NTNU

Norwegian University of
Science and Technology

Collaboration-based intelligent service composition at runtime by end users

K.M Imtiaz-Ud-Din

Master in Security and Mobile Computing
Submission date: July 2011
Supervisor: Peter Herrmann, ITEM
Co-supervisor: Mohammad Ullah Khan, ITEM

Norwegian University of Science and Technology
Department of Telematics

Collaboration-based intelligent service composition at runtime by end users

K. M. Imtiaz-Ud-Din

Academic Advisors:

Prof. Peter Herrmann
Norwegian University of Science and Technology, Norway

&

Prof. Gerald Q. Maguire Jr.
Royal Institute of Technology, Sweden

Supervisor:

Dr. Mohammad Ullah Khan
Norwegian University of Science and Technology, Norway

July 15, 2011

Abstract

In recent years, technologies in the area of ubiquitous computing have experienced a great advancement. This has resulted in a wide-spread use of services in order to improve the quality of our daily life. For example, a person with a mobile device can use the services available in the ubiquitous computing environment to plan and execute his or her travel, to connect to family and friends, to perform his or her researches, even to manage his or her business. However, most of the services are dynamic in nature in terms of their availability, robustness and the mobility of the user. These services also appear impermeable to the end users i.e. the end users do not get to control and configure the services in a way so as to feel like programmers developing services to accomplish certain goals. We envisage that in such a context end users, with no programming knowledge, will have a hard time to find services of their choice and that it will be hard for these end users to derive substantial benefits from these services. Unguided automation is not the answer to this problem as a particular service suggested automatically by a dynamic composition mechanism may not be suitable for a specific user at a certain point of time and in a given context. On the other hand explicit specification of service instances will mean that the user will be bogged down with the problem of runtime optimization in a dynamic environment where several factors as indicated earlier determine the availability of the services having required functionality. In order to address this issue we introduce the notion of intelligent service composition where the end user will have a great degree of flexibility to define his or her own rules or conditions based on which an optimal composition will be made automatically from a set of collaborative services by runtime adaptation in a specific context and point in time. This is a step forward compared to the present dynamic composition mechanisms which do not facilitate end users defining their own conditions dictating the selection of specific service instance at runtime. We have developed this conceptual solution to bring end users towards adaptive use of services. We validated our conceptual solution through a scenario-based evaluation approach with an implementation of a proof-of-concept prototype.

Acknowledgment

This thesis project would not have been possible without the support of my academic advisors and supervisor. I wish to express my sincere gratitude to Professor Gerald Q. Maguire, my academic advisor at KTH, and Professor Peter Herman, my academic advisor at NTNU, for their timely and detailed comments regarding this report. Their continuous feedback based on an in depth analysis of my work proved very fruitful for me in completing this research project.

I would also like to acknowledge the support and guidance that I received from my supervisor Mohammad Ullah Khan. It has been a learning experience to have worked with him throughout the different phases of this thesis project. His calm nature and a clear perspective regarding the problem domain have added strength to my work.

At last, I would like to specially thank my parents and uncle for being the source of inspiration. This work is dedicated to them.

Contents

List of Tables

List of Figures

Listings

List of Acronyms and Abbreviations

DOM	Document Object Model
EMF	Eclipse Modeling Framework
GUI	Graphical User Interface
MUSIC	Self-Adapting Applications for Mobile USers In Ubiquitous Computing Environments
OSGi	Open Services Gateway initiative framework
PC	Personal Computer
QoS	Quality of Service
SLP	Service Location Protocol
SMS	Short Message Service
SOA	Service Oriented Architecture
UbiCompForAll	Ubiquitous Computing For All Users
UML	Unified Modeling Language
UPnP	Universal Plug and Play
XML	Extensible Markup Language

Chapter 1

Introduction

This chapter provides an introduction to this thesis project. Section 1.1 describes the motivation behind this research work. Section 1.2 specifies the goals of the thesis project and breaks down the research problem in to a set of tasks.In section 1.3 we present the thesis background. Section 1.4 describes the research method that we followed and finally section 1.5 provides the report outline.

1.1 Motivation

Over the last decade there has been rapid progress in mobile communication, along with this progress the number of services that assist us in our everyday life has increased significantly. The advent of new technologies such as smart phones, tablet computers, and high speed wireless networks have fueled this growth in services. Furthermore this development has also enhanced the possibility for the realization of a ubiquitous computing environment[1]. The foreseeable future as forecasted by this trend is likely to lead to quite a complex computing environment. End users[1] are bound to face the problem of selecting the services that best serve their individual purpose from a set of hundreds of similar services. This is because such purpose may be

[1]In the role of an end user an actor uses an application via some user interface. End users include both service composers and service users. Typically end users do not have a programming background.

very different from other users. In addition to this problem, there looms the inconvenience of making use of multiple services separately in order to accomplish even a simple task. Thus, a typical user may require half day to find out how to do what he or she wants to do. This is quite contrary to the concept of ubiquitous computing which is supposed to provide unobtrusive assistance to the users[2]. Questions such as "How can the user adapt to this environment easily?", "What can be done to reduce the user's burden?" appear as important concerns.

In order to avoid increasing the effort required by an end user, there needs to be a certain level of abstraction in the system that will hide the underlying intricate functionalities while at the same time allow the user to tailor his or her own solution according to the user's requirement(s). Attempting to automate the process by adapting service usage without taking into consideration a user's preferences is undesirable as a particular service suggested automatically by a composition mechanism may not be suitable for a specific user at a certain point of time and in a given context. This concept of automatically selecting services by the means of adaptation is seemingly contradictory to allowing end users to selecting services of their own choice. In contrast, we argue that these concepts are actually complementary when we consider the fact that services' main purpose are to satisfy end users needs.

From the end user's point of view all he or she needs is an intelligent system that understands how much intervention or how much adaptation this end user expects. The idea is to allow a user to combine several services at a high level in order to achieve the his/her goal while automating the details of coordinating and integrating the tasks within the combined set of services. This notion of goal oriented composition requires that design environment facilitates the users' definition of their own set of objectives and priorities when constructing a solution. Thus, we not only want the user to be comfortable when employing the services, but also want to ensure that the user is able to compose the best service that meets the user's requirements and preferences in a given context. This defines the focus of our research in this thesis.

1.2 Goals

The goal of this thesis project is to systematically devise, document, and present a novel approach with which an end user will design and develop composite services based on the user's preferences. These preferences can be updated dynamically at runtime. The service composition can be adapted taking into account the context and the service landscape, as well as user preferences. In order to fulfill this objective, we work towards developing an intelligent software system that supports enough flexibility for end users in defining the service composition, while ensuring both the end user control and the expected adaptation.

A task description to realize these goals includes the following set of milestones:

- Review related work in end user service composition and the adaptation support
- Review the end user-friendly notation and the composition tool provided by the UbiCompForAll project
- Review the support for developing self-adaptive applications using the Self-Adapting Applications for Mobile USers in Ubiquitous Computing Environments (MUSIC) middleware
- Investigate how the notation and tool for end user service composition and the middleware can work together to support end users building composite services of their choice
- Develope a mechanism for end user service composition to include:
  - Concepts related to the involvement of end users in service composition
  - Concepts related to the selection of the best service from possible alternatives at runtime based on user's preferences, available services, and current context.
  - Automatically adjust the service based upon changing user needs, as for example in the case of re-composition at runtime.
- Implement the concepts developed above

- Test the validity of the concept and the implementation using a simple scenario and by conducting a user survey
- Document the work by writing the thesis report

Figure 1.1 shows the estimated time distribution for the tasks defined above

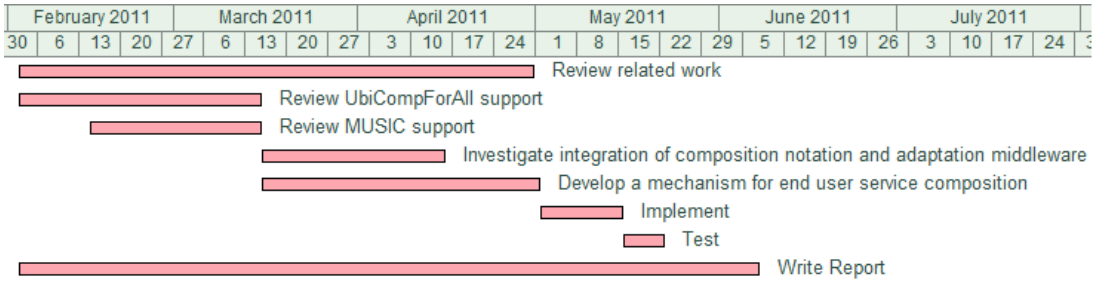

Figure 1.1: Thesis Project Plan

1.3 Thesis Background

The motivation for this thesis stems from two research projects: the UbiCompForAll project[3], which deals with end user service composition and the Self-Adapting Applications for Mobile USers in Ubiquitous Computing Environments (MUSIC) project[4], which deals with composing adaptive applications from collaborations of components and services. Both of these projects target a Ubiquitous computing environment in which end user composition and adaptation are often inter-related. Therefore, this thesis project attempts to bring the results of these two projects together, by looking at the problem from the end users' perspective.

The thesis project serves as input to the UbiCompForAll project, especially in supporting the projects research on runtime issues. It is therefore necessary to carefully adhere to the core principles of UbiCompForAll throughout the design and development phases. Adhering to these principles will be an implicit goals in addition to those goals explicitly described in section 1.2. As laid out at the UbiCompForAll website [5] the principles relevant to our work are as follows:

- Service composition can be supported by generic solutions in the form of methods and middleware that significantly reduce the complexity of developing composite services.
- Collaboration-based composition can be applied at runtime and enable the flexible adaptation and the runtime description of composition of collaborative services to meet the user's goals.
- Ontologies can be exploited to describe end user's goals and to augment the runtime description of collaborative services with goals, facilitating effective semantic service discovery and the automated composition of service collaborations.

MUSIC supports adapting mobile applications and services based on the service landscape and the context that the execution of services depends on.

Both MUSIC and UbiCompForAll will be used as example platforms for service adaptation and end user service composition, while this thesis project focuses on generic solutions independent of any particular research project.

1.4 Research Method

We have followed a systematic research method in order to achieve the goals of the thesis project. The method comprises of a number of different sequential steps, which also take care of the feedback from a later step. Different steps of the research method are illustrated in figure 1.2.

We started with the description of the problem extracted from motivating scenarios. We studied the state of the art related to the end user development and the adaptation support. We also went through several versions of the UbiCompForAll end user-friendly service description notations as they evolved over time in the UbiCompForAll project. The next step was to critically evaluate the capabilities of middlewares in addition to MUSIC which we believed could help us in designing our solution. Based on all of these studies, we developed concepts and designed a generic solution to address the problem. We further extended the solution to develop a domain dependent prototype using UbiCompForAll and MUSIC. Implementation and testing followed. We then rigorously analyzed and reviewed our proposed

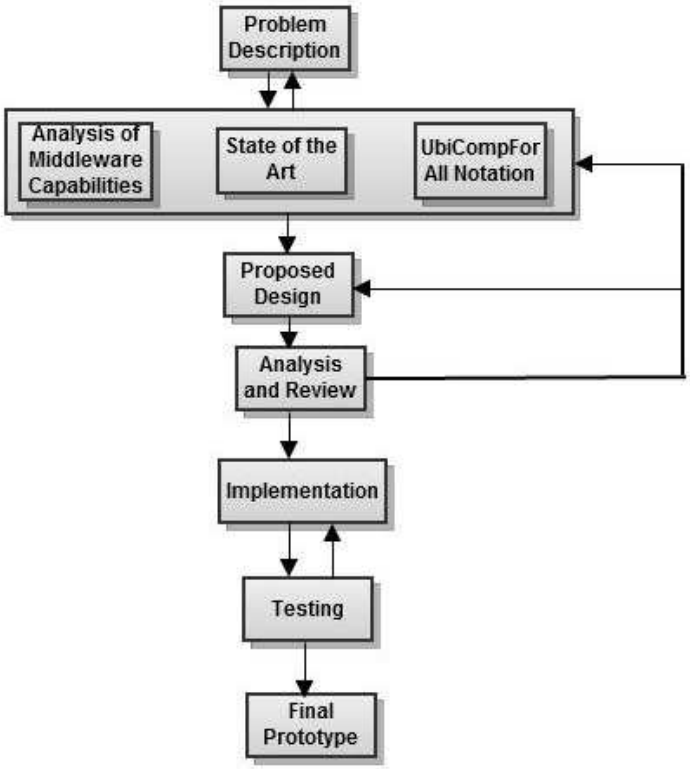

Figure 1.2: Research Method Flowchart

solution. We adopted an incremental development approach. Each time a flaw was detected while working in a particular step, we had traced it back to its origin in the concept phase. This process was repeated until the design was clean. The final results were implemented in a functional prototype that translates user requirements to composite services and are documented in this thesis report.

1.5 Report Outline

After this introductory first chapter, **Chapter 2** contains the **Scenario Based Problem Description**. The first section presents an example scenario that will depict the problem at hand. The next section identifies the points of user control and the points where adaptation is a necessity.

Chapter 3 reviews the **State of the Art**, including both previous and ongoing work in the areas of end user service composition and adaptation. From this discussion we identify the features that should be part of our proposed solution. We then evaluate the UbiCompForAll notation and show that it is suitable for our solution. We also outline the capabilities of some adaptive middleware that we believe will be useful in designing our solution and argue that MUSIC provides a sufficiently good solution to serve our purpose.

Chapter 4 presents the **The Proposed Solution**. Here we present our detailed design based on the preceding chapter. This solution combines the UbiCompForAll notation and the MUSIC middleware support. We use an example flow to show how the scenario in Chapter 2 is realized with the proposed architecture.

Chapter 5 describes the **Implementation** of the solution proposed in Chapter 4. In this chapter we explain the functionality of each of the building blocks of the proposed solution.

Chapter 6 presents the **Evaluation** of the solution by providing a proof of concept using a simplified scenario, conducting a user survey, analyzing the characteristic features of the solution using a set of parameters, and finally comparing with the existing dynamic service composition mechanisms based on a set of parameters relevant to our work.

Finally **Chapter 7** presents our concluding **Discussion** by summarizing the main achievements and limitations of the project. We also provide some suggestions for future work based on the limitations and then conclude the report in the last section.

Chapter 2

Scenario Based Problem Description

In this chapter we will use a scenario to clarify the problem presented in Chapter 1. Section 2.1 provides a scene by scene description of the scenario. Section 2.2 analyzes the scenario and identifies the points where user control and automatic adaptation are needed. The chapter ends with a discussion of possible variants that can arise due to change in user's need. In this thesis, we work on developing a software system that is competent enough to support such aspects of end user control and adaptation while providing the flexibility of facing such variations in end users' needs and adaptation.

2.1 The Scenario

The scenario is written from the perspective of a businessman (end user) making a travel plan.

Scene 1: John is a businessman. He is flying to Rome on June 12 at 2 p.m. in order to meet his clients. The airport bus arrives at the bus stop near his residence every 30 minutes. Using his mobile device, he wants to remind himself about the relevant bus arrival in order to catch the flight he wants to receive an SMS reminder at 11:55 a.m. He also wishes to remind himself via an email if Internet connectivity is available.

Scene 2: After the business meeting, John decides to go out in the afternoon. He takes a number of photographs and suddenly realizes that his phone memory is full. He might have to deal with the hassle of manually uploading the pictures to his web repository in order to free up enough memory. But by the time he would have been done, it would be evening and there might be insufficient light to take any more pictures. Therefore, he was in a serious need of a service that can automatically upload pictures only when the phone memory was critically low. This is because he not only wants to have enough memory to take the pictures but also wants to keep the pictures in mobile phone so that he can access the pictures offline as well.

Scene 3: After coming back from Rome, John knows that he has a presentation to make in a couple of days. For this presentation he needs stock exchange data for a particular day. Therefore, he wants to initiate a task that will obtain the relevant stock exchange data at the lowest price for the day that he has selected. Additionally, he wishes to receive this stock exchange information via e-mail.He uses the same e-mail service that he previously used for the notification about the time to leave for catching the bus. However, this time the e-mail service will be used to send him stock exchange data instead of giving him a reminder as in scene 1.

2.2 Analysis of the Scenario

In this section we will have a closer look at the scenario described in section 2.1 in order to identify the points where 1) end user control is expected and 2) adaptation is preferable. We also notice that a number of variations may occur in both the expected user control and adaptation possibilities. It is possible to obtain variants by switching between one user controlled option to another one, one adaptation option to another one, or by changing between a user controlled option and an adaptation option. We will identify and list a few of such variants.

End user control

With respect to the scenario described in section 2.1 the points where end user control is required are:

- In scene 1, John sets the flight date and time.

  We can think of a number of variants to this user choice.

  - Instead of wanting to set the appointment time and the flight time himself, John might want the composition to be driven by his calendar and earlier established preferences of when he wants to be notified of flights and when he wants to be notified to leave for the bus to go to the airport.
  - In another instance, John might want to set the exact time when the notification that he should depart to catch the bus must be sent.

- In scene 2, John wants to use a service to upload the photos to a web repository only when his mobile device lacks sufficient free storage.

  We can think of a number of variants to this user choice.

  - Instead of having the constraint to be able to keep the pictures in the phone memory unless absolutely necessary, John's goal could have been to have enough space to take pictures, to decrease the cost of management, and to enhance performance
  - In another instance, he might want the system to select the specific web service which will upload the photos to an online repository based on some specific conditions.

- In scene 3, John controls the criteria to select the lowest cost service in order to fetch stock exchange data.

  We can think of a number of variants to this user choice.

  - Instead of wanting to get the data from the cheapest source, John might want to select the stock data provider himself from a list of automatically extracted information providers as the cheapest option may not always be the most appropriate one.
  - John might want the stock exchange data for a particular time period instead of a particular day.

Need for adaptation

In the scenario described in section 2.1, the points where automatic adaptation is desirable are:

- In scene 1, the appropriate time to send the SMS and/or E-mail notification is determined based on the flight time and the bus time table. We can think of a variant to this point of adaptation.
  - The desired time of notifications is automatically determined from John's calendar and his previously set preferences.
- In scene 1, a copy of the generated reminder by e-mail is sent if Internet connection is available. We can think of a variant to this point of adaptation.
  - A copy of the generated reminder by e-mail is sent only if free Wifi Internet connection is available.
- In scene 2, the best web service instance to be used is determined merely from the knowledge of its functionality and quality of service. We can think of a number of variants to this point of adaptation.
  - The web service instance to be used is selected based on its cost.
  - In case of emergency, the web service instance to be used is selected only based on its functionality.
- In scene 2, a web service instance is automatically used instead of the local memory management component in order to upload the pictures from the device to the online repository when the phone memory is low. We can think of a variant to this point of adaptation.
  - A web service instance is automatically used instead of the local memory management component in order to upload the pictures from the device to the online repository when a lot of uploading throughput is available and when the costs of uploading are low.
- In scene 3, the provider offering the lowest price for the stock data is automatically determined for the selected date. We can think of a variant to this point of adaptation.
  - Instead of using cost as the parameter, the selection of stock data provider can depend upon other quality of service parameters.

Based on the above analysis we can conclude that the software system that supports John's managing these tasks must not only take into account his conditions and preferences while automating the tasks but also allow him to

control the degree of automation. This means that the number and type of software variation in such systems should actually originate from the different ways a user wishes to accomplish a given task. It is also worth mentioning that John, as an end user, is capable of using a PC or mobile devices; but does not have any programming skills and therefore, the specification means must be sufficiently end user-friendly that he can specify his choices and no more.

Chapter 3

State of the Art

In this chapter we will review the state of the art in the research areas relevant to this thesis. In section 3.1, we study the literature related to end user service composition in ubiquitous computing. Based on this review we identify the design elements in section 3.2 that can later be used to construct our own solution. In section 3.3, we will then evaluate the suitability of UbiComForAll notation for the user's specification of service composition. Finally in section 3.4, we will evaluate the potential of some adaptive frameworks and justify why MUSIC is sufficient to fulfill our requirements.

3.1 A Review of Service Composition Issues in Ubiquitous Computing

Composition of services has been addressed as one of the key features of ubiquitous computing [6, 7, 8]. For this reason a large number of service composition frameworks have been developed. We will focus on two service composition aspects that are of specific interest to our project: Composition Specification and Adaptability. Each of these aspects will be described along with their individual roles, to the extent to which they have developed. Additionally some examples of frameworks implementing these two elements of service composition are presented in the following subsections.

3.1.1 Composition Specification

A composition specification refers to a description outlining the structure of a composition from its constituent services. The important aspects related to this mechanism are the level of detail and intuitiveness with which the composition is specified, the phase of the service life-cycle which is specified, the means of specification, the entity specifying the composition, and finally the support for specifying particular criteria to drive/direct the composition.

How much detail?
Services can be specified with varying levels of depth. The most basic form is to provide an instance in the definition itself. In this case a particular service is coupled with the composition as in [9]. Numerous authors ([6], [10], [11, 12], [8], and [13, 14, 15]) have made use of individual instances while defining the service. If instead the service description is abstracted somewhat and represented with type information, then after translation by the framework the type is replaced by an actual instance of service at runtime. Some of the middleware that makes use of this technique are: [16], [17], [18], and [19]. Finally, in the third approach the user provides a high level task description that has to be resolved by the middleware into a composition of services. This method of implicit specification is followed by several authors e.g. [20, 21], [22], and [23].

When to specify?
A composition of services can either be specified during the development phase by the service developer and/or at runtime by the service user. The frameworks described in [20, 21], [16], [17], [6], [10], [11, 12], [8], [23], [19], and [13, 14, 15] all employ service specification at runtime; while the frameworks described in [18] and [22] make use of the development time specification. Some middleware with the runtime support, such as described in [6], require the users to first select the set of services that will be used to build the composite service, and then based on the choice an interface is generated for the service composition. Another alternative, described in [19], requires application developers to define a description of the service flow at runtime, and then have actual services dynamically bound to the service references. Alternatively, platforms similar to that described in [22] require the user to provide information via an ontology and corresponding classifiers before running the application. Therefore modification at runtime becomes impossible.

How to specify?
Services can be specified in various forms ranging from a configuration file and source code to an interactive tool. If a user specifies the composition via an interface, then an underlying representation has to be available to provide persistence of the specification. Alternatively, as described in [20, 21], [6], [10], and [13, 14, 15] it is possible to use an interactive tool for end users, while the frameworks described in [16], [17], [18], [8], [23], and [19] make use of a configuration file. The authors of [11, 12] and [22] require specifying the service as the source code.

Who is the composer?
Composition can be carried out either by developers who construct general applications or by end users who tailor an application directly to their individual needs. Developers try to anticipate which services will be available at runtime. In contrast, when users build the composite, they can do so based on the currently available services, thus forming a composition in a variety of unanticipated ways as described in [10, 8].

As described in [19] application developers specify the composition through a description in which sequences of services are described. Other examples of frameworks featuring end user composition include those described in [20, 21], [16], [6], [10], [8], and [13, 14, 15]. In contrast, very few instances of middleware that have been designed for ubiquitous computing environment allow application developers to describe a composition. Some examples of such middleware are described in [18], [11, 12], [23], [19], and [22].

Can the composer specify parameters that can be used to select a composition variant?
Very few of the reviewed frameworks [20, 21], [16], and [17] specifically support users specifying parameters in order to select a composition variant. Even for middleware that does have this feature, the middleware only allows users to specify one particular type of property: the quality of service (QoS). Users can usually define a threshold value for a QoS parameter. Service instances are then evaluated against this value and if the constraint is satisfied then this service is incorporated into the composition, by replacing an abstract service specification.

3.1.2 Adaptability

Adaptability is the *"capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered"* [24]. This is also termed runtime 'modifiability' in [25].

Casati, et al. [26] and majority of other surveyed frameworks allow the composition definition to be changed at runtime. In addition to this, most mechanisms perform service discovery and co-ordination based on previously defined service types. All of these features support adaptability.

Specifying QoS parameters in order to select a specific service instance enables adaptability during runtime. Sheng et al. [27] describe some of the other methods in which users require contextual information in order to obtain a personalized service. Thus quite a few user interactions are involved when composing a service. This makes this form of modifiability rather unrealistic in an ubiquitous environment where a large number of service compositions need to occur dynamically and simultaneously.

Sheng et al. [27] and IBM's Tivoli Personalized Services [28] provide personalized services to users in a pervasive environment based on static data about the user's preferences. Therefore, these platforms do **not** take into account the dynamic contextual data while realizing personalization.

Yang et al.[17] emphasize the notion of using user specified criteria in order to attain runtime adaptability, but they fail to specify an actual mechanism for context awareness that would be crucial for adapting compositions dynamically at runtime in a real environment. It remains unclear as to when and how the rules for adaptation are defined in the system and whether the users are able to easily replace these rules in at their convenience.

3.2 Inference

From the discussion in sections 3.1 and 3.2 we can safely infer that although a number of works have addressed the area of end user composition and adaptive composition, a solution to the fundamental problem of this thesis

, i.e., supporting both user control and adaptation from an end user's perspective, remains very much unfulfilled by the state of the art. However, the review has helped us identifying the following characteristic features which we believe are required to design a solution:

- The composition has to be specified by an end user, who is usually not a programmer or application developer.
- There should be an interface for describing the composition. Also an underlying representation has to be available so that persistence is ensured.
- Service descriptions can be both implicit and explicit in nature.
- Specification and updating of the composition can be made by users both at design and runtime.
- Users should be able to specify parameters that can be used to select the composition variant that they wish to use.
- Compositions can be adapted based on user-specified criteria. A context aware environment should facilitate the adaptation process, so as to dynamically build application variants at runtime.
- Users should be able to plug adaptation rules into the system at their convenience.

3.3 UbiCompForAll Notation and Language for Composition Specification

Figure 3.1 shows an overview of the UbiCompForAll system. We are mainly concerned with the UbiComposer tool within this system that allows non-ICT professionals to define service composition. The tool is still being developed and the first release will be available in August, 2011. The tool supports an end user-friendly composition notation that allows an end user in specifying service composition both at design time and at runtime in an *implicit* manner. This means that services can be defined in terms of types. The notation also allows the specification of criteria upon which service selection can take place at runtime. Additionally the notation (shown in figure 4.3) is

accompanied with a meta-model (shown in figure 4.4) that can be used to generate a persistent definition for service composition. Finally the notation is sufficiently flexible to accommodate for the inclusion or exclusion of service selection criteria at any time. In summary, this notation satisfies all of our requirements, as pointed out in section 3.3, for a composition specification language.

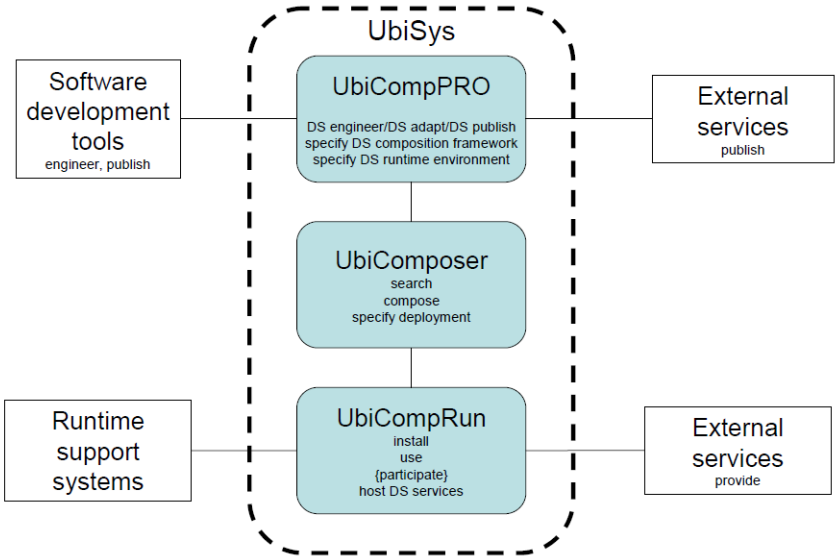

Figure 3.1: UbiCompForAll System Overview

3.4 The Adaptation Middleware

A system which can adapt and reconfigure itself according to user and execution context is a self-adaptive system. Self-adaptive systems are usually managed using middleware which senses the context and realizes the relevant changes in the adaptive application. Figure 3.2 illustrates such adaptive system. The context models take information from the user's context or from the system's resources. The middleware retrieves the necessary information required for a particular application from the corresponding context model. It then adapts the application accordingly to provide a variation of the application which is best suited to satisfy the user's requirements in the current situation.

As stated in section 3.3 building our solution requires a context-aware

environment, as shown in figure 3.2. Thus the software should have facilities to sense information and adapt the composition accordingly. MUSIC (Self-Adapting Applications for Mobile USers In Ubiquitous Computing Environments) [29] is one such framework to build adaptive systems. But it is not the only such middleware. Rouvoy, et al. in [30] proposes middleware-triggered dynamic reconfiguration methods to enable application adaptation to be based on resource availability in the case of Internet-scale shared computational grids. Other middleware, such as described in [31], is designed for the adaptation of distributed multi-user applications. Both of these frameworks were actually built for a specific kind of environment in order to make use of the changing context entities related to the domain. In addition to these some frameworks, others have taken a similar approach and tried to focus on a particular area by taking advantage of the related resources and context entities. As we are working with mobile applications in a ubiquitous computing environment, MUSIC which mainly supports mobile applications seems to be a very good choice. Additionally, MUSIC primarily models the system and its constituent properties, hence the developer is relieved of the underlying core implementation challenges related to adaptations. MUSIC facilitates development of self adaptive applications in the domain of mobile applications with relative ease as compared to other middleware platforms. This motivates us to make use of the MUSIC middleware in supporting runtime adaptation as a proof of concept to our solution.

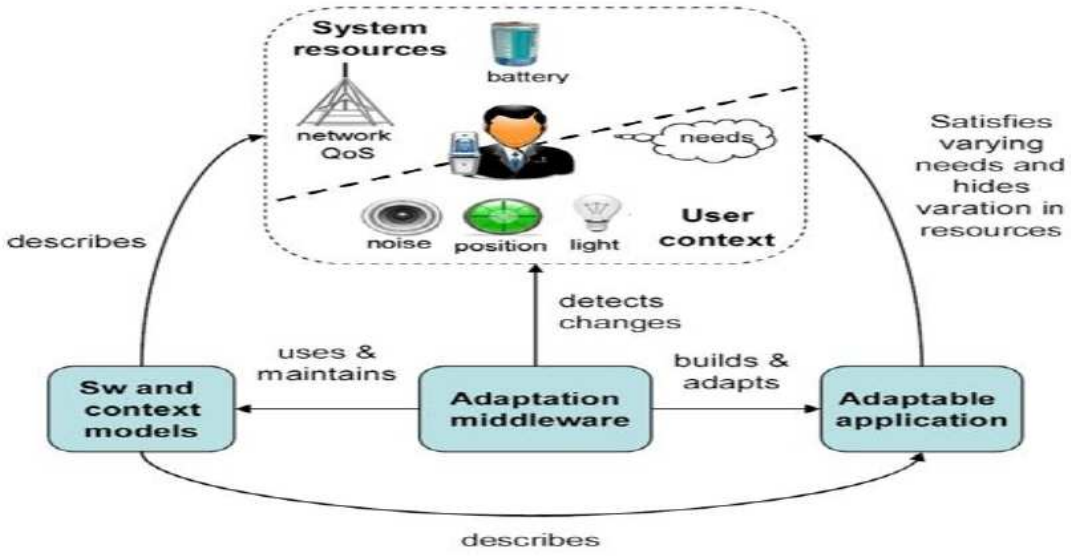

Figure 3.2: A Self Adaptive System

Chapter 4

Proposed Solution

This chapter presents our proposed solution based on the design principles elucidated in section 3.2 and the goals defined in section 1.2. We start with defining the key concepts that form the basis of the work. Afterwards, we outline the overall architecture of the solution and then explain in depth the working principle behind each of the building blocks in that architecture.

4.1 Key Concepts

This chapter utilizes several concepts that should have a clear definition in terms of how they are used in this thesis. These concepts are each described below.

End User

In the role of an end user an actor uses an application via some user interface. End users include both service composers and service users. Typically end users do not have a programming background. Examples of an end user can be a business man scheduling his business trip. In this thesis the objective of our solution is to facilitate the end users in controlling and obtaining an adaptive service usage in ubiquitous computing environment.

End User Development

End user development refers to a set of methods, techniques, and tools that facilitate users of software systems to act as non-professional software developers by providing the support for creating, modifying, or extending a software artifact [38]. The idea is to empower end users by providing as much flexibility as possible at runtime.

The browser extensions in most modern web browsers (such as Mozilla Firefox, Google Chrome, and others) are examples of software that supports end user development. Users in this case are able to modify the basic functionality of their browsers by installing or removing extensions.

In this thesis, end user development plays the key role, since the motivation of the work has stemmed from the notion of empowering end users by providing the user with as much control as possible in the development process.

Service

A service is a process that enables its users to access a set of pre-defined capabilities focused on accomplishing a specific goal. Services support our day to day life and can range from a simple calculator to complex social networks.

Services in this thesis are viewed from the perspective of end users. Our objective is to facilitate as much as possible the user's creation of composite services of their choice.

Adaptation

Adaptation can be defined as the process of changing by which a system becomes better suited to its context or environment. Adaptive systems have two essential functional properties: context sensing and reacting to the sensed information. One example of such a system is Google search which adapts the results of queries based on several criteria such as user's location and search pattern.

In this thesis project, we focus on adaptation in service composition based on the requirements of end users.

Ubiquitous Computing Weiser introduced the notion of ubiquitous computing which he defined as an environment where people will have non-intrusive availability of computational resources providing information and services when and where required[2].

The "non-intrusive" criteria is an important driving force for the research carried out in this thesis as we dedicate ourselves to exploring

opportunities that can be exploited to provide unobtrusive support to end users during service composition.

Collaboration-based Service Composition

Collaboration-based service composition refers to combining together multiple services to form a composite service providing better capabilities compared to any of the individual constituents. This is the problem domain of our research. Our objective is to provide end users the maximum amount of control over the composite that they design.

4.2 Architecture

End user service composition faces an inherent dilemma. Users most often do not want to manually select the best service that fits their requirements. However, they still would like to dictate in general from an abstract set of constraints what kind of service is acceptable to them. This means that they opt for the freedom to impose any number and types of conditions that determines the services they use. Thus automation of service composition without end user influence is not desirable, as the users want some level of control.

Automating service composition through adaptation has its own challenges which are integrating adaptation capabilities within the composition architecture, interoperability and heterogeneity of services, dynamic service discovery, dynamic updates of requirements, context sensing and reasoning, performance and scalability issues incurred upon by adaptation reasoning, robustness, usability, security, and testing and validation of adaptive systems [32]. But when viewed from the perspective of an end user's desire to have the ultimate control over service composition, adaptation falls into an unchartered territory where novel challenges exist. For this reason, our idea is to bridge the gap between adaptation reasoning and end user requirements so that a composition can be adapted dynamically at runtime based on user preferences. Figure 4.1 portrays a general architecture for solving such problems.

The *Composition Engine* generates and runs composite services and the *Adaptation Engine* is responsible for adaptation of the composition. *Our Framework* extracts *End User Requirements* constituting the composition

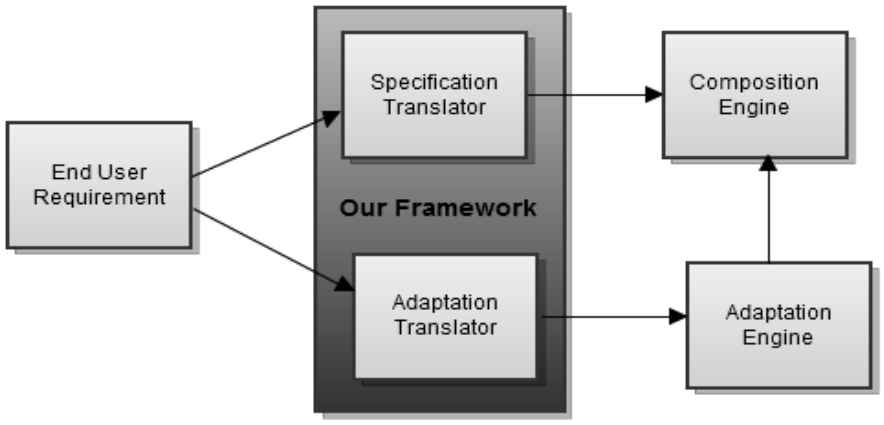

Figure 4.1: Generic Architecture of our support

specification and the user's preferences. It then translates the specification into a composition definition that will be understood by the composition engine and the preferences are translated to adaptation rules understood by the adaptation engine. These rules are subsequently used by the Adaptation Engine along with context information in order to adapt the composition generated by the composition engine from the composition definition at runtime into the variant that the user prefers.

Figure 4.2 outlines the architecture of our solution with a domain specific reference to end user requirements and composition and adaptation middleware as provided by the UbiCompForAll project and the MUSIC project. As MUSIC provides support for both composition and adaptation, the two translation phases specified in the generic design are integrated in this solution.

The first block represents user interface that is used to specify the service composition. After the user has finished describing the composition via this interface, an XML file corresponding to the description of the composition is generated. This specification is then fed into the UbiComptoMusic Code generator engine which maps the UbiCompForAll notations from the XML specification into MUSIC specific Java source code. During this process service types are either cast as external services or internal MUSIC components. If a service type refers to an external service, then the actual service instances are discovered and bound to the type by the middleware

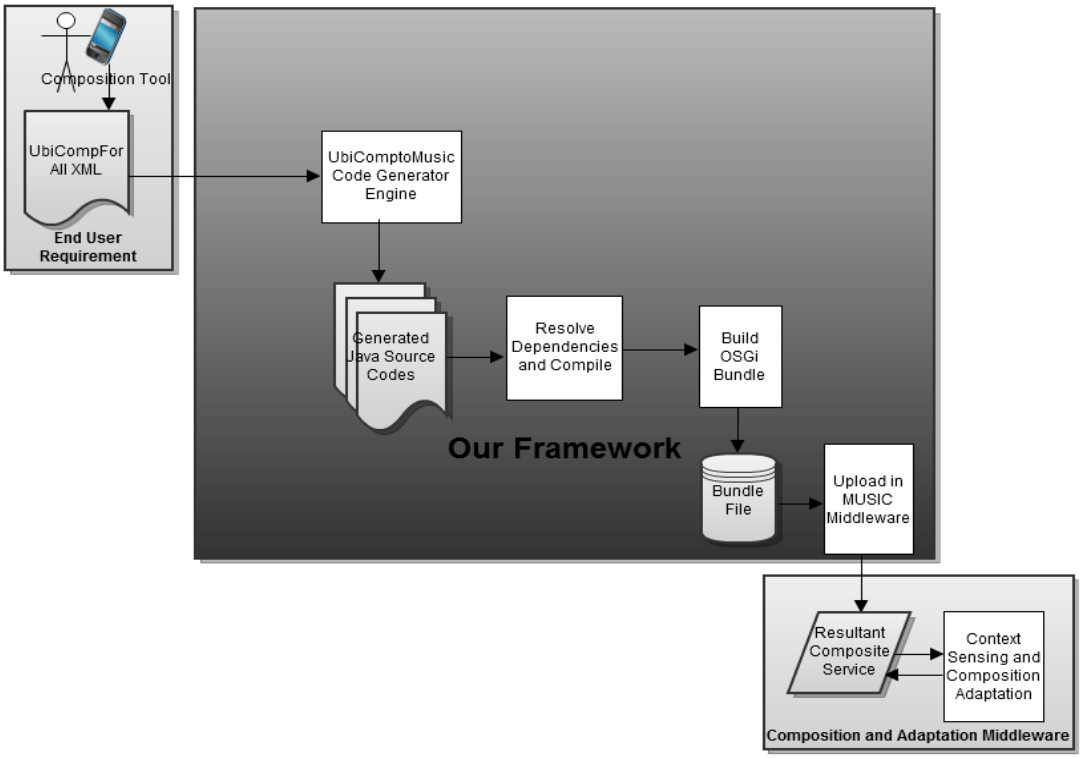

Figure 4.2: Domain-specific System Architecture of Our Solution Using UbiCompForAll and MUSIC

at runtime. However, for internal MUSIC components, as is the case for our Proof of Concept (see section 6.1), references to these internal MUSIC components are generated automatically by the UbiComptoMusic code genarator and included in the generated Java source code. An OSGi bundle [1] is then built following the compilation and the dependency resolution of the source code. Consequently, this bundle is uploaded to the MUSIC middleware to realize the composite service designed by the user. The middleware adapts the composition at runtime based on the context information in order to optimize the achievement of the user's goals. Pre-developed context sensors can be deployed to the MUSIC middleware as OSGi bundles as well.

In order to support runtime updates of the composition by end users,

[1]The Open Services Gateway initiative framework (OSGi) specification describes the bundle as a unit of modularization that is comprised of Java classes and other resources which together can provide functions to end users.

reflecting updated requirements, the process should be able to automatically change the updated composition and make the necessary adjustments in the generated source code; and consequently the service may be adapted automatically.

4.3 UbiComposer Composition Tool

In the UbiCompForAll project, work on developing a service composition tool for end users is currently in progress. Figure 4.3 shows a screen dump of the graphical user interface of the tool, providing an idea of the visual notation that are used by end users in specifying their service compositions.

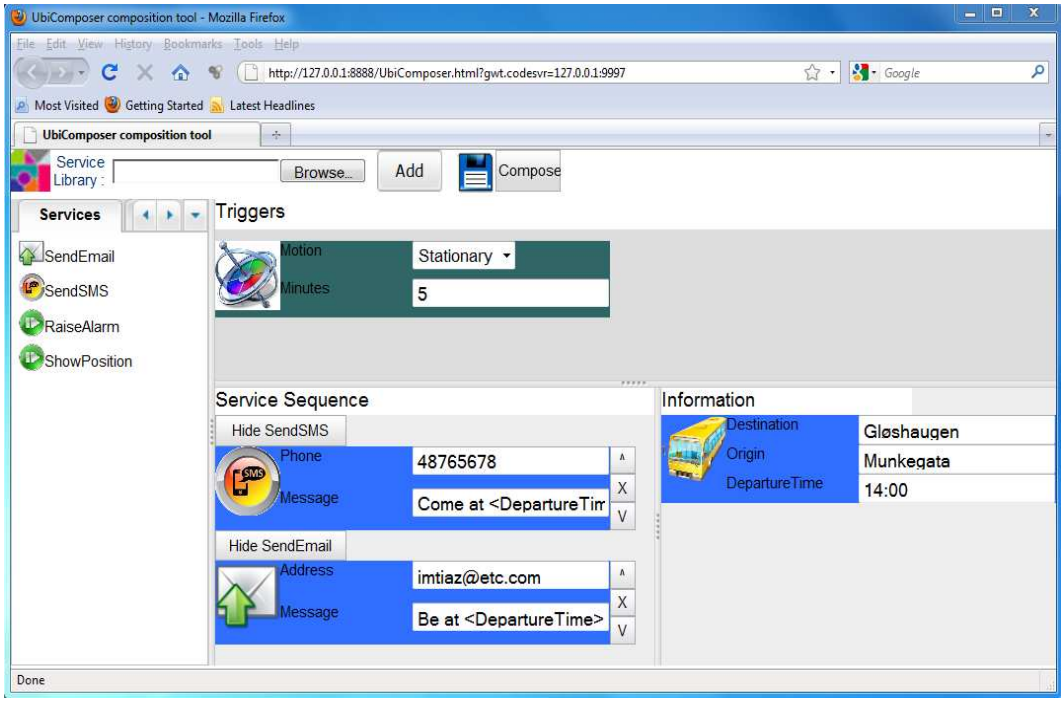

Figure 4.3: UbiComposer User Interface

Triggers define the events (e.g., context change, intents from the Android system, users moving in or out of a particular area etc.) and conditions that must be valid to trigger a particular composition. A sequence of services describe the actions that the services perform in a sequential manner. Information objects do not perform any task themselves; but they are used

by services to obtain certain information. Details about concepts such as Trigger, Information Object, and Service Sequence(Step), are discussed in section 4.4.

Note that in this tool, services are specified at the type level, for example, the SendSMS service type can be realized by a number of different providers and the best-suited provider can be chosen at runtime, either by the end user himself/herself or automatically through the adaptation reasoning process of the adaptation middleware.

4.4 UbiCompForAll Service Composition Concepts and Notation

Figure 4.4 shows the meta-model representing UbiCompForAll service composition concepts. A composition of services can be formed with one or more *Tasks* which is an entity composed of one or more *Triggers*, a number of *Steps*, and a number of *InformationObjects*.

A *Condition* is a statement whose validity can be verified at any given instant of time. Other entities such as triggers and steps can depend upon conditions for their activation.

A *Trigger* is an event which upon its realization will initiate the functioning of an entire composition or a task. A trigger may be accompanied by a number of conditions. In this case even if the triggering event has occurred, the task will only be carried out if the condition is satisfied.

A *Step* is essentially a collaborative service that performs computations on input sets and produces output sets. In some cases, a condition may be associated with a step. This means, the step will only participate in the composition if the condition is true. This genre of step is known as a *ConditionalStep*.

InformationObjects are elements that themselves are not an integral part of the composition in terms of performing any part of a task; but rather these objects provide necessary information that helps to realize a specific service. The two different types of InformationObjects are *Queries* and

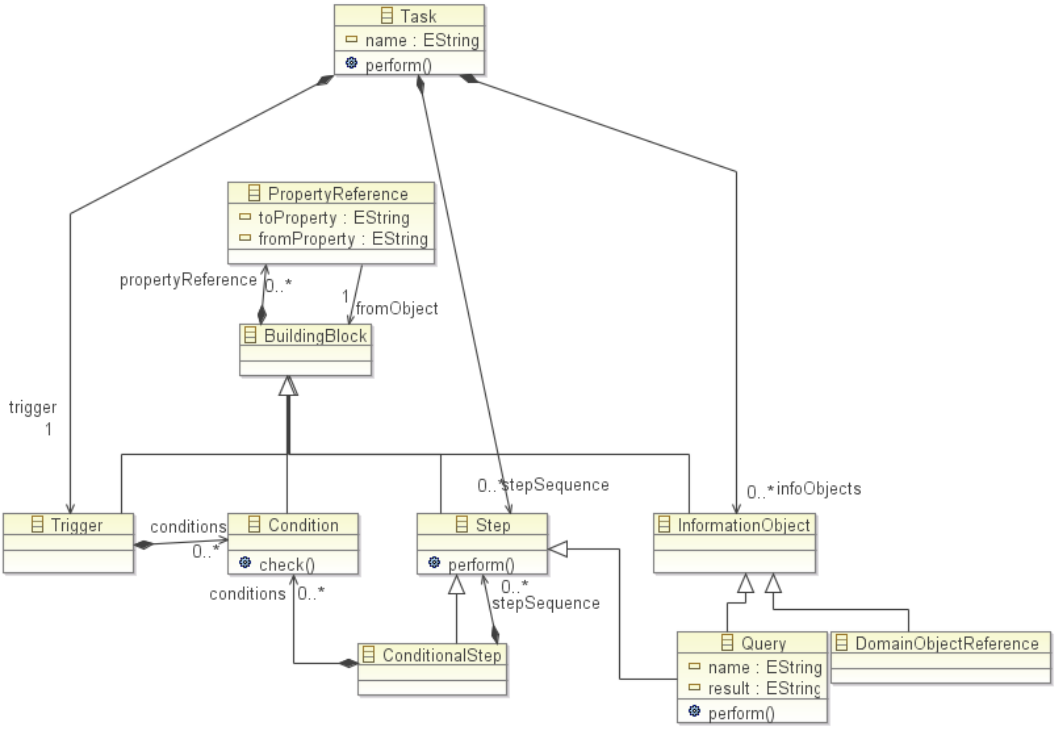

Figure 4.4: UbiCompForAll service composition meta-model

DomainObjectReference. Queries are processes that retrieves a set of information from a particular domain given a set of input parameters, while DomainObjectReferences are direct references to information that is readily available.

As seen above, Triggers, Conditions, Steps, and InformationObjects are *BuildingBlocks* of a service composition. Each of these building blocks can be associated with zero or more *Property References*. A PropertyReference is an element that maps a property from one BuildingBlock to another property of another BuildingBlock. For example, the value of attribute A of step A can be derived from the value of attribute B of step B via a PropertyReference. In this case attribute A is the toProperty and attribute B is the fromProperty.

A visual notation shown in figure 4.3 is derived for expressing the concepts defined in the simple meta-model by end users. Domain-specific descriptions of services can be added in order to define a set of icons representing the service types in a palette. Service types can be added to the composition by simply clicking on the appropriate icons. Certain attributes/properties

of each service type are expected to be defined by end users and these are made visible to the user during editing. Most of these properties are optional, so that the end user may control them, allow them to be set automatically to their default values, or allow the adaptation middleware to find the best value for them.

The end user-friendliness of the notation lies in its simplicity; thus end users are not over burdened by all the details. The use of icons representing services is quite intuitive for users with smart phones or PCs as they are already familiar with the use of icons representing services while invoking these services. The setting of specific properties are made easy by using end user-friendly names hiding the corresponding technical descriptions or representations of these properties and services.

4.5 Composition Description in XML

The composition description in XML (in EMF .ecore format) contained in the file generated by the user interface of section 4.3 was designed based on the concepts of UbiCompForAll service description meta-model described in section 4.4. All the components except DomainObjectReference are modeled as separate XML nodes. DomainObjectRefernce is a tag that is used to refer to a component from another component and therefore it is included within individual component node if applicable.

The following listing shows the template, enhancing the corresponding meta-model of figure 4.4 for the XML file used in this thesis for providing the proof of concept (see Chapter 6) implementing a subset of the scenario described in section 2.1.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/
      XMI"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
5   name="DefaultNameOfComposition" nsURI="http://ubicompforall.
        org/"
6   nsPrefix="DefaultNameOfComposition">
7   <eClassifiers xsi:type="ecore:EClass" name="NameOfTrigger"
8       eSuperTypes="../../org.ubicompforall.simplelanguage/model/
            SimpleLanguage.ecore#//Trigger">
9       <eStructuralFeatures xsi:type="ecore:EAttribute"
```

```
10          name="NameOfProperty"
11          eType="ecore:EDataType http://www.eclipse.org/emf/2002/
                Ecore#//EDATATYPE"
12          value="operand1,operator,operand2" />
13
14    </eClassifiers>
15    <eClassifiers xsi:type="ecore:EClass" name="NameOfCondition"
16      eSuperTypes="../../org.ubicompforall.simplelanguage/model/
            SimpleLanguage.ecore#
17                //Condition">
18        <eStructuralFeatures xsi:type="ecore:EAttribute"
19          name="NameOfProperty"
20          eType="ecore:EDataType http://www.eclipse.org/emf/2002/
                Ecore#//EDATATYPE"
21          value="operand1,operator,operand2" />
22
23    </eClassifiers>
24    <eClassifiers xsi:type="ecore:EClass" name="NameOfTask"
25        eSuperTypes="../../org.ubicompforall.simplelanguage/model/
              SimpleLanguage.ecore#//Task"
26        Refersto="NameOfTrigger/NameOfCondition">
27        <eClassifiers xsi:type="ecore:EClass" name="
              NameOfConditionalStep"
28      eSuperTypes="../../org.ubicompforall.simplelanguage/model/
            SimpleLanguage.ecore#
29                //ConditionalStep"
30          Refersto="NameOfCondition">
31          <eStructuralFeatures xsi:type="ecore:EAttribute"
32            name="NameOfProperty"
33            eType="ecore:EDataType http://www.eclipse.org/emf
                  /2002/Ecore#//EDATATYPE"
34            value="ValueOfProperty" />
35
36        </eClassifiers>
37        <eClassifiers xsi:type="ecore:EClass" name="NameOfStep"
38          eSuperTypes="../../org.ubicompforall.simplelanguage/
                model/SimpleLanguage.ecore#//Step">
39            <eStructuralFeatures xsi:type="ecore:EAttribute"
40              name="NameOfProperty"
41              eType="ecore:EDataType http://www.eclipse.org/emf
                    /2002/Ecore#//EDATATYPE"
42              value="ValueOfProperty" />
43
44        </eClassifiers>
45        <eClassifiers xsi:type="ecore:EClass" name="NameOfQuery"
46          eSuperTypes="../../org.ubicompforall.simplelanguage/
                model/SimpleLanguage.ecore#//Query">
47            <eStructuralFeatures xsi:type="ecore:EAttribute"
48              name="NameOfProperty"
```

```
49                eType="ecore:EDataType http://www.eclipse.org/emf
                      /2002/Ecore#//EString"
50                value="ValueOfProperty" />
51
52          </eClassifiers>
53      </eClassifiers>
54  </ecore:EPackage>
```

Listing 4.1: Template for Composition Description

In order to simplify the design of the framework, we restrict our discussion to only compositions made up of single task. However, compositions consisting of more than one task can be constructed by the inclusion of multiple tasks in the same composition XML file. In this case end users can select a task using conditions and triggers set for each of the tasks in a composition [33]. There can be a problem of having conflicting conditions or triggers so that more than one task become active at a given instance. MUSIC automatically solves such conflicts through its adaptation reasoning mechanism by selecting the task that has the highest utility. Even if the utility becomes equal for a number of tasks, MUSIC selects only one of them.

In the above template, triggers and conditions are placed outside the task description node. This is done so that multiple tasks and multiple elements within them (such as conditional steps) can refer to a single condition or trigger without having the trigger or condition having to be re-written over and over again.

Let us now have a closer look at the structures of each of the elements from the composition. Our primary focus here is to identify the enhancements to the basic UbiCompForAll XML structure with which we started. First of all, the representation technique used to store the contents of the "value" attribute in triggers and conditions is enhanced. The content of this attribute is represented in the form of a triplet: constant/variable, operator, and operand. The first element in the triplet refers to whether the value derived from the context will be matched against a constant or a variable for the evaluation of this trigger or condition. The second element defines the operator used for the comparison. The third element is the constant or variable against which the context value will be compared. An example can be (c,eq,4). This is interpreted as a trigger or a condition that will be deemed as true if the value of the variable used in the trigger or condition and derived from the context is equal to 4.

Secondly, we introduce the "Refersto" attribute in tasks and conditional steps. As the name suggests, this defines the relationship between a task and triggers and conditions; or between a conditional step and a condition. It should be noted that a task can depend on multiple triggers and conditions in which case their names will be represented as comma separated values.

The final enhancement is the convention for a reference used to address the attributes of one component from within the attribute of the other. The rule is to form a conjugate of the component and the attribute by placing a "." in between them. For example, if we are to refer to attribute "deptime" of component BusService, then the resulting conjugate would be "BusService.deptime".

4.6 The MUSIC Conceptual Model

Before we present the design of the UbiComptoMusic Code generator engine, we want to familiarize the reader with the MUSIC conceptual model that is required to build the design. As the MUSIC manual[33] suggests, MUSIC is a generic middleware that facilitates the creation of context-aware and self-adaptive software. MUSIC and its runtime representation uses a conceptual metamodel. The next few paragraphs will explain the core elements of this metamodel.

A *System* in MUSIC is defined as the collection of entities that provide and use services among each other. Entities can be anything from software to computers, networks, humans, or any phenomena that influences the needs of humans in their environments. When an entity provides a service to another entity the relationship between them is termed collaboration. The notion of *service type*, *entity type*, *role*, *composition*, and *connector* are used in MUSIC's model of systems.

A *service type* is used to represent a particular type of service in terms of the collaboration between the provider and the consumer. As a collaboration has limited duration due to its irregular availability, a collaboration can repeatedly call the provided and required operations according to the protocol used by this collaboration.

An *entity type* models an entity with respect to its required and provided

services. Entities can either be atomic or composed of other entities.

A *composition specification* models a composite entity in terms of the types of constituent entities that it is composed of and their internal and external collaboration with other entities. An entity type within a composition is usually referred to as a *role*. A *conditional role* is used to model entity types that may be present in one variant of the composition specification and absent in another.

Connectors are used to model the collaborations between the roles of a composition specification. A connector is related to a service type and with a provider role and/or a client role. This relationship implies that the provider role provides the service to the client role.

The concepts explained above are formalized with the entity relationship diagrams 4.6 and 4.7 adapted from [33].

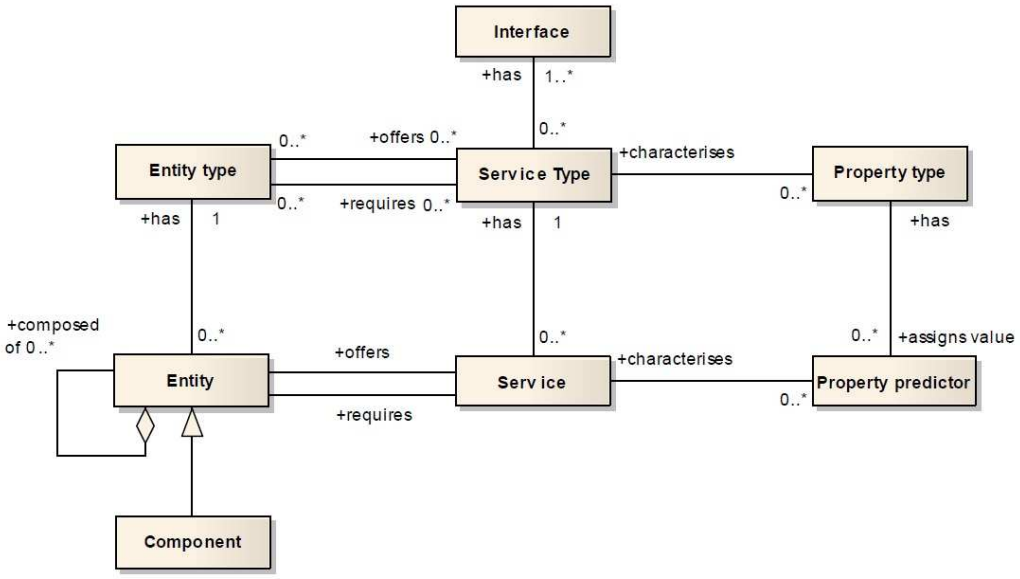

Figure 4.5: Basic Music Concepts

In MUSIC, software entities are referred to as *components*. The modeling of a component by a set of service types specifying the services that the component provides or requires can be carried out by a *component type*.

The realization of an atomic component is specified by an *atomic plan*. Similarly a *composition plan* is used to define the realization of several

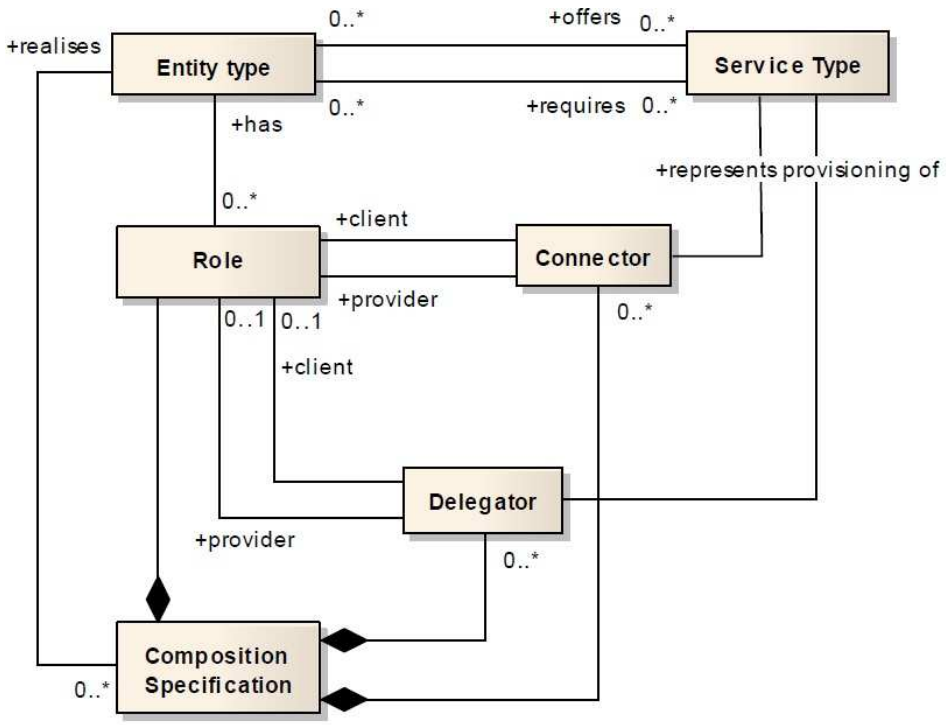

Figure 4.6: Composition Concept

components combined to form a composite component. This is carried out by defining the roles included in the composite, the connections between them, and the logical node on which each node is located.

An *application* is normally a composite component that can be launched in the MUSIC middleware to provide a service to an user via an interface and/or to provide services to other applications. In other words an application is a realization of an *application type* which is again a specialization of component type. Usually the application is deployed as a *MUSIC bundle*. This is a flexible deployment unit that allows the deployment of individual components as well as full applications and which also allows us to download the meta-information or *plans* separately from code.

The variability in an adaptive application designed using MUSIC system is based upon the definition of a set of *varying properties* specified by a component type. These properties vary from one realization of the component type to another. *Property Types* are associated with the service

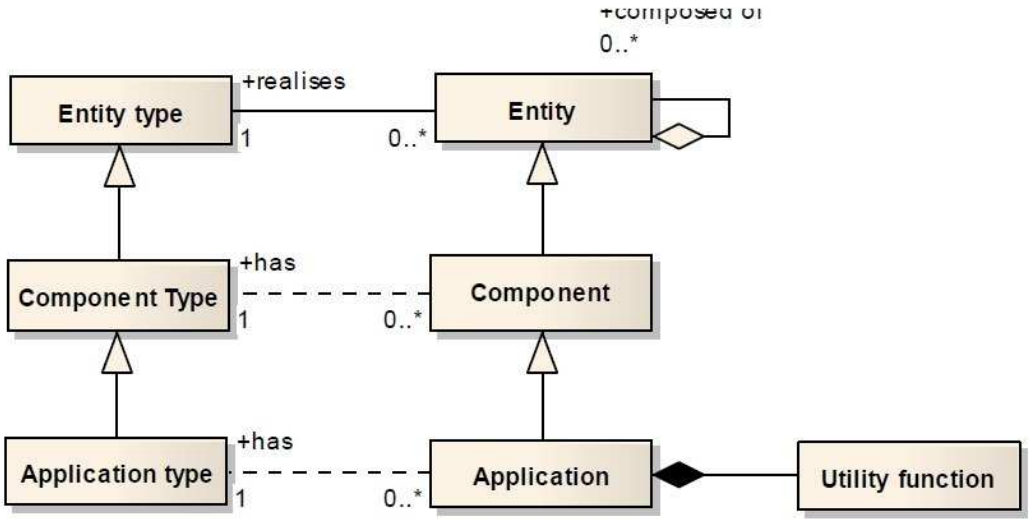

Figure 4.7: MUSIC System

types and the corresponding *Property Evaluator* functions are associated with component realizations that are used to model them. Property evaluator functions are expressed in terms of the context, resources, property evaluators of collaborating components, and also the property evaluators of constituting components in the case of composite components. These are required when the property values are not explicitly provided; in which case a Provided Property Type would have been used to directly retrieve the value of the property.

Variation in extra-functional properties and resource needs and variation in functionality - can both be modeled by varying some properties of the components. This means that it is possible to build systems with different properties from the same model, as well as to modify the characteristic features of a running system by replacing one or more components. Also, it is possible to build variants by setting some property values at instantiation time and then modify them at runtime.

The selection of a variant is dependent on the notion of *utility function* which reflects the suitability of a given configuration in a given context based on the evaluated values of the varying properties. In this case a Context Query is used to retrieve the value of certain properties in the current context. This is then matched against the value from either the provided property or the

property evaluator function to compute the utility of a certain application variant. By maximizing the overall utility, the middleware tries to continually adapt a running system to provide the optimal solution with respect to a specific situation.

4.7 Bridging between UbiCompForAll and MUSIC Concepts

The preceding two sections introduced the basic concepts of the UbiCompForAll service description notation and the MUSIC middleware. We have integrated them so that the representation of a service composition in UbiCompForAll notation can be correctly translated to its representation in MUSIC. In this section, we will use formal UML notations to establish the relationship between them. UbiCompForAll concepts are represented as Class names, while MUSIC concepts are represented as Stereotype names.

In a composition description file, Steps, Conditional Steps, and Queries are represented as abstract types, rather than specific service instances. Thus, it is intuitive to model them as MUSIC component types as shown in figure 4.8. On the other hand, the actual services realizing these building blocks are represented with atomic realizations, which is a stereotype used to model the realization details of atomic components. The condition in a conditional step is modeled as a required property implying that it must be fulfilled in order for the step to be picked up for the execution plan. In MUSIC, such component types are modeled as optional roles so that in some compositions they may be present while in some other they may be skipped. In the case of a query or a step, there is no need of mentioning the required property, because a query or a step is present in all variants of the composition.

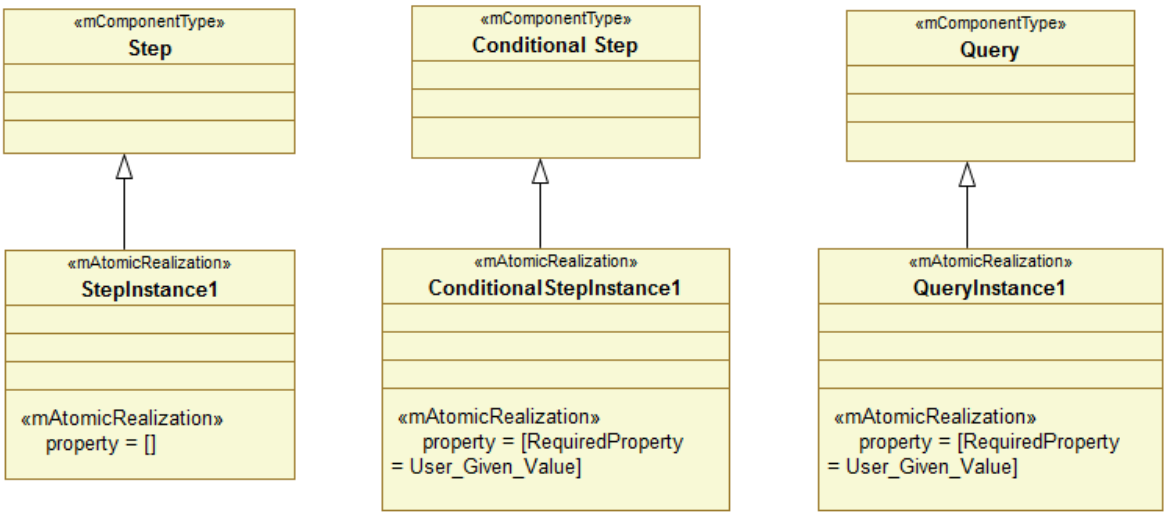

Figure 4.8: Component Type in MUSIC corresponds to Step, Query and ConditionalStep in UbiCompForAll

A composition can have components where the output attribute of one is actually the input to another. Such components are modeled in MUSIC as connected in a composition and therefore, they appear with ports providing and requiring certain properties. The mapping of UbiCompForAll concepts for such component types or service types are shown in figure figure 4.9. If they have an attribute whose value is provided to others, then it is regarded as a provided property. While if they have an attribute whose value is derived from others, it is represented as a required property. Connectors are then used to draw the relationship between a required property and provided property in a composition.

Disjoint components that take part in the composition are modeled without any provision for required or provided properties.

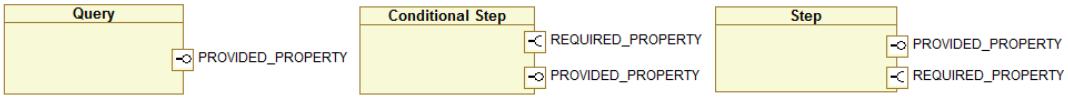

Figure 4.9: Components with provided/required properties

A trigger or a condition requires the operation where either explicitly given attribute values or attribute values that are to be calculated at runtime will be matched against values derived in the current context. The provided

property type is the representation of the explicitly mentioned attribute value while the current context value of the attribute of a trigger or a condition is modeled as a property type as shown in figure 4.10. A context query is used to evaluate the current value of the property type from the context.

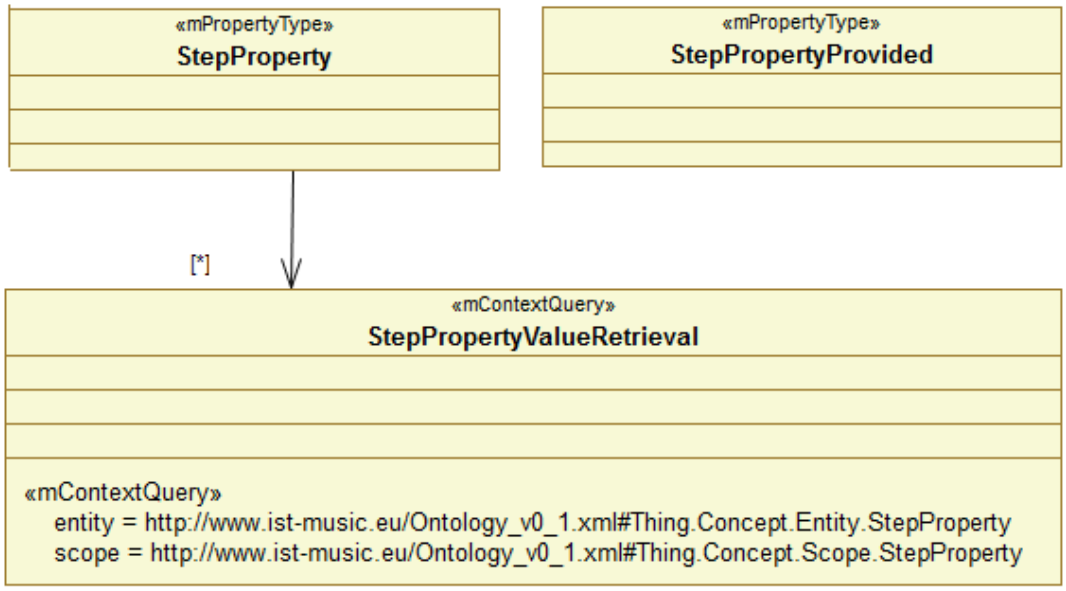

Figure 4.10: Property Types and their evaluation using context query

The attribute values of a trigger or a condition that are to be calculated and are not explicitly given in the composition can be derived from the property evaluator function as shown in figure 4.11.

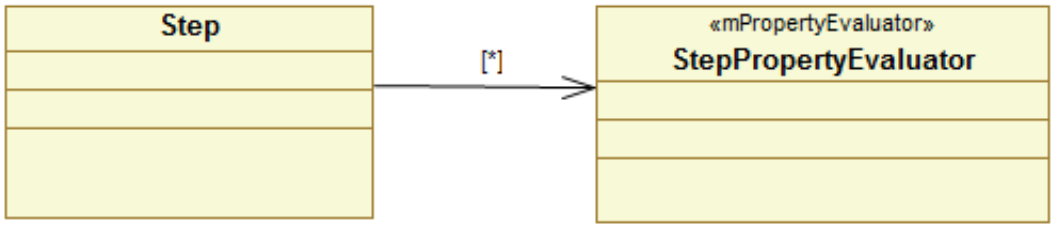

Figure 4.11: Property Evaluator

In figure 4.12, the application framework is represented as a MUSIC Application Type. As pointed out earlier this application can include one or more compositions. In the simplest of cases the application type is realized with just one composite realization. There is a MUSIC stereotype, called

¡¡mCompositeRealization¿¿ to represent the concept of composition plan that can be used to model a composition of steps , conditional steps, queries and domain object references. The presence of conditional steps in the composition actually introduces variants of the composition itself because they will only be executed if their conditions are evaluated to be true. Steps without condition will always be a part of the execution plan and therefore, they are present in all compositions.

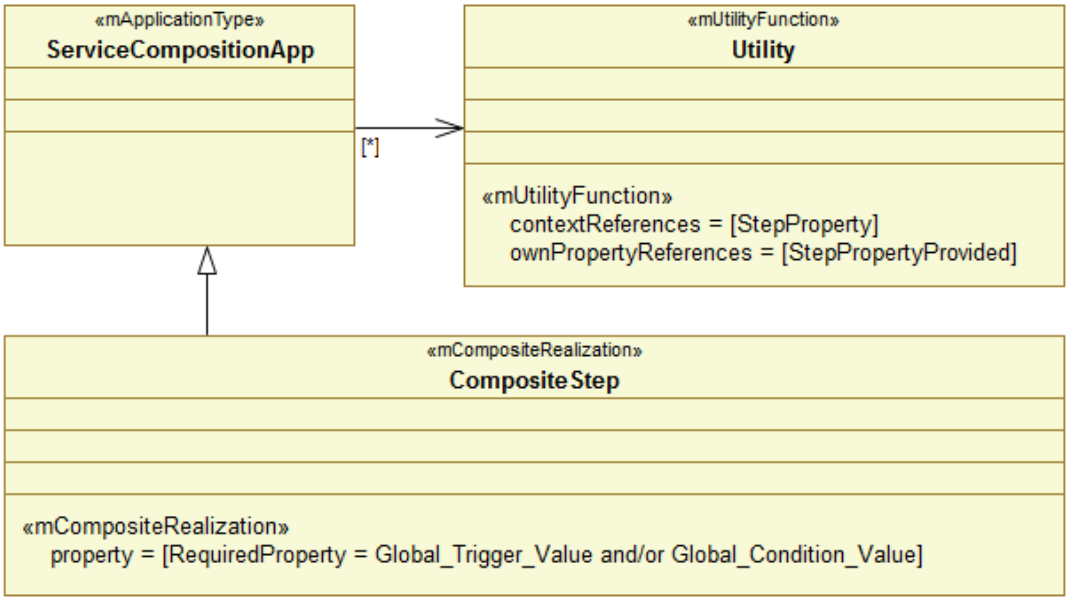

Figure 4.12: A Composition consisting of an ApplicationType, a Utility function and a CompositeRealization

In order for the composite realization or a composition to be a part of the execution plan, the global condition and/or trigger associated with it need to be realized first. Therefore, these entities are represented as required properties for the realization.

The utility function in figure 4.12 is the function that computes the suitability of a certain application variant by comparing the property values of triggers and/or conditions derived from the current context against those that are either explicitly provided or computed via property evaluator functions. This means that the utility function is used to first figure out whether the entire composition will be executed or not and then to find out which if any of the conditional steps present in the composition will be executed.

Figure 4.13 shows the inner detail of a composition represented by a composite realization diagram. In this example, the realization is comprised of two steps(SendSMS and BuyBusTicket), a conditional step(SendEmail) and a query(RetrieveBusTime). Each of the steps and the query are represented as MUSIC roles while the conditional step is realized as an optional role in the composition. The "SendSMS" step and the "SendEmail" step depend on the "RetrieveBusTime" query. The attribute values of the steps are derived from the result of the query. This relationship between a step and a query is modeled with a connector that binds the required property of the step and the provided property of the query. Two steps that are dependent on each other can be modeled in exactly the same way. Disjoint steps, that do not depend on any other entity for their execution, such as "BuyBusTicket" from figure 4.13 can also be a part of a composition.

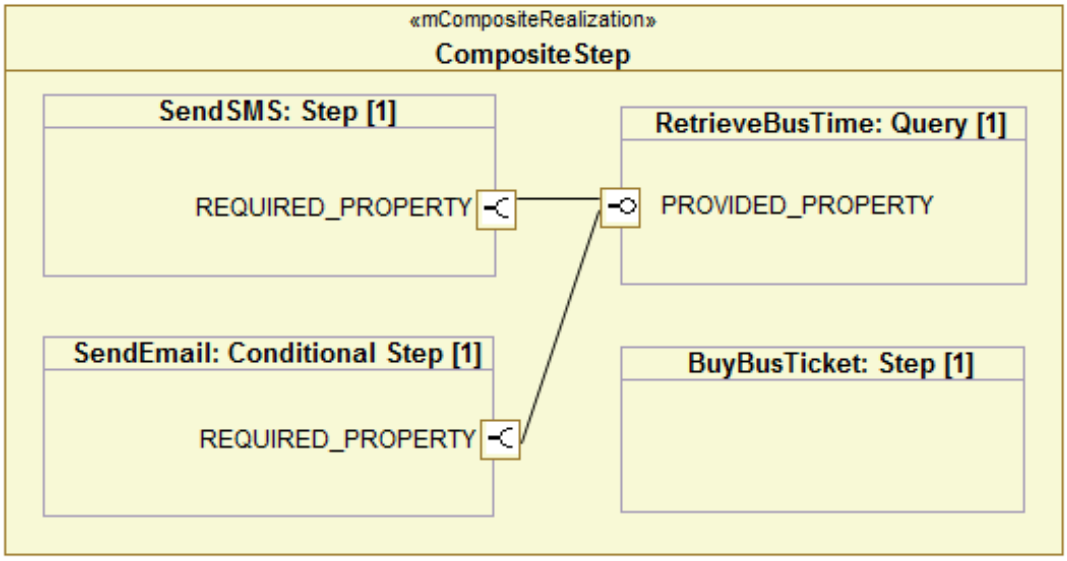

Figure 4.13: An Example Composition employing two steps, a conditional step and a query

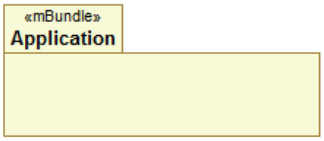

Figure 4.14: Application Bundle

In figure 4.14 entire application, composed of services, is shown modeled as a MUSIC bundle.

The discussion in this section is summarized in table 4.1 which shows the mapping between UbiCompForAll and MUSIC concepts.

Table 4.1: Bridging between the UbiCompForAll and the MUSIC concepts

UbiCompForAll Concept/Generic Concept	**MUSIC Concept**
Step	Component Type
Conditional Step	Component Type
Query	Component Type
Service Instance	Service/Component
Trigger	Required Property
Condition	Required Property
Provided Attribute	Provided Property
Required Attribute	Required Property
Relationship	Connector
Provided value of Attribute	Provided Property Type
Value of Attribute from the context	Property Type
Function to retrieve the value of Attribute from the context	Context Query
Function to evaluate Attribute values	Property Evaluator Function
Application Framework	Bundle and Application Type
Composition	Composite Realization
Function to compute suitability of application variant based on provided or evaluated current property values of Triggers and/or Conditions	Utility Function
Step/Query within a Composition	Role
Conditional Step within a Composition	Optional Role

4.8 Service Discovery

The communication services of the MUSIC middleware are responsible for carrying out discovery of remote external services and their integration to the platform[33]. The idea is that depending on the type of the services defined in the UbiCompForAll service composition file by the user, specific instances of a particular type will be discovered. These instances can then be modeled as individual atomic realizations of the service type by the UbiComptoMusic code generator. At runtime, the middleware will pick only one service based on the defined condition if it is a conditional step or will randomly select one instnace of the service otherwise.

Depending on the particular type of service discovery protocol, specific plug-ins should be used. A number of implementations such as Universal Plug and Play(UPnP), Service Location Protocol(SLP), etc. already exist in the latest version of the middleware.

If the composition of an application only makes use of internal MUSIC services (or, components), then the service discovery phase can be avoided-as we can directly use the of services that are available within the system. In our prototype of the framework, we have only made use of internal services, but have made sure that the nature of the code structure allows for the integration of a service discovery phase. We will further explain this issue in Chapter 5.

4.9 Building the Transformation Engine

The transformation engine that translates UbiCompForAll service compositions to MUSIC-compliant Java source code simply makes use of the solution described in section 4.4. The design of this tool was carried out by closely examining the Java source code files generated from UML MUSIC notations for adaptive applications. As part of our motto to automate the whole process starting from identifying the changes in the end user-defined service composition to automatically adapt the service usage, we have made a major update to the MUSIC development as well.

In MUSIC, they provide a UML2Java transformation tool that also generates

Java source code corresponding to the adaptation model and component skeletons from UML models. However, the source code needs to be manually filled out with the implementation of Utility functions. Since we would like to avoid such manual intervention in the process, we need to generate the implementation of utility function as well. The details of this implementation are discussed in Chapter 5.

4.10 Resolving Dependencies, Compilation, and Building of the Bundle

After the code is generated, it is necessary to ensure that all the MUSIC packages that are required by the code are properly included before compilation. Once the code is compiled, it is necessary to generate a customized Manifest file following the OSGi standard and then build the application bundle using this Manifest file. The final task is to upload the bundle to run on top of the MUSIC middleware. We will describe this procedure in more practical terms in Chapter 6.

4.11 Context Sensing and Composition Adaptation

After the middleware has started running the application bundle, relevant context information as described in the composition by the means of context queries are automatically sensed from the environment by using context plug-ins. We assume that these context plug-ins are pre-installed in the system. Additionally, such context plug-ins could be uploaded as MUSIC bundles when needed. These plug-ins can be developed by the framework developers or service developers beforehand, depending on the type of services supported by the framework and the parameters that are to be used by the end-users in order to optimize their composition. Composition adaptation is then carried out by the middleware by comparing the values derived from the context and the values already provided in the composition description. The next chapter outlines the implementation details of a context plug-in and its working principle will be depicted in Chapter 6.

Chapter 5

Implementation

In this chapter we discuss the implementation details of the thesis project. The implementation phase has been divided in to four parts: Detect changes in composition, Retrieve composition specification, Generate MUSIC code, and Create bundle and upload to middleware. Section 5.1 presents an overview of the framework and each of the subsequent sections except the last describes a part. Section 5.5 discusses the implementation challenges related to building context plug-ins.

5.1 Framework Overview

In this version of the prototype we have only included support for internal services. Thus, the service discovery module is not included. We begin our discussion with figure 5.1. This figure shows the sequential flow of execution of the entire system.

The process is triggered whenever there is a change in the composition description XML file. This can happen when the end user updates his/her composition. Information related to individual components forming the composition are extracted from the description file. The code generator engine then generates MUSIC code from the UbiCompForAll service description concepts following the relationships established in the chapter 4 and summarized in table 4.1 on page 37. Building this generic parser which

generates MUSIC-compliant Java source code was the most difficult and time consuming phase in the entire implementation process. We had to carefully analyze all the possible variation of input from the composition description XML file and then devise a generic algorithm so that all of these variations are correctly mapped to MUSIC-compliant Java source code. Finally, an OSGi bundle is created from the generated Java source code and deployed to the MUSIC middleware. The middleware replaces any previous instance of the composition bundle with the most recently uploaded one.

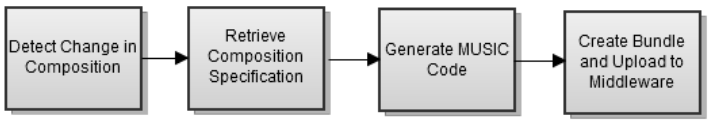

Figure 5.1: Sequential flow of execution of the phases of the framework

Three out of the four processes, except for the process labelled *create bundle and upload to middleware* shown in the diagram, have been automated. This means that whenever there is a change in the composition description XML file it leads to the automatic generation of MUSIC-compliant Java source code, but the user has to manually create the bundle and upload it to cause the application to change its behavior. Users are able to modify their composition at runtime. In the next version of the framework we plan to automate the entire flow of execution.

5.2 Composition Change Detection Mechanism

The *FlowControl* class given in listing B.2 on page 67 uses the *FileWatcher* class given in listing B.1 on page 66 to detect a change in the composition description file. The *FileWatcher* class runs in a thread and continuously monitors the composition description file in XML. We detect a change by comparing the modification time of the file and the length of the file with the earlier modification time and file length. If there is a change, then the parser is invoked to translate the composition description to MUSIC compliant JAVA source code. In this phase our objective is only to detect a

change, if any, by the end user in the composition description file.

5.3 Composition Specification Extraction

In order to extract the composition specification from the XML file, we employ Document Object Model(DOM) tree parsing. We traverse each of the nodes and their children in a recursive fashion (lines 112 through 144 in listing B.3 on page 70). The generic information regarding the UbiCompForAll components from the composition are stored in a data structure called *Component* (lines 100 through 102 in listing B.3 on pages 69-70). Their attributes are stored in another data structure known as *Attribute* (line 91 in listing B.3 on page 69).

5.4 MUSIC-compliant Java Source Code Generation

In order to generate MUSIC-compliant Java source code, we had to first figure out the structure of code recognized by the middleware. We designed, implemented, and carefully evaluated several example self-adaptive applications for deducing the code structure. The parser in listing B.3 on pages 67-99 implements this code structure. We now discuss the code generation process carried out by the parser in detail.

The MUSIC-compliant Java source code generation process starts with the generation of the bundle file (lines 207 through 216 in listing B.3 on page 72). The name of this bundle is automatically generated based upon the name of the composition in the composition description file. The *Type Names* (for component types and application types) (lines 282 through 297 in listing B.3 on page 74)and *Names*(for realizing plans)(lines 299 through 311 in listing B.3 on pages 74-75) are generated inside the bundle from the names of the composition and the constituent services.

Next the *Application Type* is generated (lines 313 through 321 in listing B.3 on page 75). This part of the code is mostly static except for its name which is taken from the name of the composition application. The *Component Types*

(lines 323 through 341 in listing B.3 on pages 75-76)are then generated from the the Steps and Queries.

Next comes the *Atomic Plans* (lines 343 through 534 in listing B.3 on pages 76-80) with specific reference to the *context dependencies*. These are generated from the conditions of Conditional Steps.

The *Service Plans* (lines 536 through 539 in listing B.3 on page 80) are generated as an empty block, since as we are not using any external services for this prototype.

A *composition plan* (lines 541 through 978 in listing B.3 on pages 80-90) with explicit mention of *roles* and *optional roles* are generated next. This includes the specification of the global condition and/or trigger in the form of context dependencies.

As an extension to the MUSIC support (which generates only a skeleton of the utility function, while it must be implemented manually), the implementation of the Utility Function is also generated. The *Utility Function* (lines 995 through 1207 in listing B.3 on pages 91-96) is generated next. Its code contains a comparison between all the property values and the current context values retrieved from the environment. If the property value is not directly provided, then a *Property Evaluator*(lines 1209 through 1247 in listing B.3 on pages 96-97) is generated in order to evaluate its value.

After the bundle file has been generated, a file is created for each of the components and the entire composition (lines 1261 through 1348 in listing B.3 on pages 97-99). The generated code for the component files instantiate an object of the actual service that realizes the component. Also, code is generated to pass on the property values of the components to the actual services and finally to run the service.

5.5 Building the OSGi Bundle and Deploying to the Middleware

We have created a Java plug-in project with the generated source code. Figure 5.2 shows the project structure of our prototype.

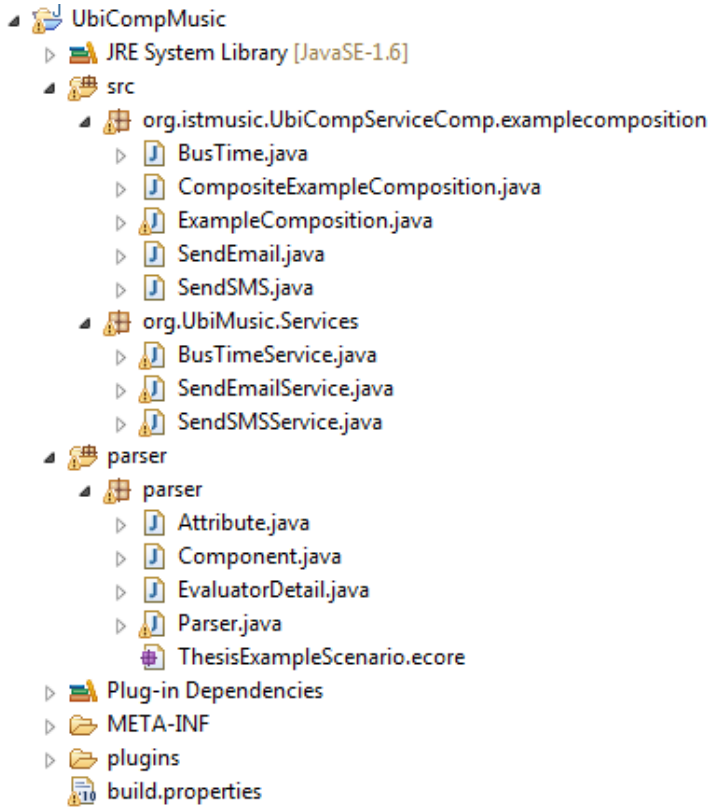

Figure 5.2: Project Structure

The following steps were followed in order to create the OSGi bundle from the project in figure 5.2 and to deploy it to run on top of the MUSIC middleware:

- The package structure containing all the generated files was copied to the "src" directory of the project from which the bundle will be built.
- A components.xml file containing project specific information was added to the projects META-INF directory.
- Next we manually edited the MANIFEST.MF file. The general information such as the name of the application was adjusted by editing the relevant fields in the MANIFEST.MF file.
- Packages were added using "Imported packages" option in the bundle creation utility. The list of required packages in this case depended on the MUSIC libraries used in the project.

- The project was added to "Exported Packages" in the bundle creation utility. The application was then exported using the "Export Wizard" of the bundle creation utility. The output was a OSGi bundle in the form of a ".jar" file.
- The bundle was uploaded to the middleware using the MUSIC GUI. The application was started using the GUI as well.

5.6 Context Plugins

The MUSIC manual[34] for developing plug-ins provides a detail guide to building of context plug-ins that can be used in general for any context-aware system. For our work, we did not develop any context sensor plug-in by ourselves; rather the screen sensor context plug-in of the middleware has been used by us to evaluate the functionality of the test case discussed in the next chapter.

Chapter 6

Evaluation

This chapter presents the evaluation of our solution. We evaluate the work from a number of different points of view. For example, a proof of concept prototype evaluates the feasibility of our proposed conceptual solution and its implementation, a user survey measures the applicability of the solution to the needs of real users and a qualitative analysis of the solution against a set of relevant criteria helps measuring the fulfillment of its scientific goals. Section 6.1 provides a proof of concept using a simplified scenario. We carry out a small user survey in section 6.2 to complement the evaluation process. In section 6.3 we use a set of parameters to analyze the solution in depth. Finally we use the criteria defined in section 3.2 to draw a comparison between our solution and other existing dynamic service composition frameworks in section 6.6.

6.1 Proof of Concept

In this section we take a subset of the scenario described in chapter 2 and use it to develop a composite service using our prototype implementation. Section 6.2.1 outlines the test case. The composition specification based on the test case is presented in section 6.2.2. The outputs from the composition shown in section 6.2.3 serve as a proof of the concept presented in earlier chapters.

6.1.1 The Description of the Test Case

The simplified scenario is as follows:

John is a businessman. He is flying to Rome on June 12 at 2 p.m. in order to meet his clients. The airport bus arrives at the bus stop near his house every 30 minutes. In order to remind himself about the suitable bus time so that he can catch the flight he wants to be sent an SMS reminder at 11:55 a.m. He also wishes to remind himself with an email service, but only if Internet connectivity is available.

From the above scenario we identify the key elements and their corresponding mapping to the UbiCompForAll concepts:

- 11:55 a.m. on June 12 - Trigger
- SMS service - Step
- Email service - Conditional Step
- Internet Availability - Condition

6.2 Composition Specification

Based on the UbiCompForAll concepts derived in section 6.2, the following service composition has been manually composed using the UbiCompForAll service composition template from section 4.3.3:

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <ecore:EPackage xmi:version="2.0"
4                 xmlns:xmi="http://www.omg.org/XMI"
5                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                      instance"
6                 xmlns:ecore="http://www.eclipse.org/em
7                 name="collabservcompexample"
8                 nsURI="http://ubicompforall.org/"
9                 nsPrefix="collabservcompexample">
10   <eClassifiers xsi:type="ecore:EClass"
11                 name="CalendarAlert"
```

```
12              eSuperTypes="../../org.ubicompforall.
                    simplelanguage/model/SimpleLanguage.ecore#//
                    Trigger">
13    <eStructuralFeatures xsi:type="ecore:EAttribute"
14                         name="executiondatetime"
15                         eType="ecore:EDataType http://www.eclipse.
                               org/emf/2002/Ecore#//EDouble"
16                         value="c,eq,1155120611" />
17    </eClassifiers>
18    <eClassifiers xsi:type="ecore:EClass"
19                  name="NetworkAvCheck"
20                  eSuperTypes="../../org.ubicompforall.
                        simplelanguage/model/SimpleLanguage.ecore#//
                        Condition">
21      <eStructuralFeatures xsi:type="ecore:EAttribute"
22                           name="InternetAv"
23                           eType="ecore:EDataType http://www.
                                eclipse.org/emf/2002/Ecore#//
                                EBoolean"
24                           value="c,eq,true" />
25      <eClassifiers xsi:type="ecore:EClass"
26                    name="ExampleComposition"
27                    eSuperTypes="../../org.ubicompforall.
                        simplelanguage/model/SimpleLanguage.ecore#//
                        Task"
28                    Refersto="CalendarAlert">
29      <eClassifiers xsi:type="ecore:EClass"
30                    name="SendSMS"
31                    eSuperTypes="../../org.ubicompforall.
                        simplelanguage/model/SimpleLanguage.ecore#//
                        Step">
32        <eStructuralFeatures xsi:type="ecore:EAttribute"
33                             name="to"
34                             eType="ecore:EDataType http://www.
                                  eclipse.org/emf/2002/Ecore#//
                                  EString"
35                             value="+4748341631" />
36        <eStructuralFeatures xsi:type="ecore:EAttribute"
37                             name="message"
38                             eType="ecore:EDataType http://www.
                                  eclipse.org/emf/2002/Ecore#//
                                  EString"
39                             value="bus has arrived" />
40     </eClassifiers>
41     <eClassifiers xsi:type="ecore:EClass"
42                   name="SendEmail"
43                   eSuperTypes="../../org.ubicompforall.
                        simplelanguage/model/SimpleLanguage.ecore
                        #//Conditional">
```

```
44                    Refersto="NetworkAvCheck">
45          <eStructuralFeatures xsi:type="ecore:EAttribute"
46                               name="from"
47                               eType="ecore:EDataType http://www.
                                     eclipse.org/emf/2002/Ecore#//
                                     EString"
48                               value="imtiazud@stud.ntnu.no" />
49          <eStructuralFeatures xsi:type="ecore:EAttribute"
50                               name="to"
51                               eType="ecore:EDataType http://www.
                                     eclipse.org/emf/2002/Ecore#//
                                     EString"
52                               value="imtu7986@gmail.com" />
53          <eStructuralFeatures xsi:type="ecore:EAttribute"
54                               name="title"
55                               eType="ecore:EDataType http://www.
                                     eclipse.org/emf/2002/Ecore#//
                                     EString"
56                               value="Bus Schedule" />
57          <eStructuralFeatures xsi:type="ecore:EAttribute"
58                               name="msg"
59                               eType="ecore:EDataType http://www.
                                     eclipse.org/emf/2002/Ecore#//
                                     EString"
60                               value="bus has arrived" />
61      </eClassifiers>
62    </eClassifiers>
63
64 </ecore:EPackage>
```

Listing 6.1: Composition Description of the Proof of Concept Scenario

6.2.1 Resulting Composition

The composition description in the last section was used as an input to the prototype. For convenience, we simulated the functionality of the "Internet Availability" with the screen orientation context plug-in provided by MUSIC, instead of building our own context plug-in. We also built simple services "SendEmailService.java" shown in listing B.7 on pages 100-101 and "SendSMSService.java" shown in listing B.8 on pages 101-102 for the purpose of evaluation. Figure 6.1 shows the output when the Internet is not available. It should be noted that the application uses and displays values that were input by the user during composition as shown in the XML in section 6.2.

This includes the phone number ("+4748341631") and message ("bus has arrived") that should be delivered by the SMS service.

Figure 6.2 shows the output of the composition when the Internet is available. Both SMS and Email services are executed this time. The contents of the email service such as sender("imtiazud@stud.ntnu.no"), receiver("imtu7986@gmail.com"), title("bus schedule"), and body("bus has arrived") are obtained from the composition provided by the end user in the XML shown in section 6.2. The application in the figure displays this information.

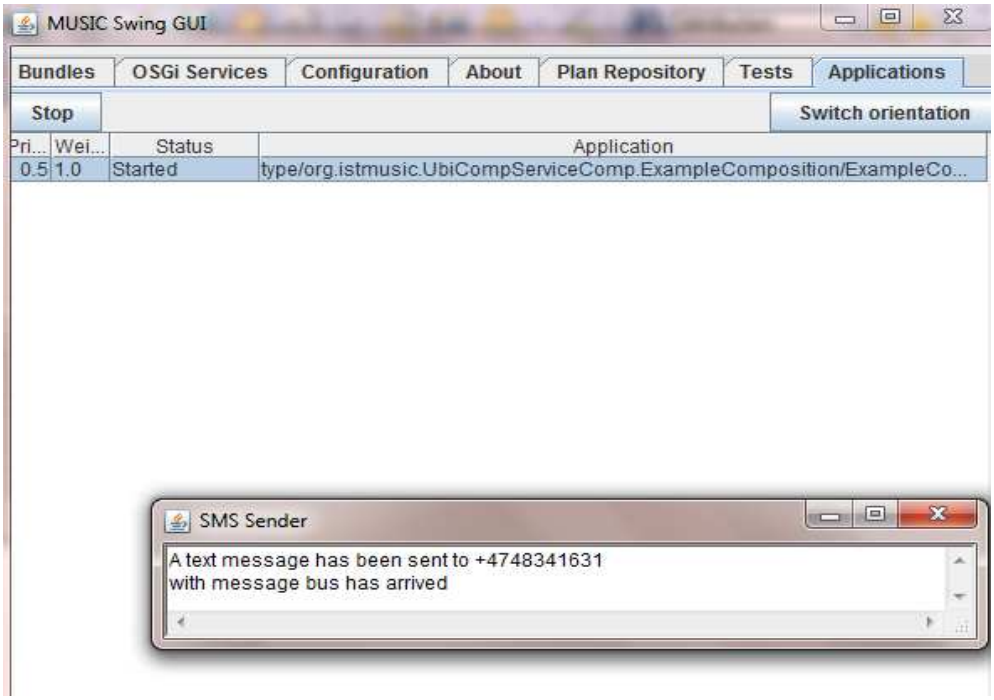

Figure 6.1: Composition without Internet Connectivity

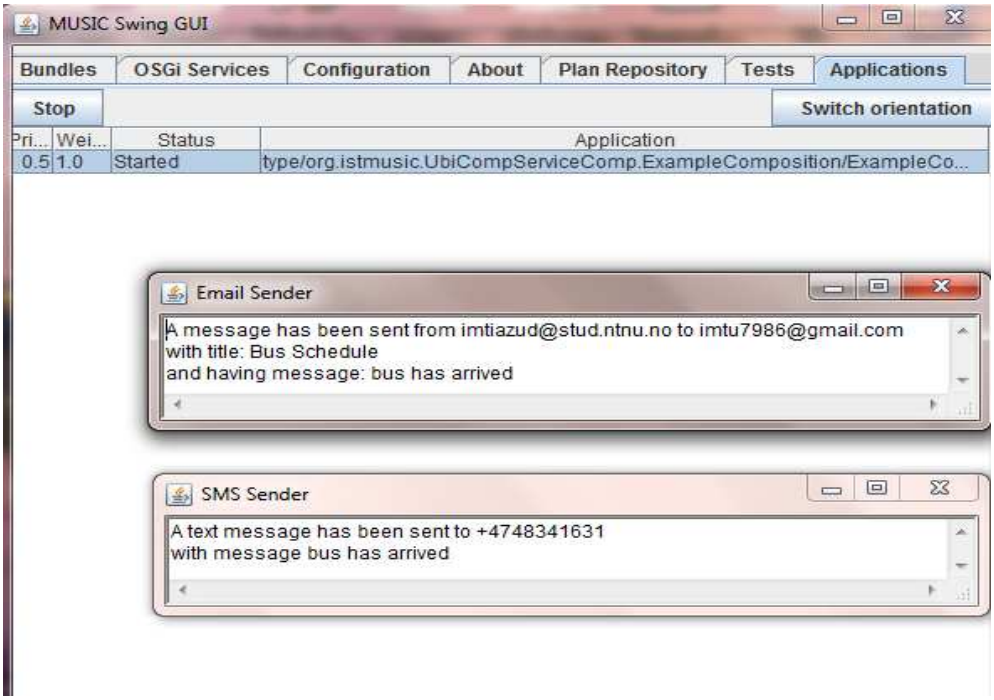

Figure 6.2: Composition with Internet Connectivity

The outputs of the composition certainly prove the validity of our solution. The user in this case had been able to define "Internet connectivity" as the condition that was then used by the composition engine at runtime to decide whether "E-mail" service would be included in the composed service or not. Runtime adaptation of the composition based on user requirement was verified first by running the composition in a computer without Internet connectivity as shown in figure 6.1 and then by connecting the computer to the Internet without stopping the application as shown in figure 6.2. Additionally, the system required the user to specify the composition implicitly, thus the user mentioned the type of the service to be used in the composition rather than indicating the actual service instance.

6.3 User Survey

Although it was not possible to carry out a formal usability study due to time constraint, we tried to evaluate the validity of our solution by conducting an informal survey. Five candidates participated in this study. All of them were

Master's students studying Telematics at NTNU. While they have previous knowledge of computer programming languages such as C, Java, and others, they had no previous knowledge regarding UbiCompForAll or MUSIC. As the fully functioning composition tool was unavailable, the participants were given instructions regarding the process of specifying their own compositions. Although initially it was cumbersome, the participants managed to write their service compositions by specifying both service types and conditions using the template shown in section 4.4. The example used in section 6.1 is also used for explaining the solution the survey participants. Their comments on the entire mechanism, after they observed the execution of their compositions, were summarized in the form of a question/answer session:

How easy was it to compose and how easy was it to include the conditions?

The participants said that had there been a fully functioning end user-friendly service composition tool available, it would have been really easy to compose services as it took some time to understand the composition from the XML file. They further concluded that it was very easy to incorporate conditions in the composition as the UbiCompForAll notation is rather intuitive in nature.

How efficient was the nature of the composition?

In response to the above question, the users said that they did not need to find the specific service that meets each of their criteria. According to them, this made the system user-friendly. Additionally, the resulting composition behaved exactly as they wanted it to behave, that is the composition adapted according to the runtime environment and users' needs. This led them to conclude that the system generated the optimal composition (the most desired composition according to user's requirement in a given context and point in time). Additionally, after we briefly explained how the system functioned they said it was efficient, as they felt that in this system existing services were combined together as per their requirements to carry out a task which otherwise would have been really difficult and inconvenient to achieve if new programs had to be written each time.

6.4 Qualitative Analysis

In this section we make our qualitative analysis from three different perspectives: design, development, and user perspectives. Starting with the general objective in section 6.5.1 we continue towards a comprehensive evaluation of the prototype based upon our own parameters: extensibility, flexibility and control, re-usability and portability, modularity, development approach, compatibility with user interface, and usability, in sections 6.5.2, 6.5.3, and 6.5.4. Alongside these parameters we use the key issues: service discovery, service coordination and management, uniform information exchange infrastructure, fault tolerance and scalability, and adaptability , that are related to service composition described by Chakrabarty and Joshi[35] to evaluate the solution in the section 6.5.2.

6.4.1 Meeting the Objective

The primary objective of this thesis project was to devise a service composition solution that would bridge the gap between end user control and runtime adaptation. In an ideal ubiquitous computing environment, the user should be able to dictate both the quality and substance of a composition from an abstract level with as much or as little detail as he or she prefers. In our prototype the user orchestrates a composite composition by defining the set of constituent services, their corresponding properties, and the goal driving the composition. We have also incorporated the continuous re-evaluation of adaptation parameters and re-configuration of adaptation throughout the life cycle of the composition in a given context. This fulfills the characteristic requirement of a ubiquitous computing environment whose state is almost never constant.

Beside meeting the overall objective of the project, the realization of the generic solution with UbiCompForAll and MUSIC fulfills the implicit goals stated in section 1.2 in the first chapter. The service composition notation of UbiCompForAll provided us a formal method for describing services. Coupling this with the adaptation capabilities of the MUSIC middleware allows users to easily design and compose services of their choice. The evaluation of service variants at runtime by the utility function ensures composition optimization (the generation of the most desired composition according to user requirement in a given context and point in time) to

meet the user's goal. Finally, the meta-modeling support in the form of triggers and conditions in UbiCompForAll notation enables users to define their goals along with the service components, thus making the process of forming collaborative services a lot easier. This is not only applicable for UbiCompForAll but rather is true for any other composition notation mechanism supporting specification of both the goals and the components of a composition.

6.4.2 Design

In this section we evaluate the framework in terms of extensibility in section 7.3.1, flexibility and control in section 7.3.2, re-usability and portability in section 7.3.3, service discovery in section 7.3.4, service coordination and management in section 7.3.5, uniform information exchange infrastructure in section 7.3.6, fault tolerance and scalability in section 7.3.7, and adaptability in section 7.3.8.

Extensibility

The design of the framework is such that it can be easily extended in order to eventually fulfill all the requirements of a ubiquitous environment. For example, in the future if users are allowed to specify the composition using some high level definition, then the only modification in the system that would be required is to build a language parser which translates this high level definition to generate a composition description in UbiCompForAll notation. The rest of the system should operate as it currently does.

Similarly the notion of allowing users to specify "as much or as little detail" as they like while describing the composition is easily achievable by incorporating an additional module that extracts previously stored information from the user's profile and uses this information to automatically fill out property values of services that are not filled in by the user himself. Learning can also be introduced by enhancing the module to automatically update the user's profile with the information drawn by sensing his or her activities.

Flexibility and Control

The flexibility of the design is best portrayed by the plug and play mechanism with which users can introduce conditions in the composition. Since triggers and conditions are modeled as context plug-ins, a user simply acquires the particular plug-in that will evaluate the specified condition and plugs it in with the rest of the bundle. The only other change needed is in the service description template file where the information regarding the trigger or condition must be added along with the other components. The UbiCompToMusic translator does not have any difficulty with the newly added conditions or triggers because of its generic translation. Also, the runtime environment must have this context plug-in included in the application's bundle. This approach of adding the context plug-in to the framework is more end user friendly when compared to the process of adding libraries to source codes written in typical programming languages. This is because in this case the user does not provide instructions to the system as to how the plug-in should be used in order to evaluate his provided condition, rather he simply adds the plug-in as a library and selects the trigger or condition that will be evaluated by the middleware using that plug-in.

The level of flexibility achieved by using context plug-ins has a tremendous significance in terms of the user's control of the composition. Users will not only be able to select components based on their functional and other non-functional properties retrieved from the service description file, but they will also be able to tune the nature of the composition based on innovative parameters such as best-rated service among friends in a social network. This can be achieved by building plug-ins that will query the necessary information from different sources, such as the social network in this case, at runtime and provide this information to the system which then uses the utility function to select the most desired service instance as per the user's requirement(s). Quality of Service including selection of secure services can also be incorporated into the composition in the same way by the use of plug-ins. The user in this case will specify the Quality of Service requirement in the form of a condition and a context plug-in will be used to evaluate the actual value for the specified Quality of Service parameter at runtime for a given component. The system will then use the utility function to compare whether the component fulfills the user's Quality of Service requirement or not. The component becomes a part of the resulting composition only if it fulfills the requirements of the user. Other forms of security such as secure communication between the components can be

ensured by developing a secure communication protocol and adding it to the existing pool of communication protocols supported by MUSIC.

The pluggable nature of the framework discussed above makes its enhancement a relatively simple job as new types of plug-ins can be easily developed by third party developers and provided to the end users.

Re-usability and Portability

The re-usability of composition descriptions in the proposed solution increases the efficiency of service composition. Users can use compositions that they have created in the past either by retaining the composition in its entirety or by editing it to whatever extent they wish.

Another feature that enhances efficiency is portability. Due to the use of Java, any composition that is composed on one platform can be transferred and used in another platform given that the necessary context plug-ins are made available.

Service Discovery

Since the services in a pervasive environment can follow a wide range of definitions, it is important for any good service composition framework to be able to support a variety of discovery techniques. While MUSIC currently supports quite a few including UPnP and SLP, the range can be extended by simply plugging in new discovery and communication protocols directly into the middleware as discussed in [33]. This is possible because MUSIC has a pluggable architecture.

Service Coordination and Management

The MUSIC middleware itself manages and coordinates the components in our framework. The advantage of using MUSIC to manage the entities is that it already has built-in functions for co-ordination and management in both stand-alone and distributed environments. So, the measure of efficiency

in handling co-ordination and management of our solution is exactly equal to the already proven level of efficiency of MUSIC as discussed in [33].

Uniform Information Exchange Infrastructure

In a ubiquitous environment, we can have different services following different information exchange mechanisms operating together. It is important that an ideal composition framework is able to communicate with a variety of services and blend them together in a composition even though they might utilize quite different types of information exchange paradigms. MUSIC currently can only support services that are tailored following the Service Oriented Architecture (SOA). However, the general solution provided in this project is not limited by the built-in capability of MUSIC and therefore further research is needed to formulate domain specific solution for building an uniform information exchange infrastructure.

Fault Tolerance and Scalability

In the ideal case, a fault tolerant composition system should be able to detect when services become unavailable in the distributed environment and invoke a fault control mechanism. Because MUSIC is at the core of our framework we can exploit MUSIC's inherent ability to detect the event that will occur when services become unavailable. Once the event has been generated, an automatic adaptation process can be initiated. New services can replace unavailable ones, but because no state information was retained, the mechanism is not that robust for stateful services, while it can operate smoothly for stateless services.

Scalability is another point of concern in a system where a user can compose large complex composite services that can have a lot of application variants. Fortunately MUSIC has an inherent adaptation mechanism to deal with the computational complexity of a large number of application variants. Depending on the number of application variants it can either use brute force, greedy, or a more sophisticated algorithm[36, 37, 32].

Adaptability

Services are not persistent in terms of their existence in a ubiquitous environment due to various reasons, such as network failure, service down-time, etc. This fact makes it vary important for a service composition platform to be adaptive in nature and make the best use of currently available services. As mentioned earlier the proposed framework has the ability to dynamically adapt according to the current state of the context in order to optimize itself to meet the user's requirement(s).

6.4.3 Development

In this section we evaluate the framework in terms of modularity in section 7.4.1, development approach in section 7.4.2, and compatibility with user interface in section 7.4.3.

Modularity

The code structure of the prototype is divided into separate sections. This means that developers will be able to enhance one part without having to worry about a ripple effect that any change might have on rest of the code. For example, a developer is free to construct the utility function completely in their own way. In order to do so, they simply modify the portion of the UbiCompToMusic method that generates the utility function.

Development Approach

We followed the incremental development methodology for engineering the prototype framework. This means that feedback provided during the previous step of development was followed as necessary.

Compatibility with user interface

Any front-end generating the composition file in the UbiCompForAll service description notation can be used as the user interface for the proposed framework. From the developers' perspective, they are thus relieved of the task of integrating an interface with the framework engine.

6.4.4 Usability

As described earlier in Chapter 6, the usability review had five persons who gave positive feedback regarding the framework. These five persons have commented that they have been empowered through the notion of user-driven adaptive composition as they have control over both the content and quality of the service without having to explicitly identify specific service instances at design time.

6.4.5 Comparison with Existing Frameworks

In order to evaluate our work, we use the same set of parameters that we devised in section 3.1 to analyze existing end user service composition frameworks. Our framework fulfills all the criteria defined in section 3.2. Just like the other frameworks supporting dynamic composition discussed in Chapter 3, it allows the end users to specify the composition at runtime implicitly at type level. Additionally, it has a provision for including an user interface which we have stated is under development by others.

In addition to the above features, our framework allows users to specify parameters that can be used to select the composition variant that they wish to use. Composition adaptation is carried out in a context aware environment by dynamically building application variants based on user criteria provided at runtime. To the best of our knowledge, this is unique among all the reviewed frameworks and other dynamic service composition techniques. Thus evaluating the performance of our primary contribution, composition adaptation based on end user preferences, with respect to other works is not possible at this time.

Chapter 7

Discussion

In this chapter we first outline the achievements of this thesis project in section 8.1. Next we list the shortcomings of the proposed solution and based on them we suggest some future work in section 8.2. We also foresee the possibility of a research project at a bigger scale. We conclude the discussion in section 8.3 portraying our view to the problem addressed in this thesis project based on our experience throughout the whole process.

7.1 Achievements and Lessons Learned

The achievements and lessons learned from this thesis project are described below:

- We have learned a systematic approach of conducting research. Although we had to re-schedule the submission date, mainly because of a longer than anticipated implementation phase, we followed a specific research method by starting with a task description, the specification of milestones, and the estimation of time required for accomplishing the tasks as shown in Chapter 1.
- We have studied and acquired in depth understanding of the state of the art in end user development, dynamic service composition, and

adaptation. We have also developed a usage scenario and analyzed it in order to formulate the problem and acquire insight in it.

- We have developed a conceptual solution to bridge the gap between end user development and automatic service adaptation by providing user-driven runtime composition adaptation mechanism. In doing so we have enhanced the UbiCompForAll service composition notation support and designed a pluggable architecture that provides users to easily define their conditions based on which composition adaptation takes place.

- We have implemented a prototype of our conceptual solution. In that process, we have also enhanced the MUSIC support by adding the automatic generation of the source code for the utility function.

- We have evaluated the solution from a number of different perspectives: a proof of concept prototype to evaluate the feasibility, a user survey to measure the applicability and a qualitative analysis to determine the scientific value of the work.

7.2 Limitations and Future Work

We believe that a scientific work is a part of a bigger process towards the development of knowledge. Therefore, besides our achievements we have also identified a number of points that offer an opportunity of improvement. Such points are listed below:

- We developed the complete conceptual solution to bridge the gap between end user development and automatic service adaptation. However, in the implementation of the prototype, we did not have enough time to implement the facility in order to automate the last phase of our framework, i.e., the automatic deployment of the application bundle to run on the MUSIC middleware.

- In our Proof of Concept we only used a subset of the original scenario to keep things simple. While the chosen subset of the scenario is good enough to evaluate the work, a more complete implementation of the scenario, especially by realizing the actual functionality of the services could provide a more comprehensive understanding of the solution.

- In the implementation phase, we used simplified and internal services only.
- We could not use the UbiComposer service composition tool, because it is still under development. This deprived us from employing the survey participants in composing their services using a graphical user interface.

Thus, most of the limitations concern the implementation. The work can be further improved by:

- Completing the entire automation process of the composition life cycle by automatic creation of the application bundle and uploading it to the MUSIC middleware
- Enhancing the framework in order to test connected services and external service support
- Developing basic Quality of Service related context plug-ins
- Enhancing the UbiComptoMusic translator so that end users may dictate the design of the utility function from an abstract level if they wish
- Evaluating the framework with the complete scenario and employing the end user-friendly service composition tool, which is planned to be released in August, 2011.

Apart from these, we also believe that the problem addressed in this thesis will gain more importance in the near future and a commercially viable solution might be needed. We believe that a future research project on a larger scale than this thesis project can build upon our work.

7.3 Conclusions

In this thesis project the problem we have addressed was how to empower end users in a ubiquitous computing environment. In this context, automation

and adaptation are often the most desirable goals. However, automatic service composition without the consideration of users' preferences does not bring about user satisfaction to a great extent as it is natural for each user to want different things in a given context and point in time. In order to understand this issue further we have critically evaluated the state of the art in end user service composition. Existing works in this field have only addressed areas related to the development of a service composition tool without looking too deeply into the issue of end user control. We thus go one step further and propose a solution for intelligent service composition where users are able to define a composition both in terms of quality and the type of constituent services based on which this service will be dynamically adapted at runtime. Next we use self adaptive middleware and a service description notation to instantiate this solution. With the help of a test case and the prototype, we proved the feasibility of our idea.

We believe that the automation achieved by integrating end user control and adaptation are key to obtaining the user's desired service composition despite the ever changing characteristic of ubiquitous computing environment and its service landscape. Analysis of our design shows that it has the potential to evolve into a method with which the user in a given context will be able to provide himself or herself with the optimal composition according to his or her requirements without the user having to explicitly define all of the requirements. We, therefore, conclude that our work is a firm step towards providing the support for intelligent service composition which maximizes end user satisfaction, control, and adaptation by using services in a ubiquitous computing environment.

Appendix A

Original Problem Description

Title: **Collaboration-based intelligent service** composition at runtime by end users

With mobile devices being an integral part of the daily life of millions of users having little or no ICT knowledge, mobile services are being developed to save them from difficult or tedious tasks without compromising with their needs. In the UbiCompForAll project, starting with a number of real life scenarios we have been working towards supporting end-users in managing their services in an efficient and user-friendly manner. We observe that these scenarios consist of sub-tasks that can be solved with collaborative service units. Therefore, a composition of such service units will serve the needs of the end-user for the complete scenario. We envisage that a visual formalism and tools can be developed to support these end-users in creating such service compositions. Moreover, methodologies and middleware can significantly reduce the complexity of developing composite services.

In this thesis, the student will investigate the collaboration-based composition of services at runtime. Collaborative services are performed by objects that may take initiatives towards the service users. This is typical for telecom services, but also for many new services such as attentive services, context aware services, notification services and ambient intelligence. Such services in general entail a collaboration among several active objects. Partial service functionalities can be specified as collaborating roles, where such roles are realized by the collaborative objects. Service composition involves in specifying the collaboration both in a static way at design time as well as

supporting dynamic discovery, learning, adaptation and binding at runtime. The expected results of the work include evaluation of different end user-friendly specification techniques for collaborative services, utilizing the tool support in composing services by the end users and applying the concepts and developments in one or more application domain(s). Throughout the process the main focus will be on runtime issues; e.g., adjusting the service usage to the changing needs of the user at runtime, issues related to the discovery of new services and selection among alternatives etc.

Appendix B

Project Source Code

B.1 FileWatcher.java

```
1  import java.util.*;
2  import java.io.*;
3
4  public abstract class FileWatcher extends TimerTask {
5    private long timeStamp;
6    private File file;
7    private long length;
8
9    public FileWatcher( File file ) {
10     this.file = file;
11     this.length = file.length();
12     this.timeStamp = file.lastModified();
13   }
14
15   public final void run() {
16     long timeStamp = file.lastModified();
17
18     if( this.timeStamp != timeStamp ) {
19
20       if(this.length != file.length()){
21       this.length = file.length();
22       this.timeStamp = timeStamp;
23       onChange(file);
24       }
25     }
26   }
27
```

```
28   protected abstract void onChange( File file );
29 }
```

Listing B.1: FileWatcher.java

B.2 FlowControl.java

```
1
2 import java.io.File;
3 import java.util.Date;
4 import java.util.Timer;
5 import java.util.TimerTask;
6
7
8 public class FlowControl {
9
10   /**
11    * @param args
12    */
13   public static void main(String[] args) {
14     // TODO Auto-generated method stub
15
16     String fileaddress = new String("C:/Users/IMTIAZ/workspace/
         Flow_Test/src/example.ecore");
17     // monitor a single file
18       TimerTask task = new FileWatcher( new File(fileaddress) )
           {
19         protected void onChange( File file ) {
20           // here we code the action on a change
21           //System.out.println( "File "+ file.getName() +" have
               change !" );
22         Parser parser = new ParserTest();
23         parser.UbiCompToMUSICconverter(fileaddress);
24
25         }
26       };
27
28       Timer timer = new Timer();
29       // repeat the check every second
30       timer.schedule( task , new Date(), 1000 );
31
32   }
33
34 }
```

Listing B.2: FlowControl.java

B.3 Parser.java

```
1
2  //package parser;
3  import java.io.FileNotFoundException;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.PrintStream;
7  import java.text.DateFormat;
8  import java.text.SimpleDateFormat;
9  import java.util.ArrayList;
10 import java.util.Calendar;
11 import java.util.Date;
12
13 import java.util.List;
14 import javax.xml.parsers.DocumentBuilder;
15 import javax.xml.parsers.DocumentBuilderFactory;
16 import javax.xml.parsers.FactoryConfigurationError;
17 import javax.xml.parsers.ParserConfigurationException;
18
19 import org.w3c.dom.Document;
20 import org.w3c.dom.Element;
21 import org.w3c.dom.NodeList;
22 import org.xml.sax.SAXException;
23
24 public class Parser{
25
26   /**
27    * @param args
28    */
29   private static List<Component> ComponentArray = new ArrayList<
       Component>();
30   private static List<EvaluatorDetail> EvalArray = new ArrayList
       <EvaluatorDetail>();
31
32   private static Component getComponent(Element comp) {
33
34     // for each <Compnent> element get type, name, RefersTo and
35     // AttributeArray
36
37     String AttributeName = null;
38     String DataType = null;
39     String Value = null;
40     String RefersTo = null;
41     String ComponentType = comp.getAttribute("eSuperTypes").
         split("//")[1];
42     String ComponentName = comp.getAttribute("name");
43     // if (comp.hasAttribute("Refersto"))
```

```
44      RefersTo = comp.getAttribute("Refersto");
45
46      // retrieve the children nodes of the Task
47
48      NodeList taskchildren = comp.getElementsByTagName("
            eClassifiers");
49
50      if (taskchildren != null && taskchildren.getLength() > 0) {
51        for (int i = 0; i < taskchildren.getLength(); i++) {
52
53          // get the Component element
54          Element elm = (Element) taskchildren.item(i);
55
56          // get the Component object
57          Component taskchilcomp = getComponent(elm);
58
59          // add it to list
60          ComponentArray.add(taskchilcomp);
61
62        }
63      }
64
65      // retrieve the children nodes of the Component
66
67      NodeList children = comp.getElementsByTagName("
            eStructuralFeatures");
68
69      // create a new component
70      Component newcomponent = new Component();
71
72      // iterate through the children and parse their information
73      if (children != null && children.getLength() > 0) {
74        for (int i = 0; i < children.getLength(); i++) {
75
76          // if(children.item(i).getLocalName()!=null){
77          Element attribute = (Element) children.item(i);
78
79          AttributeName = attribute.getAttribute("name");
80          DataType = attribute.getAttribute("eType").split("//")
                [2].split("E")[1];
81          Value = attribute.getAttribute("value");
82
83          Attribute newattribute = new Attribute();
84          newattribute.AttributeName = AttributeName;
85
86          if(DataType.equals("Int"))
87            DataType = new String("Integer");
88          newattribute.DataType = DataType;
89          newattribute.Value = Value;
```

```
90
91          newcomponent.AttributeArray.add(newattribute);
92
93          // }
94
95        }
96      }
97
98      // Create a new Component with the value read from the xml
            nodes
99
100     newcomponent.ComponentType = ComponentType;
101     newcomponent.ComponentName = ComponentName;
102     newcomponent.RefersTo = RefersTo;
103
104     return newcomponent;
105   }
106
107   public void UbiCompToMUSICconverter(String fileaddress) {
108     // TODO Auto-generated method stub
109
110     // TODO Auto-generated method stub
111
112     try {
113       // instantiating a DOM implementation
114       DocumentBuilderFactory factory = DocumentBuilderFactory
115       .newInstance();
116       // create a document loader
117       DocumentBuilder loader = factory.newDocumentBuilder();
118
119       // loading a DOM tree
120       Document document = loader
121       .parse(fileaddress);
122       // Retrieve the root element
123       Element tree = document.getDocumentElement();
124
125       // get a node list of the Components in the xml
126       NodeList nl = tree.getElementsByTagName("eClassifiers");
127
128       // iterate through the xml tree, parse the information and
              build a
129       // Component Array
130
131       if (nl != null && nl.getLength() > 0) {
132         for (int i = 0; i < nl.getLength(); i++) {
133
134           // get the Component element
135           Element elm = (Element) nl.item(i);
136           Component comp = new Component();
```

```
137             if (elm.getAttribute("eSuperTypes").split("//")[1].
                    equals("Task")||elm.getAttribute("eSuperTypes").
                    split("//")[1].equals("Trigger")||elm.getAttribute(
                    "eSuperTypes").split("//")[1].equals("Condition"))
                    {
138               // get the Component object
139               comp = getComponent(elm);
140               // add it to list
141               ComponentArray.add(comp);
142             }
143
144           }
145         }
146
147         // Iterate through the ComponentArray and Print the
              elements
148
149         System.out.println("Iterating through ArrayList elements
              ...");
150         for (int i = 0; i < ComponentArray.size(); i++) {
151           System.out.println(i);
152           System.out.println(ComponentArray.get(i).ComponentName);
153           System.out.println(ComponentArray.get(i).ComponentType);
154           System.out.println(ComponentArray.get(i).RefersTo);
155           for (int j = 0; j < ComponentArray.get(i).AttributeArray
                  .size(); j++) {
156             System.out.println(ComponentArray.get(i).
                    AttributeArray
157                     .get(j).AttributeName);
158             System.out.println(ComponentArray.get(i).
                    AttributeArray
159                     .get(j).DataType);
160             System.out.println(ComponentArray.get(i).
                    AttributeArray
161                     .get(j).Value);
162           }
163         }
164
165         // Generate the Music Code using the parsed UbiCompforAll
              notation
166         MusicCodeGenerator();
167
168         /*
169          * ClassName = tree.getTagName();
170          *
171          * NodeList nlist = tree.getChildNodes(); for (int i = 0;
                i <
172          * nlist.getLength(); i++) { System.out.println("value of
                node " + i
```

```
173         * + " " + nlist.item(i).getAttributes()); }
174         */
175
176        // create a class file with the name of the root element
               of the tree
177
178        /*
179         * FileOutputStream out = new FileOutputStream(ClassName+"
               .java");
180         *
181         * PrintStream p = new PrintStream (out);
182         *
183         * p.println("public class "+ClassName+
184         * "{public static void main(String args[]){System.out.
               println(\"hi\");}}"
185         * );
186         *
187         * p.close();
188         */
189
190      } catch (ParserConfigurationException e) {
191        e.printStackTrace();
192      } catch (IOException e) {
193        e.printStackTrace();
194
195      } catch (SAXException e) {
196        e.printStackTrace();
197      } catch (FactoryConfigurationError e) {
198        e.printStackTrace();
199      }
200
201    }
202
203    private static void MusicCodeGenerator() throws
           FileNotFoundException {
204
205      String CompositionName = null;
206
207      for (int j = 0; j < ComponentArray.size(); j++) {
208        if (ComponentArray.get(j).ComponentType.equals("Task")) {
209          CompositionName = ComponentArray.get(j).ComponentName;
210          System.out.println(ComponentArray.get(j).ComponentName);
211        }
212      }
213
214      // create a class file with the name of the root element of
           the tree
215
```

```
216     FileOutputStream out = new FileOutputStream(CompositionName
            + ".java");
217
218     PrintStream p = new PrintStream(out);
219
220     p.println("/**");
221     p.println("* The MUSIC project (Contract No. IST-035166) is
            an Integrated Project (IP)");
222     p.println("* within the 6th Framework Programme, Priority
            2.5.5 (Software and Services).");
223     p.println("*");
224     p.println("* More information about the project is available
             at: http://www.ist-music.eu");
225     p.println("*");
226     p.println("* This library is free software; you can
            redistribute it and/or");
227     p.println("* modify it under the terms of the GNU Lesser
            General Public");
228     p.println("* License as published by the Free Software
            Foundation; either");
229     p.println("* version 2.1 of the License, or (at your option)
             any later version.");
230     p.println("*");
231     p.println("* This library is distributed in the hope that it
             will be useful,");
232     p.println(" * but WITHOUT ANY WARRANTY; without even the
            implied warranty of");
233     p.println("* MERCHANTABILITY or FITNESS FOR A PARTICULAR
            PURPOSE. See the");
234     p.println("* GNU Lesser General Public License for more
            details.");
235     p.println("*");
236     p.println("* You should have received a copy of the GNU
            Lesser General");
237     p.println("* Public License along with this library; if not,
             write to the");
238     p.println("* Free Software Foundation, Inc., 59 Temple Place
            , Suite 330,");
239     p.println("* Boston, MA 02111-1307 USA");
240     p.println("*/");
241     p.println("\n");
242
243     p.println("// TODO: package name generation/checking");
244     if (CompositionName != null)
245       p.println("package org.istmusic.UbiCompServiceComp."
246           + CompositionName.toLowerCase() + ";");
247     p.println("\n");
248
249     p.println("import org.istmusic.mw.model.*;");
```

```
250     p.println("import org.istmusic.mw.model.property.*;");
251     p.println("import org.istmusic.mw.resources.impl.descriptors
            .ResourceVocabulary;");
252     p.println("import java.util.HashMap;");
253     p.println("import java.util.Map;");
254     p.println("import java.util.Vector;");
255     p.println("\n");
256     p.println("\n");
257
258     p.println("/**");
259     p.println("* This class implements the Hierarchical bundle")
            ;
260     p.println("* <p/>");
261
262     DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
263     Date date = new Date();
264     p.println("* Date: " + dateFormat.format(date));
265
266     Calendar now = Calendar.getInstance();
267     p.println("* Time: " + now.get(Calendar.HOUR_OF_DAY) + ":"
268         + now.get(Calendar.MINUTE) + ":" + now.get(Calendar.
                SECOND));
269     p.println("*");
270     p.println("* @author UbiCompToMusic Transformation Tool");
271     p.println("*/");
272     p.println("\n");
273
274     p.println("public class " + CompositionName + " implements
            IBundle {");
275
276     long number = (long) Math.floor(Math.random() *
            9000000000000L) + 1000000000000L;
277     p.println("\t private static final long serialVersionUID = "
            + number
278         + "L;");
279
280     p.println("\n");
281
282     p.println("// Type Names");
283     p.println("private static MusicName "
284         + CompositionName
285         + "App = MusicName.nameFromString(\"/type/org.istmusic.
                UbiCompServiceComp."
286         + CompositionName + "/" + CompositionName + "App\");");
287     for (int i = 0; i < ComponentArray.size(); i++) {
288       if (ComponentArray.get(i).ComponentType.equals("
              ConditionalStep")
289           || ComponentArray.get(i).ComponentType.equals("Step")
```

```
290             || ComponentArray.get(i).ComponentType.equals("Query")
                    ) {
291         p.println("private static MusicName "
292             + ComponentArray.get(i).ComponentName
293             + "_Type = MusicName.nameFromString(\"/type/org.
                    istmusic.UbiCompServiceComp."
294             + CompositionName + "/"
295             + ComponentArray.get(i).ComponentName + "_Type\");")
                    ;
296       }
297     }
298
299     p.println("// Names");
300     p.println("private static String Composite" +
            CompositionName
301         + "_Name = \"Composite" + CompositionName + "\";");
302     for (int i = 0; i < ComponentArray.size(); i++) {
303       if (ComponentArray.get(i).ComponentType.equals("
              ConditionalStep")
304           || ComponentArray.get(i).ComponentType.equals("Step")
305           || ComponentArray.get(i).ComponentType.equals("Query")
                  ) {
306         p.println("private static String "
307             + ComponentArray.get(i).ComponentName + "_Name = \""
308             + ComponentArray.get(i).ComponentName + "\";");
309       }
310     }
311     p.println("\n");
312
313     p.println("//get application types");
314     p.println("public ApplicationType[] getApplicationTypes(){")
            ;
315     p.println("\n\t ApplicationType appType_0 = new
            ApplicationType("
316         + CompositionName + "App, null);");
317     p.println("\n\t ApplicationType[] appTypes = new
            ApplicationType[] {");
318     p.println("\t\t appType_0");
319     p.println("\t};");
320     p.println("\t return appTypes;");
321     p.println("}");
322
323     p.println("//get component types");
324     p.println("public ComponentType[] getComponentTypes(){");
325     int compno = 0;
326     for (int i = 0; i < ComponentArray.size(); i++) {
327       if (ComponentArray.get(i).ComponentType.equals("
              ConditionalStep")
328           || ComponentArray.get(i).ComponentType.equals("Step")
```

```
329             || ComponentArray.get(i).ComponentType.equals("Query")
                    ) {
330           p.println("\n\t ComponentType compType_" + compno
331               + " = new ComponentType("
332               + ComponentArray.get(i).ComponentName + "_Type, null
                      );");
333           compno++;
334         }
335       }
336       p.println("\n\t ComponentType[] compTypes = new
            ComponentType[] {");
337       for (int i = 0; i < compno; i++)
338         p.println("\t\t compType_" + i + ",");
339       p.println("\t };");
340       p.println("\t return compTypes;");
341       p.println("}");
342
343       p.println("//Create and install plans");
344       p.println("private AtomicPlan[] getAtomicPlans(){");
345       int noofatplans = 0;
346       for (int m = 0; m < ComponentArray.size(); m++) {
347         if (ComponentArray.get(m).ComponentType.equals("
              ConditionalStep")
348             || ComponentArray.get(m).ComponentType
349             .equals("ConditionalQuery"))
350           noofatplans++;
351
352       }
353
354       p.println("\t AtomicPlan[] ATOMIC_PLANS = new AtomicPlan["
355           + noofatplans + "];");
356
357       int counter = 0;
358       for (int i = 0; i < ComponentArray.size(); i++) {
359         if (ComponentArray.get(i).ComponentType.equals("
              ConditionalStep")
360             || ComponentArray.get(i).ComponentType
361             .equals("ConditionalQuery")) {
362           p.println("\n\t String[] contextDep_" + counter + " = {"
                );
363
364           if (ComponentArray.get(i).RefersTo != null) {
365             p.println("\t\t new String (\"http://www.ist-music.eu/
                    Ontology_v0_1.xml#Thing.Concept.Entity."
366                 + ComponentArray.get(i).RefersTo
367                 + ";http://www.ist-music.eu/Ontology_v0_1.xml#
                      Thing.Concept.Scope."
368                 + ComponentArray.get(i).RefersTo + "\")");
369             p.println("\t };");
```

```
370
371             p.println("\t ATOMIC_PLANS[" + counter
372                 + "] = new AtomicPlan("
373                 + ComponentArray.get(i).ComponentName + "_Name, "
374                 + ComponentArray.get(i).ComponentName + "_Type, "
375                 + ComponentArray.get(i).ComponentName
376                 + ".class.getName(), contextDep_" + counter + ");"
                        );
377             p.println("\n\t {");
378             p.println("\t\t Map propertyMap = myCreateMap(");
379
380             int j;
381             int index = 0;
382             for (j = 0; j < ComponentArray.size(); j++) {
383               if (ComponentArray.get(j).ComponentName
384                    .equals(ComponentArray.get(i).RefersTo)) {
385                 // p.println("\t\t\t new String[]{\""+
386                 // ComponentArray.get(j).AttributeArray.get(0).
                        AttributeName
387                 // + "\"},");
388                 index = j;
389
390               }
391
392             }
393             // p.println("\t\t\t\t new Object[]{");
394
395             int flag = 0;
396             String temp[] = null;
397             EvaluatorDetail evaldet = new EvaluatorDetail();
398
399             if (ComponentArray.get(index).AttributeArray.get(0).
                    Value
400                  .contains(",p,")) {
401
402               temp = ComponentArray.get(index).AttributeArray.get
                        (0).Value
403               .split(",p,");
404               evaldet.operator = new String("+");
405             } else if (ComponentArray.get(index).AttributeArray.
                    get(0).Value
406                  .contains(",s,")) {
407
408               temp = ComponentArray.get(index).AttributeArray.get
                        (0).Value
409               .split(",s,");
410               evaldet.operator = new String("-");
411             } else if (ComponentArray.get(index).AttributeArray.
                    get(0).Value
```

```
412                 .contains(",d,")) {
413
414             temp = ComponentArray.get(index).AttributeArray.get
                    (0).Value
415             .split(",d,");
416             evaldet.operator = new String("/");
417         } else if (ComponentArray.get(index).AttributeArray.
                get(0).Value
418                 .contains(",m,")) {
419
420             temp = ComponentArray.get(index).AttributeArray.get
                    (0).Value
421             .split(",m,");
422             evaldet.operator = new String("*");
423         } else if (ComponentArray.get(index).AttributeArray.
                get(0).Value
424                 .contains(",g,")) {
425
426             temp = ComponentArray.get(index).AttributeArray.get
                    (0).Value
427             .split(",g,");
428             evaldet.operator = new String(">");
429         } else if (ComponentArray.get(index).AttributeArray.
                get(0).Value
430                 .contains(",ge,")) {
431
432             temp = ComponentArray.get(index).AttributeArray.get
                    (0).Value
433             .split(",ge,");
434             evaldet.operator = new String(">=");
435         } else if (ComponentArray.get(index).AttributeArray.
                get(0).Value
436                 .contains(",l,")) {
437
438             temp = ComponentArray.get(index).AttributeArray.get
                    (0).Value
439             .split(",l,");
440             evaldet.operator = new String("<");
441         } else if (ComponentArray.get(index).AttributeArray.
                get(0).Value
442                 .contains(",le,")) {
443
444             temp = ComponentArray.get(index).AttributeArray.get
                    (0).Value
445             .split(",le,");
446             evaldet.operator = new String("<=");
447         }
448
```

```
449             else if (ComponentArray.get(index).AttributeArray.get
                    (0).Value
450                  .contains(",eq,")) {
451
452               temp = ComponentArray.get(index).AttributeArray.get
                        (0).Value
453               .split(",eq,");
454               evaldet.operator = new String("==");
455             }
456
457             //evaldet.componentname = ComponentArray.get(index).
                    ComponentName;
458
459             evaldet.evalname = ComponentArray.get(index).
                    AttributeArray
460             .get(0).AttributeName;
461             evaldet.operands = temp;
462             System.out.println("\n\n\n\n\n" + temp[0]);
463             System.out.println("\n\n\n\n\n" + temp[1]);
464             evaldet.DataType = ComponentArray.get(index).
                    AttributeArray
465             .get(0).DataType;
466
467             EvalArray.add(evaldet);
468
469             if (!ComponentArray.get(index).AttributeArray.get(0).
                    Value
470                  .contains("c,")) {
471               p.println("\t\t\t new String[]{\""
472                    + ComponentArray.get(index).AttributeArray
473                    .get(0).AttributeName
474                    + "\"},");
475
476               p.println("\t\t\t\t new Object[]{");
477               p.println("\t\t\t\t\t new "
478                    + ComponentArray.get(index).AttributeArray
479                    .get(0).AttributeName + "Evaluator ()});");
480               //p.println("\t\t\t\t\t new Utility()});");
481
482             }
483
484             else if (ComponentArray.get(index).AttributeArray.get
                    (0).Value
485                  .contains("c,")) {
486               p.println("\t\t\t new String[]{\""
487                    + ComponentArray.get(index).AttributeArray
488                    .get(0).AttributeName + "\"},");
489               p.println("\t\t\t\t new Object[]{");
```

```
490             if (ComponentArray.get(index).AttributeArray.get(0).
                    DataType
491                     .equals("String"))
492               p.println("\t\t\t\t\t new ConstProperty(new "
493                     + ComponentArray.get(index).AttributeArray
494                     .get(0).DataType + "(\"" + temp[1]
495                                                     + "\"))});");
496             else
497               p.println("\t\t\t\t\t new ConstProperty(new "
498                     + ComponentArray.get(index).AttributeArray
499                     .get(0).DataType + "(" + temp[1]
500                                                   + "))});");
501           }
502
503         } else {
504           p.println("\t };");
505           p.println("\t ATOMIC_PLANS[" + i + "] = new AtomicPlan
                  ("
506               + ComponentArray.get(i).ComponentName + "_Name, "
507               + ComponentArray.get(i).ComponentName + "_Type, "
508               + ComponentArray.get(i).ComponentName
509               + ".class.getName(), contextDep_" + counter + ");"
                      );
510           p.println("\n\t {");
511           p.println("\t\t Map propertyMap = myCreateMap(");
512           p.println("\t\t\t new String[]{},");
513           p.println("\t\t\t\t\t new Object[]{});");
514         }
515
516         p.println("\n\t\t Map resourceMap = myCreateMap(");
517         p.println("\t\t\t new String[]{},");
518         p.println("\t\t\t\t\t new Object[]{});");
519
520         p.println("\n\t\t Feature[] features = {");
521         p.println("\n\t\t };");
522
523         p.println("\n\t\t ATOMIC_PLANS["
524             + counter
525             + "].addPlanVariant(propertyMap, null, resourceMap,
                    features);");
526         p.println("\t }");
527
528         counter++;
529       }
530
531     }
532
533     p.println("\n\t return ATOMIC_PLANS;");
534     p.println("}//getAtomicPlans()");
```

```
535
536     p.println("\n private ServicePlan[] getServicePlans(){");
537     p.println("\t ServicePlan[] SERVICE_PLANS = new ServicePlan
            [0];");
538     p.println("\n\t return SERVICE_PLANS;");
539     p.println("\n}//getServicePlans()");
540
541     p.println("\n\n private CompositionPlan[]
            getCompositionPlans(){");
542     p.println("\t Vector compositionPlans = new Vector();");
543
544     p.println("\t HashMap roleMap = new HashMap();");
545     p.println("\t roleMap.put(\"" + CompositionName
546         + "App_Role\", new Role(\"" + CompositionName + "
                App_Role\", "
547         + CompositionName + "App));");
548     for (int i = 0; i < ComponentArray.size(); i++) {
549       if (ComponentArray.get(i).ComponentType.equals("
              ConditionalStep")
550           || ComponentArray.get(i).ComponentType.equals("Step")
551           || ComponentArray.get(i).ComponentType.equals("Query")
                  ) {
552         p.println("\t roleMap.put(\""
553             + ComponentArray.get(i).ComponentName
554             + "_Type_Role\", new Role(\""
555             + ComponentArray.get(i).ComponentName
556             + "_Type_Role\", "
557             + ComponentArray.get(i).ComponentName + "_Type));");
558       }
559     }
560
561     p.println("\n\t //Create ROLES for the Component Framework")
            ;
562     p.println("\t Role[] ROLES = new Role[]{");
563     p.println("\t\t (Role)roleMap.get(\"" + CompositionName
564         + "App_Role\"),");
565     for (int i = 0; i < ComponentArray.size(); i++) {
566       if (ComponentArray.get(i).ComponentType.equals("
              ConditionalStep")
567           || ComponentArray.get(i).ComponentType.equals("Step")
568           || ComponentArray.get(i).ComponentType.equals("Query")
                  ) {
569         p.println("\t\t (Role)roleMap.get(\""
570             + ComponentArray.get(i).ComponentName
571             + "_Type_Role\"),");
572       }
573     }
574
575     p.println("\t };");
```

```
576     p.println("\t ComponentFrameworkModel fwModel = new
            ComponentFrameworkModel(ROLES);");
577
578     p.println("\n\t //Create composition plan(s) for Composite"
579         + CompositionName + " composite realization");
580
581     p.println("\n\t String[] contextDep_Composite" +
            CompositionName
582         + " = {");
583
584     int index = 0;
585     for (int k = 0; k < ComponentArray.size(); k++) {
586       if (ComponentArray.get(k).ComponentType.equals("Task"))
587         index = k;
588     }
589
590     String[] taskdep = ComponentArray.get(index).RefersTo.split(
            ",");
591
592     if (ComponentArray.get(index).RefersTo != null) {
593       for (int i = 0; i < taskdep.length; i++) {
594         String comma = new String(",");
595
596         if (i != taskdep.length - 1)
597           p.println("\t\t new String (\"http://www.ist-music.eu/
                Ontology_v0_1.xml#Thing.Concept.Entity."
598               + taskdep[i]
599                       + ";http://www.ist-music.eu/
                            Ontology_v0_1.xml#Thing.Concept.Scope
                            ."
600                       + taskdep[i] + "\")" + comma);
601         else
602           p.println("\t\t new String (\"http://www.ist-music.eu/
                Ontology_v0_1.xml#Thing.Concept.Entity."
603               + taskdep[i]
604                       + ";http://www.ist-music.eu/
                            Ontology_v0_1.xml#Thing.Concept.Scope
                            ."
605                       + taskdep[i] + "\")");
606
607       }
608     }
609     p.println("\t };");
610
611     p.println("\n\t //TODO: Constraints of composition are not
            used");
612     p.println("\n\t //Feature specification");
613
```

```
614     p.println("\n\t Vector featureSpec_Composite" +
            CompositionName
615         + " = new Vector();");
616
617     p.println("\n\t FeatureTypeAssociation[]
            featureTypeAssocs_Composite"
618         + CompositionName
619         + " = new FeatureTypeAssociation[featureSpec_Composite"
620         + CompositionName + ".size()];");
621     p.println("\t featureTypeAssocs_Composite" + CompositionName
622         + " = (FeatureTypeAssociation[])featureSpec_Composite"
623         + CompositionName
624         + ".toArray(new FeatureTypeAssociation[
                featureSpec_Composite"
625         + CompositionName + ".size()]);");
626
627     p.println("\n\t //Composition_0 of realization
            CompositeHelloWorld");
628     p.println("\n\t {");
629     p.println("\t\t //Connection spec");
630     p.println("\t\t ConnectionSpec[] connections = new
            ConnectionSpec[]{");
631     p.println("\t\t };");
632     p.println("\t\t //Port delegation specification");
633     p.println("\t\t PortDelegationSpec[] delegations = new
            PortDelegationSpec[]{");
634     p.println("\t\t };");
635
636     p.println("\n\t\t //Plans for node deployment spec_0");
637     p.println("\n\t\t {");
638     p.println("\n\t\t\t NodeDeploymentSpec[] deployments = new
            NodeDeploymentSpec[]{");
639     p.println("\n\t\t\t };");
640     p.println("\n\t\t\t CompositionSpec composition = new
            CompositionSpec(fwModel, deployments, connections,
            delegations, featureTypeAssocs_Composite"
641         + CompositionName + ");");
642     p.println("\n\t\t\t CompositionPlan compPlan = new
            CompositionPlan(Composite"
643         + CompositionName
644         + "_Name, "
645         + CompositionName
646         + "App, Composite"
647         + CompositionName
648         + ".class.getName(), contextDep_Composite"
649         + CompositionName
650         + ", composition);");
651     p.println("\n\t\t\t compositionPlans.add(compPlan);");
652
```

```
653     p.println("\n\t\t\t\t {");
654     p.println("\n\t\t\t\t\t Map propertyMap = myCreateMap(");
655
656     int k;
657     int ind = 0;
658     for (k = 0; k < ComponentArray.size(); k++) {
659       if (ComponentArray.get(k).ComponentType.equals("Task"))
660         ind = k;
661     }
662
663     String[] taskprop = null;
664
665     if(ComponentArray.get(ind).RefersTo!=null)
666       taskprop = ComponentArray.get(ind).RefersTo.split(",");
667
668     // Global Context Dependencies (trigger and condition)
669
670     String compcomma = new String(",");
671
672     int com = 0;
673
674     p.print("\n\t\t\t\t\t new String[]{");
675
676     for (int a = 0; a < taskprop.length; a++) {
677       for (int j = 0; j < ComponentArray.size(); j++) {
678         if (ComponentArray.get(j).ComponentName.equals(taskprop[
              a])) {
679           // p.println("\t\t\t new String[]{\""+
680           // ComponentArray.get(j).AttributeArray.get(0).
                AttributeName
681           // + "\"},");
682           com = j;
683
684
685
686           if (!ComponentArray.get(com).AttributeArray.get(0).
                Value
687                .contains("c,")) {
688             //EvalArray.add(compevaldet);
689
690             //p.println("\t\t\t\t new String[]{");
691
692
693
694             if(a!=taskprop.length-1)
695               p.print("\""
696                   + ComponentArray.get(com).AttributeArray
697                   .get(0).AttributeName+"\",");
698             else {
```

```
699                 p.print("\""
700                     + ComponentArray.get(com).AttributeArray
701                     .get(0).AttributeName+"\",IPropertyEvaluator.
                            UTILITY_PROPERTY},");
702
703               }
704               //
705               //+ "\",IPropertyEvaluator.UTILITY_PROPERTY},");
706
707
708             }else if(ComponentArray.get(com).AttributeArray
709                 .get(0).Value.contains("c,")) {
710
711               //p.println("\t\t\t\t new String[]{");
712
713               if(a!=taskprop.length-1)
714                 p.print("\""
715                     + ComponentArray.get(com).AttributeArray
716                     .get(0).AttributeName+"\",");
717
718
719               else {
720                 p.print("\""
721                     + ComponentArray.get(com).AttributeArray
722                       .get(0).AttributeName+"\",IPropertyEvaluator
                              .UTILITY_PROPERTY},");
723               }
724             }
725
726             else {
727               p.print("},");
728
729             }
730
731           }
732         }
733       }
734
735       p.println("\n\t\t\t\t new Object[]{");
736
737       EvaluatorDetail compevaldet = new EvaluatorDetail();
738
739       int com1 = 0;
740       for (int z = 0; z < taskprop.length; z++) {
741         for (int y = 0; y < ComponentArray.size(); y++) {
742           if (ComponentArray.get(y).ComponentName.equals(taskprop[
                  z])) {
743
744             com1 = y;
```

```
745
746
747             int compflag = 0;
748             String comptemp[] = null;
749
750
751             if (ComponentArray.get(com1).AttributeArray.get(0).
                    Value
752                  .contains(",p,")) {
753
754               comptemp = ComponentArray.get(com1).AttributeArray
755               .get(0).Value.split(",p,");
756               compevaldet.operator = new String("+");
757             } else if (ComponentArray.get(com1).AttributeArray
758                  .get(0).Value.contains(",s,")) {
759
760               comptemp = ComponentArray.get(com1).AttributeArray
761               .get(0).Value.split(",s,");
762               compevaldet.operator = new String("-");
763             } else if (ComponentArray.get(com1).AttributeArray
764                  .get(0).Value.contains(",d,")) {
765
766               comptemp = ComponentArray.get(com1).AttributeArray
767               .get(0).Value.split(",d,");
768               compevaldet.operator = new String("/");
769             } else if (ComponentArray.get(com1).AttributeArray
770                  .get(0).Value.contains(",m,")) {
771
772               comptemp = ComponentArray.get(com1).AttributeArray
773               .get(0).Value.split(",m,");
774               compevaldet.operator = new String("*");
775             } else if (ComponentArray.get(com1).AttributeArray
776                  .get(0).Value.contains(",g,")) {
777
778               comptemp = ComponentArray.get(com1).AttributeArray
779               .get(0).Value.split(",g,");
780               compevaldet.operator = new String(">");
781             } else if (ComponentArray.get(com1).AttributeArray
782                  .get(0).Value.contains(",ge,")) {
783
784               comptemp = ComponentArray.get(com1).AttributeArray
785               .get(0).Value.split(",ge,");
786               compevaldet.operator = new String(">=");
787             } else if (ComponentArray.get(com1).AttributeArray
788                  .get(0).Value.contains(",l,")) {
789
790               comptemp = ComponentArray.get(com1).AttributeArray
791               .get(0).Value.split(",l,");
792               compevaldet.operator = new String("<");
```

```
793             } else if (ComponentArray.get(com1).AttributeArray
794                 .get(0).Value.contains(",le,")) {
795
796               comptemp = ComponentArray.get(com1).AttributeArray
797               .get(0).Value.split(",le,");
798               compevaldet.operator = new String("<=");
799             }
800
801             else if (ComponentArray.get(com1).AttributeArray
802                 .get(0).Value.contains(",eq,")) {
803
804               comptemp = ComponentArray.get(com1).AttributeArray
805               .get(0).Value.split(",eq,");
806               compevaldet.operator = new String("==");
807
808               System.out.println(ComponentArray.get(com1).
                      AttributeArray
809               .get(0).AttributeName+" "+ComponentArray.get(com1).
                      AttributeArray
810               .get(0).Value);
811
812               System.out.println(compevaldet.operator);
813             }
814
815             //compevaldet.componentname = ComponentArray.get(com1)
                    .ComponentName;
816             compevaldet.evalname = ComponentArray.get(com1).
                    AttributeArray
817             .get(0).AttributeName;
818             compevaldet.operands = comptemp;
819             System.out.println("\n\n\n\n\n" + comptemp[0]);
820             System.out.println("\n\n\n\n\n" + comptemp[1]);
821             compevaldet.DataType = ComponentArray.get(com1).
                    AttributeArray
822             .get(0).DataType;
823
824
825
826             EvalArray.add(compevaldet);
827
828             if (!ComponentArray.get(com1).AttributeArray.get(0).
                    Value
829                 .contains("c,")) {
830
831               if(z!=taskprop.length-1)
832                 p.println("\t\t\t\t\t new "
833                     + ComponentArray.get(com1).AttributeArray.get
                            (0).AttributeName + "Evaluator (),");
834
```

```
835             else {
836               p.println("\t\t\t\t\t\t new "
837                   + ComponentArray.get(com1).AttributeArray.get
                        (0).AttributeName + "Evaluator (),");
838
839               p.println("\t\t\t\t\t\t new Utility()});");
840
841
842             }
843
844             //
845             //p.println("\t\t\t\t\t\t new Utility()});");
846           }
847
848           else if(ComponentArray.get(com1).AttributeArray
849               .get(0).Value.contains("c,")) {
850
851             if(z!=taskprop.length-1){
852
853               if (ComponentArray.get(com1).AttributeArray.get(0)
                      .DataType.equals("String"))
854                 p.println("\t\t\t\t\t\t new ConstProperty(new "+
                        ComponentArray.get(com1).AttributeArray.get
                        (0).DataType + "(\""+ comptemp[1] + "\")),");
855               else
856                 p.println("\t\t\t\t\t\t new ConstProperty(new "+
                        ComponentArray.get(com1).AttributeArray.get
                        (0).DataType + "("+ comptemp[1] + ")),");
857             }
858
859             else{
860
861               if (ComponentArray.get(com1).AttributeArray.get(0)
                      .DataType.equals("String"))
862                 p.println("\t\t\t\t\t\t new ConstProperty(new "+
                        ComponentArray.get(com1).AttributeArray.get
                        (0).DataType + "(\""+ comptemp[1] + "\")),");
863               else
864                 p.println("\t\t\t\t\t\t new ConstProperty(new "+
                        ComponentArray.get(com1).AttributeArray.get
                        (0).DataType + "("+ comptemp[1] + ")),");
865
866               p.println("\t\t\t\t\t\t new Utility()});");
867
868             }
869
870
871
872           }
```

```
873
874             else {
875               p.println("});");
876             }
877
878
879
880           }
881         }
882
883       }
884
885
886
887
888
889
890
891
892
893
894
895
896
897 //
898 //     p.println("\t\t\t\t\t new Object[]{");
899 //     p.println("\t\t\t\t\t\t new "
900 //         + ComponentArray.get(com).AttributeArray
901 //         .get(0).AttributeName + "Evaluator (),");
902 //     p.println("\t\t\t\t\t\t new Utility()});");
903 //
904 //   }
905 //
906 //   else if (ComponentArray.get(compcondind).AttributeArray
907 //       .get(0).Value.contains("c,")) {
908 //     p.println("\t\t\t\t new String[]{\""
909 //         + ComponentArray.get(compcondind).AttributeArray
910 //         .get(0).AttributeName + "\"},");
911 //
912 //     p.println("\t\t\t\t\t new Object[]{");
913 //     if (ComponentArray.get(compcondind).AttributeArray
914 //         .get(0).DataType.equals("String"))
915 //       p.println("\t\t\t\t\t\t new ConstProperty(new "
916 //           + ComponentArray.get(compcondind).AttributeArray
917 //           .get(0).DataType + "(\""
918 //           + comptemp[1] + "\"))});");
919 //     else
920 //       p.println("\t\t\t\t\t\t new ConstProperty(new "
921 //           + ComponentArray.get(compcondind).AttributeArray
```

```
922 //             .get(0).DataType + "("
923 //            + comptemp[1] + "))});");
924 //   }
925 //
926 //   else {
927 //     p.println("\n\t\t\t\t new String[]{},");
928 //     p.println("\n\t\t\t\t\t new Object[]{});");
929 //   }
930 //
931 //}
932 //}
933 //}
934
935 /*
936  * if (ComponentArray.get(index).RefersTo != null) {
937  * p.print("\n\t\t\t\t\t new String[]{\"");  for (int l = 0; l <
938  * taskprop.length; l++) { for (int j = 0; j < ComponentArray.
       size();
939  * j++) {
940  *
941  * if (ComponentArray.get(j).ComponentName .equals(taskprop[l]))
       {
942  *
943  * if (l!=taskprop.length -1)
944  * p.print(ComponentArray.get(j).AttributeArray
945  * .get(0).AttributeName+compcomma);
946  *
947  * else if(l==taskprop.length -1)
948  * p.print(ComponentArray.get(j).AttributeArray
949  * .get(0).AttributeName+"\",IPropertyEvaluator.UTILITY_PROPERTY
       },");
950  *
951  *
952  * index = j; }
953  *
954  * }
955  *
956  * }
957  *
958  * p.println("\n\t\t\t\t\t\t new Object[]{");  // System.out.
       println(j);
959  * p.println("\t\t\t\t\t new ConstProperty(new " +
960  * ComponentArray.get(index).AttributeArray.get(0).DataType + "(
       " +
961  * ComponentArray.get(index).AttributeArray.get(0).Value + "))})
       ;");
962  *
963  * } else {
964  *
```

```
965  * p.println("\n\t\t\t\t\t new String[]{},");
966  * p.println("\n\t\t\t\t\t\t new Object[]{});"); }
967  */
968 // end of global context dependencies
969
970 p.println("\n\t\t\t\t Map resourceMap = myCreateMap(");
971 p.println("\n\t\t\t\t\t new String[]{},");
972 p.println("\n\t\t\t\t\t\t new Object[]{});");
973 p.println("\n\t\t\t\t compPlan.addPlanVariant(propertyMap, null,
        resourceMap, null);");
974 p.println("\t\t\t }");
975 p.println("\t\t }");
976 p.println("\t }");
977 p.println("\t return (CompositionPlan[]) compositionPlans.
      toArray(new CompositionPlan[compositionPlans.size()]);");
978 p.println("}//getCompositionPlans()");
979
980 p.println("\n public IPlan[] getPlans(){");
981 p.println("\t IPlan[] atomicPlans = getAtomicPlans();");
982 p.println("\t IPlan[] compositionPlans = getCompositionPlans();"
      );
983 p.println("\t IPlan[] servicePlans = getServicePlans();");
984 p.println("\n\t IPlan[] plans = new IPlan[atomicPlans.length +
      compositionPlans.length + servicePlans.length];");
985 p.println("\t System.arraycopy(atomicPlans, 0, plans, 0,
      atomicPlans.length);");
986 p.println("\t System.arraycopy(compositionPlans, 0, plans,
      atomicPlans.length, compositionPlans.length);");
987 p.println("\t System.arraycopy(servicePlans, 0, plans,
      atomicPlans.length+compositionPlans.length, servicePlans.
      length);");
988 p.println("\t return plans;");
989 p.println("}");
990
991 p.println("\n public IPlan[] getExtensionPlans(){");
992 p.println("\t return new IPlan[0];");
993 p.println("}");
994
995 // start of utility function
996
997 p.println("\n class Utility extends AbstractPropertyEvaluator{")
      ;
998
999 long number2 = (long) Math.floor(Math.random() * 9000000000000L)
       + 1000000000000L;
1000 p.println("\t private static final long serialVersionUID = " +
      number2
1001      + "L;");
1002
```

```
1003 p.println("\n\t public Object evaluate(IContextValueAccess
         context, IPropertyEvaluatorContext evalContext)");
1004 p.println("\t {");
1005 p.println("\n\t\t double utility = 0.0;");
1006 p.println("\n\t\t double temputility = 0.0;");
1007 p.println("\n\t\t int count = 0;");
1008 p.println("\n\t\t //contextReferences");
1009
1010 for (int i = 0; i < ComponentArray.size(); i++) {
1011   if (ComponentArray.get(i).ComponentType.equals("Task")) {
1012
1013
1014
1015     String datatype = null;
1016     int search = 0;
1017
1018     String[] tasdep = ComponentArray.get(index).RefersTo.split("
           ,");
1019
1020
1021     for(int t =0;t<tasdep.length;t++){
1022
1023       for (int a = 0; a < ComponentArray.size(); a++) {
1024         if (ComponentArray.get(a).ComponentName
1025              .equals(tasdep[t])){
1026           search = a;
1027           break;
1028         }
1029       }
1030
1031       if (ComponentArray.get(search).AttributeArray.get(0).
             DataType
1032              .equals("Boolean"))
1033         datatype = "Bool";
1034       else
1035         datatype = ComponentArray.get(search).AttributeArray.get
               (0).DataType;
1036
1037       String defaultval = null;
1038       if (datatype.equals("Bool"))
1039         defaultval = new String("false");
1040       else if(datatype.equals("Int"))
1041         defaultval = new String("0");
1042       else if(datatype.equals("Double"))
1043         defaultval = new String("0.0");
1044
1045
1046
1047
```

```
1048
1049        p.println("\n\t\t "
1050            + ComponentArray.get(search).AttributeArray.get(0).
                    DataType
1051            .toLowerCase()
1052            + " "
1053            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1054            + "=  context.get"
1055            + datatype
1056            + "Value(\"http://www.ist-music.eu/Ontology_v0_1.xml#
                    Thing.Concept.Entity."
1057            + ComponentArray.get(i).RefersTo
1058            + ";http://www.ist-music.eu/Ontology_v0_1.xml#Thing.
                    Concept.Scope."
1059            + ComponentArray.get(i).RefersTo + "\", "+defaultval+"
                    );");
1060        p.println("\n\t\t //rolePropertyReferences");
1061        p.println("\n\t\t "
1062            + ComponentArray.get(search).AttributeArray.get(0).
                    DataType
1063            .toLowerCase()
1064            + " "
1065            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1066            + "Provided = (("
1067            + ComponentArray.get(search).AttributeArray.get(0).
                    DataType
1068            + ") evalContext.evaluate(\""
1069            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1070            + "Provided\", context))."
1071            + ComponentArray.get(search).AttributeArray.get(0).
                    DataType
1072            .toLowerCase() + "Value();");
1073        p.println("\n\t\t //ownPropertyReferences");
1074
1075        int opind = 0;
1076
1077        String abc = null;
1078
1079        for (int q = 0; q < EvalArray.size(); q++) {
1080          abc = EvalArray.get(q).evalname;
1081          if (abc.equals(ComponentArray
1082                .get(search).AttributeArray.get(0).AttributeName)){
1083            opind = q;
1084            break;
1085          }
1086        }
```

```
1087
1088        System.out.println(ComponentArray
1089                .get(search).AttributeArray.get(0).AttributeName+"1"
                        );
1090
1091        System.out.println(EvalArray.get(opind).evalname+"2");
1092        System.out.println("\n\n\n\n"+EvalArray.get(opind).
                operator+"3");
1093
1094        System.out.println("\n\n\n\n");
1095
1096        p.println("\n\t\t if ("
1097            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1098            + " "
1099            + EvalArray.get(opind).operator
1100            + " "
1101            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1102            + "Provided) utility = 1.0;");
1103        p.println("\n\t\t else utility = 0.0;");
1104      }
1105    }
1106
1107 }
1108
1109 // other context dependencies
1110
1111 for (int i = 0; i < ComponentArray.size(); i++) {
1112   if (ComponentArray.get(i).ComponentType.equals("
          ConditionalStep")) {
1113
1114     if (ComponentArray.get(i).RefersTo != null) {
1115       String datatype = null;
1116       int search = 0;
1117       for (int a = 0; a < ComponentArray.size(); a++) {
1118         if (ComponentArray.get(a).ComponentName
1119               .equals(ComponentArray.get(i).RefersTo)){
1120           search = a;
1121           break;
1122         }
1123       }
1124
1125       if (ComponentArray.get(search).AttributeArray.get(0).
              DataType
1126             .equals("Boolean"))
1127         datatype = "Bool";
1128       else
1129         datatype = ComponentArray.get(search).AttributeArray
```

```
1130            .get(0).DataType;
1131
1132        String def = null;
1133        if (datatype.equals("Bool"))
1134          def = new String("false");
1135        else if(datatype.equals("Int"))
1136          def = new String("0");
1137        else if(datatype.equals("Double"))
1138          def = new String("0.0");
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149        p.println("\n\t\t "
1150            + ComponentArray.get(search).AttributeArray.get(0).
                  DataType
1151            .toLowerCase()
1152            + " "
1153            + ComponentArray.get(search).AttributeArray.get(0).
                  AttributeName
1154            + "=  context.get"
1155            + datatype
1156            + "Value(\"http://www.ist-music.eu/Ontology_v0_1.xml#
                  Thing.Concept.Entity."
1157            + ComponentArray.get(i).RefersTo
1158            + ";http://www.ist-music.eu/Ontology_v0_1.xml#Thing.
                  Concept.Scope."
1159            + ComponentArray.get(i).RefersTo + "\", "+def+");");
1160
1161        p.println("\n\t\t //rolePropertyReferences");
1162
1163        p.println("\n\t\t "
1164            + ComponentArray.get(search).AttributeArray.get(0).
                  DataType
1165            .toLowerCase()
1166            + " "
1167            + ComponentArray.get(search).AttributeArray.get(0).
                  AttributeName
1168            + "Provided = (("
1169            + ComponentArray.get(search).AttributeArray.get(0).
                  DataType
1170            + ") evalContext.evaluate(\""
```

```
1171            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1172            + "Provided\", context))."
1173            + ComponentArray.get(search).AttributeArray.get(0).
                    DataType
1174            .toLowerCase() + "Value();");
1175        p.println("\n\t\t //ownPropertyReferences");
1176
1177        int opind = 0;
1178
1179        for (int q = 0; q < EvalArray.size(); q++) {
1180          if (EvalArray.get(q).evalname.equals(ComponentArray
1181              .get(search).ComponentName))
1182            opind = q;
1183        }
1184
1185        p.println("\n\t\t if ("
1186            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1187            + " "
1188            + EvalArray.get(opind).operator
1189            + " "
1190            + ComponentArray.get(search).AttributeArray.get(0).
                    AttributeName
1191            + "Provided) temputility = 1.0;");
1192        p.println("\n\t\t else temputility = 0.0;");
1193        p.println("\n\t\t utility = utility + temputility;");
1194        p.println("\n\t\t count++;");
1195
1196      }
1197    }
1198 }
1199
1200 p.println("\n\t\t utility = utility/count;");
1201
1202 p.println("\n\t\t return new Double(utility);");
1203
1204 p.println("\n\t }");
1205 p.println("\n }");
1206
1207 // end of utility function
1208
1209 // Property Evaluator Class
1210
1211 for (int m = 0; m < EvalArray.size(); m++) {
1212   if (!EvalArray.get(m).operands[0].equals("c")) {
1213     p.println("\n class " + EvalArray.get(m).evalname
1214         + "Evaluator extends AbstractPropertyEvaluator{");
1215     long newnumber = (long) Math
```

```
1216        .floor(Math.random() * 9000000000000L) + 1000000000000L;
1217        p.println("\t private static final long serialVersionUID = "
1218            + newnumber + "L;");
1219
1220        p.println("\n\t public Object evaluate(IContextValueAccess
                context, IPropertyEvaluatorContext evalContext){");
1221
1222        String[] newtemp = new String[2];
1223
1224        newtemp[0] = EvalArray.get(m).operands[0].substring(0,
1225            EvalArray.get(m).operands[0].indexOf("."));
1226        newtemp[1] = EvalArray.get(m).operands[0].substring(
1227            EvalArray.get(m).operands[0].indexOf(".") + 1,
1228            EvalArray.get(m).operands[0].length());
1229        System.out.println(newtemp[0]);
1230        p.println("\n\t\t " + EvalArray.get(m).DataType + " "
1231            + newtemp[0] + "_Type_" + EvalArray.get(m).evalname
1232            + " = ((" + EvalArray.get(m).DataType
1233            + ") evalContext.evaluateForRole(\"" + newtemp[1]
1234            + "\", context, \"" + newtemp[0] + "_Type_Role\")).")
1235            + EvalArray.get(m).DataType.toLowerCase() + "Value ();")
                ;
1236        p.println("\n\t\t " + EvalArray.get(m).DataType + " "
1237            + EvalArray.get(m).evalname + " = " + newtemp[0]
1238            + "_Type_" + EvalArray.get(m).evalname + " "
1239            + EvalArray.get(m).operator + " "
1240            + EvalArray.get(m).operands[1] + ";");
1241        p.println("\n\t\t return new " + EvalArray.get(m).DataType
1242            + "(" + EvalArray.get(m).evalname + ");");
1243        p.println("\n\t }");
1244        p.println("\n }");
1245
1246    }
1247 }
1248
1249 p.println("\n private HashMap myCreateMap(String[] keys, Object
        [] values){");
1250 p.println("\t HashMap propMap = new HashMap();");
1251 p.println("\t for(int i = 0; i < keys.length; i++){");
1252 p.println("\t\t propMap.put(keys[i], values[i]);");
1253 p.println("\t }");
1254 p.println("\t return propMap;");
1255 p.println(" }");
1256 p.println("}");
1257
1258 // end of main file
1259 p.close();
1260
1261 // generate other files
```

```
1262
1263 for (int i = 0; i < ComponentArray.size(); i++) {
1264   if (ComponentArray.get(i).ComponentType.equals("
         ConditionalStep")
1265       || ComponentArray.get(i).ComponentType.equals("Step")
1266       || ComponentArray.get(i).ComponentType.equals("Query")) {
1267
1268     FileOutputStream output = new FileOutputStream(
1269         ComponentArray.get(i).ComponentName + ".java");
1270
1271     PrintStream pout = new PrintStream(output);
1272
1273     pout.println("/*");
1274     pout.println("* @author K M Imtiaz-Ud-Din");
1275
1276     DateFormat newdateFormat = new SimpleDateFormat("dd-MMM-yyyy
         ");
1277     Date newdate = new Date();
1278     pout.println("* Created on " + newdateFormat.format(newdate)
         );
1279     pout.println("*");
1280     pout.println("*/");
1281
1282     if (CompositionName != null)
1283       pout.println("\n\npackage org.istmusic.UbiCompServiceComp.
           "
1284           + CompositionName.toLowerCase() + ";");
1285
1286
1287
1288
1289
1290
1291
1292     pout.println("\nimport org.istmusic.mw.adaptation.
         configuration.ConfigurableImpl;");
1293     pout.println("\nimport org.UbiMusic.Services."+
         ComponentArray.get(i).ComponentName+"Service;");
1294     pout.println("\npublic class "
1295         + ComponentArray.get(i).ComponentName
1296         + " extends ConfigurableImpl {");
1297     pout.println("\t public void startActivity() {");
1298     pout.println("\t\t " + ComponentArray.get(i).ComponentName+"
         Service "
1299         + "service = new "
1300         + ComponentArray.get(i).ComponentName + "Service();");
1301     for (int s = 0; s < ComponentArray.get(i).AttributeArray.
         size(); s++)
1302       pout.println("\t\t service.set"
```

```
1303            + ComponentArray.get(i).AttributeArray.get(s).
                    AttributeName
1304            + "(\""
1305            + ComponentArray.get(i).AttributeArray.get(s).Value
1306            + "\");");
1307
1308      pout.println("\t\t service.start();");
1309      pout.println("\t }");
1310      pout.println("}");
1311
1312    }
1313 }
1314
1315 // generate the composite file
1316
1317 FileOutputStream compoutput = new FileOutputStream("Composite"
1318      + CompositionName + ".java");
1319
1320 PrintStream cpout = new PrintStream(compoutput);
1321
1322 cpout.println();
1323 cpout.println("/*");
1324
1325 cpout.println("* @author K M Imtiaz-Ud-Din");
1326
1327 DateFormat ndateFormat = new SimpleDateFormat("dd-MMM-yyyy");
1328 Date ndate = new Date();
1329 cpout.println("* Created on " + ndateFormat.format(ndate));
1330 cpout.println("*");
1331 cpout.println("*/");
1332
1333 if (CompositionName != null)
1334   cpout.println("\n\npackage org.istmusic.UbiCompServiceComp."
1335       + CompositionName.toLowerCase() + ";");
1336
1337 cpout.println("\nimport org.istmusic.mw.adaptation.configuration
       .ConfigurableImpl;");
1338 cpout.println("\npublic class " + "Composite" + CompositionName
1339      + " extends ConfigurableImpl {");
1340 cpout.println("\n\t public " + "Composite" + CompositionName + "
       (){");
1341 cpout.println("\n\t\t super();");
1342 cpout.println("\n\t\t System.out.println(\"Composition created
       ...\");");
1343 cpout.println("\n\t }");
1344 cpout.println("\n }");
1345
1346 }
1347
```

```
1348 }
```

Listing B.3: Parser.java

B.4 EvaluatorDetail.java

```
1
2
3 public class EvaluatorDetail {
4   //String componentname;
5   String evalname;
6   String operands[];
7   String operator;
8   String DataType;
9
10 }
```

Listing B.4: EvaluatorDetail.java

B.5 Component.java

```
1
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Component {
7
8   String ComponentType;
9   String ComponentName;
10  String RefersTo;
11  List<Attribute> AttributeArray = new ArrayList<Attribute>();
12 }
```

Listing B.5: Component.java

B.6 Attribute.java

```
1
2
3 public class Attribute {
4
5   String AttributeName;
6   String DataType;
7   String Value;
8
9 }
```

Listing B.6: Attribute.java

B.7 SendEmailService.java

```
1  package org.UbiMusic.Services;
2
3  import java.awt.Component;
4  import java.awt.Frame;
5  import java.awt.TextArea;
6  import java.awt.TextComponent;
7
8  public class SendEmailService{
9
10   private static String from, to, title, msg;
11
12   public void setfrom(String from){
13
14     this.from = from;
15   }
16
17   public void setto(String to){
18
19     this.to = to;
20   }
21
22   public void settitle(String title){
23
24     this.title = title;
25
26   }
27
28   public void setmsg(String msg){
29
30     this.msg = msg;
31   }
```

```
32
33    public void start(){
34      Frame msgBox = new Frame();
35      msgBox.setTitle("Email Sender");
36      msgBox.setSize(500, 100);
37      msgBox.setLocation(100, 100);
38
39      TextArea textArea = new TextArea("default text for Email
            sender");
40
41      String text = "A message has been sent from "+from+" to "+
            to+ "\nwith title: "+ title+ " \nand having message: "+
            msg;
42      textArea.setText(text);
43
44
45      msgBox.add(textArea);
46      msgBox.setVisible(true);
47
48      System.out.println();
49      System.out.println("=======Email sender =========");
50      System.out.println ("A message has been sent from "+from+"
            to "+ to+ "\nwith title: "+ title+ " \nand having message
            : "+msg);
51      System.out.println();
52    }
53
54  }
```

Listing B.7: SendEmailService.java

B.8 SendSMSService.java

```
1  package org.UbiMusic.Services;
2
3  import java.awt.Frame;
4  import java.awt.TextArea;
5
6  public class SendSMSService{
7
8    private static String to, message;
9
10   public void setto(String to){
11
12     this.to = to;
13   }
```

```
14
15
16    public void setmessage(String message){
17
18      this.message = message;
19    }
20
21    public void start(){
22      Frame msgBox = new Frame();
23      msgBox.setTitle("SMS Sender");
24      msgBox.setSize(500, 100);
25      msgBox.setLocation(650, 100);
26
27      TextArea textArea = new TextArea("default text for SMS
            Sender");
28
29      String text = "A text message has been sent to "+ to+ "\
            nwith message "+message;
30      textArea.setText(text);
31
32      msgBox.add(textArea);
33
34      msgBox.setVisible(true);
35
36      System.out.println();
37      System.out.println("=======SMS sender =========");
38      System.out.println ("A text message has been sent to "+ to+
            "\nwith message "+message);
39      System.out.println();
40    }
41
42 }
```

Listing B.8: SendSMSService.java

Bibliography

[1] F. Mattern and P. Sturm. From Distributed Systems to Ubiquitous Computing The State of the Art, Trends, and Prospects of Future Networked Systems. In K. Irmscher and K.-P. Fahnrich, editors, Proc. KIVS 2003, pages 325, Springer-Verlag, Feb. 2003.

[2] Mark Weiser (1993), Hot Topic: Ubiquitous Computing. IEEE Computer, 26(10): 71-72

[3] UbiCompForAll - Ubiquitous service composition for all users / The project, http://www.sintef.no/Projectweb/UbiCompForAll/The-project/ (Accessed: 28 June 2011)

[4] The Project, http://ist-music.berlios.de/site/index.html#project (Accessed: 28 June 2011)

[5] Approach,UbiCompForAll - Ubiquitous service composition for all users, http://www.sintef.no/Projectweb/UbiCompForAll/The-project/Approach/ (Accessed: 27 June 2011).

[6] S. Ponnekanti et al., ICrafter: A Service Framework for Ubiquitous Computing Environments. Proc. 3rd Conf. Ubiquitous Computing (UbiComp), LNCS 2201, Springer, 2001, pp. 56-75.

[7] M. Romn, B. Ziebart, and R.H. Campbell, Dynamic Application Composition: Customizing the Behavior of an Active Space. Proc. 1st IEEE Intl Conf. Pervasive Computing and Communications(PerCom 03), IEEE CS Press, 2003, pp.169-176

[8] D. Svensson, G. Hedin, and B. Magnusson, Pervasive Applications through Scripted Assemblies of Services. Proc. IEEE Intl Conf. Pervasive Services, IEEE Press, 2007, pp. 301-307

[9] R. Moats, URN Syntax. RFC 2141 (Proposed Standard), May 1997.

[10] W.K. Edwards, M. W. Newman, J. Z. Sedivy, and T. F. Smith, Bringing network effects to pervasive spaces. IEEE Pervasive Computing 4, 3 (2005), 15-17.

[11] R. Grimm, One. world: Experiences with a Pervasive Computing Architecture. IEEE Pervasive Computing 3, 3 (2004), 22-30.

[12] R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall, System support for pervasive applications. ACM Trans. Comput, Syst. 22, 4 (2004), 421-486.

[13] W. K. Edwards, M. W. Newman, J. Sedivy, and S. Izadi, Challenge: recombinant computing and the speakeasy approach. In Proceedings of ACM MobiCom (New York, USA, 2002), ACM Press, pp. 279-286.

[14] K. Edwards, et al., Using speakeasy for ad hoc peer-to-peer collaboration. In Proceedings of ACM CSCW '02(2002), ACM Press, pp. 256-265.

[15] M. Newman, et al., Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In Proceedings of DIS '02 (New York, NY,USA, 2002), ACM Press, pp. 147-156.

[16] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, Project Aura: toward distraction-free pervasive computing. Pervasive Computing, IEEE 1, 2(2002), 22-31.

[17] Y. Yang, F. Mahon, M. H. Williams, and T. Pfeifer, Context-aware dynamic personalised service re-composition in a pervasive service environment. In Proceedings of Ubiquitous Intelligence and Computing(2006), vol. 4159 of LNCS, Springer, pp. 724-735.

[18] M. Roman, et al., Dynamic application composition: Customizing the behavior of an active space. In Proceedings of IEEE PERCOM (Washington, DC, USA, 2003), IEEE Computer Society, p. 169.

[19] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, Service composition for mobile environment. Mobile Networks and Applications 4, 10 (August 2005), 435-451.

[20] S. Ben Mokhtar, J. Liu, N. Georgantas, and V. Issarny. Qosaware dynamic service composition in ambient intelligence environments. In Proceedings of the 20th IEEE/ACM Int. Conf. on Automated software engineering (ASE 05), pages 317320, 2005.

[21] M. Vallee, F. Ramparany, and L. Vercouter, Flexible Composition of Smart Device Services. In The 2005 International Conference on Pervasive Systems and Computing (PSC-05), Las Vegas, Nevada, USA., June(2005), pp. 27-30.

[22] S. Maffioletti, M. Kouadri, and B. Hirsbrunner, Automatic resource and service management for ubiquitous computing environments. Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on (2004), 219-223.

[23] E. Kiciman, and A. Fox, Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In Proceedings of HUC2000 (2000), no. 1927 in LNCS, pp. 211-226.

[24] ISO/IEC. Software engineering—product quality, part 1-4, 2001. ISO-9126-1,-2,,3,-4, page-11.

[25] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practise. 2nd ed. Addison-Wesley, 2003, page-80.

[26] F. Casati, S. Ilnicki, Li-Jie Jin, V. Krishnamoorthy, and Ming-Chien Shan, "eFlow: a platform for developing and managing composite e-services", Academia/Industry Working Conference on Research Challenges (AIWORC'00), pp.341-348, 2000, doi: http://doi.ieeecomputersociety.org/10.1109/AIWORC.2000.843314.

[27] Q. Z. Sheng, B. Benatallah, et al, Enabling Personalized Composition and Adaptive Provisioning of Web Services. The 16th International Conference on Advanced Information Systems Engineering (CAiSE), Riga, Latvia, June 7-11. (2004)

[28] IBM. Tivoli Personalized Services Manager, Version 1.2. ftp://ftp.software.ibm.com/software/pervasive/info/tech/tpsm_ ss.pdf, 2002.

[29] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein O. Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz, Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In Software Engineering for Self-Adaptive Systems, pages 164-182, 2009.

[30] K. E. Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela, The Internet Operating System: Middleware for adaptive distributed computing. International Journal of High Performance Computing

Applications (IJHPCA), Special Issue on Scheduling Techniques for Large-Scale Distributed Platforms, 20(4):467480, 2006.

[31] M. Mochizuki, H. Tokuda, "Possession System: adaptation support middleware for collaborative multimedia applications in Java," Multimedia Computing and Systems, 1999. IEEE International Conference on , vol.1, no., pp.339-344 vol.1, Jul 1999.

[32] Mohammad Ullah Khan, "Unanticipated Dynamic Adaptation of Mobile Applications",Ph.D. dissertation, University of Kassel, Electrical Engineering and Computer Science, Kassel Germany, March 2010, http://www.uni-kassel.de/upress/online/frei/978-3-89958-918-4.volltext.frei.pdf

[33] Hewlett Packard Italiana,System design of the MUSIC architecture,Deliverable reference: D4.3, 02 October 2009, http://svn.berlios.de/svnroot/repos/ist-music/music-documentation/trunk/deliverables/D04.3

[34] Developing the Context Sensor Bundle, https://docs.google.com/View?id=dhq9qbc3_38gcrc37fg (Accessed: 27 June 2011).

[35] Dipanjan Chakrabarty and Anupam Joshi (2001), Dynamic Service Composition: State-of-the-Art and Research Directions. University of Maryland, Baltimore County: Baltimore(USA). TR-CS-01-19

[36] Romain Rouvoy, Roman Vitenberg, Frank Eliassen, Enhancing Planning-Based Adaptation Middleware with Support for Dependability: a Case Study,1st International Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS08). Electronic Communications of EASST (ECEASST), vol. 11. Oslo, Norway. June 3, 2008.

[37] Mourad Alia, Geir Horn, Frank Eliassen, Mohammad Ullah Khan, Rolf Fricke, and Roland Reichle, A Component-based Planning Framework for Adaptive Systems. The 8th International Symposium on Distributed Objects and Applications (DOA), Oct 30 Nov 1, 2006, Montpellier, France.

[38] H. Lieberman, F. Paterno, and V. Wulf. End-User Development. Springer, October 200