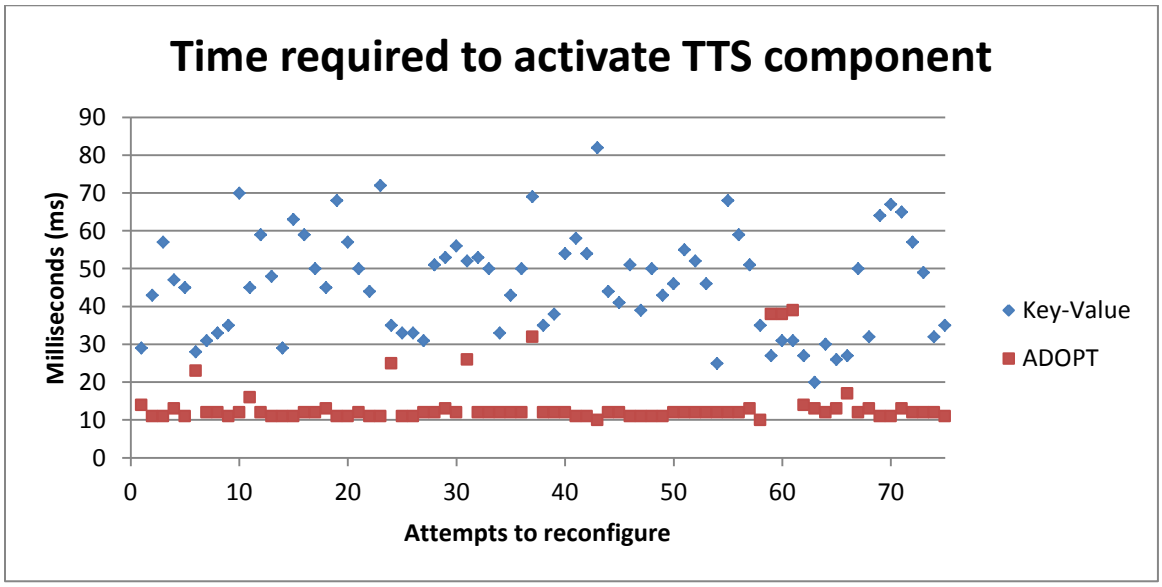

Figure 4.24 - Minimum time needed for TTS component to reconfigure upon obtaining disability context

The final time attribute calculated by the PRM application was the time it took for an individual *service* to reconfigure based on a detected change in the retrieved signal strength based on the

specific strength thresholds discussed earlier. As Table 4.10 demonstrates, there is an improvement using the ADOPT approach compared to that of the linear key-value approach by over 29.62% when a *service* is reconfigured from a "high" signal strength variant to a "low" signal strength variant. There is also a 40.87% reduction in the amount of time it took for the ADOPT approach to successful reconfigure the displayed *service* from a "low" signal strength variant to a "high" signal strength variant. The tabulated results of Table 4.10 can be seen in Figure 4.25 and Figure 4.26.

Table 4.10 - Average time for reconfiguration of low and high signal strength services (milliseconds)

Reconfiguration	Linear Approach	Proposed Ontology Approach
High to Low	2269.77	1597.37
Low to High	4369.73	2583.9

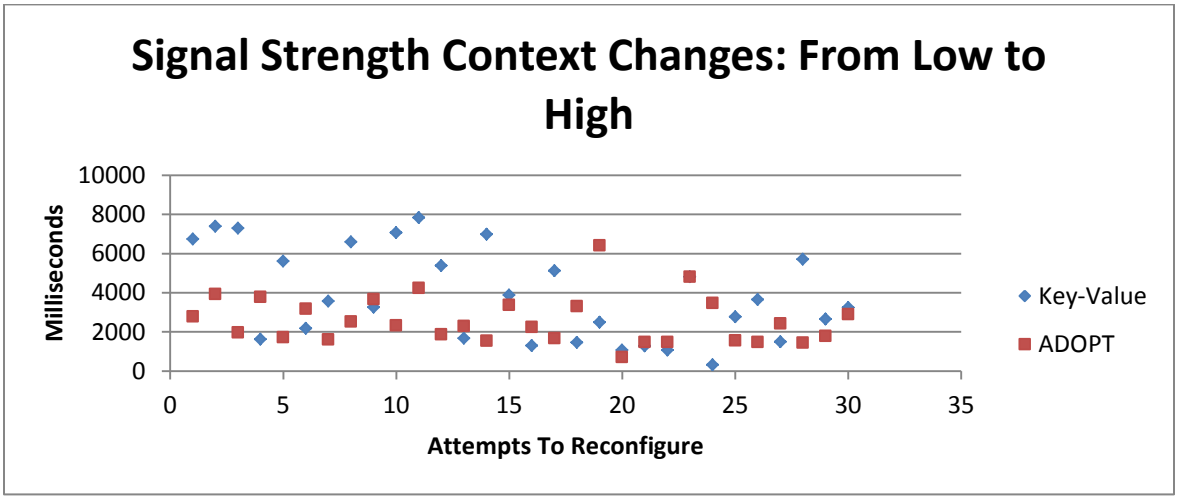

Figure 4.25 - Charted time to reconfigure a service from low to high signal strength

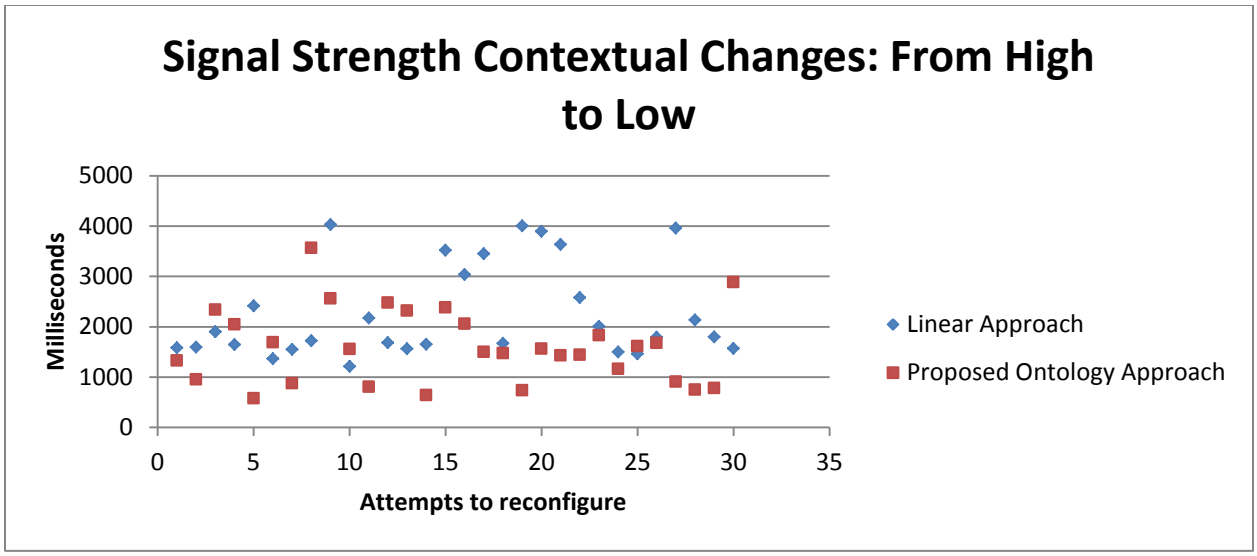

Figure 4.26 - Charted time to reconfigure a service from high to low signal strength

4.3.3 Focus Group Results

The results that appear in this section were obtained from a three-person developer survey of Android application developers that vary in experience; including a beginner, intermediate, and an expert in Android application development. The goal of survey was to access the benefits of the ADOPT solution over the linear key-value approach. The first task that was requested of the participants was to implement their own context widgets for each dynamic context variable found in the PRM application. All three participants were required to use Android Services in order to implement each of the context widget. If the participant was unsure of how to create a service that obtains specific sensor data, they were allowed to research the topic before starting each task with the research time recorded by each developer. For the purpose of this experiment, the developers were asked to perform two unique tasks:

- *Task 1*: Create a service that broadcasts the identification number of the cellular tower the phone is currently connected to. When there is a change in cellular tower id, the service

should broadcast the new cell tower id to any application listening to that specific broadcast.

- *Task 2*: Create a service that broadcasts the signal strength of the 3G connection of your phone. When the signal strength changes, the service needs to broadcast the new signal strength to any listening applications.

Through these two tasks the participants would familiarize themselves with the need to create context widgets, regardless of the application these widgets were being built for. Once these two tasks were completed the participants had created mechanisms to retrieve dynamic contextual information and broadcast that information to an application. Using this broadcast information an application would then have the ability to initiate the appropriate reconfiguration within the application.

Following the completion of the task 1 and task 2 the participants were given two more application scenarios and asked how they would obtain the necessary contextual information. The two additional applications were:

- *Application 1*: A music streaming application that downgrades or upgrades the quality of the music based on the data connection strength.
- *Application 2*: An application that notifies the user that their device is currently in roaming mode due to the fact it is not connected to one of their network provider's cellular towers.

Using these two new application scenarios the participants were asked how they obtain the necessary contextual information needed to create both application. In response, all three participants respond that they would reuse the two services they created in task 1 and 2, in order

to retrieve the necessary information. Based on this observation there is a clear need to create reusable contextual information that can be used regardless of the application that a developer intends to create. Although the participants were not required to build each requested application the notion of reusing contextual information was apparent.

Through this example another benefit of ADOPT is evident; it provides the ability to create pluggable context widgets that can be obtained directly from the scopes that are implemented within both the domain-specific and application ontologies. Once a user has implemented a scope into their application they will have access to the corresponding widget that notifies the application of any contextual changes. Therefore whenever an application developer incorporates ADOPT into their application they also incorporate the repository of all the context widgets that can be associated to a corresponding scope. The benefit of such a feature is that it reduces the research time that application developers need to obtain the necessary knowledge needed to retrieve specific contextual information, such as the 3G signal strength from the mobile platform's system resources. Table 4.11 and Table 4.12 demonstrate the time it took for each participant in the study to research each task and implement them. By using ADOPT, this time would be negligible as all the information about the various contextual information (scopes) is included within the system structure.

Table 4.11 - Time, in minutes, it took each participant to research and implement task 1

Participant	Research Time	Implementation Time
Participant A (Beginner)	45	20
Participant B (Intermediate)	17	13
Participant C (Expert)	10	8

Table 4.12 - Time, in minutes, it took for each participant to research and implement task 2

Participant	Research Time	Implementation Time
Participant A (Beginner)	0	10
Participant B (Intermediate)	2	4
Participant C (Expert)	4	6

Chapter 5. Conclusion

5.1 Contributions

The work presented in this thesis proposed a framework, ADOPT, designed for creating context-aware public transportation mobile applications. The goal of ADOPT was to construct a framework that allows application developers to create mobile applications, designed for the public transportation domain, in a quick and efficient manner. Through the presented framework, application developers will be able to better understand all of the concepts and their relationships within the public transportation domain in order to reduce the amount of time needed to research and implement these concepts within the public transportation domain. Within ADOPT, the proposed domain ontology is constructed using hundreds of different concepts that can be found within any public transportation domain, through these concepts, application developers will save both research and implementation time by quickly understanding their desired domain. By understanding their desired domain, application developers will know which sensor-based information their applications will need. The framework also has the ability to allow application developers to introduce their own contextual information either from a combination of already pre-defined scopes or completely new concepts that will perhaps be available in the future.

Through ADOPT, application developers have the opportunity to create dynamic mobile applications that use various environment embedded sensors along with any user-defined attributes to provide public transportation passengers with a more personalized mobile application. In order to personalize an application, software components must be able to reconfigure based on the passenger's current needs and the ever-changing environment variables that surround them. These embedded sensors communicate with various applications by informing them of changing contextual data. Based on this changing contextual data,

applications designed using ADOPT can reconfigure various software components in order to enhance the user's travel experience. Enhancing the travel experience can include reconfiguring the user interface based on the passenger's disability, providing relevant travel information depending on the location of the user, and automatically informing the user if there is a delay on their route.

Another goal of the proposed ontology based framework was to improve the flexibility that application developers have when customizing a mobile application for a specific need. Through the use of variants and components, application developers are able to define under which contextual conditions a specific software component should be reconfigured in order to enhance the experience of a public transportation user. By personalizing the experience for the user it allows both application developers and public transportation operators the ability to reach more people who have specific travel needs that might not be met by generalized applications. As applications need to react quickly and efficiently the proposed ADOPT-based solution needed to also demonstrate an enhancement in the amount of time that it took for a software reconfiguration to occur. By using the ADOPT system, applications demonstrated an average improvement of 27.64% when compared to linear key-value models, therefore proving that the ADOPT-based solution is in fact a superior method of development.

Therefore, the main contribution of the proposed solution was to create an ontological system that allows application developers to quickly design public transportation applications that can reconfigure software components based on the changing contextual information received from sensors embedded within the public transportation environment. The goal of reconfiguring various application components is to enhance the user experience of someone who is using public transportation as a means of traveling from one location to another.

5.2 Limitations

Although the proposed ontological solution provides an enhancement over a linear key-value solution, limitations in the proposed approach are present. The largest limitation of ADOPT is that once an application ontology has been constructed for a particular domain and scenario, with all of the scopes and entity types defined, the defined contexts cannot be changed or altered by the application developer therefore creating a static environment. This becomes apparent when the application developer of the PRM application decides to remove the ability of reconfiguring the user interface based on the disability attribute of the passenger. If the developer was to remove this from the application ontology and redeploy a new ontology the application would not be able to display a user interface as the application logic will still try to search for the previously defined scopes that are associated to the user interface. Currently the only solution to this problem is to remove all the code associated to the application's user interface and recompile the application. However, this is not an optimal solution as the mobile device would need to update the new application code base before obtaining the new application ontology. Unfortunately, mobile device users do not always update their applications in a timely manner therefore providing a new application ontology would in fact deteriorate the functionality of the application as it will not understand how to process the missing scopes and entity types. This problem persists within the entire realm of ADOPT and therefore changes to a deployed application ontology have been disallowed to avoid inconstancies within the code.

The second major limitation is that all context sensor widgets need to be manually created before the ontology is constructed. As of the time of this document's creation not all context widgets that correspond to the scopes defined in the domain specific ontology have not been created due in part to the sheer size of the ADOPT ontology as it is comprised of over fifty basic scopes and

twenty-five composite scopes. Additionally context widgets are not platform independent, hence to use this system across any platform still require platform-dependent context widgets to be created for each desired mobile platform.

5.3 Future Research

The majority of the future work should be spent on solving the limitations presented in the previous section. Firstly, creating a more dynamic ontology model would greatly enhance the usability of ADOPT for aspiring application developers, as it would provide them with the ability to enhance their applications at a future time. Creating a more dynamic ontology structure would also further enhance the overall user experience for users as the inclusion/exclusion/modification of various reconfigurable software components can be achieved without requesting the application user to re-download a new version of the application. By achieving this feature, ADOPT based applications would provide a more ubiquitous and transparent experience to the end user. In order to construct a more dynamic development environment several areas of research need to be investigated on a further basis. It is the opinion of the author of this thesis that the following research areas would allow developers to further improve on the limitations of the proposed ADOPT framework;

- *Topic 1*: Somehow indicate, within the domain ontology, which ontological concepts have been added/removed/modified from a previous version.
- *Topic 2*: The creation of a client-based library that will be able to understand which ontological concepts have been added/removed/modified and according activate/deactivate the necessary component types. In short, improving the connection of ontological concepts to the client software components.

- *Topic 3*: The separation of each software variants from the component types themselves. Currently all components and their variants are closely linked so the removal of one variant will cause the entire component to malfunction.
- *Topic 4*: The transfer of external contextual models to the application ontology layer. Currently each piece of contextual information needs to be defined through both the application ontology and then a context model in order for the utility function to obtain the necessary information. However, as the information is identical, the utility function can obtain the necessary contextual information directly from the scopes instead of having the developers re-define their application's desired context models.
- *Topic 5*: The creation of dynamic scopes. Currently all scopes are pre-defined by the application developers and the proposed domain ontology, meaning that the LocationGeo composite scope is always be made up of the same basic scopes such as Latitude and Longitude. However, if under specific circumstances the application developer wants to include the Altitude basic scope as well they must create a new composite scope entirely and redeploy the ontology again, but by making the ontological scopes more flexible, it allows application developers to fully use the capabilities of the utility function.

Through these five research areas, future development of a more dynamic development environment will be made easier as the burden placed on application developers will be greatly diminished. Additionally creating platform independent context widgets would greatly improve the development time of ADOPT-based application, however there is currently very limited similarities between the various mobile platforms, therefore to enhance ADOPT, a transformation tool that automatically generates context widgets for a particular sensor would need to be constructed. The proposed transformation tool would greatly benefit not only the

proposed ADOPT solution based application but also any application that a user might want to build regardless of the platform they are developing for.

Appendix 1. List of Acronyms

ADOPT – context-Aware Domain Ontology-based framework for Public Transportation

MUSIC – self-adapting applications for Mobile USers In ubiquitous Computing environments

RFID – Radio Frequency IDentification

KVM – Key-Value Model

UML – Unified Modeling Language

ORM – Object-Role Model

LBM – Logic-Based Model

SVO – Subject Verb Object

OWL – Web Ontology Language

RDF – Resource Description Framework

CONON – CONtext ONtology

COBRA – COntext BRoker Architecture

SOCAM – Service-Oriented Context-Aware Middleware

PACE – Pervasive, Autonomic, Context-aware Environments

PSCM – Peer-to-peer Context Sharing Model

CSM – Context Sharing Messages

ICT – Information and Communication Technologies

ITS – Intelligent Transportation Systems

APTS – Advanced Public Transportation Systems

ATMS – Advanced Traffic Management Systems

ATIS – Advanced Traveler Information Systems

V2V – Vehicle-to-Vehicle

UTS – Ubiquitous Transportation Systems

PDA – Personal Digital Assistant

SVS – Smart Vehicle Space

MDD – Model-Driven Development

SoS – System of Systems

ASC – Aspect Scale Context

NFC – Near Field Communication

PRM – Person with Reduced Mobility application

TTS – Text-To-Speech

GPS – Global Positioning System

DMZ – Digital Media Zone

References

[1] X. Wang, J. S. Dong, C. Y. Chin, S. R. Hettiarachchi and D. Zhang, "Semantic Space: an infrastructure for smart spaces," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 32-39, 2004.

[2] J. M. Neighbors, "The Draco approach to constructing software from reusable components," in *Readings in artificial intelligence and software engineering*, San Francisco, Morgan Kaufmann Publishers Inc, 1984, pp. 525-535.

[3] J. Neff and L. Pham, "A profile of public transportation passenger demographics and travel characteristics reported in on-board surveys," American Public Transportation Association, Washington, 2007.

[4] M. Tweed, "Study on transit in Canada: Report of the standing committee on transport, infrastructure and communities," Public Works and Government Services Canada, Ottawa, 2012.

[5] M. V. Vugt, P. A. M. V. Lange and R. M. Meertens, "Commuting by car or public transportation? A social dilemma analysis of travel mode judgements," *European Journal of Social Psychology*, vol. 26, no. 3, pp. 373-395, 1996.

[6] D. Wise, " Public transportation: federal role in value capture strategies for transit Is limited, but additional guidance could help clarify policies," Diane Publishing, 2010.

[7] M. Foth and R. Schroeter, "Enhancing the experience of public transport users with urban screens and mobile applications," in *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, Tampere, Finland, pp. 33-40, 2010.

[8] "Google Play: Go Mobile Application," Retrieved July 15, 2012 from https://play.google.com/store/apps/details?id=com.ryerson.go.

[9] "iTunes: MiWay," Retrieved July 15, 2012 from http://itunes.apple.com/ca/app/id423819918?mt=8&ign-mpt=uo%3D4.

[10] "Google Play: RATP Subway Paris," Retrieved July 15, 2012 from https://play.google.com/store/apps/details?id=com.fabernovel.ratp.

[11] L. Magrini, M. Nati and E. Panizzi, "RMob - a mobile app for real time information in urban," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, Capri Island, Italy, pp. 776-777, 2012.

[12] J. Biagioni, T. Gerlich, T. Merrifield and J. Eriksson, "EasyTracker: automatic transit tracking, mapping, and arrival time prediction using smartphones," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, Seattle, pp. 68-81, 2011.

[13] B. Ferris, K. Watkins and A. Borning, "OneBusAway: results from providing real-time arrival information for public transit," in *Proceedings of the 28th international conference on Human factors in computing systems*, Atlanta, pp. 1807-1816, 2010.

[14] S. Carmien, M. Dawe, G. Fischer, A. Gorman, A. Kintsch and J. J. F. Sullivan, "Socio-technical environments supporting people with cognitive disabilities using public transportation," *ACM Transactions on Computer-Human Interaction,* vol. 12, no. 2, pp. 233-262, 2005.

[15] P. Baumann, "User context recognition for navigation systems in public transportation," in *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, Lugano, pp. 552-553, 2012.

[16] R. Reichle, M. Wagner, M. U. Khan, K. Geihs, J. Lorenzo, M. Valla, C. Fra, N. Paspallis and G. A. Papadopoulos, "A comprehensive context modeling framework for pervasive computing systems," in *Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, Oslo, pp. 281-295, 2008.

[17] "IST-MUSIC: Context-aware self-adaptive platform for mobile applications," Retrieved July 15, 2012 from http://ist-music.berlios.de/site/.

[18] M. Weiser, "The computer for the 21st century," *Scientific American*, pp. 94-104, 1991.

[19] M. Baldauf, S. Dustdar and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing,* vol. 2, no. 4, pp. 263-277, 2007.

[20] B. Schilit and M. Theimer, "Disseminating active map information to mobile hosts," *IEEE Network,* vol. 8, no. 5, pp. 22-32, 1994.

[21] N. Ryan, J. Pascoe and D. Morse, "Enhanced reality fieldwork: the context-aware archaeological assistant," in *Computer Applications in Archaeology*, Oxford, British Archaeological Reports, 1997, pp. 34-45.

[22] A. K. Dey, G. D. Abowd and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction,* vol. 16, no. 2, pp. 97-166, 2001.

[23] D. Salber, A. K. Dey and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems,* Pittsburgh, pp. 434-441, 1999.

[24] P. Brown, "The stick-e document: a framework for creating context-aware applications," in *Proceedings of the Conference on Electronic Publishing and document manipulation*, Palo Alto, pp. 259-272, 1996.

[25] M. F. Mokbel and J. J. Levandoski, "Toward context and preference-aware location-based services," in *Proceedings of the Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, Providence, pp. 25-32, 2009.

[26] J. Indulska and P. Sutton, "Location management in pervasive systems," in *Proceedings of the Australasian information security workshop conference on ACSW frontiers*, Adelaide, pp. 143-151, 2003.

[27] K. Henricksen and J. Indulska, "Modelling and using imperfect context information," in *Second IEEE International Conference on Pervasive Computing and Communications. Workshop on Context Modelling and Reasoning*, Orlando, pp. 33-37, 2004.

[28] A. Ferscha, S. Vogl and W. Beer, "Context sensing, aggregation, representation and exploitation in wireless networks," *Scalable Computing: Practice and Experience*, vol. 6, no. 2, pp. 77-81, 2005.

[29] A. M. Bernardos, P. Tarrío and J. R. Casar, "A data fusion framework for context-aware mobile services," in *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Seoul, pp. 606-613, 2008.

[30] S. Sigg, D. Gordon, G. v. Zengen, M. Beigl, S. Haseloff and K. David, "Investigation of context prediction accuracy for different context abstraction levels," *Mobile Computing*, vol. 99, pp. 1-14, 2011.

[31] J.-Z. Sun, J. Sauvola and J. Riekki, "Application of connectivity information for context interpretation and derivation," in *Proceedings of the 8th International Conference on Telecommunications*, Zagreb, pp. 303-310, 2005.

[32] A. Dey, "Understanding and Using Context," *Personal Ubiquitous Computing*, vol. 5, no. 1, pp. 4-7, 2001.

[33] A. Schmidt, M. Beigl and H.-W. Gellersen, "There is more to context than location," *Computers and Graphics*, vol. 23, no. 6, pp. 893-901, 1999.

[34] D. Hong, H. R. Schmidtke and W. Woo, "Linking context modelling and contextual reasoning," in *Proceedings of the 4th International Workshop on Modelling and Reasoning in Context*, Roskilde, pp. 37-48, 2007.

[35] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive Mobile Computing*, vol. 6, no. 2, pp. 161-180, 2010.

[36] D. Zhang, M. Guo, J. Zhou, D. Kang and J. Cao, "Context reasoning using extended evidence theory in pervasive computing environments," *Future Generation Computer Systems*, vol. 26, no. 2, pp. 207-216, 2010.

[37] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning And Management*, Nottingham, pp. 33-40, 2004.

[38] J. F. Sowa, "Conceptual Structures Summary," in *Conceptual Structures: Current Research and Practice*, Ellis Horwood, 1992, pp. 3-51.

[39] K. Henricksen, J. Indulska and A. Rakotonirainy, "Generating context management infrastructure from high-level context models," in *Proceedings of the 4th International*

Conference on Mobile Data Management (MDM) - Industrial Track, Melbourne, pp. 1-6, 2003.

[40] T. Halpin, Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design, San Francisco: Morgan Kaufman Publishers, 2001.

[41] A. Ranganathan and R. H. Campbell, "An infrastructure for context-awareness based on first order logic," *Personal Ubiquitous Computing*, vol. 7, no. 6, pp. 353-364, 2003.

[42] V. Kabilan, Ontology for information systems (04IS) design methodology: conceptualizing, designing and representing domain ontologies, KTH, 2007.

[43] T. Strang, C. Linnhoff-Popien and F. Korbinian, "CoOL: A context ontology language to enable contextual interoperability," in *Proceedings of the 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, Paris, pp. 236-247, 2003.

[44] X. H. Wang, D. Q. Zhang, T. G. Gu and H. K. Pung, "Ontology based context modeling and reasoning using OWL," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Orlando, pp. 18-22, 2004.

[45] H. Chen, T. Finin and A. Joshi, "An ontology for context-aware pervasive computing environments," *The Knowledge Engineering Review*, vol. 18, no. 3, pp. 197-207, 2003.

[46] T. Gu, H. K. Pung and D. Q. Zhang, "A middleware for building context-aware mobile services," in *Proceedings of IEEE Vehicular Technology Conference*, Milan, pp. 2656-2660, 2004.

[47] M. Kirsch-Pinheiro, Y. Vanrompay, K. Victor, Y. Berbers, M. Valla, C. Fra, A. Mamelli, P. Barone, X. Hu, A. Devlic and G. Panagiotou, "Context grouping mechanism for context distribution in ubiquitous environments," in *Proceedings of the OTM 2008 Confederated International Conferences*, Monterrey, pp. 571-588, 2008.

[48] K. Henricksen, J. Indulska, T. McFadden and S. Balasubramaniam, "Middleware for distributed context-aware systems," in *Proceedings of On the Move to Meaningful Internet Systems*, Agia Napa, pp. 846-863, 2005.

[49] J. Ye, J. Li, Z. Zhu, X. Gu and H. Shi, "PCSM: a context sharing model in peer-to-peer," in *International Conference on Convergence Information Technology*, Gyeongju, pp. 1868-1873, 2007.

[50] A.-U.-H. Yasar, Y. Vanrompay, D. Preuveneers and Y. Berbers, "Optimizing information dissemination in large scale mobile peer-to-peer networks using context-based grouping," in *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems*, Madeira Island, pp. 1065-1071, 2010.

[51] L. Han, S. Jyri, J. Ma and K. Yu, "Research on context-aware mobile computing," in *Proceedings of 22nd International Conference on Advanced Information Networking and Applications*, Ginowan, pp. 24-30, 2008.

[52] D. Shank and D. Roberts, "Assessment of ITS benefits-results from the field," in *Proceedings of the 1996 Annual Meeting of ITS America.*, Houston, pp. 740-749, 1996.

[53] X. Yang and X. Zhou, "Conceptual study on evaluation of advanced public transportation systems," in *Proceedings of 2003 IEEE Intelligent Transportation Systems*, Shanghai, pp. 1683-1687, 2003.

[54] J. L. Kay, "Advanced traffic management systems - an element of intelligent vehicle-highway systems," in *Proceedings of the International Congress on Transportation Electronics*, pp. 73-84, 1990.

[55] S. Shekhar and D.-R. Liu, "Genesis and advanced traveler information systems (ATIS): killer applications for mobile computing," in *Mobile Computing*, Springer, 1994, pp. 699-723.

[56] M. Torrent-Moreno, J. Mittag, P. Santi and H. Hartenstein, "Vehicle-to-vehicle communication: fair transmit power control for safety-critical information," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 7, pp. 3684-3703, 2009.

[57] M. Foth and R. Schroeter, "Enhancing the experience of public transport users with urban screens and mobile applications," in *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, Tampere, pp. 33-40, 2010.

[58] H. Karvonen, "Different aspects of trust in ubiquitous intelligent transportation systems," in *Proceedings of the 28th Annual European Conference on Cognitive Ergonomics*, Delft, pp. 311-314, 2010.

[59] J. Kjeldskov, S. Howard, J. Murphy, J. Carroll, F. Vetere and C. Graham, "Designing TramMateña context-aware mobile system supporting use of public transportation," in *Proceedings of the 2003 conference on Designing for user experiences*, San Francisco, pp. 1-4, 2003.

[60] E. Lee, K. Ryu and I. Paik, "A concept for ubiquitous transportation systems and related development methodology," in *Proceedings of the 11th International IEEE Conference on Intelligent Transportation Systems*, Beijing, pp. 37-42, 2008.

[61] M. Bertolotto, G. O'Hare, R. Strahan, A. Brophy, A. Martin and E. McLoughlin, "Bus Catcher: a context sensitive prototype system for public transportation users," in *Proceedings Second International Workshop on Web and Wireless Geographical Information Systems*, Singapore, pp. 64-72, 2002.

[62] Z. Wu, Q. Wu, J. Sun, Z. Gao, B. Wu and M. Zhao, "ScudWare: a context-aware and lightweight middleware for smart vehicle space," in *Proceedings of the First international Conference on Embedded Software and System*, Hangzhou, pp. 266-273, 2004.

[63] P. A. Ruiz and J. M. Sánchez, "D20.3: Evaluation Framework Specification," in *MUSIC (IST-035166) Report*, 2009.

[64] J. Li, Y. Bu, S. Chen, X. Tao and J. Lu, "FollowMe: On research of pluggable infrastructure for context-awareness," in *Proceedings of the 20th International Conference on Advanced*

Information Networking and Applications, Vienna, pp. 199-204, 2006.

[65] R. Reichle, Information exchange and fusion in dynamic and heterogeneous distributed environments, Kassel, 2010.

[66] M. Wagner, "D6.5: Modelling notation and software development method for adaptive applications in ubiquitous computing environments," in *MUSIC (IST-035166) Report*, 2010.

[67] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267-284, 1998.

[68] N. Paspallis, R. Rouvoy, P. Barone, G. A. Papadopoulos, F. Eliassen and A. Mamell, "A pluggable and reconfigurable architecture for a," in *Proceedings of the OTM 2008 Confederated International Conferences*, Monterrey, pp. 553-570, 2008.

[69] G. Gottlob, M. Schrefl and B. Rock, "Extending object-oriented systems with roles," *ACM Transactions on Information and System Security*, vol. 14, no. 3, pp. 268-296, 1996.

[70] R. Wieringa, W. d. Jonge and P. Spruit, "Roles and dynamic subclasses: a modal logic approach," in *Proceedings of the 8th European Conference on Object-Oriented Programming*, Bologna, pp. 32-59, 1994.

[71] M. Hwang, J. Kemp, E. Lerner-Lam, N. Neuerburg and P. Okunieff, "Advanced public transportation systems: state of the art update 2006," Report from United States Department of Transportation, Washington, 2006.

[72] R. Raskin, "Guide To SWEET Ontologies," NASA Jet Propulsion Lab, Pasadena, 2006.

[73] M. Horridge, H. Knublauch, A. Rector, R. Stevens and C. Wroe, "A practical guide to building OWL ontologies using Protege 4 and CO-ODE tools," University of Manchester Reports, 2004.

[74] N. Paspallis and S. Hallsteinsen, "D2.4: Final Research Results on methods, languages, algorithms, and tools to modeling and management of context," in *MUSIC (IST-035166) Report*, 2010.

[75] A. Mamelli, A. Devlic and S. Hallsteinsen, "D1.4: Final research results on mechanisms and planning algorithms for self-adaptation," in *MUSIC (IST-035166) Report*, 2010.

[76] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle and E. Stav, "A comprehensive solution for application-level adaptation," *Software Practice & Experience*, vol. 39, no. 4, pp. 385-422, 2009.

[77] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Technical Report TR2000-381, Dartmouth, 2000.

[78] M. Anvaari and S. Jansen, "Evaluating architectural openness in mobile software platforms," in *Proceedings of the Fourth European Conference on Software Architecture*, Copenhagen, pp. 85-92, 2010.