Middleware for ad hoc user task composition in heterogeneous environments considering user preferences

Hamid Mukhtar

Thèse de doctorat de l'INSTITUT NATIONAL DES TELECOMMUNICATIONS dans le cadre de l'école doctorale S&I en co-accréditation avec l'UNIVERSITE D'EVRY-VAL D'ESSONNE

Spécialité : Informatique

Par :

HAMID MUKHTAR

Thèse présentée pour l'obtention du grade de Docteur de l'INSTITUT NATIONAL DES TELECOMMUNICATIONS

Middleware for Ad hoc User Task Composition in Heterogeneous Environments Considering User Preferences

Soutenue le 16 novembre 2009 devant le jury composé de :

Rapporteurs :

M. Noël De Palma	Maître de conférences, HDR, INP/ENSIMAG
M. Lionel Seinturier	Professeur, Université des Sciences et Technologies de Lille 1

Examinateurs :

Mme Françoise André	Professeur, Université de Rennes 1
M. Gordon Blair	Professeur, Lancaster University
M. Djamel Belaïd	Maître de conférences, Telecom SudParis (Encadrant)
M. Guy Bernard	Professeur, Telecom SudParis (Directeur de thèse)

Thèse numéro : 2009TELE0015

Dedicated to my wife and parents who supported me throughout my doctoral studies.

Acknowledgements

At the outset, I would like to express my sincere gratitude to my thesis advisors. Most beneficial to my doctoral research was the vision, direction, and significant feedback from my advisor, Professor Guy Bernard; and the guidance and committed concentration towards technical quality from my co-advisor, Dr. Djamel Belaïd.

I also thank the honorable members of the jury — Pr. Françoise André, Pr. Gordon Blair, Dr. Noël De Palma, and Pr. Lionel Seinturier — for their attention and thoughtful comments.

I owe gratitude to the members of the computer science department of Telecom Sud-Paris, notably Denis Conan and Samir Tata, for their sincere help and guidance at various occasions during my thesis.

I want to thank all my doctoral fellows whose friendly company, during my thesis work, was a source of great pleasure. I would like to thank administrative and technical staff members of Telecom SudParis who have been kind enough to help in their respective roles.

Last but certainly not the least, I am proud to acknowledge the generous and enduring support of my wife who supported me through all the tough times, and prayers of my parents throughout the years of efforts toward this dissertation.

Résumé

En raison du grand succès des réseaux sans fil et des appareils portatifs, le paradigme de l'informatique pervasive est devenu une réalité. L'un des plus difficiles objectifs à atteindre dans de tels environnements est de permettre à l'utilisateur d'exécuter une tâche en composant à la volée, les services et les ressources de l'environnement.

Cela implique la correspondance et la sélection automatique de services à travers divers dispositifs de l'environnement pervasif. Les approches existantes considèrent souvent seulement les aspects fonctionnels des services et ne prennent pas en compte différents aspects non-fonctionnels tels que les préférences utilisateur, les capacités des dispositifs en termes matériels et logiciels, et l'hétérogénéité du réseau de ces dispositifs.

Nous présentons une approche pour la sélection dynamique des composants et des dispositifs dans un environnement pervasif en considérant simultanément tous les aspects précédemment mentionnés. Premièrement, nous proposons une modélisation abstraite et concrète de l'application, des capacités des terminaux et des ressources, des préférences des utilisateurs, ainsi que la modélisation de la plate-forme réseau sous-jacente. Les capacités des dispositifs sont représentées par notre extension du modèle CC/PP et les préférences des utilisateurs en utilisant notre extension du modèle CP-Net. Nous modélisons sous forme d'un graphe la tâche de l'utilisateur et des services réseau sous-jacent, ainsi que les exigences des services, des préférences utilisateur et les capacités des dispositifs. L'hétérogénéité des protocoles de communication est également considérée dans les graphes. Les aspects algorithmiques ont été traités en fournissant des algorithmes pour la correspondance entre les services et les composants, pour la projection des applications sur la plate-forme de composants existants et pour l'évaluation des préférences utilisateurs.

Pour la description de la composition de l'application nous proposons un modèle SCA étendu. Partant d'une composition abstraite de services, nous arrivons à réaliser une composition concrète de l'application distribuée à travers les dispositifs existants. Si pendant l'exécution un nouveau meilleur dispositif apparaît, l'application est recomposée en tenant compte des nouveaux composants. Cela permet de réaliser la continuité de la session d'un dispositif vers un autre. Une mise en oeuvre d'un prototype et son évaluation sont également fournis.

Abstract

Due to the large success of wireless networks and portable devices, the pervasive computing paradigm is becoming a reality. One of the most challenging objectives to be achieved in pervasive computing environments is to allow a user to perform a task by composing on the fly the environment's service and resource components.

This involves automatic matching and selection of services across various devices in the pervasive environment. Existing approaches mostly consider only functional aspects for service and component matching and do not consider various non-functional aspects such as user preferences, device capabilities in terms of software and hardware, and network heterogeneity of devices.

We present an approach for dynamic selection of components and devices in a pervasive environments considering all the aforementioned aspects simultaneously. First, we provide a modeling of abstract and concrete application, device capabilities and resources, user preferences as well as modeling of the underlying connected platform. Device capabilities are represented by our extended CC/PP model and user preferences using our extended CP-net model. We model both the user task and the underlying network services, along with service requirements, user preferences and device capabilities, as graphs. The heterogeneity of communication protocols is also considered in the graph. The algorithmic aspects have been treated by providing algorithms for service and component matching, application mapping on network platform and user preference evaluation.

For description of application composition extended SCA model is used. Departing from an abstract composition, we arrive on achieving a concrete application composition which may be distributed across more than one device. If during the application execution a new, better device appears, the application is recomposed to replace the existing components by the newer ones. This also implies the continuity of session from one device to another. A prototype implementation and its evaluation are also provided.

Keywords: Ad hoc user task, user preferences, device capabilities, service requirements, middleware.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a typical computing environment of modern days, a user may have access to several computational devices at the same time. These include small handheld devices such as PDA's and smart-phones, televisions, and audio players as well as multimedia-rich laptop and desktop computers. These devices offer their capabilities to the users in the form of services. These devices are able to communicate with each other on the basis of peer-to-peer protocols and a device may offer services that can also be utilised by other devices in the pervasive environment. As the technology is advancing, manufacturers are pushing hard to create such devices which offer the user a plethora of services ranging from voice and text communication, photography and image manipulation, audio and video streaming, to high bit-rate Internet access.

We can identify several main factors which have given rise to the concept of a digital multimedia home network in the last few years: 1) *increased device capabilities*: a small hand-held device of modern day is no less powerful than a large desktop found in homes a decade ago, in terms of its functionalities and usefulness for an end-user. 2) *Increased number of devices*: the number of such devices has also increased sharply and a typical household has several such devices. Due to the fact that devices have become *personal*, such number has jumped from per family or per house to per person. 3) *Increased demand for multimedia*: as the number of devices around a user is increased, users are enjoying more and more multimedia due to easy access and user-friendliness of the devices. 4) *Increased connectivity*: recently, device communication protocols like Bluetooth have been developed for connecting heterogeneous devices in home networks with minimum user intervention. Moreover, Wi-Fi is becoming the access mechanism for Internet connectivity in home networks instead of traditional wired cables. 5) *Increased mobility*: due to small sizes of the devices and by using wireless connectivity, a user is able to move around in the home network, along with the device, without being disconnected from other devices in the home network.

All these factors have created new opportunities for making the applications more intelligent and supportive to the user. Intelligent applications are able to adapt themselves to the context of the user and the environment whenever the user move from one location to another in a mobile computing environment.

1.1 Motivation

As the computing needs of the individual users have evolved and as the technology is advancing at a rapid pace, we have witnessed the accomplishment of Mark Weiser's vision (Weiser, 1999) that he envisioned in early 90's, in the form of *pervasive computing environment*. In such an environment, information processing is thoroughly integrated into everyday objects and activities. A user in a pervasive computing environment engages many computational devices and systems simultaneously, and may not necessarily even be aware that they are doing so. They devices and systems may be available in the environment as part of the underlying computing infrastructure or provided by other users in the environment.

As a direct result of this progress, we will also witness service- and resource- rich environments called *smart spaces*. Such environments are equipped with plethora of services and resources provided by the numerous computing devices. Most of these services and resources can also be accessed by the user through direct intervention or indirectly by the user applications. Due to the difficulty of selecting the best service and the increased usage of these services, the interaction models between users and their surroundings can no longer be based on a single application or a service. Therefore, pervasive applications can be composed or assembled from *software components* which provide these services and which reside physically across multiple nodes or devices in the pervasive environment, in which the user is present at a given time.

Applications which are composed of such components are called composite applications and they provide a number of advantages which are not available for monolithic applications. For example, composite applications are able to aggregate functionality across several resource-limited devices, thus, sharing their resources. Further enhancements on these applications come in the form of task-based computing (Sousa and Garlan, 2002)(Ben Mokhtar et al., 2007), which enables users to explicitly define their tasks which are then realized using networking components and resources. Example user tasks are *writing an email*, *chatting with friends*, *watching movies over the Internet*, etc. These tasks may be requested anytime, anywhere and then constructed at runtime as required by the situation in hand. These tasks integrate the functionalities offered by the environment's components and, thus, their instantiation is dependent on the environment in which they are carried out. In other words, we say that such *user tasks* are composed *dynamically* in terms of available network components and resources.

However, implementing a solution for the dynamic composition of user tasks is not an easy task. The users are mobile and therefore the binding of the user task to the available network components and resources, such as wall projectors, cameras, and interactive displays, has to be realized at run-time. But the availability of such components cannot be determined in advance as the participation of users and their devices in such environments is dynamic and the availability of resources required by such components cannot be known in advance. Furthermore, because the situation and the user needs can change, it might be appropriate to choose different sets of components and resources even for the realization of the same application, when used in a different user context. These challenges require automated mechanisms that take care of application adaptation and lifecycle management.

Due to heterogeneity of the resources, the mechanisms need to satisfy the functional requirements in an effective way but also to consider non-functional requirements when modelling the application behaviour and the limitations of the available resources. Furthermore, the planning of application composition has to be abstracted: it should not be tailored to certain types of applications with specific properties.

As a network becomes saturated with devices, the number of components available on the network also grows rapidly. In such a case, it is possible to compose various applications through a combination of different sets of components. The optimal allocation of resources needs to be determined for cases in which there are multiple ways to compose the same application across networking devices. Considering the multi-faceted problem of having varying device capabilities, supporting a different set of protocols and each device hosting a number of components providing the same functionality, it becomes practically impossible to choose one particular component on a specific device without considering the mentioned factors. For example, a chat application can be mapped to a text chat, a voice chat or video chat component, because they all provide similar functionality: to chat.

However, selection of one component or another should not be arbitrary. One particular driving force for component selection can be user preferences. For one user, it would be more convenient to chat using video method, yet for another user it would of more pleasure to send/receive text chat message. Another important factor for component selection is the capabilities of devices. If a device does not have the capability to display video, the device cannot be used for video chatting and, hence, should be avoided for component selection. The third important factor to consider, when selecting components on different devices, is the communication capabilities of each device. The components required by a user task can be selected on different devices only if they can communicate with each other, otherwise, if the devices do not share common communication protocol, the task cannot be coordinated.

Thus, we have identified several challenges that need to be addressed:

1. **Application design** the application should be designed so that if the required resources are distributed across various devices in the environment, so should be the functional aspects of the application.
2. **Environment's heterogeneity** the user application should be executable on different hardware platforms, ranging from desktop, multimedia systems with abundant resources to mobile handheld devices with limited resources. Moreover, the heterogeneity of the underlying network, in terms of access protocols, should also be considered.
3. **Resource requirement** the resource requirements of the user application should be clearly and unambiguously specified to make sure that the application can be deployed within the available resources of a particular device whenever it moves.
4. **Resource usage** the supporting infrastructure should be able to enforce appropriate admission control and resource usage policies.
5. **User preferences** the supporting infrastructure should be aware of user's preferences so that optimum allocation of resources, according to user preferences, can be carried out.

6. **Automatic selection** the selection of components and required resources should be done automatically, with minimum involvement from the user side.

1.2 Contribution

To address the above challenges, this thesis introduces MATCH-UP: Middleware for *Ad hoc* user Task Composition in Heterogeneous environments considering User Preferences. Although there have been a significant number of recent approaches, which we will outline in next chapter, we position ourselves in the research community, by contributing in those axes that have either been treated partially or not at all. In this regard, the most significant contributions of the proposed middleware are as follows:

1.2.1 User Preferences Consideration

Very few of the existing approaches consider user preferences for *ad hoc* user task composition; some of them will be discussed in the related work. As part of this thesis, we consider user preferences mainly for device capabilities, including various hardware, software, and network characteristics of a device. Given a user task and given various user preferences, our goal is to select the components, required by the user task, on the best device for the user, i.e., a device which satisfies most of user preferences. However, it might happen that the selected device will not have all the required components. In that case, we need to look for other devices to find the required components. Thus, the user task may be instantiated with its components distributed across several devices. Depending upon the number and type of devices, the user task may be realized differently in different environments.

Assuming that when using the same task for the next time in the environment, with the same set of devices, the user changes some of his preferences. This might lead to selection of a different device or set of devices. Thus, user preferences play an important role in our user task composition process.

1.2.2 Service Requirements for Capabilities

Services in the user task may also describe their requirements for device capabilities. These requirements may be specified by the service architect or the user task developer and they tailor the selection of required components on the devices. One may argue that this is an unnecessary constraint [1], we claim that by introducing service requirements, we not only reduce the solution space largely by decreasing the number of candidate components, but it also serves in solving other issues, related to security and authentication, etc. in pervasive computing environments.

First, they can be used for security reasons: for example, when the runtime instantiates the user task, it will restrict the use of device resources to only those which are required by the services in the task. This will ensure that when an un-trusted component is selected for task realisation, it will not be able to access the resources which are not required by the services in the user task.

1. Assuming that components can only be available on devices, which fulfil their requirements

Second, it also enforces authorization of access to device resources. For example, assuming that in a pervasive environment the devices have different policies for resource usage by different persons. Thus, for a user with full access to resources, the task can be instantiated according to his preferences. However, another user with limited resource access may not be allowed to use certain resources and, thus, the components requiring access to such resources will not be selected on those devices, even though they may best satisfy user preferences.

This also has business implications for service providers to distinguish their premium service users from the other users. For example, in pervasive environments in public places such as airports, railway stations, etc. a *premium traveller* may have access to use the ad screens available in an airport terminal, for example, for some activity of a short duration, but other travellers may not. This distinction will be made by the runtime after checking the user's access capabilities in the company's database.

Thus, the same user task may be realised differently for different users in the same pervasive environment, and even for the same user in different pervasive environments. That's why we consider service requirements an important aspect of our user task composition process.

1.2.3 Network Heterogeneity

Existing user task composition approaches assume that services can be instantiated across all the devices uniformly and there is a common communication protocol that is understood by all the devices equally. However, such assumptions fail when we consider heterogeneity of the hardware and software of devices and a variety of communication capabilities such as Bluetooth, GPRS, WiFi, and IrDA etc., available on different devices. For example, consider that a user's mobile phone is equipped with Bluetooth only. When the user instantiates a task on his device, the device will be able to communicate only with those devices that support Bluetooth. However, assuming that the other devices may support additional communication technologies, they can further communicate among them for realization of the user task. Thus, while the user device has limited network visibility, our approach allows for user task composition by facilitating the discovery of resources required by the user task.

1.3 Organization of the Thesis

The rest of this thesis is organised as following. In Chapter 2, we discuss the principles of service-oriented pervasive computing that form the foundation of our work and then provide a survey of the state-of-the-art in the area of user task composition.

In Chapter 3, first we survey existing languages for describing devices in a pervasive environment. Then we propose our own model for describing device capabilities as an extension of the existing standard, Composite Capability/Preference Profile (CC/PP). Based on our extended model, we then describe how user preferences can be specified for device capabilities. The preference elicitation is also based on an existing model for qualitative user preferences called Conditional Preference Networks (CP-nets). However, due to limitations of CP-nets, we propose to extend it by combining it with our quantitative preference model. We then describe an algorithm for selection of devices in pervasive

environments based on our extended CP-nets specifications. Finally, in Chapter 3, we also explain the modelling of user task and the underlying network for mapping of the user task on the available network components. An algorithm for matching user task with the network components is also presented.

In Chapter 4, we present our proposed middleware for the composition of user tasks. However, first we provide a comparison of the existing approaches for user task composition with our proposed work. Then we explain Service Component Architecture (SCA), a component assembly model that we use for describing our task. We then propose extensions to SCA to overcome its limitation with respect to our work; these extensions concern the semantic description capability, service requirement for device capabilities and service collocation description in SCA. All these extensions are required by our proposed user task model and they are done without modifying the existing SCA assembly model specifications. Finally, the architecture and functionality of our proposed middleware, MATCH-UP, is explained a few performance results are given.

Chapter 5 is dedicated to another aspect of our thesis that is presented as an application of our middleware and demonstrates our intention for continuing our work in that direction. It concerns the seamless continuity of user session across devices in the pervasive environments. Our initial proposition is based on a limited scenario: using the Universal Plug-n-Play protocol for transferring of session from one device to another for multimedia applications involving audio, video, and controlling capabilities. We also explain how our proposed user preferences model can be used for deciding when to transfer a session on another device.

Chapter 6 concludes this thesis with a summary of our important contributions and discusses further, remaining issues to be explored in the future.

Chapter 2

User Task Composition: State of the Art

2.1 Introduction

Pervasive computing is the outcome of computer technology advancing at exponential speeds —resulting in the trend towards increasingly ubiquitous [1], connected computing devices in the environment, a trend being brought about by a convergence of advanced electronic — and particularly, wireless — technologies and the Internet. Pervasive computing goes beyond the realm of personal computers: it is the idea that almost any device, from the human body to clothing to tools to appliances to cars to homes can be embedded with chips to connect the device to an infinite network of other devices. The goal of pervasive computing is to create an environment where the connectivity of devices is embedded in such a way that the connectivity is unobtrusive and always available. These devices may range from resource rich devices such as desktops and laptops to resource constrained devices such as PDA's, smart-phones and sensors deployed in the environment. Figure 2.1 shows various devices that may be present in a pervasive environment.

2.1.1 Towards Service-Oriented Pervasive Computing

The devices participating in a pervasive environment may be stationary or mobile and may contribute resources to the environment by sharing their hardware and software capabilities. This leads to the creation of *service-oriented pervasive environments* in which *services* become first-class entities. The enabling technology behind such environments is the Service-Oriented Architecture (SOA) (Papazoglou, 2003). Using SOA principles, resources provided by devices are abstracted as services.

A service is any software component, data, or hardware resource made accessible to pervasive environment by a device. A service is established as an independent software entity with well defined interfaces that may be accessed without any knowledge of their underlying technologies. As far as hardware resources are concerned, they are also exported as services in the environment through a software abstraction. Examples

1. Some other names for pervasive computing are ubiquitous computing and ambient intelligence. However, due to minor distinguishes between these terms, we prefer to use the term pervasive computing throughout the remaining of this thesis.

Figure 2.1: Different Devices in a Pervasive Environment

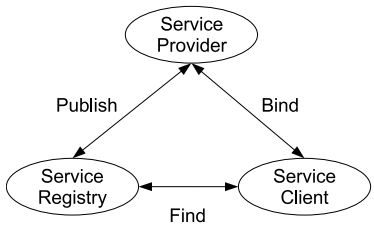

Figure 2.2: Relationship Between the Basic Entities in SOA

of services include printing service provided by a printer, audio/video service provided by TV set, web-search service provided by a PC, temperature reading service provided by a thermometer, etc. These services are provided by devices by being connected to the pervasive networking environment constituted of heterogeneous networks including wired (e.g., WAN, LAN, ADSL Internet connection) and/or wireless networks (e.g., PAN, Bluetooth, WiFi in *ad hoc* or infrastructure mode).

Thus, to satisfy these requirements, services should have the following characteristics as described in (Papazoglou, 2003):

- *Technology neutral* services and clients can be written in any language and deployed in any platform, as long as they can speak the standard languages and protocols that are used.
- *Loosely coupled* services should hide any unnecessary complexity, which is not required for the usage of the service and provide a way to encapsulate it in order to hide it from the consumer of the service.
- *Location transparency* services should be accessible by a variety of clients that can locate and invoke the services irrespective of their location.

Based on the characteristics, the basic SOA is a relationship of three kinds of participants (Papazoglou, 2003): the *service provider*, the *service discovery agency*, and the *service requestor*. The interactions involve the *publish*, *find* and *bind* operations as shown in figure 2.2. Service provider is the role assumed by a software entity offering a service, service requester is the role of a client entity seeking to consume a specific service, and service discovery agency is the role of an entity maintaining information on available services and the way to access them.

While the service provider and service requestor are essentially required for service usage in any scenario, the service discovery agency may not be present in all cases. For example, as mentioned previously, the participation of devices in a pervasive environment may be dynamic and their interconnection may be on *ad hoc* basis. In such *ad hoc* situations, it will not always be possible to expect the presence of an entity with a predetermined functionality.

A user, being part of the pervasive environment, may use the services provided by the environment for realization of their tasks. Previously, in Section 1.1, we identified the challenges arising in such environments if the user task is to be composed dynamically in terms of available services. Before detailing our solution of how to meet these challenges, which will be done in the next chapters, in the remaining of this chapter, we will discuss the existing approaches for user task composition.

Since different works have treated different aspects of user task composition, using different solutions, we will categorise these approaches as: 1) those which want to compose a user task by emphasizing on the resource utilisation in the environment — they will be described in Section 2.2; and 2) those approaches which have emphasized on the modelling aspects of the user task, specifically, modelling the problem as a graph-theoretic approach; they will be described in Section 2.3.

2.2 Resource/QoS-Aware Middleware for User Task Composition

Middleware in this category deal primarily with the usage of resources in the pervasive environment for execution of the user task. These middleware include Aura, Gaia, PERSE and the Platform Composition middleware. Aura and Gaia are two of the most cited approaches with long list of objectives: these research works have been carried out over several years and have resulted in developed solutions for different problems related to user task in pervasive environments. As such it would not be possible to explain all the aspects of their works, but, we will only describe the aspects that closely relate to our contributions. PERSE is a relatively recent work, but due to its contribution in different area of service composition in general, some of its aspects are comparable to our work and, hence, we treat it in the same section. Platform Composition is a work in progress at Intel; however, some of its aspect have been known and will be discussed in the corresponding section.

2.2.1 AURA

The Aura project (Sousa and Garlan, 2002) defines an architecture that allows users to dynamically realize daily tasks modelled as abstract software applications, in a transparent way, without manually dealing with the configuration and reconfiguration issues of these applications. The solution to their problem is based on the concept of personal Aura. The intuition behind a personal *Aura* is that it acts as a proxy for the mobile user it represents: when a user enters a new environment, their Aura marshals the appropriate resources in the environment to support their task. Furthermore, an Aura captures constraints that the physical context around the user imposes on tasks. These constraints are modelled both as non-functional requirements of the user task and as user preferences.

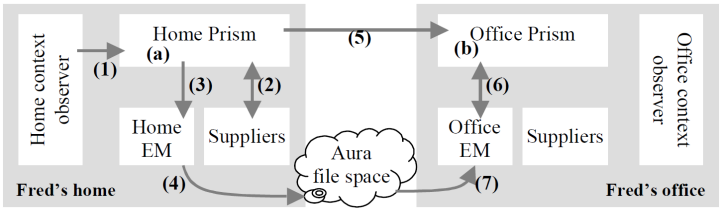

Figure 2.3: Collaboration Between the Architectural Components in Aura

Examples of user tasks are: writing a paper, preparing a presentation or buying a house. Each of these tasks may involve several information sources and applications. Thus, the notion of task is extended by describing a task as a coalition of abstract services, such as "edit text" and "play video." This form of abstraction allows such tasks to be successfully instantiated in different environments using different supporting applications. For example, in a Windows environment Microsoft Word and Media Player might be used to provide the "edit text" and "play video" services, whereas in a Unix environment Emacs and Xanim could be used.

The work mainly focuses on defining an infrastructure for the automatic configuration and reconfiguration of ubiquitous computing environments. There are three aspects to this sub-problem (Sousa and Garlan, 2003). First, as users move from one location to the next, the infrastructure automatically handles the chores of transferring their computer-supported tasks: finding and configuring suitable services to support their tasks, and dealing with migrating/accessing the relevant information. Second, as users switch from one task to another, or resume previous tasks, the infrastructure automatically sets up all of the relevant capabilities. This is done by suspending the state of the task on a file system and then resuming it later in the same or different environment. Third, the infrastructure shields the user as much as possible from distractions by automatically adapting to dynamically changing resources and capabilities.

To make this happen, four architectural component types are defined in Aura: first, the *Task Manager*, called Prism, embodies the concept of personal Aura. Second, the *Context Observer* provides information on the physical context and reports relevant events in the physical context back to Prism and the Environment Manager. Third, the *Environment Manager* embodies the gateway to the environment; and fourth, *Suppliers* provide the abstract services that tasks are composed of: text editing, video playing, etc.

One of the main innovative features of Aura is that user tasks adapt themselves according to the resources available in each pervasive computing environment, thus taking the full advantage of the diverse capabilities of each environment. Furthermore, each environment is able to renegotiate task support with respect to the run time variation of service capabilities and resources. Thus, we can see that mainly Aura is about configuration and reconfiguration of user task based on the available resources in the environment.

Another important aspect of Aura is the consideration of user preferences. The dynamic configuration and reconfiguration of a user task is influenced by their preferences for task configuration and QoS, etc. This aspect will be further discussed in more detail in Section 3.4.1.

The abstract modelling of user tasks allows platform-independence regarding the application-related services that are actually invoked. However, the instantiation and

adaptation of architectures still place high demand on the underlying platform; it requires availability of components implementing task manager, service suppliers, context observer and environment manager in every environment.

Figure 2.3 shows a situation where a user named Fred moves from his home to office. It shows how the various architectural entities of Aura inter-communicate to coordinate the task migration from the home environment to office environment.

The algorithm that the Aura infrastructure performs for the automatic configuration of the user tasks is based on the Knapsack problem solver and aims to maximize user task feasibility in a specific context which is the abstract measure of "user happiness". Since the user task in Aura consists of services, which do not conform to Service-Oriented Architecture (SOA), the stress is on resources in pervasive environment rather than services. The services can only be bound to particular resources that are "wrappers" of legacy applications. This constrains the usefulness of the approach to a pre-defined set of applications, with very little possibility of extension.

2.2.2 GAIA

The Gaia project (Román and Campbell, 2003) is a distributed middleware infrastructure that coordinates networked devices and software components in a physical space, called an *active space*, in order to enable the dynamic binding and execution of software applications. Gaia is a component based meta-operating system, or middleware operating system, that runs on top of existing systems, such as Windows 2000, Windows CE, and Solaris.

The main objectives of the Gaia project include contributions such as: to construct applications that use multiple devices simultaneously, to take advantage of resources contained in the user environment, to exploit context information (e.g., location, mood, and social activity), to benefit from automatic data transformation and to alter application composition dynamically (e.g., attaching and detaching components), to adapt to changes in the environment, and to move the application with the users to different active spaces.

An application is mapped to available resources of a specific active space. This mapping can be either assisted by the user or it can be automatic. This is possible due to the fact that applications based on the Gaia application framework are independent of a particular Active Space by using generic application descriptions that list the application components and their requirements, called Application Generic Descriptions (AGD). These descriptions are used to create a specific application description that uses resources present in the Active Space, which match the application requirements listed in the generic description, called Application Customized Descriptions (ACD). Gaia uses a STRIPS planning algorithm (Ranganathan and Campbell, 2004) for task-based computing, which takes the abstract description of AGD and transforms it in to an ACD using user's current context into account

Figure 2.4 illustrates the AGD for an example music application, Music Jukebox, which provides functionality to organize and play a collection of music files using resources present in the ubiquitous computing environment. The model is implemented by a component named MusicJukeboxModel, requires a parameter with the location of the files, has a cardinality of one (a Music Jukebox application has exactly one model), and

```
Model {
  ClassName  JukeboxModel
  Params     -f <files' location>
  Cardinality  1 1
  Requirements device=ExecutionNode
           and OS=Windows2000
}
Presentation {
  ClassName   MusicPlayer
  Cardinality   1 *
  Requirements device=ExecutionNode
           and type=AudioOutput
           and OS=Windows2000
}
Presentation {
  ClassName   ListViewer
  Cardinality   1 *
  Requirements device=ExecutionNode
           and OS=Windows2000
           or OS=WindowsCE }
```

```
InputSensor {
  ClassName   VCRInputSensor
  Cardinality  0 *
  Requirements device=ExecutionNode
           and device=Touchscreen
           and OS=Windows2000
           or OS=WindowsCE
}
Coordinator {
  ClassName  Coordinator
  Cardinality  1 1
  Requirements device=ExecutionNode
           and OS=Windows2000
}
```

Figure 2.4: An Example Application Description in AGD in the Gaia framework

requires an ExecutionNode device running Windows 2000. Similarly, resource requirement is also specified for other elements in the description.

Gaia supports the dynamic reconfiguration of pervasive applications. For instance, it allows changing the composition of an application dynamically upon a user request (e.g., the user may specify a new device providing a component that should replace a component currently used). Furthermore, Gaia supports the mobility of applications between active spaces by saving the state of the application and retrieving it later in the same or different active space. This is similar to the suspend-resume concept of user tasks in Aura.

Gaia also supports two different types of mobility: intra-space mobility and inter-space mobility. Intra-space mobility is related to the migration of application components inside an active space and is the result of application partitioning among different devices. Intra-space mobility allows users and external services to move application components among different devices. Inter-space mobility concerns moving applications across different spaces.

To configure and coordinate applications and OS components in an easy manner, a high-level scripting tool called LuaOrb is used. LuaOrb is a binding between the interpreted language Lua and CORBA. It extends the interpreted language Lua with a set of facilities to access component infrastructures, and it can perform several of the important tasks at run-time, which makes the functionality of Gaia possible. This dependency of Gaia on Lua is one of the significant shortcomings of Gaia, restricting its usage to particular technologies.

The STRIPS algorithm used by Gaia for task concretization finds a sequence of actions which lead to the best realization of the user task. This sequence of actions is later executed by the Gaia application framework, which ensures that none of the action executions fail because of resource unavailability. However, Gaia's planning algorithm is too restrictive; it only allows planning for Gaia applications which are based on extended Model-View-Controller architecture. The task description also uses proprietary language description. In contrast, we use SCA-based task description, which is technology/protocol independent. In addition, our algorithm for component selection considers user prefer-

ences, device characteristics as well as the service requirements, simultaneously, when selecting components on the devices.

2.2.3 PERSE

PERSE (Ben Mokhtar, 2007) is a semantic middleware, that deals with several aspects of user task composition such as services discovery, registration, composition and QoS evaluation. The middleware is capable of composing user task out of heterogeneous services described in different description languages and supports interoperability between them at syntactic and semantic levels. This is carried out by specifying service conversations as finite state automata, which enables the automated reasoning about service behaviour independently from the underlying conversation specification language. Additionally, it supports the specification of non-functional service properties based on existing QoS models to meet the specific requirements of each pervasive application through the *QoS-aware service Composition*. These aspects have been detailed in (Ben Mokhtar et al., 2007).

To manage interoperability among different service description mechanisms, PERSE introduces the architecture of a semantic service registry for pervasive computing. This registry allows heterogeneous service capabilities to be registered and retrieved by translating their corresponding descriptions to a predefined service model through the *Description Translator*. This service model is defined in OWL-S. The interoperability between different service discovery protocols requires the translation of service advertisements into a common service description language for enabling service matching and composition to be performed independently from the specific underlying languages. Once the translation is done, the services can be published, stored, compared or composed depending on what is needed from the environment.

For user task composition, the first step PERSE performs is a semantic matching of interfaces, called *Service Matching*, that leads to the selection of the set of services that may be useful during the composition. Then, PERSE performs a conversation matching starting from the set of previously selected services, thus, obtaining a conversation composition that behaves as the task's conversation. The matching is based on a mapping of OWL-S conversations to finite state automata. This mapping facilitates the conversation composition process, as it transforms this problem to an automaton equivalence issue. Once the list of sub-automata that behaves like the task automaton is produced, a last step consists in checking —through the *Service Conformance* and *Service Coordination*— whether the atomic conversation constraints have been respected in each sub-automaton. After rejecting those sub-automata that don't verify the atomic conversation constraints, PERSE selects arbitrarily one of the remainders, as they all behave as the user task. Using the sub automaton that has been selected, an executable description of the user task that includes references to existing environment's services is generated, and sent to the *Service Discovery & Invocation* that executes this description by invoking the appropriate service operations.

Figure 2.5 illustrates an example application scenario. In this example, a service GetVideoStream on the user device is to be matched with two network services ProvideGame and SendDigitalStream. Each service specifies its capabilities (input, output) and a semantic matching between these services is performed using the capabilities. These capabilities

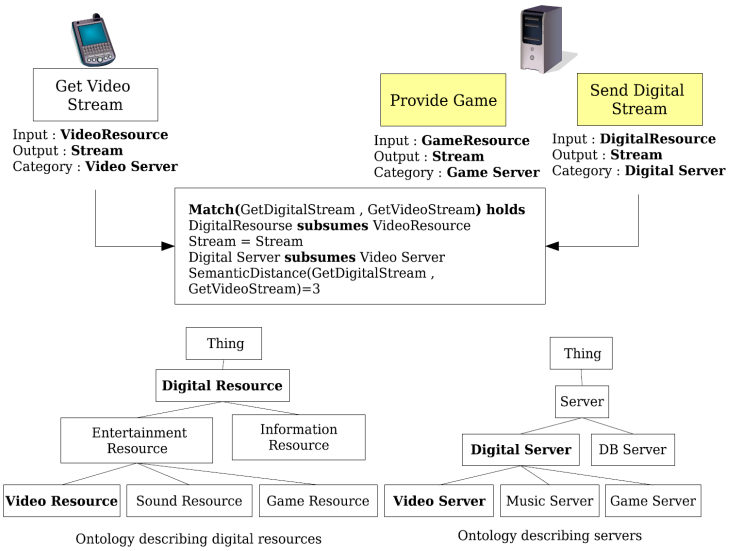

Figure 2.5: Describing and matching capabilities of pervasive services in PERSE (Ben Mokhtar et al., 2006)

are modelled as concepts in an ontology and the matching will result in exact, weak, subsumed or plug-in relations between them, which identifies the degree closeness of concepts (Ben Mokhtar et al., 2007). In the figure, the requested capability GetVideoStream is matched with the provided capability SendDigitalStream, using the two underlying ontologies describing digital resources and servers. The relation Match(SendDigitalStream, GetVideoStream) holds, and the semantic distance between these capabilities is equal to 3. Similarly, a matching between non-functional QoS is also supported by PERSE.

PERSE contributes mainly to solve the interoperability issues arising due to heterogenety of service description mechanisms. As far as the heterogeneity of network access protocols is concerned, PERSE assumes the availability of an underlying infrastructure such as the one proposed by (Raverdy et al., 2006a), that takes care of the network protocol heterogeneity. Thus, PERSE does not deal directly with the heterogeneity issues arising from the network protocols. Another limitation of PERSE is the availability of a central registry in pervasive environments. As described in the beginning of this chapter, in pervasive environments such as home networks such an assumption cannot be held true and hence PERSE cannot be employed in such situations.

Compared to their approach, we also proposes the use of semantic matching but instead of being bound to a particular semantic description language such as OWL-S, we propose to use semantic annotations, which are independent of description languages and can be used with any semantic model. However, due to the fact that such approaches require use of a central repository for description of the ontology, we do not integrate it in our final solution.

2.2.4 Platform Composition

Platform Composition (Pering et al., 2009) is a technique that integrates standard computing components to support effective collaborative work by wirelessly combining

the most suitable set of resources available on nearby devices of a user. It refers to connecting devices together using standard distributed network protocols in such a way that existing applications can be run unmodified.

Platform Composition builds on top of Dynamic Composable computing (Want et al., 2008), which enables the on-the-fly assembly of a logical computer from the best set of wireless component parts available nearby. DCC is an extension of the Intel's previous related work termed as *Personal Server* (Want et al., 2002). There are three emerging technology pillars that support Dynamic Composable Computing: 1) availability of high-bandwidth wireless communication networks using standards such as Ultra-Wideband and WiMax, 2) effective processing — due to improvements in processor technology enabling new levels of interoperability between mobile devices and desktop processing systems, and 3) platform sensing — to support many alternative forms of interactions. The approach, based on these three technologies, allows the users to access the components available on the nearby devices in an intuitive manner.

The implementation of their composition system enables sensor-based interaction techniques for manipulating compositions. For example, a simple gesture-based interaction allows a user to rotate her mobile device, and with a quick shake downward, create a connection between her device and another composition-enabled computer. To disconnect the composition, a second metaphor and gesture based command is applied. This type of interaction can be achieved by using a magnetometer to sense the orientation of the mobile device. Similarly, other sensors such as accelerometer and GPS-augmented devices can be used to select the device positioning, orientation and its distance from the user.

The data from one sensor can be used individually, or multiple sensing modes can be used in conjunction. The information can be used to filter irrelevant devices. For example, spatial sensing can be used to only show nearby devices, the motion sensors might be used to filter out any non-static devices, etc. Likewise, multi-modal interfaces can provide additional mechanisms for representing and controlling this filtering process. This context information can be presented in a user interface creating a virtual representation of the devices in the physical space around a user.

At its core, the Platform Composition framework is a distributed message passing framework that has modules for user interfaces, device/service discovery and service integration. The framework exists as a thin middleware layer that is used to orchestrate service connections among devices. The user-interface of the prototype middleware uses a "join-the-dots" metaphor to create logical computer system. A circle graphic is used to represent each discovered computer; while a set of linked surrounding circles represent services that each computer can export. In order to effect a composition, the user can simply draw a line from a service to the desired destination device: active connections are represented by a permanent link between the nodes. This system both allows the user to graphically see what devices and services are available for composition, and also provide an intuitive mechanism to form multi-device compositions, while displaying the entire state of the system. An example composition shown in figure 2.6 demonstrates how components such as clipboard, files, and display are shared among three devices.

Each individual service for sharing a resource is specified by an XML service descriptor file, which encodes basic properties of the service (name, icon, etc.) and provides details

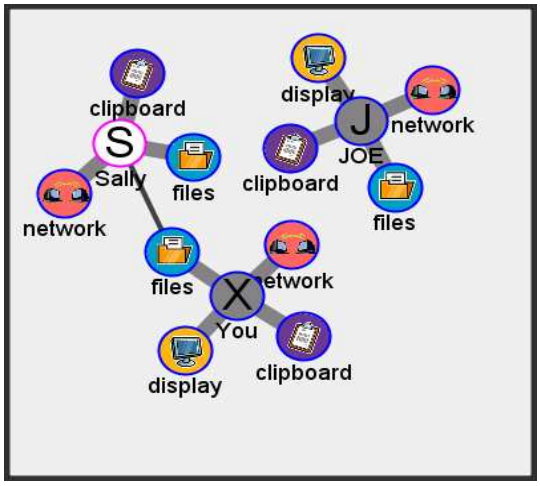

Figure 2.6: Dynamic Composable Computing: Example Composition Across Three Devices

on how to probe, invoke, monitor, and disconnect the service. Services are handled using an explicit client/server model based on commonly available standard systems.

The work on Platform Composition is still in progress and it is expected that, within a few years, this technology will be embedded in the devices. As such most of the work emphasizes the commercial aspects of the research based on the current trends in the hardware technology. Most of the work, described in (Pering et al., 2009)(Want et al., 2008) and (Lyons et al., 2009), concerns the usage of different sensors for collecting context information and then transfer of data among different devices using high-bandwidth protocols, which are not yet commercialised. The work also suggests different approaches for how context information can be used for representation of devices and components on user interfaces. Thus, the research focuses more on the human-computer interaction part of the problem. Some challenges that still need to be addressed are: considering user preference for device ranking and selection of the best available components, dealing with the security and privacy related issues arising from such approach, and the unavailability of wireless standards required by the platform for devices interoperability.

On the common grounds, Platform Composition shares the common objective of creating a user task from the assembly of component available across the devices in the pervasive environment. However, their work focuses more on the use of lower-level technology for providing a higher-level user interface representation. Compared to this approach, our work builds up on the theoretical aspects rather than treating the technological aspects. Instead of presenting a solution based on lower-level technologies, such as sensors and link-layer radio protocols, our approach builds on top of the existing standards. What has been considered a challenge in the cited work has been incorporating as a solution in our work. For example, our work focuses on the selection of devices and components based on the user preferences and thus device ranking is already achieved. More over, we also consider service requirements for execution on devices, which helps in solving the problems related to security and authentication. Built on the SOA architecture, our proposed framework is interoperable and technology-agnostic and does not enforce any proprietary components, which is not the case in the cited work.

2.3 Graph-based User Task Composition

There are some existing approaches, which model the problem of user task composition as a graph. Such approaches are described in this section. These include CoSMoS, PICO, and the PCOM systems.

2.3.1 CoSMoS

One of the most recent works of user task composition using a graph-based approach, proposed by (Fujii and Suda, 2009), presents an approach to dynamic service composition that relies on automatic composition of services based on user queries and preferences in a natural language such as "if I am in a meeting do not use a cell phone." Their proposed framework composes an application through combining distributed components based on the semantics of components and contexts of users. The framework composes applications differently to individual users based on their contexts and preferences. The user preferences are acquiresd from user-specified rules and also via learning. The framework monitors the change in user contexts while executing the requested application, and upon detecting any change, it composes a new application which better suits user's new context (e.g., using newly available components or better satisfying user preference), and seamlessly migrates the execution status from the old application to the new one.

To satisfy the requirement for semantic support, the system comprises of three subsystems: Component Service Model with Semantic (CoSMoS), Component Runtime Environment (CoRE), and Semantic Graph based Service Composition (SeGSeC). CoSMoS integrates the semantic information and functional information into a single semantic graph representation. CoSMoS is an abstract component model designed to model the functions (i.e., inputs, outputs or properties), semantics (i.e., what each input, output and property semantically corresponds to), and contexts (i.e., its location, capability) of components. Each input (and output) of an operation is modelled as a component, representing that the operation accepts (or generates) another component as its input (or output). Similarly, a property of a component is also modelled as a component, representing that it can be retrieved as another component. CoSMoS also models contexts of users and user-specified rules. CoSMoS models those information about a component/user as a semantic graph, that is, a directed graph that consists of labelled nodes and links. CoSMoS is an abstract model and it can be described in different formats, for example, in WSDL and RDF.

CoRE provides a unified interface to discover and access components implemented in various component technologies to make them interoperable with CoSMoS components. SeGSeC is a semantic-based service composition mechanism that allows users to request a service using a natural language sentence and it generates the execution path as a workflow that can then be executed by CoRE.

Figure 2.7 shows the example CoSMoS representations of a microphone component, a voice recognition service component, and a telephone component.

A benefit of the CoSMoS model is that it models functional, semantic and contextual information, and user-specified rules in the same semantic graph format. This allows CoRE and SeGSeC to manage and analyze components and users in the same manner. Also, CoSMoS can be easily applied to different component technologies because it is an abstract model and can be described in different formats.

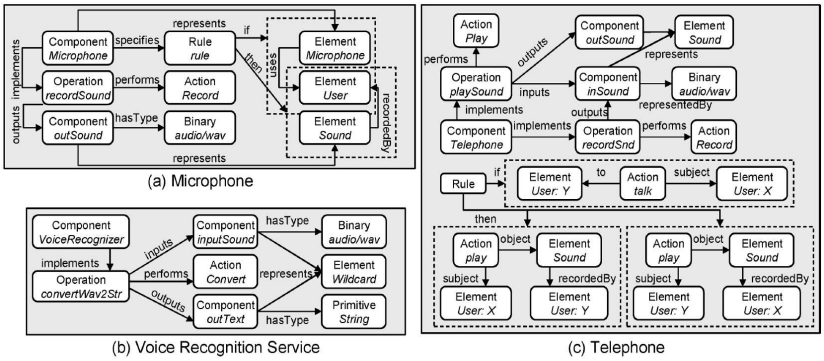

Figure 2.7: Examples of CoSMoS (Fujii and Suda, 2009)

However, on the negative side, CoSMoS also suffers some serious problems. The set of queries that have been reported in (Fujii and Suda, 2009), to test the system's functionality is quite limited indicating that the system is useful only in a few specific scenarios and it needs a lot of add-ons to satisfy additional types of user queries. Another major problem with this approach is that it requires an infrastructure to deliver the deducted service configuration. Contrary to that, our approach of specifying a user task as an assembly of components, embraces the highly distributed and ad-hoc nature of pervasive and ubiquitous computing, and does not depend on computationally intensive deductions.

2.3.2 PCOM

The Pervasive Component System (PCOM) (Handte et al., 2007) is a light-weight component system that allows specification of distributed applications made up of components. The objective of PCOM is to automate the configuration and runtime adaptation of component-based applications using a set of pluggable assembling algorithms. These algorithms compute a valid application configuration based on the functional properties required by component interfaces. Components reside within a component container that is running on every device. Each component explicitly specifies its dependencies in a contract. This contract defines the functionality offered by the component, i.e., its offer, and its requirements with respect to local resources and other components (see fig. 2.8). Offered and required functionalities are described indirectly through interface names. To model the non-functional properties, the syntactical description can be enriched with properties (statically or dynamically), i.e., typed name-value pairs for offers and typed name-value-comparator triples for requirements. Using this description, the system can automatically determine whether an offer can satisfy a requirement. An offer satisfies a certain requirement if the offer specifies a superset of the interfaces and properties contained in the requirement specification and all comparators of the requirement specification evaluate to true when the corresponding properties of the requirement and offer are compared. A component can only be used if its local resource and component requirements can be satisfied by existing offers. Utilizing a component can lead to new requirements recursively, e.g. in fig. 2.8, the Converter requires two additional components. Thus, the application model supported by PCOM is a tree that starts from a root component, the so-called application anchor.

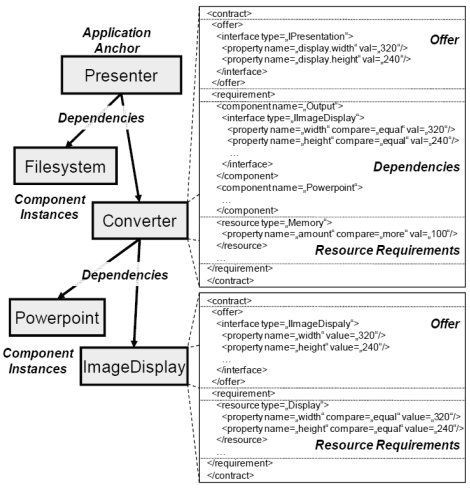

Figure 2.8: PCOM concepts

As such the system supports composite applications with recursive dependencies, which force the assembling algorithms to resolve configurations for only one component at a time thus adopting the greedy approach. While sufficient in some cases, the greedy approach has a number of drawbacks. Firstly, it does not guarantee to find a valid solution when one exists. Secondly, the greedy approach fails to achieve optimality in terms of minimizing the overall resource usage or other criteria in the presence of multiple constraints. Besides, PCOM assembling algorithms only consider the functional properties of the applications, thus limiting the configuration of the assembly to elementary property matching.

Components can be embedded in multiple applications simultaneously (however, one instance of a component per application). To enable this, components are instantiated for each usage. As soon as they are no longer used, they are automatically removed. To do this, PCOM defines a basic lifecycle that consists of a START and a STOP state. The lifecycle of the application anchor defines the overall lifecycle of the application. If the anchor is instantiated and started by the user, the system automatically resolves its contractually specified resource and component requirements recursively. If the anchor (i.e., the whole application) is stopped, the containers release all component instances and resources that have been allocated.

To resolve component requirements, each container is equipped with a remote query interface that lets another container search for matching offers. Dependencies on local resources are automatically resolved by the container that hosts the component. The resources available on a container can be strictly limited, e.g. to model exclusive resources such as input devices. Thus, using a certain component on a device might prohibit the use of another component on this device due to a lack of resources. The container guarantees that the available resources are not allocated in a conflicting way at any point in time.

The configuration process that is cooperatively performed by the containers available in a given environment ensure that only valid configurations are started. A valid configuration is defined as a tree of components starting from an anchor where all contractually specified component and resource requirements are met. Since the set of available re-

sources and reachable devices might fluctuate at runtime, e.g., because a user unplugged an input device or because he carried a Laptop into another room, the container cannot guarantee that the configuration stays valid all the times.

2.3.3 PICO

PICO (Pervasive Information Community Organization) (Kumar et al., 2003), is a middleware framework mainly intended for time-critical applications (e.g., tele-medicine and military applications). This middleware supports the automated, continual, unobtrusive provision of services. The main contributions of their work include (Kalasapur et al., 2007) : 1) modelling of services as graphs containing semantic descriptions, 2) a graph-theory-based service composition mechanism, and 3) a hierarchical service overlay in pervasive computing environments.

The PICO middleware consists of autonomous software entities called *delegents* (or intelligent delegates) and hardware devices that provide services called *camileuns* (which stands for connected, adaptive, mobile, intelligent, learned, efficient, ubiquitous nodes). A camileun is a device possessing one or more functionalities: such as see, hear, adapt, compute, communicate, learn, or process information. The main objective of PICO is to allow the dynamic creation of delegent communities in order to perform tasks on behalf of users. A sequence of events can lead to creation of communities. These events may be generated by camileuns, delegents or users.

A camileun can be described by three tuples, $C = \{C_{id}, C_h, F\}$, where C_{id} is the camileun identifier, C_h is the set of system characteristics, and F is the set of functionalities. The delegents work on behalf of a user or a camileun. A delegent is represented by the tuple $D = \{D_{id}, F_d\}$, where D_{id} is the delegent's identity and Fd is its functional description. Functionally, we can describe a delegent by a three tuple, $F_d = \{M, R, S\}$, where M is the set of program modules, R is the set of rules to operate on modules specified in event-condition-action style, i.e., state machines, and S is the delegent's goal or mission. The delegents are mobile and can move inside and between the communities. A delegent receives inputs from many sources such as external events, internal events, and intentions.

Each delegent (service) is described as a graph $G = \{V, E, v, e\}$ where V is a set of vertices in the graph representing the services. E is the set of edges representing services' input/output. v represent service attributes such as name, location, cost, and semantic description. e represents semantic description of the parameter, the parameter type, the data rates, formats, etc. The services are described semantically in a repository.

In PICO a user tasks contain description of required services, which are then looked for into the environment. The services in the environment are aggregated in the form of a graph and then a matching between the user task and the aggregated services graph is performed at semantic level to find the service instances that match the user task. An important aspect of PICO is that if there is a direct match between a task and a service, the service is returned, otherwise task components are matched against available basic services and a composition of services is returned.

Figure 2.9 (a) shows the representation of a service in PICO that performs text-to-voice conversion. The semantic attribute associated with the input is text, whereas the syntactic attribute specifies the expected form of text, which is ASCII in this case, along

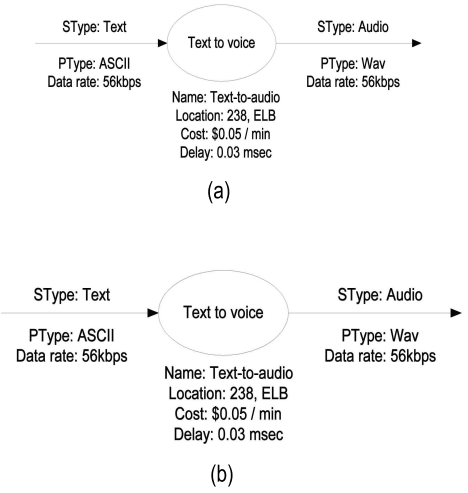

Figure 2.9: (a) Graph representation of a service and (b) Graph representing the task of reading out a file in PICO

with other associated parameters such as the data rate. The vertex and other edges are described similarly. Figure 2.9 (b) shows an example of a task graph, representing the task of reading out a file to a user.

A serious disadvantage of approach is the presence of a centralized directory for services aggregation, which is not always possible in *ad hoc* pervasive environments. It is also assumed that a single aggregated service graph will always exist in the directory, which is not valid in a heterogeneous environment covering a plethora of service types. On the contrary, we propose the aggregation of a graph based on the current task requirements only. User preferences and protocol heterogeneity are not considered as well in their approach.

2.4 Other Approaches for Ad hoc User Task Composition

The above described research efforts have been carried out over several years of research work and as such have been cited significantly by other approaches. In this section, we provide an overview of the work carried out at the University of Oulu, Finland. The interesting aspect of their work is that it they consider both the aspects of service composition that we considered in the previous two sections. That is, they model a user task in terms of services that require resources available in the environment; they also model their task as graph However, while it seems that both of them were done in the research group, it is not clear exactly which of these directions will be used in their future work.

2.4.1 Task Composition Systems at Oulu

The first approach, described in (Perttunen et al., 2007), present a system supporting task-based service composition in smart spaces. The main objective of their system is to compose a user task dynamically such that the QoS required by the user task is maximised. This means to utilize those services in the environment that maximize the

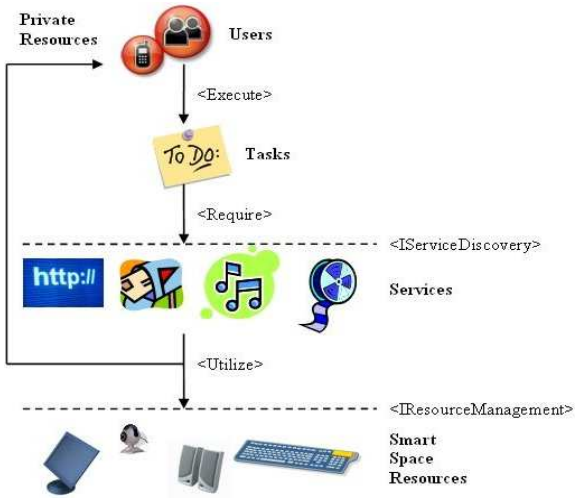

Figure 2.10: Task composition in (Perttunen et al., 2007)

QoS given the user's current situation and active task. In order to utilize the services, suitable resources need to be acquired for them. Both smart space resources and the mobile terminal's resources can be utilized. Figure 2.10 shows the overview of their task model.

The user task description lists the required service types. Services are objects to which computational access is possible through an interface. Service descriptions may specify their resource requirements and they may utilise resources according to binding rules described in policies. The availability of a service depends on the resources it requires. Resources are objects that are used in executing services, and they feature management interfaces decoupled from the services. The rules governing the usage of a resource are defined in its usage policy. The runtime binding of resources to services can be supported by two different QoS models: static QoS and dynamic QoS.

Static QoS refers to the degree of match between the requirements of the user's active task and the qualities and capabilities of a service composition. Dynamic QoS extends the static QoS by taking into account the current state and availability of the service. The similarity between the requirements of the services and the qualities and capabilities of resources determine the static QoS. Modelling the dynamic QoS involves describing the availability estimates of resources. This includes both estimating when a resource becomes available and the length of the period the resource would be available for the requesting user.

User preferences are also considered for resolution of the user task. A user may have one or more preferences (e.g., "use external display when larger than current display"). User's task descriptions automatically include all user preferences, and may over-write and add new task-specific preferences (e.g. "usage of external displays not allowed"). This way the system needs to handle only task descriptions when it discovers and assembles services.

Another aspect of their resource modelling is shared versus exclusive access to a resource. In the special case of shared resources (such as bandwidth), multiple simultaneous services can be supported at runtime. However, usage of resources which require exclusive

excess is done only for specific time period. These periods depend on the usage policy of the resource. These resources are granted on lease for the specific time period using the concept of leases. A lease is negotiated agreement between the mobile client and the resource manager of the smart space, concerning the resources the user needs in performing their task.

When starting the composition process, the Service Assembler (SA) first builds a query to be sent to Service Discovery (SD). The query contains a service composition template (SCT) that specifies the service requirements for the active task. The SCT contains a slot for each required service type, along with possible additional constraints (e.g., maximum allowed distance of a service from the user). SD builds possible service composition candidates (SCC) for the response through matchmaking. Each SCC contains a valid service in each slot of the SCT. Each service in a given SCC contains unique identifiers of the resources it requires for execution. After receiving the SCCs, SA ranks them based on their static QoS. The ranked list of SCCs is used to streamline the process of service selection by selecting the top-ranked services.

Davidyuk et al. (Davidyuk et al., 2008) propose two planning algorithms for dynamic allocation of application components to multiple networking devices. These algorithms, based on evolutionary and genetic computing, optimize the selection of the networking devices and the structure of composite applications according to a given criteria, such as minimizing hardware requirements, maximizing the application QoS or other criteria specified by the user. The algorithms are based on generic models and allow the approach to be used in multiple application domains.

The core of the system consists of the Application Assembly, the Resource Management and the Service Discovery. These components control the applications' lifecycle, monitor the utilization of the network resources and perform dynamic application adaptation when necessary. The Application Assembly uses planning algorithms to find an optimal deployment plan which specifies how the application components are allocated onto the network resources. A deployment plan may become infeasible during the application execution due to resource unavailability or changes in user preferences. Therefore, the Application Assembly has to reallocate the executing application at run-time if needed. The Service Discovery is a ubiquitous database which manages declarative descriptions of applications and network resources. Its main function is to find matches between discovery requests and the descriptions stored in the database. The responsibility of the Resource Management is to monitor and control the environment and also to trigger application adaptation, if some of the application components start to consume more resources than expected.

The application allocation problem is defined by an application and a platform model. The application model is a graph which describes an application. Each node in the application model describes a software component, which implements an interface and has properties, such as resource consumptions. Each link in the application model describes a communication channel between two software components. This means, that one component uses other component's interface. The platform model is a graph where nodes and links describe a network topology. Each node and link is assigned with weights, which model network properties, such as currently available resource capacities. The application allocation algorithm uses these two models as an input and computes a deployment plan, which determines the mapping of the application components onto the resources from the

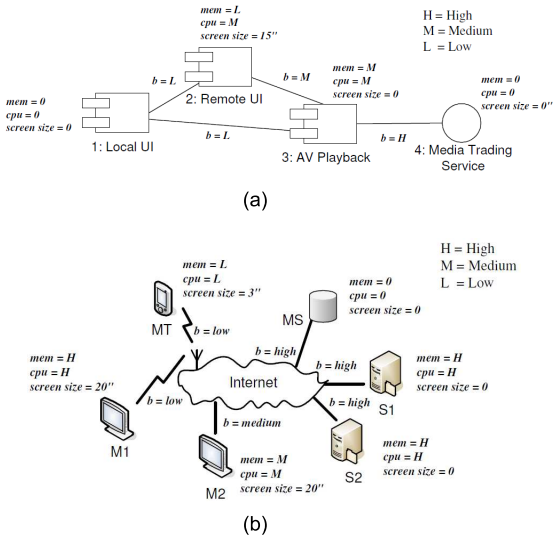

Figure 2.11: Task Composition in (Davidyuk et al., 2008) (a) Example Application Model (b) Platform Model

platform model. Then, the deployment plan is executed by the Resource Management which performs the actual deployment of the components. The Resource Management is also responsible for triggering the adaptation of the application, if some of the previously used resources are not available anymore.

An example of the application allocation problem, consisting of 3 hosts and 4 service components is illustrated in Figure 2.11.

The disadvantage of their algorithms is that due to inherent problems with genetic and evolutionary algorithms, it is possible that the algorithms find no solution at all even though a valid solution may exist. For example, as described in (Davidyuk et al., 2008), even for relatively small number of devices, such as 9 devices or more, the failure ratio of not finding a solution is more than 10%. Another problem with their approach is that they may consider several (possibly thousands of) infeasible solutions before arriving at a feasible solution. This also results in poor performance of the algorithms and in certain cases the time required to compute a valid solution may be of order of several hundreds of seconds. Apart from this, their model considers only CPU and memory utilization, and bandwidth consumption for optimization and do not consider other attributes including user preferences and devices' interoperability.

2.4.2 Work at University of Leuven

In the research work performed at the University of Leuven, Belgium (Preuveneers and Berbers, 2005), the objective is to enable the automated composition of pervasive services by tailoring them according to user preferences in a specific context or the capabilities

of specific devices. This approach uses a centralized algorithm entirely based on OWL and Protégé reasoning tools. The algorithm is based on backtracking as it tries to find a minimal composition of component instances targeted at a certain terminal client device and it cuts down on the user preferences if no suitable solution is found. Thus, the algorithm only focuses on resource constraint satisfaction and it does not optimize either the structure of the composite services or the set of network resources needed. In contrast to their work, we use user preference to guide the selection of devices mainly, although some user preferences are also related to software platform of a device, and this may affect the selection of components on the device as well. Compared to their semantic matching technique for user preferences, which include temporal and locality conditions, the user preferences in our case are only about device characteristics. This is because we consider the *ad hoc* task composition, whose composition in terms of concrete components depends only on the devices and components available in the current user context and the notions of time or locality have no significant meaning.

2.5 Summary

In this chapter we provided an overview of the existing approaches for user task composition in pervasive environments. Most of these approaches assume the availability of a the task description, usually represented by a service template, for which matching services with the required resources are acquired from the environment. This allows the selection of services and resources according to user's current context.

These approaches were divided into two main categories depending upon how they arrive on a solution. First type of approach consists of those research works, which define a user task in terms of the explicit description of the resources they require. Two classical examples of such approaches, Aura and Gaia, were elaborated in detail. Both of these works were initiated in the early 2000's when technologies like SOA were not yet established properly. Thus, they suffered seriously from the issues, which were later addressed by the SOA approach. For example, both Aura and Gaia use a single description to describe the functionality of the user task along with their resource requirements. The granularity of the resources is also not coarse, e.g., instead of describing *what* should be required, they describe *who* should provide it. Similarly, due to unavailability of sophisticated description languages, such as RDF, OWL-S, etc., both of them use proprietary description languages. Issues like these raise the interoperability problems, which restrict the design and development of applications to only the corresponding platforms.

With the arrival of technologies like SOA and description languages such as OWL-S, new horizons were opened. They became quickly the *de-facto* standards for building such applications in which interoperability is the main issue, such as in the case of user task composition in heterogeneous environments. One particular approach that builds on top of SOA and using OWL-S, the PERSE middleware, was also described in this chapter. Taking advantage of such technologies, PERSE solves the interoperability issue to a large extent. PERSE integrates semantic matching for service capabilities during various phases of service publication, advertisement, and matching.

One of the recent work dealing with user task composition is the Platform Composition work at Intel. Although this work focuses more on the technological aspects related to wireless networking, user interface management, and multi-modal interfaces, Intel has

claimed that this technology will be in practice commercially in a few years. It resembles our approach because it also carries out on-the-fly user task composition based on component selection across devices. The utility of the approach will be clear when it is commercialised.

The second type of approaches we described above consists of those research efforts which model the user task as a graph of services and then search for components in the environment, which can be "plugged" into the graph in place of services to realise the user task. The difference between them arises, among other things, on how to model the graph and to find the most suitable components for each service modelled in the graph. PICO uses semantic matching, while PCOM uses simple, syntactic matching of component interface as well as matching the resource requirement. Compared to them CoSMoS is much sophisticated and large. CoSMoS is an abstract component model designed to model the functions, semantics, and contexts of components.

In the next chapters, we will explain our own approach for user task composition that will consider many of the limitations of the existing approaches that we identified in this chapter.

Chapter 3

Device Capabilities, User Preferences, and User Task

3.1 Introduction

In the previous two chapters we discussed the problem we are trying to solve, and analyzed some of the existing approaches for achieving solutions to similar problems. This chapter describes the modelling and algorithmic aspects of our approach. We begin our discussion of characterizing devices in a pervasive environment by describing the differences and commonalities between different devices found in a typical pervasive environment. We then examine several existing proposals for device description and list their limitations in Section 3.2. Then in Section 3.3 we propose our own modelling of device characteristics and present it as an extension of one of the existing standard specification, CC/PP, for device description.

Since an important contribution of this thesis is the resolution of the user task using user preferences, in Section 3.4 we explain what we mean by user preferences and how they can be specified, and then evaluated, to select the best user device (or devices) for execution of the user task. In Section 3.5 we propose a quantitative model for user preferences and in Section 3.6 we extend that model by combining it with a qualitative preference model. Algorithms for selection of devices based on user preferences and example usage scenarios illustrating how device capabilities and user preferences can be used together are also explained the corresponding sections.

Finally, in Section 3.7 we describe the modelling of user task and underlying network as a graph, to consider the network heterogeneity, and in Section 3.8, we provide the algorithm for matching of these graphs to obtain a sub-graph consisting of components that are selected for execution of the user task.

3.1.1 Devices in Pervasive Environments

When we think about pervasive environments, the very first thing we consider is the availability of a variety of devices that may communicate with each other through some peer-to-peer or some intra/inter-net protocol. These devices may have a wide range of different characteristics: hardware, software, communication capabilities, etc. They possess different kinds of technologies for sending and receiving information (GSM, GPRS,

UMTS, Bluetooth, Infrared, wireless and wired LAN, etc.) Specifically, mobile devices — such as smart-phones, PDA's, tablet PC's etc. — are small to enable mobility so they have tiny displays for presenting the visual output. As devices have became smaller and smaller in recent years, the displays have also become smaller. Besides display sizes there are also differences between display resolution, colour representation, image/video capabilities, support of 3D rendering, etc.

Moreover, no standard, unique input method for devices exists; usual keyboard and mouse input are not practical for mobile devices. Featured input methods like soft-keyboards, keypads, thumb-sticks or touch-screens require attainment of new input modalities from the user. Therefore the interaction with the device (e.g., the input of text) will be more time consuming than on a desktop computer with classical input hardware.

There are also certain limitations on mobile devices that set them apart from large devices. For example, the processing power, memory, data storage capacities, and battery life-time of mobile devices is much more limited. As we discussed, the applications designed for pervasive environments do not consider the availability of particular software/hardware components at the time of their conception. It is only at the time of execution of such applications, based on the availability of devices and the available components on them, that such an application is instantiated. Thus, the framework responsible for application instantiation must know about the characteristics of the devices in the environment, in order to determine the device for selection of component for instantiating the application.

As mentioned above, due to heterogeneity of devices, it is hard to assume in advance about the characteristics of individual devices. What we need is a unified way for devices to share their characteristics with other entities such as our application instantiation framework. This can be easily done by adopting a device description mechanism. Such a description contains device's characteristics — hardware, software, and network, etc. — that can be used by other entities in the pervasive environment.

In the literature, there have been a number of device description mechanisms and we will be detailing on some of them in the next section.

3.2 Existing Approaches for Device Description

As the number of user devices is proliferating and as the capabilities of the devices are augmenting on daily basis, there have been a number of different approaches for describing device capabilities. Due to large number of research efforts put into defining new standards and mechanisms, recently, for devices descriptions, it is not possible to mention them all here. However, in what follows below, we present some of the most cited and used works that are adopted by the rest of the research community.

3.2.1 CC/PP and UAProf

Before the dawn of ubiquitous and pervasive computing, when the mobile devices had to access Web contents, the preferred method to determine device capabilities was to use information encoded in HTTP headers. For example, the HTTP/1.1 specification (Fielding et al., 1999) defines the syntax and semantics of some HTTP headers that can be useful for representing device capabilities. Unfortunately, the information conveyed by

HTTP/1.1 headers that can be useful for representing device capabilities is quite limited and includes only information about the user agent, media types (MIME types), character sets, preferred natural languages, and encoding.

To overcome the limitations of the HTTP headers approach, the World Wide Web Consortium (W3C) defined the structure and vocabularies of Composite Capability/Preference Profiles (CC/PP) (Klyne et al., 2004)(Kiss, 2007). A CC/PP profile describes the capabilities of the device and, possibly, the preferences of the user. It serves a model and provides core vocabulary for the devices' capabilities description. It has been designed for small, wireless devices such as PDA's and smart phones. The basic purpose of CC/PP is to allow a server to adapt and deliver contents to its clients based on their capabilities. When a device requests a server, it sends its CC/PP profile embedded in the request, which is typically in HTTP. The server can then filter and adapt its contents according to the requesting device's capabilities.

A CC/PP profile is an XML document based on Resource Description Framework (RDF/XML [1]). Since RDF is meant for describing resources and their characteristics, it is well-suited for specifying a device's capabilities in the form of its resources and their associated properties. The use of RDF enables an extensibility mechanism for CC/PP-based schemas that addresses the evolution of new types of devices, applications, or hardware/software.

The CC/PP standard defines a two-level hierarchy consisting of components, and attributes attached to each component. The standard also specifies several components along with a set of attributes for each one of them. A typical CC/PP profile may, for example, specify the input character set of the device and the installed Java runtime such as CDC/CLDC etc. The CC/PP profiles are provided by the device manufacturers or software vendors, and since devices share most of the common characteristics, vendors may set default values for various attributes of the components.

In a typical application, a device capability and user preferences profile is transmitted from a mobile device through a WAP network via a WAP gateway to the Internet. Here, the request is converted to HTTP, and passed on to the server. The server uses the profile to guide the presentation of content, and the response then passes back through the Internet, past the gateway, and so on back to the end device. The format of the profiles, together with the way they are handled by the network (CC/PPex protocol (Ohto and Hjelm, 1999)), is defined in CC/PP.

As of the writing, the User Agent Profile (UAProf) standard (Open Mobile Alliance (OMA), 2001) is the only well-known CC/PP compliant vocabulary. It has been proposed by the Open Mobile Alliance [2] (OMA) for representing the hardware, software, and network capabilities of mobile devices. The listing in figure 3.2.1 shows a fragment of CC/PP description showing only the SoftwarePlatform component.

One of the key considerations for mobile devices is the limited bandwidth available on the network. For this reason, the UAProf and CC/PP standards provide a mechanism for sending profiles as a URI link to a default profile, and optionally a collection of profile differences that override the defaults. The profiles themselves reside in a profile repository provided by the manufacturer of the device. Thus, instead of sending the complete profile over the low-bandwidth to the server, the mobile device instead sends

1. RDF can be serialized in several ways. CC/PP uses the XML serialization of RDF for its description.
2. http://www.openmobilealliance.org/

```
<rdf:Description rdf:ID="Profile">
      <ccpp:component>
         <rdf:Description rdf:ID="SoftwarePlatform">
            <rdf:type rdf:resource="http://www.openmobilealliance.org/tech/
                 profiles/UAPROF/ccppschema-20021212#SoftwarePlatform"/>
            <prf:AcceptDownloadableSoftware>Yes</prf:AcceptDownloadableSoftware>
            </prf:AudioInputEncoder>
            <prf:CcppAccept>
               <rdf:Bag>
                  <rdf:li>application/java</rdf:li>
                  <rdf:li>application/msword</rdf:li>
                  <rdf:li>application/pdf</rdf:li>
                  <rdf:li>audio/3gpp</rdf:li>
                  <rdf:li>image/bmp</rdf:li>
                  <rdf:li>text/calendar</rdf:li>
                  <rdf:li>text/html</rdf:li>
                  <rdf:li>video/3gpp</rdf:li>
                        . . .
               </rdf:Bag>
            </prf:CcppAccept>
            <prf:CcppAccept-Charset>
               <rdf:Bag>
                  <rdf:li>ISO-8859-1</rdf:li>
                        . . .
               </rdf:Bag>
            </prf:CcppAccept-Charset>
            <prf:CcppAccept-Language>
               <rdf:Seq>
                  <rdf:li>en-US</rdf:li>
               </rdf:Seq>
            </prf:CcppAccept-Language>
            <prf:JavaEnabled>Yes</prf:JavaEnabled>
            <prf:JavaPlatform>
               <rdf:Bag>
                  <rdf:li>CLDC</rdf:li>
                  <rdf:li>MIDP</rdf:li>
                        . . .
               </rdf:Bag>
            </prf:JavaPlatform>
            <prf:JVMVersion>
               <rdf:Bag>
                  <rdf:li>Monty VM</rdf:li>
               </rdf:Bag>
            </prf:JVMVersion>
            <prf:OSName>Series60</prf:OSName>
            <prf:OSVendor>Symbian LTD</prf:OSVendor>
            <prf:OSVersion>5.0</prf:OSVersion>
            <prf:SoftwareNumber>9.4</prf:SoftwareNumber>
            <prf:VideoInputEncoder>
               <rdf:Bag>
                  <rdf:li>MPEG-4</rdf:li>
                  <rdf:li>H.264</rdf:li>
               </rdf:Bag>
            </prf:VideoInputEncoder>
            . . .
         </rdf:Description>
      </ccpp:component>
 </rdf:Description>
```

Figure 3.1: Description of the SoftwarePlatform Component in CC/PP

the URI of the profile and, optionally, the differences that may exist between the device current capabilities and the standard profile. Currently, many hardware vendors make publicly available on their Web sites the UAProf profiles of their devices. At the time of this writing, the list of UAProf descriptions provided by important vendors such as Nokia, Sony Ericsson, and BlackBerry are kept up to date with the new models, suggesting that this technology is considered interesting by hardware vendors.

CC/PP and UAProf Extensibility

While the CC/PP standard has already defined a set of generic components and attributes, a CC/PP profile is not limited to only them. Through the introduction of new vocabularies, new attributes and relationships can be defined as needed. Third parties, such as device manufacturers and application developers, may introduce new vocabularies by defining schemas containing specialized components which extend the CC/PP Component class and by adding attribute to these specialized components.

3.2.2 WURFL

CC/PP defines a rather simple object model, although the underlying language, RDF, is capable of describing not only objects and their properties, but also classes of objects and relationships between classes. The standards are rigid, defining features in a theoretical way, but far from the reality of the various implementations provided by the manufacturers of mobile devices. Also, there are no formalized mechanisms that simplify the identification process of new terminals with respect to the existing ones. Finally, the CC/PP standard constrains the device description hierarchy to be limited to two levels only. This is insufficient for describing devices and — as it will be evident after knowing the rest of the approaches — this limitation results in aggregation of device attributes in only a few components, which leads to incoherence and inconsistency, as new devices, with additional attributes, emerge.

WURFL (Wireless Universal Resource File) (Passani and Trasatti,) has emerged as an open software project mainly to resolve shortcomings of CC/PP and UAProf. WURFL provides a repository of wireless device capabilities and a simple API that permits the access to capability repository. WURLF is based on XML so to be open and platform independent. The advantages of WURFL is that it provides different APIs to access to the terminal repository using Java, PHP, .NET, Perl, Ruby and Python to recognize and identify device capabilities.

All device capabilities are grouped in a single XML file called `wurfl.xml` that is globally accessible to everyone over the Internet. `wurfl.xml` includes information regarding more than 7000 devices (as of this writing). The number of capabilities represented by WURFL is over 300. This makes WURFL a huge database and it would be very heavy if each device is described independently in the file (i.e., more than 2.1 million capabilities). For this reason and from the fact that new devices inherit old ones from the same family [3], WURFL represents information structured in an inheritance tree where devices are positioned relative to each other.

3. Around 10% of new capabilities in the worst cases according to WURFL creators

Each node of the tree represents a physical device or an entire device family. A device identifier `user_agent` and parent device identifier `fall_back` are associated to each device (or each family of devices). The highest parts of the tree define template families that provide *standard* values for the features. The lowest parts of the inheritance tree define specific families and only overwrite the features they actually redefine. The root of the database consists of a device called generic, which establishes all the possible features that can be defined by WURFL. To know about the capabilities of a device, one only needs to know its `user_agent` identifier, which is enough to allow the repository to infer the value of the device's capabilities from another existing device(s).

WURFL is a relatively new approach and it is not yet clear whether the industry is going to embrace it and if there are any chances that it will emerge as a standard. Since the philosophy behind WURFL is to describe devices in a central repository, accessible over the Internet, and then comparing a given device description with it, this raises a big problem for consideration of WURFL in environments where devices are not connected to the Internet. Considering such pervasive environments, it is not desirable to use WURFL for device description.

3.2.3 Other Device Ontologies

The Delivery Context Ontology (DCO) is a work in progress published as part of the W3C Ubiquitous Web Applications Activity by the Ubiquitous Web Applications Working Group [4]. DCO provides a formal model of the characteristics of the environment in which devices interact with the Web. It describes an OWL (W3C, 2003) ontology for device properties as a basis for adaptation to the context in which an application is executed. The delivery context includes the characteristics of the device, the software used to access the Web and the network providing the connection among others. The DCO initial specifications do not specify their position in relation to CC/PP or UAProf. Properties defined do not use the same syntax used for UAProf, however, for each property, alternative names including UAProf syntax are mentioned.

On the positive side, a good aspect of DCO is that it defines classes of units, and conversion between units. It also provides `maximum`, `minimum`, `default` and `actual` value of a given property. For example `pixelCount`, `maximumPixelCount`, `minimumPixelCount` and `defaultPixelCount` represent different types of count of pixels associated with a display or camera. This detailed description is quite important for any device ontology. However, on the negative side, the draw back of DCo includes the fact that it is tailored only for Web-based applications and so its vocabulary is mainly influenced by device characteristics related to Web browsing. This also limits its usability for future devices with yet unknown characteristics.

The FIPA device ontology specified by Foundation for Intelligent Physical Agents can be used by agents when communicating about devices. It specifies a frame-based structure to describe devices, and is intended to facilitate agent communication for purposes such as content adaptation. Agents pass profiles of devices to each other and validate them against the FIPA device ontology. The standard frames defined in FIPA device ontology include `hw-description`, `sw-description`, `connection-description`,

4. http://www.w3.org/2007/uwa/

`memory-description`, `ui-description`, `qos`, `screen-description`, etc. Many of these frames are related to each other in the ontology.

As can be observed, the main problem with FIPA device ontology is its limited set of vocabulary and the lack of mechanism for its extension. Also, the granularity of frames is not clear. This makes the usage of FIPA to be restrictive for particular scenarios and, hence, cannot be used to describe *any* type of device. For example, terminal devices like PC's, PDA's and the like could be described using this ontology, it does not facilitate an effective description of devices like printers, scanners, etc.

(Bandara et al., 2004) introduces an ontology shown in Figure 3.2 for devices description intended at providing a general framework to describe any type of device from PCs, Notebooks to printers, scanners and headsets. The information related to a device is logically divided into five classes depending on the type of information they provide: namely Device Description, Hardware Description, Software Description, Device Status (including location) and Service.

The Device Ontology, described in OWL (Web Ontology Language) (W3C, 2003), is intended to provide a general framework to describe any type of device. But to describe specific types of devices more precisely, the concept of class hierarchies can be used. A hierarchy of sub-classes can be constructed, that inherits from the Device class to provide an effective device categorization. For example there can be a Printer sub-class that inherits from the device class and builds on additional properties as necessary to effectively describe printers. In the case where a device does not fall into any of the available categories, or when it is not clear to which category a device belongs to, it could be specified as an instance of the Device class itself and thereby avoiding the use of the hierarchical classification.

The main purpose of defining the ontology is to allow reasoning about devices using semantic techniques. For example, in (Bandara et al., 2008), authors have used the ontology for semantic matching of device/resource descriptions in pervasive environments. The approach includes a ranking mechanism that orders services according to their suitability and also considers priorities placed on individual requirements in a request during the matching process.

3.2.4 RFCs for Device Capabilities

The specifications in RFC 3840 (Rosenberg et al., 2004) define mechanisms by which a Session Initiation Protocol (SIP) User Agent [5] (UA) can convey its capabilities and characteristics to other user agents and to the registrar for its domain. Capability and characteristic information about a UA is carried as parameters of the Contact header field in a SIP message. These parameters can be used within REGISTER requests and responses, OPTIONS responses, and requests and responses that create dialogs (such as INVITE). Based this RFC, a number of other specifications have emerged, such as PIDF (Presence Information Data Format) proposed in RFC 5196 (Lonnfors and Kiss, 2008), and its related RFCs, as cited in the document define different formats and extensions to represent SIP User Agent capabilities. However, as one might expect, the capabilities

5. Although the term User Agent can be applied to both the client application and to the end points (devices) in a call session, in terms of SIP, we use it here interchangeably to synonym a user device, for the sake of terminology used in most RFCs.

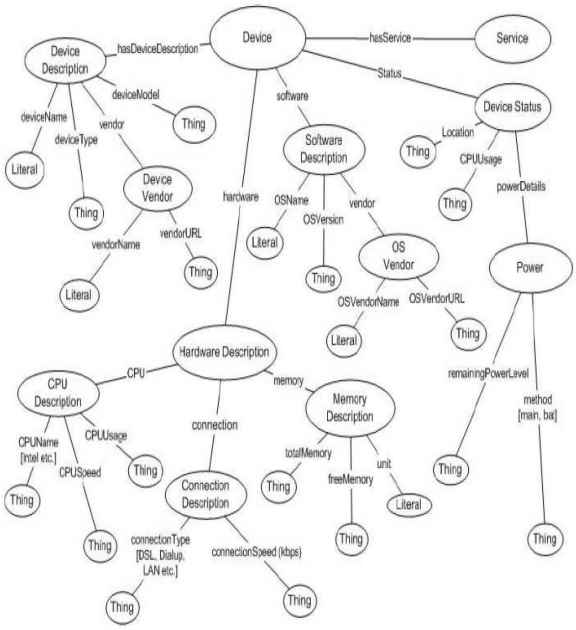

Figure 3.2: The Device Ontology Defined by (Bandara et al., 2004)

defined by these RFCs are mostly what is required only in SIP-related issues and do not cover many of the device capabilities, which might be of interest when considering devices in a truly pervasive environments.

3.2.5 Discussion

As the reader can observe, there are many approaches for device descriptions and we have pointed out the positive and negative aspects for some of those described in this section. Thus, we see that all those, three of them are more or less comprehensive for describing large number of devices: CC/PP, WURFL, and the device ontology proposed by (Bandara et al., 2004). Of these three, WURFL is based on the concept of a central repository containing the major *families* of devices and then using the family reference to conclude the attributes of a device. The limitation of this approach, as mentioned previously, is that it is not suitable for environments that are not connected to the Internet. As such this can be applied to pervasive environments in general, where connectivity to the Internet is not an essential requirement. Thus, we cannot choose WURFL for device description in pervasive environments.

The CC/PP device ontology and the device ontology proposed by (Bandara et al., 2004) have common characteristics that they are both based on languages designed for semantic matching purposes, RDF and OWL, respectively. The CC/PP standard specifications do not provide any pointers for used CC/PP for matching purposes; however, Bandara et al. have already used their ontology for matching of devices in pervasive environment (e.g., see(Bandara et al., 2008)). On the other hand, CC/PP has emerged as standard, used by most of the mobile device vendors for device description of millions of devices already shipped in the market.

Since in our approach we use device description for capturing device attributes, which can then be used for comparing user preferences, the ontology proposed by (Bandara et al., 2004) would have been a good option for device description to be used in our approach, for matching of user preferences and device attributes. However, as we will explain in the next few sections, our user preference model is based on a model that does not rely on semantic matching and it has its own formalism and semantics. For that purpose, all we need is *some* comprehensive device description mechanism. CC/PP, being an adopted standard, is the obvious choice in this case and, hence, we have selected CC/PP for device description in our approach. However, considering the fact that CC/PP descriptions are limited to only two-level hierarchy, as described in Section 3.2.2, we propose, in the next section, to extend CC/PP to multiple hierarchies and to add additional components.

Considering the heterogeneity of the pervasive environments where multi-vendor device participation is not something unusual, we should also consider the importance of other device description mechanisms. This raises an important issue of interoperability of device descriptions. In Section 4.9.1, we will consider this issue in detail.

All the approaches described above for device description emerged in order to overcome the limitations of the existing approaches. However, it is also important to realise that these limitations do not just come from the device, but also the people who use them. Developers must take into consideration user expectations and human-computer interaction (HCI) (Bowman et al., 2005) limitations. That is why, not only it is important to specify how devices can be characterized, it is also necessary to provided means to the user so that they can elicit what type of devices they prefer and based on which criteria. Thus, apart from presenting an approach for device description, we also present a user preference elicitation technique for selection of devices by ordinary users.

First, in the next section we propose a model for describing device capabilities and then based on that model, we describe in the subsequent section, how users can specify their preferences for device capabilities.

3.3 A Model for Device Capability Description

In spite of the fact that there exist a number of description mechanisms for device profiles, they have several limitations — as we mentioned in their respective sections — mostly due to the fact that they do not cover all of the characteristics of devices. It would be *yet another device description* if we are going to propose another description mechanism, and that would not yield much in terms of usefulness. Nevertheless, in the context of the discussed limitations of individual approaches, we now present an extension of the CC/PP model to overcome some of its limitations and to make it richer in vocabulary so that it is more useful. Although, our proposed model can be described by several of the above mentioned mechanisms, we have selected CC/PP as a base for our device description model, because CC/PP is a W3C standard, and it is widely used by the leading device manufacturers.

3.3.1 Capabilities, Characteristics, and Resources

As defined in RFC 2703 (Klyne, 1999), a *capability* is an attribute of a sender or receiver (often the receiver) which indicates an ability to generate or process a particular type of message content. A capability is distinct from a characteristic in that a capability may or may not be utilized in any particular communication, whereas a characteristic is a non-negotiable property of a User Agent (UA). A *characteristic* is like a capability, but describes an aspect of a UA which is not negotiable. As an example, whether or not a UA is a mobile phone is a characteristic, not a capability. The semantics of this specification do not differentiate between capability and characteristic, but the distinction is useful for illustrative purposes. Indeed, in the text below, when we say "capability", it refers to both capabilities and characteristics, unless the text explicitly says otherwise.

Similarly, it is possible to quantize a concrete capability, e.g., memory, CPU, bandwidth, etc. We call such capabilities as the *resources* available on the device.

3.3.2 Abstract vs Concrete Capabilities

A given capability of a device may be specified abstractly or concretely. For example, to specify 1 MB of memory is a *concrete* specification, but to specify or require some input mechanism is *abstract*. Input can be concretized by specifying keyboard, stylus or speech recognition capabilities of the device, for instance.

Mostly, we will consider only device *capabilities*, which will be used by the users for specifying their preferences for device characteristics.. This is because we do not expect an ordinary user to be expert in providing low-level, technical details of their preferences in terms of resources (e.g., an upper or lower bound on memory, bandwidth, etc.); we need to capture them at a more abstract level. This will be discussed in detail in the remaining of this section.

3.3.3 Modelling of Device Capabilities

From the user's point of view, device's capabilities can be further classified into hardware, software and network categories as following.

Hardware Capabilities

To capture hardware related capabilities from a user's point of view, we need to emulate how a user perceives the hardware. Obviously, it is related to the user's interaction with the device, which is characterized by the input and output of the device. Thus, we need to characterize various forms of input/output typically offered to a user. We can model the hardware resources as $Hardware = \{Input, Output\}$, where $Input$ and $Output$ are explained as follow.

Input A device can accept input from the user in various ways. These include input by a pointing device such as a mouse or a stylus, input by typing into the keyboard, and input using the voice commands. In the last case, the device recognizes user's voice and interprets them as commands, which are used to take actions in the same way as it would have been by using a pointing device or a keyboard.

Another form of input is video. A video input capable device may use video input for command processing or for user tasks like video chatting or conferencing, and may be one of the required or desired features of a device. In its simplest case, it could be a inbuilt video camera or a web-cam attached to the device.

Output The visual output of a device is mostly associated with its screen or attached monitor. For an ordinary user, the important factors are: whether the display is colour, whether images can be displayed and whether video can be played or not. Audio output must also be considered, e.g., speakers or attached audio units for listening to sound and music. One of the most important features of the output categories is the screen size. Some users may prefer bigger screens while others may prefer smaller. Thus we cannot select a user preference on the basis of largest or smallest screen size. In the next section, we describe how screen size is considered as a user preference.

Software Capabilities

Software cannot be considered as an inherent characteristic of a device. However, from user's point of view, it is considered to be a valuable capability offered by the device. When given a choice among many devices possessing same hardware capabilities, a user will prefer that device which has a particular software application installed on it. It is not possible to enumerate a list of applications which are pre-installed on the device; however, we can generalize this to specific categories. A user may be interested in generic applications including, but not limited to: calendar application, email client, messenger, SIP client and photo browser, etc. Alternatively, user may be looking for a specific application, such as an Internet browser or a word processor from a particular vendor, to be available on the device.

The set of all software resources can be modelled as: $Software = \{a_1, a_2, \ldots, a_n\}$, where a_i represents a particular software capability that is of interest to the user.

Network Capabilities

Most of the network related characteristics are resource related and, thus, should not be considered as user's preferences, such as bandwidth and network interface type. Two prominent aspects in which user would be interested are its economical aspects and security, i.e., a user will prefer a particular network access type over the other due to connection charges, and will expect the communications to be secure. The set of network capabilities is represented by: $Network = \{w_1, w_2, \ldots, ...w_n\}$, where w_i represents a network access method or protocol, or a non-functional aspect such as security or payment.

The overall capability model C can be represented by:

$$C = \{Hardware, Software, Network\} \tag{3.1}$$

All of the capabilities in C are of either type *boolean* or *literal*. With these data types, we can represent very well the user preferences for a specific device capability, such as `TextInputCapable=Yes` or `EmailClient="Mozilla"`.

Figure 3.3 shows the extended CC/PP model, as proposed by the authors in (Mukhtar et al., 2008b). The figure shows how the identified capabilities have been merged into the

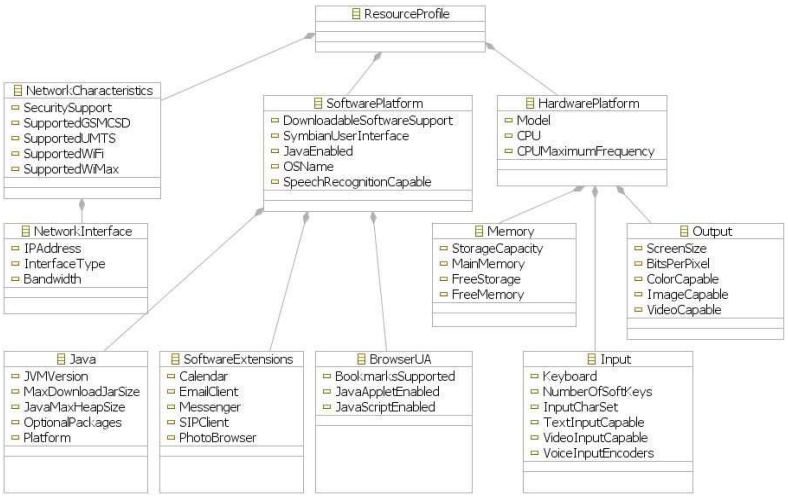

Figure 3.3: The extended CC/PP model

categories of *HardwarePlatform*, *SoftwarePlatform* and *NetworkCharacteristics* and their sub-categories. In fact, the purpose of the extension is to specify additional sub-categories in CC/PP indicating the need for refinement and re-organization in CC/PP. For example, by the addition of *SoftwareExtensions* category, new attributes such as *Calendar* and *EmailClient*, etc. have appeared in the model. Attributes like these will be required for elicitation of user preferences as explained in the next section.

Despite the fact that we have added additional categories and attributes to the existing UAProf model, we cannot claim that they are enough for complete modelling of device capabilities. This is because due to rapid advancement in technologies, we observe new hardware, software, and networking solutions on daily basis and to gain market shares, manufacturers put their maximum efforts into integrating these solutions into the existing ones[6]. For example, none of the existing device capability description mechanisms — including WURFL that is continuously updated — supports description of the features including GPS, accelerometer, etc., which have been embedded into many mobile device for quite a long time.

Thus, instead of trying to model every possible device capability, we have tried to include the most common denominator that is supported by the majority of the available mobile phones. In our opinion, these are also the characteristics that are useful in most of the situations.

6. An example is the introduction of the Wi-Fi Direct specifications by the Wi-Fi Alliance, less than a month ago as of this writing, which will enable Wi-Fi devices to connect to one another without joining a traditional home, office, or hotspot network. This will emerge as a replacing technology for Bluetooth, which might no longer be offered by the manufacturers.

3.4 User Preferences

Now that we know how devices' characteristics can be described in our extended CC/PP model, we can now describe how users can specify their preferences for device capabilities.

3.4.1 Problem Description and Assumptions

We assume that the user wants to select one or more devices for execution of a task whose description is available on his device. We also assume that a number of devices exist in the pervasive environment with different characteristics, ranging from small handheld devices with limited capabilities to powerful multimedia computers with abundant resources. These devices host one or more software components, which are required by the user task and are of interest to the user. Examples of such components include video player component, weather update component, etc.

The user task may be composed of several services, and for each service in the user task, we may find one or more matching components across different devices; however, while these components offer similar functional interfaces, they are considered different from each other in terms of the capabilities of the devices. Hence, for example, a video component on a smart phone is not considered the same as one on a laptop. It will be based on the user preferences that a particular device will be preferred over the other, when they host the same components.

There are two main reasons for this approach. First, consider a user entering in a pervasive environment, where different devices are available and the user has no prior knowledge of their characteristics. We assume that all the devices are utilizable by the user and the user has a description of a task on his device. Then he can execute his task on any of the available devices, if they have the components required by the services in the task. Thus, the user is not limited to using his own device, but can use any device in whatever environment he moves to.

Second, for task execution, the user can specify his preferences and dislikes qualitatively in the form: *I prefer big screen to small screen* or *I prefer a device which has the Firefox web browser installed on it but which connects to the Internet using Wi-Fi and not GPRS*. Using various such preferences, our proposed model selects the device that meets best his preferences for the particular task.

Since we consider *ad hoc* pervasive environments, where device participation is dynamic, we do not consider the availability of a central registry or server that contains devices descriptions, but we assume that the devices are able to share their capabilities and the available components on them, with the other devices, using some device/service discovery mechanism. All devices are able to communicate with each other using mechanisms like Wi-Fi and Bluetooth. We assume that the heterogeneity of communication protocols is taken care of by using an approach similar to (Mukhtar et al., 2007a), that will be described in Section 3.7.

We will present two approaches for device selection based on user preferences: a quantitative approach, as we have proposed in (Mukhtar et al., 2009c), that is based on a utility function, and a qualitative approach mixed with quantitative one, as we have proposed in (Mukhtar et al., 2009a), that is based on graphical representation of user preferences.

In the former case, a linear additive utility function is used for calculating devices' weights and the device with the highest weight is selected for the user. The limitations of this approach include the inability to express the conditional and relative importance statements as well as the assignment of multiple values to variables. These limitations can be overcome by our second approach, which allows users to specify their preferences qualitatively in an intuitive manner, but which evaluates the preferences quantitatively and, hence, leads to better precision as compared to using quantitative or qualitative method alone. One important aspect of both of these approaches is the selection of multiple devices: if the selected device does not host all the components required by the task, then the next better device is selected for the rest of the components and so on.

Both of these approaches will be discussed in next sections. However, before that we need to describe how user preferences were used in the Aura project(Sousa and Garlan, 2003), for comparison purposes. We are providing a comparison with Aura only because, to the best of our knowledge, this is the only approach that considers user preferences for execution of the user task comprehensively. Other approaches like (Ben Mokhtar et al., 2007) and (Fujii and Suda, 2009) also claim to consider user preferences, but their treatment of user preferences is only nominal.

User Preferences in Aura

In Aura (Sousa and Garlan, 2002)(Sousa and Garlan, 2003), computing the best match between what the user wants and what the environment has to offer corresponds to maximizing a utility function. The utility offered to the user is defined as the difference $benefit$ - $cost$. The cost refers to the cost of change when a reconfiguration is required, e.g., swapping a text editor by another. The benefit is calculated by the user preference satisfaction.

User preferences (and their formal reification, utility functions) used in the Aura infrastructure have three parts: first, *configuration preferences* capture the preferences of the user with respect to the set of services to support a task. Second, *supplier preferences* capture which specific components are preferred to supply the required services; and third, *QoS preferences* capture the acceptable Quality of Service (QoS) levels and preferred tradeoffs.

For example, for taking notes, the user may prefer to dictate the text. However, if the environment lacks the capabilities (microphone, speech recognition software,...) or resources (CPU cycles, battery charge,...) to support dictation satisfactorily, the user is willing to type or write the text. This means the user preferences related to configurations. Similarly, a user may specify their supplier preferences in the form of preferring one text editor over the other and QoS preferences as by specifying the trade off between response-time and accuracy of a query, etc.

These user preferences are managed by different architectural components of Aura. Configuration preferences are used by the Task Management when deciding what to configure for the user. The Environment Management uses supplier preferences to make a first pass at finding the best candidates to support each requested service, and then it uses QoS preferences to make a final decision on which components are better positioned to deliver the QoS expected by the user. All these different types of preferences are modelled independently by different equations involving different set of variables related to the

Table 3.1: Values for User Preferences

Importance level	Preference value
Very Important	1.0
Important	0.7
Like	0.4
Somewhat Like	0.2
Don't Care	0.0
Somewhat Dislike	-0.2
Dislike	-0.5
Dislike Very Much	-0.7
Avoid	-1.0

environment, timeliness, QoS, suppliers, user happiness, etc. Specifically, the Environment Management will consider a reconfiguration if the inequality below evaluates to true:

$$\prod_{s\in a} F_s^{c_s}(\bar{p}_s)\cdot\left(\prod_{d\in QoS\,\dim(s)} F_d^{c_d}(\bar{f}_d)\right) < F_w^{c_w}\left(\max_{s\in a|_{x_s=1}} W(p_s)\right)\cdot\prod_{s\in a} h_s^{x_s}\cdot F_s^{c_s}(\hat{p}_s)\cdot\left(\prod_{d\in QoS\,\dim(s)} F_d^{c_d}(\hat{f}_d)\right)\Bigg|_{\substack{resources,\\ QoSprof(\hat{p}_s)}}$$

As the reader can observe, deciding for a reconfiguration in AURA not only requires monitoring the environment, but it also results in continuous evaluation of several variables simultaneously and comparing the results. In fact, as concluded by the authors of Aura (Poladian et al., 2004), the multiplicative model of preferences for QoS requires that these dimensions satisfy certain independence assumptions, which is not possible in each case. Also, the model depends on accurate prediction of applications' resource demand, which is also not feasible in dynamic situations, such as pervasive environments.

This, in our opinion, is a serious drawback of the approach as it is 1) complex to process and 2) requires re-evaluation of several variables again and again. In the next sections, we also develop two user preferences models and detail on how they can be better than the one described above in terms of its simplicity of processing, as well as its efficient re-evaluation when changes are detected in the environment.

First, in Section 3.5, we specify a utility function for user preferences. However, as we will discuss there, that due to its limitation, we will also propose a qualitative model in Section 3.6 and describe its usefulness as compared to a utility-function based preference model.

3.5 A Quantitative Preference Model

Our quantitative preference model uses a utility function for evaluation of user preferences. This utility function takes as input the collection of preferences specified by the user and for each specified device, matches its characteristics with the user preferences and calculates a real-valued device value for each device. For a device satisfying more of the user preferences, the value will be higher as compared to a device which satisfies relatively less number of user preferences. This can be explained as following.

User specify their preferences for device capabilities and resources with the help of a GUI provided on the device such as by clicking certain choices, by enabling/disabling

certain options, or by specifying a preference value directly. Depending upon the value, the preference dictates the approval or dislike of the user for a particular capability of the device. It is a real number value between $(-1.0, 1.0)$. The reason for choosing such values is two folds: first, it helps in defining a normalized function with respect to user preferences; second, by including negative values, it is possible to eliminate devices with unwanted capabilities, as will be shown later. Using such values, 1.0 represents a *very important* capability, and -1.0 represents user's dislike, so much to *avoid* using a device with such a capability, and 0.0 representing a *don't care* condition, i.e., the availability or unavailability of such a capability is not important for the user. Table 3.1 summarizes some values assigned to different preference levels by the user. However, any real value between $(-1.0, 1.0)$ can be used. As can be seen, positive values indicate user's approval while negative values show their dislike for device capabilities.

While the user preferences for a capability can be in the range of $(-1.0, 1.0)$, the availability or absence of a capability c on a device is noted by `true` and `false`, which can be represented by 1 and 0, respectively. For example, for a boolean capability `TextInputCapable`, the possible value can either be `true` or `false` for a device. When a device is capable of accepting text as input, the capability will be presented as `TextInputCapable=true`, which will be translated by the value of 1.

This is treated differently for capabilities whose values can assume literals instead of boolean. For example, for the device capability, `BrowserAgent`, a user may specify any of the value among Internet Explorer, Firefox, Safari, etc. and a device for which the capability matches with the user preference, the capability will be evaluated to true, otherwise, it will be false.

3.5.1 Constraints Satisfaction

We consider two types of constraints when selecting devices and services: constraints posed by user preferences and constraints posed by the services in the user task.

3.5.2 User Preferences Based Constraints

The user would like to execute their task on a set of devices for which maximum of their preferences are satisfied. In other words, those devices should be selected in priority for whose capabilities, user preferences have higher values; and those devices should be selected with least priority for whose capabilities the user preferences are set to lower values. Thus, the user preferences present particular constraints on the selection of devices. These constraints result in weighted capability values which are used in determining the overall device value to the user. The constraints are defined as following:

Required user preferences

If the user has specified a value of 1 for a particular device capability, it means the device must have such a capability. If the device's value for such a capability is *false*, that device should not be considered for service selection.

Avoided user preferences

If the user has specified a value of -1 for a particular device capability, it means the selected device must not have such a capability. If the device's value for such a capability is *true*, that device should not be considered for service selection.

Screen size

We adopt the approach proposed in (Perttunen et al., 2007) for selecting the screen size, which is closest to the one specified by the user. Let $R_u = \{w_u, h_u\}$ and $R_d = \{w_d, h_d\}$ represent the screen resolution specified by the user and the screen resolution available on the device, and w and h represent the width and height dimensions of the screen, respectively. Then the formula to calculate the nearest matching screen size is:

$$Match(R_u, R_d) = \frac{\frac{Min(w_u,w_d)}{Max(w_u,w_d)} + \frac{Min(h_u,h_d)}{Max(h_u,h_d)}}{2} \tag{3.2}$$

Using this equation, only the exact match will return the value of 1, while both larger and smaller screen sizes will return smaller values.

Presence of a preferred capability

If a user has specified an importance value $0 < v < 1$ for a capability and the device's value for such a capability is *true*, the device should have more importance. So the weighted value w for the capability c becomes $w = v > 0$.

Absence of a preferred capability

If a user has specified an importance value $0 < v < 1$ for a capability and the device's value for such a capability is *false*, it means the device lacks the particular preferred capability; so its value should be decreased. The inverse of the preference value is used for that capability, i.e., $w = -v < 0$.

Presence of a disliked capability

If a user has specified a dislike value $-1 < v < 0$ for a capability and the device's value for such a capability is *true*, it means the device has a certain capability, which is undesirable for the user. The value $w = v < 0$ is used to decrease the device value.

Absence of a disliked capability

If a user has specified a dislike value $-1 < v < 0$ for a capability and the device's value for such a capability is *false*, the weighted capability $w = -v > 0$ is used, i.e., the device value is increased.

Don't care preferences

A don't care preference results in a weighted capability value $w = 0$. That is, the availability of unavailability of such a capability should not have any affect on the device value.

Based on the above constraints, given a capability c_i of a device and the user preference or dislike value v_i for c_i, we have the weighted capability w_i as:

$$w_i = \begin{cases} v_i & \text{if } c_i = 1, \quad -1 \leq v_i \leq 1 \\ -v_i & \text{if } c_i = 0, \quad -1 \leq v_i \leq 1 \end{cases} \tag{3.3}$$

Number of preferred capabilities

We consider the number of preferred capabilities fulfilled by a device compared to the total number of preferences specified by the user. A device satisfying more number of preferred capabilities should have more value as compared to one that satisfies fewer preferences. If I_s denotes the number of preferred capabilities satisfied or fulfilled by the device and I_t denotes the total number of preferred capabilities specified by the user, then the ratio I_s/I_t is added to the device value.

Number of disliked capabilities

We also consider the number of disliked capabilities present on a device compared to the total number of dislikes specified by the user. A device which has more number of disliked capabilities should have less value as compared to one which has fewer of them. If D_s denotes the number of disliked capabilities present on a device and D_t denotes the total number of dislikes specified by the user, then the ratio D_s/D_t is subtracted from the device value.

Finally, let us denote by DV the *device value* of a device, which is calculated as following:

$$DV = \sum w_i + Match(R_u, R_d) + \frac{I_s}{I_t} - \frac{D_s}{D_t} \tag{3.4}$$

where w_i is the weighted capability as defined in eq. (3.3) and the function $Match()$ is defined in eq. (3.2). Equation (3.4) gives us the importance of a device based on its capabilities and user preferences for those capabilities.

If a device value $DV < 0$, it is not considered for service selection. This is because the device is not considered suitable for the user, given their preferences.

3.5.3 Task Based Constraints

The constraints described above help in the selection of the *best* device for executing a service. This selection is made on the basis of user preferences. However, this cannot guarantee that all of the services specified in the user task can be executed on such a device. The selected device may not be able to execute all of any of the services. In order to know if a service is executable on a device or not, we need to consider the following constraints by the services in the user task.

Algorithm 1 ServiceSelection($Task$, $DevicesList$)

```
1: Task contains set of services
2: DevicesList contains set of devices
3: sort DevicesList by decreasing device values
4: for each service s in Task do
5:   for each device d in DeviceList do
6:     if fit(s, d) and fit(collocation(s), d) then
7:       select s for d
8:       select collocation(s) for d
9:       remove s from Task
10:      remove collocation(s) from Task
11:    end if
12:  end for
13: end for
14: return Task = ∅
```

Service fitness

Each service may specify one or more capability requirements, which must be satisfied in order for the service to be executed on the device. If any of the required capability's value is *false*, the service cannot be executed on that device.This is defined as service fitness. *Service fitness* is a boolean value; *true* indicates the service is executable on the device, *false* indicates it is not.

Service collocation dependency

Two or more services may specify their collocation dependence on each other, which means that they must be executed on the same device. They are known as *collocating services*. This may happen due to intrinsic inter-service dependency or may be the result of a user preference. For example, a user may specify that all the services should be executed on the same device, rather than on various devices apart. We define collocation as bidirectional, i.e., if a service S_1 collocates with another service S_2, then S_2 also collocates with S_1.

Based on these two types of constraints, the service selection algorithm is defined as shown in Algorithm 1. In the algorithm, the device value is used to sort the devices (line 3) and starting from the highest value device, services are allocated one by one. The algorithm will be successful only if all the services in the user task have been assigned to devices. The algorithm references two other methods, which we have not defined previously. The *collocation*() method determines if the service is to be collocated with some other service (or vice-versa) on the same device. The $fit()$ method determines the fitness of a service on a device. It returns a *false* if the component implementing the service is not available on the device or if the service requires certain capability, which is not present on the device.

In the next subsection, we depict a scenario that illustrates how quantitative user preferences can be used for selection of one or more devices in the pervasive environment.

Table 3.2: Devices capabilities and user preference values for each capability in different cases

Devices capabilities (True=1, False=0)						User preferences		
Capabilities	PDA	MM	FS	LT	DT	User 1	User 2	User 3
PhotoBrowser=Picasa	0	1	0	1	0	0.7	0.0	0.0
VideoViewer=VLC	0	1	0	0	1	0.2	1.0	0.5
VoiceInputCapable	1	0	0	0	0	0.3	0.0	0.7
OSName=Windows	0	0	1	1	0	0.7	0.5	0.5
AccessCharges	0	1	0	0	0	0.0	-0.5	-0.5
SecuritySupport	1	1	0	0	1	0.3	0.0	0.0
Keyboard=AZERTY	0	0	0	1	0	0.0	0.2	-0.5
Screen Size – Width	320	1200	1920	1024	800	x	1024	1900
Screen Size - Height	240	1024	1200	768	600	x	768	1600
VideoOutputCapable	0	1	1	1	1	x	x	x
SoundOutputCapable	1	1	0	1	0	x	x	x
InputCapable	1	1	0	1	1	x	x	x

3.5.4 Example Scenario

Consider that a user wants to execute a multimedia task, whose description is available on their device. The task allows him to browse photos, watch video and listen to the related audio, and to control the playback of audio, video and photos. Thus, essentially there are four services in the user task: photo service, video service, audio service and the controller service. We assume that on the user device, these services are described as $S = \{Photo, Video, Audio, Controller\}$. The user specifies that Video and Audio services should coexist, as it would not be of much pleasure to have audio and video coming form different devices.

In the pervasive environment, five different devices are available. These are: 1) a PDA; 2) a high-end multimedia PC (MM); 3) a flat panel screen (FS); 4) a laptop (LT); and 5) a desktop (DT). The capabilities of each device are shown in the left part of table 3.2. Although the capabilities are specified as *true* or $false$, we use 1 and 0 to represent them, respectively. For example, the table shows that only the laptop has a keyboard whose layout is AZERTY, because its capability `Keyboard=AZERTY` is true (i.e., 1); the rest of the devices have other types of keyboards. A user can provide preference for only a single value per capability at a time. That is, users cannot specify two different preferences for one capability (e.g., "I like VLC and Windows Media Player" or "I like VLC but I do not like Windows Media Player", etc.)

Finally, the fitness function for each service is calculated on the basis of their requirements for device capabilities, which are shown in the bottom right part of table 3.2. These capabilities evaluate as following: 1) the Photo and Video services can be executed on all devices except for PDA, which cannot execute the Video service, because it requires the capability VideoOutputCapable and the PDA does not satisfy this requirement because it does not have a video viewer installed on it; 2) the Audio service cannot be executed on both FS and DT, because the service requires SoundOutputCapable capability and these devices do not have any audio unit attached to them; 3) the Controller service cannot be

executed on FS as it requires the abstract requirement InputCapable and this capability is not provided by FS.

We now discuss an example case in which three users specify different preference values for various capabilities of the device. This has been shown in the right part of table 3.2. Each column represents the preference values for a particular user. Based on user preferences and calculations in eq. (3.4), the calculated *device values* and the selected devices are shown in table 3.3. Now we are going to show how different preference values for each user will result in selection of a different set of devices.

User 1 in this case, the user has assigned the preference *important* to Picasa photo browser and Windows operating system. This means the selection of the device will be influenced by these parameters mainly, which dominate the rest of the preferences. Since the user has not specified any preferences for screen size, it is considered as a *don't care* case and will evaluate to 0 for all the devices. Based on eq. (3.4), the device value is calculated for all of the devices, and as shown in table 3.3, LT has the highest device value. Since LT fulfils the requirements of all the services in the user task, it is selected for execution of all the components for the user.

User 2 the user would like to use a device which has VLC video viewer installed on it (preference specified as *very important* for `VideoViewer=VLC`) and does not like a device that charges for internet access (specified by assigning a dislike value to `AccessCharges`). The user also specifies their preferences for OS and screen size as well as has trivial importance for the AZERTY type of keyboards. Based on eq. (3.4), DT has the highest device value, which is selected for execution of all services.

Note that if we change the value of `VideoViewer=VLC` from 1.0 to 0.9, the device with the highest value will be LT, based on new calculations. Similarly, if preference for `AccessCharges` is changed from -0.5 to 0.0, MM will get the highest value. This has been shown in the User 2 row of table 3.3.

User 3 in this case, FS has the highest device value. However, the service fitness values for Audio and Controller services on FS are *false*, so FS can neither execute the Audio service nor the Controller service. Since the Video and Audio services are declared to be coexisting, the Video service cannot be executed alone on FS either. Thus, the only service capable of execution on FS is the Photo service, for which it is selected first. For the rest of the services, the device with the second highest value is selected, i.e., the PDA. Again, PDA cannot execute Video service (and so Audio service), so it is selected for Controller service only. Next, the device with the third highest value is looked for. Thus, DT is selected for executing both Audio and Video services. With this, all the services are resolved and the process is terminated. With this selection, the user will use FS for browsing photos, DT for watching video along with the associated audio and PDA for controlling photo browsing and video, much like a remote control. As mentioned previously, here we assume that all these devices are able to inter-communicate using Wi-Fi and Bluetooth, etc.

Table 3.3: Devices values and final user preference values

User	PDA	MM	FS	LT	DT	Selected Devices
User 1	-0.60	0.80	-0.60	**1.00**	-0.80	LT
User 2	-0.89	-0.06	0.72	1.87	**1.91**	DT
User 3	1.19	-0.23	**1.50**	-0.36	1.03	FS, PDA, DT

3.6 A Qualitative Preference Model

When using quantitative preference alone, as in the previous section, we cannot express the conditional and relative importance statements as well as the assignment of multiple values to variables. These limitations can be overcome by proposing a qualitative approach combined with the quantitative one.

In this section, we use a graphical representation, CP-nets (Boutilier et al., 1999) and its extension TCP-nets (Brafman and Domshlak, 2002), that can be used for specifying qualitative preference relations in a relatively compact, intuitive, and structured manner using conditional *ceteris paribus* (all else being equal) preference statements. The main advantage of using TCP-nets to represent user preference statements is that it is a qualitative graphical representation and can reflect the conditional dependence and independence of preference statements. Also, such a representation is compact and natural in many circumstances.

Users specify their preferences qualitatively, in specific order and with different importance levels. Users may specify their preferences both as likings and dislikes as well as specify their constraints. However, users are not required to specify their preferences exhaustively: preferences can be provided partially or incompletely. These preferences are then translated into quantitative values using some heuristics. A preference tree is created containing device capabilities as nodes and having user preference values as weights on these capabilities. This results in a tree where each leaf node represents the aggregated value of a device and the highest weighted leaf node gives the device that is the most favourable for the given user preferences. The created preference tree is compact, consisting of only characteristics of devices currently available and which are needed to be evaluated against user preferences. The tree is pruned for any nodes that are in conflict with user preferences. We discuss that the results obtained using the preference tree method are better than using only additive utility methods or the qualitative methods.

3.6.1 CP-nets

A CP-net (for Conditional Preference network) (Boutilier et al., 1999) is a model for compactly representing conditional and qualitative preference relations. For representation, CP-nets use directed graphs consisting of set of features $V = \{X_1, \ldots, X_n\}$ represented by nodes, where each node X_i is associated with a finite domain $\mathcal{D}(X_i) = \{x_1^i, \ldots, x_n^i\}$. A feature X_i may specify a set of parent features $Pa(X_i)$ that can affect the values of X_i based on the values assigned to $Pa(X_i)$. The relation between a node X_i and its parent nodes $Pa(X_i)$ is denoted by the arcs that lead from $Pa(X_i)$ to X_i in the graph. Associated with each variable is its Conditional Preference Table (CPT), which expresses the preferences over the values taken by it. A preference between two outcomes

x_1 and x_2 can be specified by the $\succ$ relation, e.g., $x_1 \succ x_2$ specifies that x_1 is preferred over x_2.

The graphical representation of CP-net allows us to express dependency between the connected CPTs. This results in a dependency graph in which each node X_i has some $Pa(X_i)$ as its immediate predecessors and/or a set of nodes as its immediate successors. This also defines the notion of relative preference between the preference variables themselves: a CPT associated with a specific node has a higher priority than the CPTs of its offspring. Given this structural information, the user explicitly specifies his preference over the values of X_i for each complete assignment on values of $Pa(X_i)$. That is, given a particular value assignment to $Pa(X)$, the user should be able to determine a preference ordering for the values of X, *ceteris paribus*. By the term *ceteris paribus*, meaning all other things being equal, we imply that the preference over values of X is considered only on the values of $Pa(X)$, while all other attributes are considered being assigned fixed values. Once such preferences are specified for a number of attributes, the resulting graph takes the form of total or partial order over $\mathcal{D}(X)$ (Boutilier et al., 2004)(Boutilier et al., 1999). An example CP-net, described in (Boutilier et al., 2003), for three variables A, B, and C is shown in fig. 3.4(a). Each of these variables takes a binary value. For variable A, the user specifies that a is preferred to $\overline{a}$. The value assigned to B depends upon the value of A. So given a, the value b will be preferred to $\overline{b}$. But if A takes as value $\overline{a}$, then B will also be preferred to assume the value of $\overline{b}$ and not b. Similarly, the value assigned to C depends on the value of B.

CP-nets can be used both as an input tool and as an internal representation of preference statements provided by the user directly or by means of an appropriate interface. The semantics of CP-nets do not force the graph to be essentially acyclic. However, the preference statements contained in the acyclic graph are ensured to be consistent, i.e., there is a total pre-order that satisfies all the preference statements (Boutilier et al., 1999). Cyclic CP-net graphs are not necessarily satisfiable and require more complicated procedures and, thus, are beyond the contribution of this thesis.

Two fundamental types of queries that can answer the preferences of the decision maker and can be used with CP-nets are (Boutilier et al., 2003):

1. *Outcome optimization* determining the best outcome consistent with certain evidence (or one of the outcomes, if the best outcome is not unique).
2. *Outcome comparison* preferential comparison between a pair of outcomes.

3.6.2 TCP-nets

The TCP-net (for Tradeoff-enhanced CP-nets) (Brafman and Domshlak, 2002) model is an extension of CP-nets that encodes conditional relative importance statements, as well as the conditional preferences available in CP-nets. Like CP-nets, the nodes of a TCP-net correspond to the problem variables V. However, unlike CP-nets, TCP-nets have three types of edges. The first type of (directed) edge is the same as used in CP-nets: for capturing preferential dependence, i.e., an edge from node X to Y implies that the user has different preferences over Y values given different values of X. The second edge type captures relative importance relations. The existence of such an edge from node X to node Y implies that X is more important than Y. The third edge type captures conditional constraint relations: such an edge between nodes X and Y exists if there for

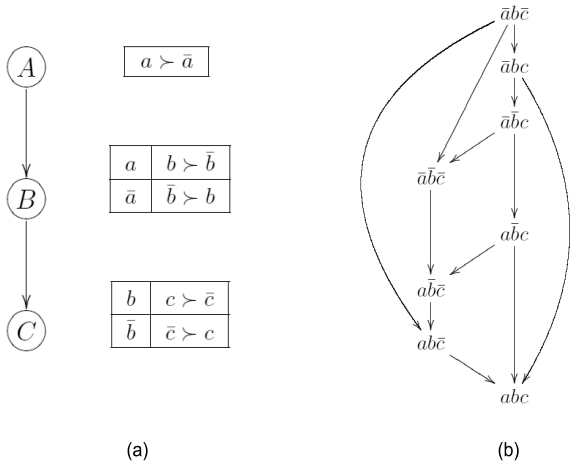

Figure 3.4: An example CP-net

some value of variable X, variable Y must also hold with a particular value. These will be explained with the help of an example in a later Section.

In TCP-nets, as in CP-nets, each node X is annotated with a conditional preference table (CPT). But in addition, in TCP-nets, each undirected edge is annotated with a conditional importance table (CIT). The CIT associated with such an edge (X, Y) describes the relative importance of X and Y given the value of the conditioning variables.

3.6.3 Interpretation of CP-nets and TCP-nets

To evaluate a CP-net or a TCP-net, the graph is traversed in topological order resulting in an induced graph whose nodes and arcs determine the various possible combinations in which the user preferences can be ordered. Figure 3.4(b) shows the induced graph for the CP-net of fig. 3.4(a). As can be seen, the induced graph specifies the worst and best preferences, respectively at the top and bottom of the graph, but it does not show any ranking between several intermediate solutions, e.g., we cannot compare the solutions $\overline{a}bc$ and $a\overline{b}c$.

Based on the concepts of CP-nets and TCP-nets, we now describe how they can be extended for user preference modelling in our case.

3.6.4 Proposed Extensions to TCP-nets

CP-nets (and so TCP-nets) use preference ordering relation $\succ$ only to order values taken by the features. This becomes restrictive as it does not imply *how much important* is the value of a feature with respect to another. For a given feature, more than one value may apply and the user may have different preferences for each one of them. In order to capture the notion of *more important* or *less important* or even *dislike*, we propose to use a preference function $feature\ value \mapsto importance\ level$. This function is used to assign different levels of importance to different values of a feature. Users specify their preferences for all of the features in which they are interested.

While users specify their preferences qualitatively, we need to map them to specific values so that they can be measured quantitatively. Table 3.1 shows various levels of

importance — that can be used by the user to specify his likings and dislikes — and their respective mappings. A user's preference is mapped to a specific real number between the range $(-1.0, 1.0)$. The reason for choosing such values is two folds: first, specifying a range between $(-1.0, 1.0)$ helps in defining a normalized function with respect to user preferences; second, by including negative values, it is possible to eliminate devices with unwanted capabilities, as will be shown later. Using such values, 1.0 represents a *very important* capability, -1.0 represents user's dislike, so much to *avoid* using a device with such a capability, and 0.0 representing a *don't care* condition, i.e., when user specifies $x \mapsto dontcare$, the availability or unavailability of x is not important for the user.

The reason for choosing unequal spaces between various preference levels is to spread the weight space to make the difference more expressive, e.g., the distance between *somewhat like* and *like* is only 0.2, but between *important* and *very important*, it is 0.3, i.e., the difference is more emphasized in the latter case.

With these notions, we can specify how much critical is a feature for the user. Using these importance levels, we can use our extended TCP-nets in several ways such as:

- *Specifying likings*: for a feature X, the user may specify $x_1 \mapsto important$ and $x_2 \mapsto like$, i.e., the assignment of x_1 to X is important than the assignment of x_2. For example, the user may say that he prefers VLC media player to QuickTime player using these notations. How much does he prefer one over the other depends on the preference levels assigned to each one.
- *Specifying dislikes*: for a feature X, the user may specify $x \mapsto dislike$, e.g., the user may say that he does not like to use a device that connect to the Internet using GPRS.
- *Specifying indifference*: for a feature $X = \{x_1, \ldots, x_n\}$, the user may specify his preferences only for x_1 and not for other values. In such a case, the unspecified preferences are considered as *don't care*.
- *Ordering preferences*: for two features X and Y, the user may specify that $X \succ Y$, i.e., he prefers to select the value of feature X before feature Y. For example, the user may specify that it is more important to have a device with longer remaining battery rather than a device having larger screen.
- *Conditioning preferences*: if for some $z \in \mathcal{D}(Z)$, we have that $X \succ Y$ or $Y \succ X$ given z, the relative importance of X and Y is conditioned on the value taken by Z. For example, the user may specify that if the battery is full (or unlimited), he would prefer a larger screen, otherwise he would prefer a better operating system, i.e., the one which provides better resource management for less power consumption.
- *Specifying resources qualitatively*: since resources are measured quantitatively, we need a mechanism for qualitative resource specification. A user can specify his preferences for resources using notions such as *maximum*, *minimum*, and *closest match*, etc. Thus, $X \mapsto maximum$ specifies that the user prefers the device which provides the maximum value for resource X.

Table 3.4 shows one way of how the preferences for the variables in fig. 3.4(a) can be specified with our extended approach. As it can be observed, instead of specifying preferences *relatively* to each other as in CP-nets, using our extended CP-nets, users can specify resources with the *degree of importance*, independent of each other. For example, for variable A, user has specified only preferences, but for variables B and C three preferences have been specified.

Table 3.4: Extended CPT for variables in fig. 3.4

Variable	Preference value
A	$a_1 \mapsto$ Very important $a_2 \mapsto$ Like
B	$b_1 \mapsto$ Dislike $b_2 \mapsto$ Important $b_3 \mapsto$ Somewhat Like
C	$A = a_1 \Rightarrow c_1 \mapsto$ Very important $B = b_1 \Rightarrow c_2 \mapsto$ Dislike $B = b_2 \Rightarrow c_2 \mapsto$ Important

One important thing to consider here that unlike traditional CP-nets, where variables are necessary dependent on each other, we do not assume that it must be like this in our case. That is, variables can be specified indepdently of each other and their order in the CP-net determines their importance in terms of their specification order by the user. The use of the extended CPT will be explained in Section 3.6.7 with the help of an example scenario.

3.6.5 Resource Modelling

Resources of a device are dynamic and may vary over time. So unlike capabilities, whose values is fixed, they need to be evaluated for their current status. Also, since resources are measured quantitatively, we need a mechanism for qualitative resource specification, because users specify their preferences qualitatively. Resources can be modelled using aggregate utility functions.

Aggregate Utility Functions

Table 3.1 provides a mapping from qualitative preferences to quantitative values, however, it does not show how the concepts such as *maximum* and *minimum* should be mapped to quantitative values. We propose the use of aggregate utility functions. These functions take as input a single resource feature, e.g., battery or screen size, for all the devices, and return the devices ranked according to the input feature. For example, consider a vector

$$X = \{X_i, 1 \leq i \leq n),$$

where X represents a particular feature and n is the number of devices, then we can have a ranking function (Zeng et al., 2004):

$$RankMin(X) = \begin{cases} \frac{X_{max} - X_i}{X_{max} - X_{min}} & \text{if } X_{max} - X_{min} \neq 0 \\ 1 & \text{if } X_{max} - X_{min} = 0. \end{cases}$$

This function can be used to specify a criterion in which the higher the value, the lower the quality of the feature, e.g. , latency, delay, packet loss, etc. We also define a function to specify a criterion in which the higher the value, the higher the quality of the feature, such as for remaining battery, CPU availability, and bandwidth, etc.:

$$RankMax(X) = \begin{cases} \frac{X_i - X_{min}}{X_{max} - X_{min}} & \text{if } X_{max} - X_{min} \neq 0 \\ 1 & \text{if } X_{max} - X_{min} = 0. \end{cases}$$

Similarly, for certain attributes, a user may specify an input value such that any provided value that is closer to the input value will be better rather than a value which is higher or lower; e.g., an input screen resolution, R_i, can be matched with a screen resolution, R_d, provided by a device using a similarity function as presented in eq. (3.2). We redefine it here as:

$$Optimum(R_i, R_d) = \frac{\frac{Min(w_i,w_d)}{Max(w_i,w_d)} + \frac{Min(h_i,h_d)}{Max(h_i,h_d)}}{2}$$

where w and h represent the width and height component of the screen resolution, respectively. For all these aggregate utility functions described above, the ranking is in the normalized range $(0.0, 1.0)$. A more promising device will have a higher value using either of these functions.

As an example, assume that there are five devices and their screen resolutions are recorded as:

$$X = \{480 \times 640, 1280 \times 800, 1920 \times 1200, 1280 \times 800, 1600 \times 1200\}$$

and the user specifies a desired screen resolution of 1600×1200. Then the *Optimum* function when applied to values in X gives us the ranked set:

$$Optimum(X) = \{0.42, 0.74, 0.92, 0.74, 1.0\}$$

while the function *RankMax* results in:

$$RankMax(X) = \{0.0, 0.93, 1.0, 0.93, 0.89\}.$$

3.6.6 The Preference Tree

In order to reason about TCP-nets, they are transformed into topological graphs, which result in ranked preferences (as was shown in fig. 3.4(b)). However, this requires exploring all the possible combinations, which is not useful when the decision maker is interested only in a subset of variables taking possibly few values from the domain set. Thus, we propose an alternative mechanism, which is much efficient for incomplete preference specification.

We use a tree structure, instead of graph, for exploring the preferences contained in a TCP-net and the extended CPTs. Instead of searching the tree for all possible values of the variables, we use only the currently available values for the variables. The tree consists of nodes from the TCP-net and each node represents only one variable. The arc from a parent node X to a child node Y represents the preference value assigned by the user as they are conditioned in the TCP-net and extended CPT. Thus, since each arc represents an *importance level*, it means that for each parent node, there will be as many arcs — and corresponding children nodes — as there are user preferences for the

Algorithm 2 $PreferenceTree(X, P, L, R)$

Input: Acyclic TCP-net X, User Preferences P, Devices list L, Results R
Output: R is a set of weighted devices according to user preferences

```
1:  R = ∅ {i.e. initially all devices have weight 0}
2:  Let n := size(L)
3:  Let x_1 ... x_k ∈ D(X)
4:  for i := 1 to k do
5:    if there is a user preference for X then
6:      Let k := pref(x_i) be the calculated preference value
7:    else
8:      k := DontCare {i.e. k := 0}
9:    end if
10:   for j := 1 to n do
11:     if k is consistent with properties(L_j) then
12:       Set R_j := R_j + k
13:       if X is not the last variable in TCP-net then
14:         Get variable Y that succeeds X in the TCP-net
15:         Create node Y and add new edge (k, Y) to X
16:         R := PreferenceTree(Y, P, L, R)
17:       end if
18:     else
19:       {the constraint is inconsistent, ignore this device}
20:       for all L_i where pref(x_i) = k set R_i := ∅
21:     end if
22:   end for
23: end for
24: return R
```

node variable. For example, for the variable B in table 3.4, the corresponding node B in the tree will contain three children connected to it by arc labelled as b_1, b_2, and b_3, respectively. This preference value is then translated into its equivalent weight based on the mappings defined in table 3.1.

Algorithm 2 shows the `PreferenceTree` procedure for creating a preference tree. This is a minimalist algorithm that recursively iterates through the nodes in the TCP-net, creating and searching the preference tree at the same time. Initially, the input of the algorithm consists of the root node of TCP-net, the list of user preferences in the CPT, the list of devices and a data structure containing the results, which is empty in the beginning. For each variable, first we determine the available values from the set of all devices (line 3). Next, we determine if there are any user preferences from the CPT corresponding to each of these values. The $pref()$ function in line 6 calculates the weighted preference value according to table 3.1. If there are no user preferences for a particular value of the variable, then we consider a *Don't care* case. In lines $10 - 22$, all devices are iterated for their capabilities and if user preferences are consistent with a device's properties (i.e., the device's capabilities and its resources as described previously) the device's value is increased (or decreased) by the preference value. If the preference

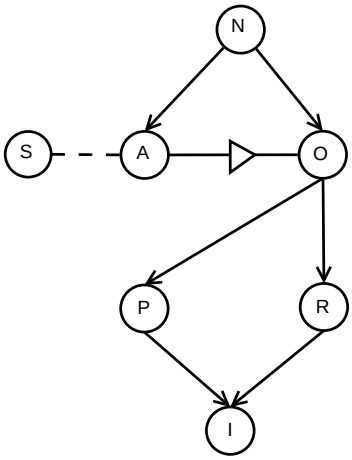

Figure 3.5: The TCP-net for my preferences

value is not consistent with the device's properties, the device is not considered further in the subsequent iterations. Based on the preference value in CPT, the next node is selected for the current preference value and the process is repeated until all nodes in the input TCP-net are evaluated. The algorithm returns the ranked list of devices according to preferences. The application of the algorithm will be more evident after the reader is acquainted with the next section.

3.6.7 Example Scenario

Suppose that I want to execute a multimedia task, whose description is available on my smart-phone. The task allows me to browse photos, watch video, and to control the playback of video and photos. The preferences for my task are specified as following and are summarised in table 3.5:

1. Since the photos and videos I will be watching are available over the Web, the device I select must be able to connect to the Internet. This criterion should be absolutely met; otherwise the device is not useful for me. Thus, I specify $NetworkAccess = \{Yes \mapsto Very\ Important, No \mapsto Avoid\}$.
2. I would not like to pay for internet connectivity, rather I would like to use a device that offers free internet access. So, I specify $AccessCharges = \{Yes \mapsto dislike, No \mapsto like\}$.
3. I work on Linux all the time and I like it very much, so I specify $Linux \mapsto important$; however, if Linux is not available, I would prefer Windows to the rest of the operating systems. I specify $Windows \mapsto Somewhat\ like$ to distinguish it from *don't care* cases, where all else will be equal.
4. When working routinely, I use a screen resolution of 1280×800 on my Linux system, so I specify $OSName = Linux \Rightarrow ScreenResolution \mapsto ClosestMatch(1280 \times 800)$. However, if a Windows machine is selected, I would prefer a higher resolution, because I think Windows is good at managing screen layout at higher resolutions, so I assign $Windows \Rightarrow ScreenResolution \mapsto Maximum()$.

Table 3.5: My preferences for device capabilities

Variable	Preference value
NetworkAccess	Yes $\mapsto$ Very important No $\mapsto$ Avoid
AccessCharges	Yes $\mapsto$ Dislike No $\mapsto$ Like
OSName	Linux $\mapsto$ Important Windows $\mapsto$ Somewhat like
ScreenResolution	OSName$\mapsto$Linux$\Rightarrow$ ScreenResolution $\mapsto$ ClosestMatch(1280x800) OSName$\mapsto$Windows$\Rightarrow$ ScreenResolution $\mapsto$ Maximum()
VideoPlayer	OSName$\mapsto$Linux$\Rightarrow$ VLC $\mapsto$ Important OSName$\mapsto$Windows$\Rightarrow$ MediaPlayer $\mapsto$ Important
InputType	TouchScreen $\mapsto$ Important Voice-command $\mapsto$ Slightly Important
Constraints	AccessCharges $\succ$ OSName AccessCharges=Yes $\Rightarrow$ SecuritySupport $\mapsto$ Very Important

5. I am also selective about the media player: on my Linux system, I am impressed with the freely available VLC media player; however, if I had to use Windows, I would prefer its built-in Windows Media Player (WMP). So I assign $OSName = \{Linux \Rightarrow VideoPlayer \mapsto VLC, Windows \Rightarrow VideoPlayer \mapsto WMP\}$.
6. Finally, it would be nice if the selected device has touch-screen; it will be good for esthetics, but if that is not the case, I would still prefer giving voice-commands rather than dealing with keyboard and mouse. So $InputType = \{TouchScreen \mapsto like, VoiceCommand \mapsto somewhat\ like\}$.
7. I also specify two additional conditions: a) if two devices have the same final weights, I would prefer a device that does not charge for Internet access, even if it has Windows, rather than a device that uses Linux but charges for Internet access, i.e., I prefer a better value for *AccessCharges* as compared to the value for *OSName*; b) in case there is no better device providing free internet access, I would pay for the Internet access only if it is secure, i.e., $AccessCharges = Yes \Rightarrow SecuritySupport \mapsto Very\ important$.

Figure 3.5 shows the TCP-net obtained based on my preferences shown in table 3.5. For convenience of representation, the various variables in the table are shortened to one letter symbols in the graph. Note the three different types of edges in the graph: the constraint between *AccessCharges* and *SecuritySupport* is represented by a dashed line, the *relative importance* by an edge with a triangular symbol, and the arc represents the order in which the preferences are to be satisfied.

After my preference specification, I allow my smart-phone to select the best device that matches my preferences. In the pervasive environment, different devices are available

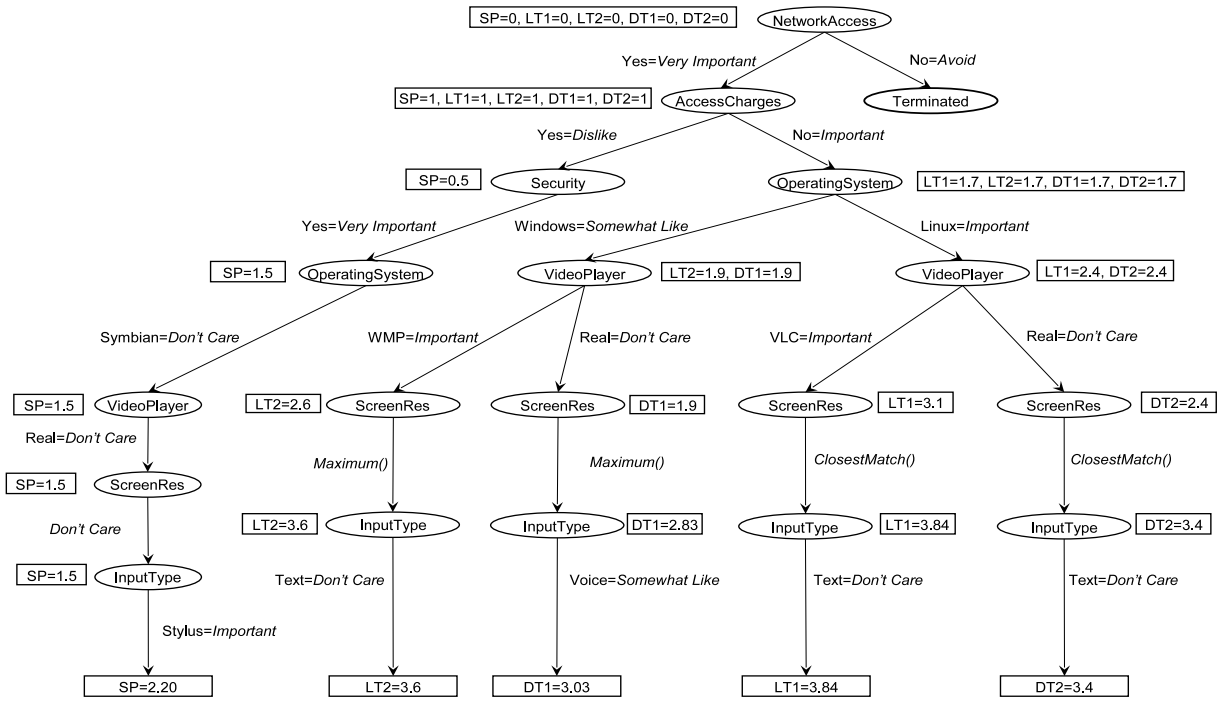

Figure 3.6: The Preference Tree for my preferences for device capabilities

and my smart-phone starts interrogating all devices for their characteristics. Of all these devices, let us assume that only five have Internet access. These are: two laptops namely LT1 and LT2, two desktops, DT1 and DT2, and my smart-phone. The capabilities of each device are shown in table 5.2.

In the matter of a few seconds, my smart-phone compares my preferences with these devices by creating the preference tree for device ranking and comparison as shown in fig. 3.6. The tree is created for the TCP-net of fig. 3.5 using the procedure described in Algorithm 2. The first node of the tree is the $NetworkAccess$ feature after which the $AccessCharges$ feature is evaluated because it is more imporant than the $OperatingSystem$ feature according to the TCP-net. Note that the comparison is made only between those devices which have Internet access. All other devices were avoided for selection in the very beginning due to user preference $NetworkAccess = No \mapsto Avoid$ of table 3.5. If $AccessCharges = Yes$ for a device, then the $Security$ feature must evaluate to Yes as specified in the constraint by the user. Alongside each node, we have also specified the value of each device as it is evaluated according to its capabilities. At each next node, the device value is updated according to a feature's value and its preference weight assigned by the user. At the bottom of the tree, the cumulated device values are shown in the rectangles. As shown, LT1 has the highest device value and, hence, is selected for the user.

3.7 Graph-based User Task Composition

So far we have described user preferences and device capabilities; in this section we will explain how they can be used for realisation of the user task.

As the technology is advancing rapidly, more and more devices are emerging. As such network technologies are also evolving and we will witness new wireless communication protocols like Wi-Fi-n (IEEE, 2009) and Ultra-Wideband (UWB) ((ECMA), 2005) in the near future. Considering pervasive environments such as large offices, airports and railway stations, etc., there will be hundreds of user devices able to communicate in a variety of network-level protocols. Thus, it would not be possible to manage these devices on a large scale. Although the problem may be manageable now, but looking to the future, it will become a more significant issue as a wider variety of wireless devices are introduced and additional wireless technologies are embedded in everyday items.

Such issues of scale will be paramount for any application that seeks to rapidly connect devices together in an environment where many other wireless devices exist. While the approach described in this work seeks to enable compositions of components distributed across devices in such pervasive environments, if the approach cannot be scaled beyond a few number of devices, its usefulness will be limited. For example, in environments such as a crowded cafe or office, there could easily be tens or even hundreds of other electronic devices nearby, providing a bewildering array of target devices for users to select from when forming a composition. Systems operating in such environments all face the same challenge of scale seen during conventional wireless discovery, and similarly needs a suitable solution: the problem is not necessarily the limited number of devices that the users wish to combine together, rather, it is the numerous other nearby devices that are not intended to be part of the collection (Want et al., 2008).

Thus, our approach will be to eliminate all those devices, which are not potentially useful during a particular situation. This decision has to be made dynamically, as devices that may be useful for realisation of one user task may not be useful for realisation of a subsequent user task, and vice-versa.

In the previous sections, we explained how devices can be selected for user task composition based on user preferences and service requirements (Section 3.6 and 3.5). Based on similar concepts, we can also eliminate devices that are not useful by considering user preferences and service requirements. That is, any device that does not fulfil service requirements should be eliminated from user task composition. Similarly, when ranking devices according to user preferences, any device whose ranking falls below a specific minimum value should not be considered suitable for the user (according to their preferences) and hence should be eliminated.

A user task may consist of various components, and different components may be executed on different devices, as demonstrated in the scenarios of Sections 3.5.4 and 3.6.7. As we assume that different devices may use different types of network protocols for communication, when two devices are selected for different components required by the user task, we must make sure that if these components need to communicate with each other, the devices must also be able to communicate with each other. In other words, when selecting components on devices, the network level compatibility between the devices must also be considered, so that components can be selected only on those devices, which can communicate with each other using a common network protocol.

In order to solve both of these issues (device elimination and protocol heterogeneity), we use a graph-theoretic approach when selecting components and devices. We model the user task as a graph of services for which we will need to find the concrete components. As these components are distributed across devices in the pervasive environments, we also need a graph representation of the devices and their interconnection. For this purpose, we create an aggregate graph consisting of all the components in the environment, which match the services in the task and then provide a one-to-one mapping of the user task graph onto the aggregate graph for selection of concrete components. When creating aggregate graph, user preferences and service requirements are used for ranking and elimination of devices.

This approach is discussed in detail in the next subsection. First we describe the modeling of the user task, and the underlying network consisting of the devices interconnected with one another using a variety of protocols, and then in the next section, we explain how these two graphs can be matched to obtain the concrete user task.

3.7.1 User Task Modelling

To carry out a task, the user provides task specifications. A user's task T is modelled as an attributed graph $G_T = \{S, I, U, R_T\}$ where $S = \{S_1, S_2, \ldots, S_n\}$ is a set of abstract services represented by the nodes in the graph, $I \subseteq S \times S$ is the set of edges between nodes and each $I_{i,j} \in I$ represents the communication or dependency interfaces between the services S_i and S_j. U is the set of user preferences for services in the task. R_T is the set of resources required by all services in the graph G_T, i.e., $R_T = \bigcup R_{S_i} \in S_i$. R_{S_i} denotes the abstract level resources required by the service S_i and is represented by the attributes on the nodes of the graph. The modelling of R_s was provided in Section 3.3 in the form of extended CC/PP profile.

G_T is an undirected, connected graph representing a two-way, request/reply, communication between the services. Each service S_i is deployed independently. The communication between services takes place through its exposed interfaces only, i.e. the services are loosely coupled. We assume that due to synchronous request/reply paradigm, services are able to maintain the temporal relationship, as specified by their dependencies and the problem of service sequencing and interleaving is taken care by the underlying infrastructure.

3.7.2 Network Modelling

The network model N is represented by a graph $G_N = \{D, L\}$ where $D = \{D1, D2, \ldots, D_m\}$ is the set of devices available at the time of user task composition and $L \subseteq D \times D$ is the set of links such that each $L_{i,j}$ denotes the connection between devices D_i and D_j

A device is modelled by $D = \{C, R_D, G_C\}$ where $C = \{C_1, C_2, \ldots, C_n\}$ is the set of components installed on the device, R_D is a set of properties characterizing the capabilities or the resource available on the device and $G_C = \{G_{C1}, G_{C2}, \ldots, G_{Cn}\}$ is a set of graphs, where each G_{Ci} represents concrete compositions of components, already deployed on the device. Each G_{Ci} represents either a single component in C exported as a network service, or a composition of various components $C_u, C_v, ...C_z \in C$ that is exported as a network service. Thus the implementation of an abstract service will be provided either by a single component C_i or a set of components represented by G_i.

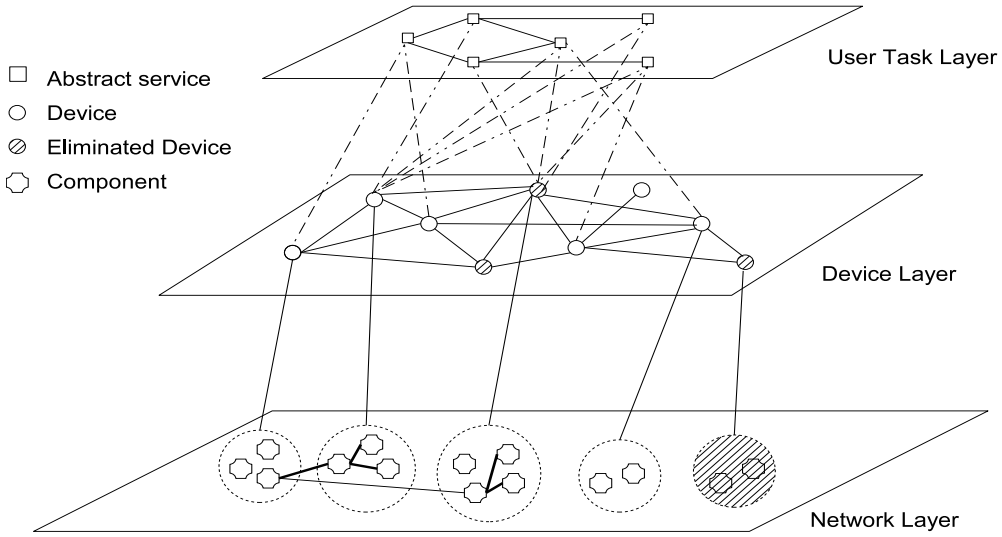

Figure 3.7: Task resolution at three different layers

The next section describes how an abstract service in the user task can be resolved into components using graph-based matching algorithms.

3.8 Task Resolution

An abstract user task is resolved into network services only at the time of execution of the task. This requires matching of abstract services with the network services, which is done at two levels: functional interfaces and abstract resource requirements. Both of them can be described using some appropriate means, e.g. an XML-based description such as SCA[7] in (Mukhtar et al., 2008a), or OWL-S[8] in (Ben Mokhtar et al., 2007). Furthermore, we assume that a Task Composer is present on the device where the user task is initiated. The Task Composer uses a three phase protocol to resolve a task into network components.

3.8.1 Task Composition Using Three Phase Protocol

In this section, we explain how a user task is resolved into components based on user preferences. Figure 3.7 shows the three layers of service composition used by our task composition algorithm. The Task Composer first determines the abstract services S, their interfaces I, and abstract service requirements R_T of all the services, from the abstract user task T. These are used to create a graph representation G_T of the user task. The *User Task layer* in fig. 3.7 shows such a graph. To create the network graph G_N, the Task Composer forwards G_T to the Service Discovery system and uses the three phase task composition protocol as following.

7. SCA: Service Component Architecture. http://www.osoa.org/
8. OWL-S: Semantic Markup for Web Service. http://www.w3.org/Submission/OWL-S/

Algorithm 3 ComposeTask(G_T, G_N)

1: $G_T = \{S, I, U, R_T\}$, $G_A = \{Q, Y, R\}$
2: sort G_T and G_A by node label
3: **while** $S \neq \varnothing$ and $C \neq \varnothing$ **do**
4: let s_1 be the first element of S
5: let c_1 be the first element of C
6: **if** $\alpha_1(s_1) = \alpha_2(c_1)$ **then**
7: $\mu(s_1) := c_1$
8: $S := S - \{s_1\}$
9: **end if**
10: $C := C - \{c_1\}$
11: **end while**
12: sort I and Y by node label
13: **while** $I \neq \varnothing$ and $Y \neq \varnothing$ **do**
14: let (s_1, s_2) and (c_1, c_2) be the first element of I and Y
15: **if** $\mu(s_1) = c_1$ and $\mu(s_2) = c_2$ and
16: $\beta_1(s_1, s_2) = \beta_2(c_1, c_2)$ **then**
17: $I := I - (s_1, s_2)$
18: **end if**
19: $Y := Y - (c_1, c_2)$
20: **end while**
21: $match := (S = \varnothing$ and $I = \varnothing)$
22: **return** $(\mu, match)$

Phase I - Device Elimination

The Service Discovery system interrogates the environment for available devices and queries them for their capabilities. Each device sends its capabilities R_D back to the requesting device. The Task Composer needs to create a network graph consisting of all the devices and their interconnection. The connection between the devices will be based on the protocol compatibility, i.e., two devices in the graph will be connected if and only if they can communicate using same protocol.

For each device D_i, the Task Composer compares R_{D_i} with the abstract resource requirements R_T of the task. Any device which does not meet at least one of the required capabilities or user preferences is eliminated from the graph (shown by the shaded circles in the *Device layer* of fig. 3.7). This results in a reduced graph to be matched.

Phase II - Graph Aggregation

The Service Discovery sends the abstract services in G_T to the selected devices. Each device sends back $G'_C \subseteq G_C$ such that each $G'_{Ci} \in G'_C$ matches with at least one of the abstract services S in G_T. The Task Composer then combines the partial network service graphs G'_c, obtained from all devices, into a single aggregate graph $G_A = \{Q, Y, R\}$, where Q is the set of components obtained from all the partial graphs and represents the nodes in the aggregate graph, $R = \bigcup R_{D_i}$ and each R_{D_i} is associated with the component $C_j \in D_i, 1 \leq j \leq C_n$. As we will see later, associating device capabilities to

its components will be useful when matching services with components. $Y = \{F \bigcup P\}$, where F is the set of functional interfaces between components and $p_i \in P$ is the network level protocol used between two components when they use functional interface $f_i \in F$. The *Network Layer* in fig. 3.7 shows an example component graph obtained.

Phase III - Matching G_T and G_A

The Task Composer can now match G_T with G_A to determine the final set of network services and their composition, as required by the user task. The goal is to find a subgraph H of G_A such that there is a one-to-one correspondence between H and G_T. The final graph obtained, H, represents the concrete user task graph that is sent to an execution engine, which executes the task by invoking the appropriate network services.

3.8.2 Graph to sub-graph matching

The graph (G_A) to sub-graph (G_T) matching is known as *matching sub-graph isomorphism* in graph theory. The problem of sub-graph isomorphism detection is known to be NP-complete (Garey and Johnson, 1979). However, by imposing certain conditions on the properties of the graphs, it is possible to derive algorithms that have a polynomially bounded complexity. One efficient algorithm for su-bgraph isomorphism, proposed recently by Valiente (Valiente, 2007) assumes that each node, in the graphs to be matched, has unique labels. Thus matching is done by their labels. A modified version of the algorithm, for our case, is shown in Algorithm 3. The algorithm works as follows.

The task graph G_T and the network graph G_A are represented as shown previously. The unique *node labels* are used to identify each service and component in G_T and G_A, respectively. In G_T, each label consists of the service interface, while in G_A it consists of a combination of component interface, devices' capabilities and the network protocol associated with the interface. This uniquely identifies each component in G_A. The α and β are node labelling and edge labelling functions in the graphs, while μ is an assignment function, which assigns a component to a service when they match. α matches functional interfaces and abstract task requirements, while β ensures protocol compatibility between different components.

3.9 Discussion

There are a few things that need special attention and were not considered in the above discussion. The are explain in the following.

3.9.1 Multi-Protocol Network

We have been considering the heterogeneity of the underlying network in terms of disparity of the communication protocols. Network heterogeneity leads to many independent networks being available to users at a location and sophisticated mechanisms are required to process the availability, location, and matching of devices and services.

One particular research work in this direction is the MSDA middleware (Raverdy et al., 2006b), which enables multi-network, multi-protocol service discovery and access in pervasive computing environments. For dealing with multiple protocols arising from

availability of different networks in the pervasive environment, MSDA dynamically composes nearby networks through application-level routing components provided on devices having multiple network interfaces, which enables the dissemination of service location and access requests in the whole environment.

Multi-protocol interoperability decomposes into service discovery protocol interoperability and service access protocol interoperability. Service access protocol interoperability is performed by translating service access messages from one protocol to another. Service discovery protocol interoperability decomposes in two parts, i.e., the translation between protocol messages and the translation of heterogeneous service advertisements into a common XML format (the MSDA service description format) (Ben Mokhtar, 2007).

There also also some low-level solutions for achieving interoperability at network protocol level. For example, the authors in (Greenwood et al., 2008) (Greenwood and Ghizzioli, 2008) have used hybrid adaptors for conversion between 3G and Wi-Fi protocols or between Infrared and Bluetooth protocols. Works in the IST Palcom project [9] have led to the creation of RASCAL (Resilience and Adaptivity System for Connectivity over Ad-hoc Links) prototype (Greenwood and Ghizzioli, 2008). The RASCAL prototype is a software component that ensures the continued operation of (PalCom) services communicating in disruptive environments, to such degrees as are possible within the operating environment. For communication in a hybrid environment, a (PalCom) device's routing manager can select either a technology-specific media manager (e.g., UDP, Bluetooth, etc.) or a RASCAL media manager. When using RASCAL, all technology-specific media managers are used concurrently to enable flexible and resilient communication.

3.9.2 Distributed Task Composition

We explained above how to consider the heterogeneity of protocols, but we have not mentioned how the devices will interact with each other for realisation of the user task, if they do not all understand a common protocol. For example, if the user device understands only one protocol, for example, then how can it discover and communicate with devices which speak different other protocols. For two devices to communicate, there must be at least one common protocol [10]. Now here is a strong assumption: devices propagate the service discovery information among one another using all of their network interfaces. Thus, the availability of a Bluetooth only enabled device will also be known to a Wi-Fi only device, and vice-versa, if there is some third device that can speak both of these languages. This type of advertisement is not supported directly by the RASCAL and MSDA approaches described above; rather they need sophisticated service discovery protocols. Issues like these have also been considered previously; for example, (Chakraborty et al., 2004)(Chakraborty et al., 2005) have proposed distributed service composition mechanism, that can work very well combined with approaches like RASCAL and MSDA.

3.10 Concluding Remarks

This chapter mainly dealt with the modelling and algorithmic aspects of involved in our approach for user task composition. This involved modelling device capabilities, user

9. http://www.ist-palcom.org/
10. If there is more than one common protocol, then any of them can be selected arbitrarily

preferences and the user task. An important aspect of our work is the matching of user task with the available components in the environment. To do that, we modelled the user task and concrete components distributed on devices as graphs. Our aim was to map the user task graph on to the component graph to obtain a concrete composition graph. Several issues involved in the creation of the aggregate graph, such as network protocol heterogeneity and distributed service composition were also discussed.

This chapter provided a foundation for the next chapter, where we will define a unified approach for user task composition considering device capabilities, user preferences, and the modelling and algorithmic aspects related to the user task matching.

Chapter 4

Middleware for *Ad hoc* User Task Composition

4.1 Introduction

In the previous chapter, we explained how devices in pervasive environment can be characterized based on their capabilities and how users can specify their preferences for these devices. We showed, with the help of two example scenarios, how user preferences can lead to selection of one or more devices in the pervasive environment. We also discussed briefly how a user task, consisting of several services, can be mapped on to devices in the pervasive environment. However, we did not define exactly, what a user task was and how it was defined.

This chapter is dedicated to the definition and composition of a user task. First, in the remaining of this section we explain some concepts related the definition a user task, the description of the problem followed by a description of our approach for achieving the solution in the next section. In Section 4.3 we explain the Service Component Architecture (SCA) and in Section 4.4 we describe how SCA can be used for user task description. Based on the identified limitations of SCA, we propose a few extensions to SCA in Section 4.5. Then we explain how SCA description can also be used for describing devices and user preferences in Section 4.6.

Finally, in Section 4.7 we introduce our middleware called MATCH-UP, which stands for *Middleware for Ad hoc user Task Composition in Heterogeneous environments considering User Preferences*. We also described its architecture and functionality in the same section. In Section 4.8 we explain the prototype implementation of MATCH-UP and in Section 4.10, we give some initial evaluation results of the implementation.

Next, we will explain once again, for the purpose of clarity, what do we mean by *ad hoc* user tasks, and what are some of the assumptions that we consider in our approach for *ad hoc* user task composition. The unified solution to all this issues will be presented in the following sections.

4.1.1 *Ad hoc* User Tasks

As the research in pervasive and ubiquitous computing is evolving, researchers have presented models for representing *daily user tasks* as composite applications. Such models

capture the needs of the user in terms of required services in the environment, their interconnections, preferred characteristics, and levels of quality of service. To address environment diversity, user tasks are expressed in terms of abstract services, such as text editing, video playing, printing, etc. (Sousa and Garlan, 2002). The user task is executed by automatically searching, setting up and maintaining service configurations that best meet the user's needs. Thus, while the user task is defined in terms of *what* needs to be done, it does not specify *how* it could be done.

Different researches related to addressing the problem of user tasks consider mostly the latter aspect of the problem, i.e., given a user task, how to map it to available components in the environment. Due to the dynamic behaviour of the task, a user task is also called as *ad hoc* user task (Ben Mokhtar et al., 2007) and the process of resolving a user task into components, dynamically, is known as *ad hoc user task composition*. The composition process is initiated when the user *instantiate*s the task, i.e., when the user invokes the task for execution.

4.1.2 Problem Description and Assumptions

We consider a user task as an abstract description of services on the user's device. The description is independent of the particular concrete components and devices that will be used for realizing the user task. More specifically, the task consists of only description of the services required by the task (in terms of service interfaces and references) but it does not contain any implementation specific information. The description of a user task can be developed by an application architect, a developer or even an expert user. Additionally, the designer of the user task also specifies certain parameters that can be used to guide the selection of devices which can provide the required components. We restrict such parameters to only device characteristics, as discussed in the previous chapter.

A service in the user task can be implemented by one component or a composition of arbitrary number of components. These components can be implemented in variety of technologies and can communicate with one other using a variety of protocols. The resolution of services into components (or the realization of the user task) is carried out dynamically based on the properties of available services, devices, network, and user preferences. Thus, the same user task can be instantiated differently in various environments depending upon the capabilities of devices and user preferences. Even if the user task is reused in the same environment, having the same set of devices, the selected components and devices may be differently, if the user specifies some of his preferences differently.

The distinguishing features of our ad hoc user task composition procedure are: 1) it is influenced by user preferences, 2) it considers service execution requirements for device capabilities when selecting components, and 3) it considers heterogeneity of communication protocols. These are described briefly in the following subsection.

4.2 Our Approach

In order to address these issues, we propose an approach for user task composition, which unifies our modeling aspects that we described in Chapter 3, as following: first of all, we assume device capabilities are modelled as discussed in the previous chapter. The users can then provide their preferences for devices and components based on the same

model. This was also explained in the previous chapter. Next, the services in the user task also define their requirements for device capabilities so that they can be executed only on devices fulfilling the required capabilities. In order to consider network heterogeneity, and to select the most promising components for the preferred devices for the user, we modelled both the user task and the underlying network, consisting of devices and the components available on them, as graphs. This ensures the compatibility of devices in terms of communication protocols as well as the matching between the functional interfaces of components present on these devices and which are required by the user task. The mapping of the user task on to distributed components is done using a proposed matching algorithm.

To address the issue of architectural and description interoperability, we build our framework on the Service Component Architecture (SCA), which is a realization of the Service Oriented Architecture (SOA). A user task described in SCA can be defined in terms of interoperable services for which the components can be implemented in a number of technologies in a protocol-independent manner.

There are further assumptions, that we need to consider, when composing *ad hoc* user tasks. We assume that the end-user has a priori such an abstract task description available on their device. Moreover, we consider that user has specified their preferences in advance, so that they are available during the task composition process. The devices surrounding the user in the pervasive environment are also capable of communicating with each using the communication technology available on them and they can also discover the capabilities and components on the peer devices.

First, we introduce Service Component Architecture in the next section, and then in the following section, we will explain how it can be used for description of user tasks in our approach.

4.3 Service Component Architecture

As the research in Service Oriented Architecture (SOA) is evolving, different architectures and languages have been proposed over the time to support application development using the SOA paradigm. For example, Web Services (WS) have been a particular architecture for developing SOA based applications. Unfortunately, due to its reliance on a particular set of description languages and protocols (WSDL/SOAP/HTTP), Web Services have suffered much from criticism. Same is the case with those approaches that rely on Web Services for specifying the user task, e.g., as in (Issarny et al., 2005).

Thus, a felt was needed for an architecture model that is independent of particular implementation technology or communication protocol. The result was in the form of the Service Component Architecture (SCA). Being a recent technology, SCA is not yet widely accepted, however, due to its characteristics it seems to be an emerging standard for developing SOA based applications. It is due to these reasons that we have also chosen to use SCA as the description and deployment model for user tasks.

Service Component Architecture (SCA) (Open SOA Collaboration, 2008), maintained and standardized by the Open SOA consortium [1], provides a programming model for building applications and systems based on a Service Oriented Architecture (Papazoglou,

1. http://www.osoa.org/

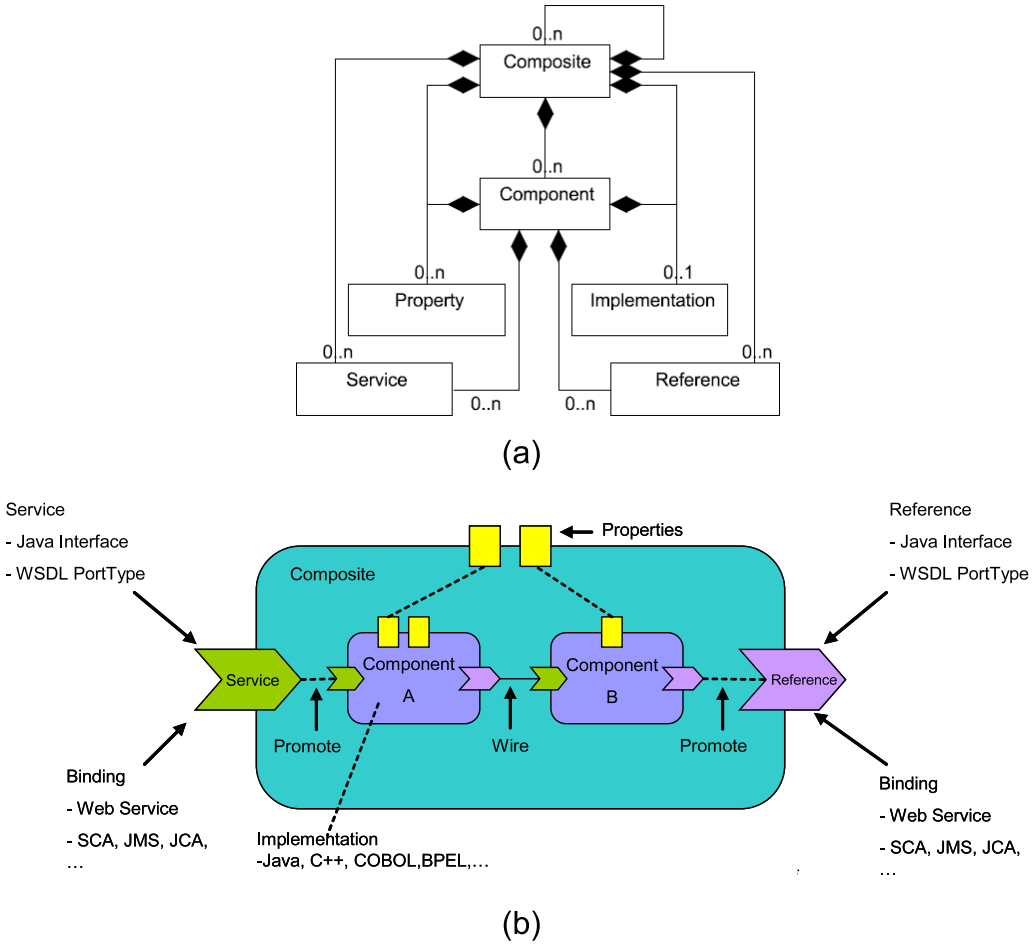

Figure 4.1: (a) SCA meta model (b) SCA composite diagram

2003). The main idea behind SCA is to be able to build distributed applications across organizations, which are independent of particular technology, protocol, and implementation. SCA extends and complements prior approaches to implementing services, and builds on open standards such as Web services.

SCA applications are deployed as composites. An SCA composite describes an assembly of heterogeneous components, which offer functionality as services and require functionality from other components in the form of references. SCA components can be implemented in Java, C++, COBOL, Web Services or as BPEL processes. Independent of whatever technology is used, every component relies on a common set of abstractions, i.e. services, references, properties, and bindings. A service describes what a component provides, i.e., its external interface. A reference specifies what a component needs from the other components or applications of the outside world. Services and references are matched and connected using bindings. A component also defines one or more properties. These properties can be customized, allowing a component to adapt its behaviour appropriately. Figure 4.1 (a) shows the SCA meta-model, while fig. 4.1 (b) shows the various SCA elements that constitutes an application in SCA.

SCA divides up the steps in building a service-oriented application into two major parts:

- The implementation of components which provide services and consume other services
- The assembly of sets of components to build business applications by connecting references to services.

SCA emphasizes the decoupling of service implementation and of service assembly from the details of infrastructure capabilities and from the details of the access methods used to invoke services. SCA components operate at a business level and use a minimum of middleware APIs (Edwards, 2007).

SCA allows dependency injection by relieving the developer from writing the code to find the required references and do the appropriate binding. The bindings are taken care of by the SCA runtime and can be specified at the time of deployment. The bindings specify how services and references communicate with each other. Each binding defines a particular protocol that can be used to communicate with a service as well as how to access them.

Because bindings separate how a component communicates from what it does, they let the component's business logic be largely divorced from the details of communication. A single service or reference can have multiple bindings, allowing different remote software to communicate with it in different ways.

4.3.1 Non-Functional Requirements Description in SCA

SCA provides a framework to support specification of non-functional constraints, capabilities, and QoS expectations from component design through to concrete deployment. The capture and expression of non-functional requirements is done through the introduction of policies in SCA. These policies are defined independently of the corresponding component assembly. The advantage is that policies can be reused for several different assemblies and can be changed without modifying the assembly itself. Thus, a separation is kept between functional and non-functional aspects of applications.

The SCA Policy Framework (Open SOA Collaboration, 2005) describes the framework and its usage for enforcement of policies in SCA. A brief overview of the SCA Policy Framework is provided in the following subsection.

SCA Policy Framework

A policy describes some capability or constraint that can be applied to components or to the interaction between components. Essentially, the policies deal with the non-functional requirements of SCA applications. The SCA Policy Framework (Open SOA Collaboration, 2005) describes how policies and policy subjects specified using WS-Policy (WS-Policy, 2007) and with other policy languages, to be associated with SCA components. The version 1.0 of the SCA Policy Framework discusses only the security and reliability policies.

SCA does not define how policies should be described within a domain —no policy language is mandated— and so vendors are free to do this in their own ways. In order to allow the inclusion of any policy language within a policy set, SCA allows using the extensibility elements in the `@policySet` attribute, which may be from any namespace and may be intermixed. One or more policy sets can be attached to any SCA element used in the definition of components and composites. The attachment is specified by using the optional `@policySet` attribute, which takes as its value a list of policy set names.

How a policy is interpreted depends on how the policy is defined within the domain in which the SCA component is running. For example, a binding for a service can have an associated policy set describing its interaction policies, while a binding for a reference can have another policy set describing its interaction policies. When a binding is created between them, these policy sets are matched, and their intersection determines the set of policies used for this communication.

The SCA policy framework defines the following key concepts:

- An *Intent* allows the SCA developer to specify abstract Quality of Service capabilities or requirements independent of their concrete realization.
- A *Profile* allows the SCA developer to express collections of abstract QoS intents.
- A *Policy Set* provides the realization of concrete policies for a set of intents. The set of intents is also provided by the policy set.

For example, a service which requires its messages to be authenticated can be marked with an intent "authentication". This marks the service as requiring message authentication capability without being prescriptive about how it is achieved. At deployment time, when the binding is chosen for the service (e.g., SOAP over HTTP), the deployer can apply suitable policies to the service which provide aspects of WS-Security, for example, and which supply a group of one or more authentication technologies.

It is also possible to use *qualified intents*, which are more restrictive than the unqualified intents. For example, assume that the intent "confidentiality" comes in two types: transport and message. An unqualified intent `confidentiality` required by a message may be satisfied by any of the transport or message confidentiality mechanism applied. However, a qualified intent `confidentiality.transport` will require to ensure transport confidentiality for messages.

Building on the notions of Intents, Profiles, and Policy Sets defined by the SCA Policy Framework, we now describe how CC/PP can be used for resource specification

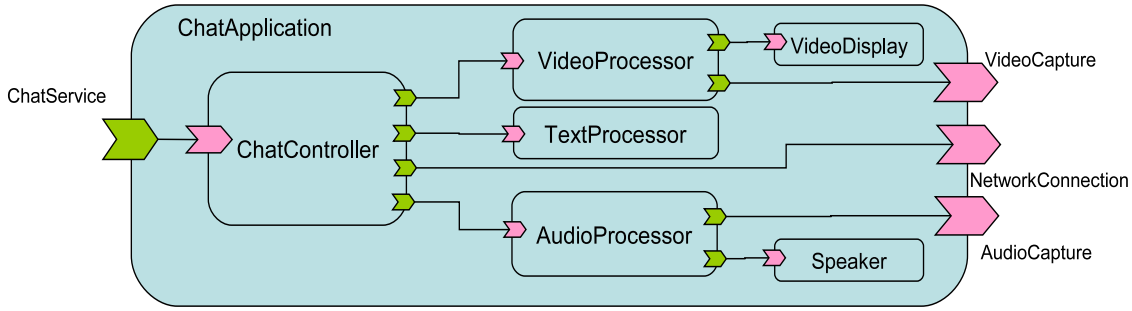

Figure 4.2: Example Chat application represented as SCA composite

in SCA. We use these concepts to distinguish between abstract and concrete resource specifications.

4.3.2 Example SCA Application

Figure 4.2 shows the example chat application modeled as SCA composite. The `ChatApplication` application provides one service called `ChatService` and consists of five components: `ChatController`, `VideoProcessor`, `AudioProcessor`, `TextProcessor`, `VideoDisplay`and `Speaker`. The composite also requires three references: `VideoCapture`, `AudioCapture` and `NetworkConnection`[2].

SCA Assembly Model Specifications (Open SOA Collaboration, 2008) has also defined how an SCA application can be described in SCDL (Service Component Description Language), an XML based description language. The listing below shows the SCDL for our example chat application as shown in figure 4.2.

```
<composite name="ChatApplication">

   <service name="ChatService" promote="ChatController/UIService"/>

   <component name="ChatController">
      <service name="UIService">
         <interface.java interface="example.chat.ChatController.controllerInterface"/>
      </service>
      <implementation.java class="example.chat.ChatController.ControllerImpl"/>
      <reference name="VideoProcessor"/>
      <reference name="AudioProcessor"/>
      <reference name="TextProcessor"/>
      <reference name="NetworkConnection"/>
   </component>

   <component name="VideoProcessor">
      <service name="VideoProcessingService">
```

2. In this thesis, we assume that hardware components offer certain programming-level abstraction and can be represented as software components and, hence, can be referenced by other software components forming the business logic. Here, the distinction between hardware and software components does not make any difference, but in our future work we will distinguish between these components for dynamic adaptation of the *ad hoc* user tasks. Here, software/hardware components will be treated equally.

```
    <interface.java interface="example.chat.video.VideoProcessorInterface"/>
     </service>
     <implementation.java class="example.chat.video.VideoProcessorImpl"/>
     <reference name="VideoDisplay"/>
     <reference name="VideoCapture"/>
  </component>

  <component name="AudioProcessor">
     <service name="AudioProcessingService">
        <interface.java interface="example.chat.audio.AudioProcessorInterface"/>
     </service>
     <implementation.java class="example.chat.Audio.AudioProcessorImpl"/>
     <reference name="Speaker"/>
     <reference name="AudioCapture"/>
  </component>

  <component name="TextProcessor">
     <service name="TextInputService">
        <interface.java interface="example.chat.input.TextInputInterface"/>
     </service>
     <implementation.java class="example.chat.input.TextInputImpl"/>
  </component>

  <component name="VideoDisplay">
     <service name="VideoDisplayService">
        <interface.java interface="example.chat.video.VideoDisplayInterface"/>
     </service>
    <implementation.java class="example.chat.video.VideoDisplayImpl"/>
  </component>

  <component name="Speaker">
     <service name="AudioOutputService">
        <interface.java interface="example.chat.audio.AudioOutputInterface"/>
     </service>
     <implementation.java class="example.chat.audio.Audio.AudioOutputImpl"/>
  </component>

  <reference name="AudioCapture" promote="AudioProcessor/AudioCapture">
     <interface.java interface="example.audio.micInterface"/>
  </reference>

  <reference name="VideoCapture" promote="VideoProcessor/VideoCapture">
    <interface.java interface="example.video.captureInterface"/>
  </reference>

  <reference name="NetworkConnection" promote="ChatController/NetworkConnection">
    <interface.java interface="example.network.connectionInterface"/>
  </reference>

  <wire source="ChatController/VideoProcessor" target="VideoProcessor/VideoProcessingService"/>
  <wire source="ChatController/AudioProcessor" target="AudioProcessor/AudioProcessingService"/>
  <wire source="VideoProcessor/VideoDisplay" target="VideoDispaly/VideoDisplayService"/>
  <wire source="AudioProcessor/Speaker" target="Speaker/AudioOutputService"/>

</composite>
```

4.3.3 Extensibility of SCA

As described in the SCA Assembly Model specifications (Open SOA Collaboration, 2008), it is possible to extend SCA without disturbing the existing assembly model specifications. This is possible by allowing additional tags and attributes in the existing SCA element description. Using them we can add custom elements and attributes originating from any namespace.

The caveat is that only those parsers and runtimes, which are aware of the semantics of the custom attributes, can process the contents. Parsers that do not understand the custom elements and attributes will simply ignore them.

4.3.4 Limitations of SCA

SCA is sharply gaining acceptance in the services-oriented industries. However, the current specifications of SCA consider a SCA composite as a static assembly of components. The binding between the services and components is also defined at deployment time. The specifications do not describe mechanisms for matching of services with components for dynamic binding and the bindings are specified manually or by the container at runtime for particular, selected types of bindings.

While this approach leads to greater flexibility, it also affects the way various services are to be considered during binding. Currently, SCA runtimes has no mechanism to determine the best component among a set of given components. The only way to connect a reference to a proper service is by using binding information or policies. To overcome these limitations, we describe in Section 4.5.1, how we can add semantic annotations to SCA applications that will allow matching of services and components for dynamic bindings. Also, since the SCA Policy Framework has defined a limited set of policies related to only the non-functional aspects such as transactions and security and does not specify any policy regarding resources or QoS, in Section 4.5.2, we propose to use CC/PP as a policy language for describing service capabilities in SCA.

4.4 User Task Description in SCA

In this section, we describe how we can use SCA for describing a user task. We also explain how semantic descriptions can be added to SCA elements and how CC/PP can be used as a policy language for describing service requirements in the user task.

4.4.1 Abstract and Concrete User Tasks

As mentioned previously, a user task is described abstractly so that its concretization can be carried out dynamically depending upon the context in which it is used. In general, we say that a user task is abstract when its description specifies the services in the task but does not specify the components that provide implementation of these services. Such a composition describes the services participating in the composition, but does not tell about how the services are implemented.

When this concept is applied to SCA, we say that a user task described in SCA is abstract if its description does not contain implementation definition. An abstract task is represented by an abstract SCA composite. An abstract composite specifies only

its services, references and the components that provide those services. However, the components specify only their interfaces but they do not specify any implementation [3].

The resources are also specified abstractly. A concrete composite, on the other hand, specifies all its interfaces as well as technology-specific implementation and , possibly, concrete resources required by it.

The listing below shows how the `VideoProcessor` component of the example SCA application of figure 4.2 can be described abstractly.

```
<component name="VideoProcessor">
   <service name="VideoProcessingService">
      <interface.java interface="example.chat.video.VideoProcessorInterface"/>
   </service>
   <reference name="VideoDisplay"/>
   <reference name="VideoCapture"/>
</component>
```

The corresponding concrete component will also have the implementation description, e.g., by referring to the Java class that implements the component's interface.

However, as know that in a pervasive environment, there may be lots of components, implemented in a variety of languages, but which provide the same functionality. Although SCA applications can be composed of components implemented in heterogeneous languages and technologies and we can use any implementation technology to plug in as the component implementation, the problem is that it is not possible to determine at run time, which particular component implementation to choose. In fact, it is hard in first place to determine whether a component implementation is exactly what is required by the component's interface.

4.5 Extensions to SCA

To overcome the limitations of SCA, identified in Section 4.3.4, in this section we propose some extensions to the SCA assembly model specifications.

4.5.1 Semantic Service Descriptions in SCA

To be able to reason about the functional properties of SCA artefacts, we add semantic descriptions to them. We have proposed Semantic Annotations for SCA (SA-SCA) (Belaïd et al., 2009a)(Belaïd et al., 2009b), which suggests how to add semantic annotations to various SCA artefacts: composite, services, components, interfaces, and properties. This extension is similar to the concept of annotations used in SA-WSDL (Semantic Annotations for Web Service Description Language) (Akkiraju and Sapkota, 2006) and is in accordance with the SCA extensibility mechanism (Open SOA Collaboration, 2008). Our proposed SA-SCA extension defines a new namespace called *sasca* and adds an extension attribute called *modelReference* so that relationships between SCA artefacts and concepts

3. We assume here that all the components in a composite may be abstract, i.e., they do not provide implementation details. However, it is possible that some components may provide implementation details, e.g., the Java class used for implementing the component.

in another semantic model are handled. This semantic model can be an ontology or any other model in which concepts can be described. This choice is motivated by the fact that applications developers can use any ontology language to annotate services rather than be bound to one particular approach. The listing below shows the description of our abstract VideoProcessor component when semantic annotations are added to it:

```
<component name="VideoProcessor">
   <service name="VideoProcessingService"
            sasca:modelReference="http://example.org/chat.owl#VideoProcessor">
      <interface.java interface="example.chat.video.VideoProcessorInterface"/>
   </service>
   <implementation.java class="example.chat.video.VideoProcessorImpl"/>
   <reference name="VideoDisplay"
            sasca:modelReference="http://example.org/chat.owl#VideoDisplay"/>
   <reference name="VideoCapture"
            sasca:modelReference="http://example.org/chat.owl#Videocard"/>/>
</component>
```

Note that the component description now has a reference to an OWL ontology, which contains the definition of the VideoProcessor concept. The references required by the component are also annotated with `@modelReference` attribute. When this abstract component is matched with concrete components, it will be ensured that both of them refer to the same VideoProcessor concept as well as their references also refer to the same concepts in ontology. Only if they match, the concrete component description can be used. For example, consider the following concrete component:

```
<component name="VideoProcessorComponent">
   <service name="VideoProcessingService"
            sasca:modelReference="http://example.org/chat.owl#VideoProcessor">
      <interface.java interface="example.app.VideoInterface"/>
   </service>
   <implementation.java class="example.chat.video.VideoProcessorImpl"/>
   <reference name="VideoDisplayComponent"
            sasca:modelReference="http://example.org/chat.owl#VideoDisplay"/>
   <reference name="VideoCaptureComponent"
            sasca:modelReference="http://example.org/chat.owl#Videocard"/>/>
</component>
```

Since the `@modelreference` attribute in both the abstract and concrete descriptions refer to the same `VideoProcessor` concept in the service and the same `VideoDisplay` and `Videocar` concepts in the references, they will match.

4.5.2 Service Capability Requirements in SCA

We propose that services in the SCA application may describe the capabilities required for their execution. These capabilities are expressed as the device capabilities. As mentioned previously, the reason is to determine whether the service can be executed on a particular device or not: this may be due restriction on usage of certain types of resources or due to the capabilities of the user.

Abstract Capability Description

As described in Section 4.3.1, the SCA policy framework allows to specify a policy abstractly at service level using intents and profile, and then to provide concrete policy implementation using the policy sets at the implementation level. Based on the same concept, we can specify capabilities at abstract and concrete levels as following.

The services in the user task specify their requirements for capabilities abstractly. The component implementations provide the corresponding concrete policies. Both the service and reference SCA elements use the @require attribute to specify their intents for resource requirements. In the description, `@requires`, `@intent`, or `@profile` attributes are used to specify abstract resource requirements, while the `@policySets` attribute is used to specify concrete resources.

Consider the example chat application once again. The `VideoChat` composite requires a service, which is specified as a reference named `NetworkConnection`, for connection to the Internet. Thus, the device executing the `VideoChat` application must be able to connect to the network; otherwise the application will not work properly. This means the `NetworkConnection` reference requires the capability of connecting to the internet. This *non-functional* capability can be requested by specifying it using the `@required` attribute in the reference definition as following:

```
<reference name="NetworkConnection" promote="ChatController/NetworkConnection"
      requires="NetworkCharacteristics.NetworkAccess">
   <interface.java interface="example.network.connectionInterface"/>
</reference>
```

The requirement is specified using the unqualified intent `NetworkCharacteristics.NetworkAccess`. A qualified intent such as `NetworkCharacteristics.NetworkAccess.Wi-Fi` or `NetworkCharacteristics.NetworkAccess.GPRS` could have also been used to restrict the network access type. Similarly, the `VideoDisplay` and `AudioOutput` services may also specify their requirements for video and audio capabilities of a device as following:

```
<component name="VideoDisplay">
   <service name="VideoDisplayService" requires="HardwarePlatform.VideoCapable">
      <interface.java interface="example.chat.video.VideoDisplayInterface"/>
   </service>
  <implementation.java class="example.chat.video.VideoDisplayImpl"/>
</component>

<component name="Speaker">
   <service name="AudioOutputService" requires="Hardware.SoundOutputCapable">
      <interface.java interface="example.chat.audio.AudioOutputInterface"/>
   </service>
   <implementation.java class="example.chat.audio.Audio.AudioOutputImpl"/>
</component>
```

Now consider the particular case of the `TextInputService` service. This service requires that the device must be capable of accepting input from the user. Depending upon the interactivity model used by the device, text input can be accepted by a device in

several ways: touch-screen, keypad, soft-keypad, keyboard, and mouse (using virtual, on-screen keyboard by clicking), etc. Thus, instead of specifying one of these several types as a required capability, the service may specify the capability abstractly as following:

```
<component name="TextProcessor">
   <service name="TextInputService" requires="HardwarePlatform.InputCapable">
      <interface.java interface="example.chat.input.TextInputInterface"/>
   </service>
</component>
```

The service uses an unqualified intent *HardwarePlatform.InputCapable*, which specifies that in order to use the service, the Input resource from the Hardware category must be available. The service does not specify the type of particular input method. It will depends on the policy set which realizes the *HardwarePlatform.InputCapable* intent. This policy set is defined by the system administrator or the deployer who is also responsible for defining the intent.

Concrete Capability Description

Concrete policies can be specified using the `@policySets` attribute in SCA. Such policies can be applied to implementations and bindings and they dictate the requirements that should be fulfilled before executing the components to which the policy sets are attached. For example, consider the abstract `TextProcessor` component described above. One particular implementation of this component can be provided in Java. The concrete component is shown in the following listing:

```
<component name="TextProcessor">
   <service name="TextInputService" requires="HardwarePlatform.InputCapable">
      <interface.java interface="example.chat.input.TextInputInterface"/>
   </service>
   <implementation.java class="example.chat.input.TextInputImpl" requires="JavaEnvironment"/>
</component>
```

The implementation part of the description requires a concrete policy using the *JavaEnvironment* intent. The concrete policy set *JavaExecutable* providing the intent *JavaEnvironment* is as followig:

```
<policySet name="JavaExecutable" provides="JavaEnvironment" appliesTo="implementation.java">
   <ccpp:Profile xmlns:ccpp=http://example.com>
      <SoftwarePlatform JavaEnabled="true">
         <Java Platform="CDC" OptionalPackages=VirtualKB/>
      </SoftwarePlatform>
      <HardwarePlatform>
         <Memory freeMemory="256"/>
      </HardwarePlatform>
   </ccpp:Profile>

   <security>
```

```
      <allow roles="customers">
   </security>
</policySet>
```

As one might expect, the *JavaEnvironment* policy set describes the resource and security policies simultaneously. The resource policy, described in CC/PP fragment dictates that the software platform must be Java enabled. It also specifies the required Java profile, CDC, to execute the class as well as the optional package required by it. Note that this package is not specified as a reference, because it is native to the platform, and is not deployed as a SCA component. The class also requires a free memory of 256 kb for loading its resources. The CC/PP related policies have been specified by the developer of the `TextInputImpl` Java class. Since the developer has the knowledge of the resources required for the developed Java class to be executed, he/she explicitly include them in implementation definition. This means that the container must fulfil these requirements before the class is executed.

The security policy dictates that only users of type *customers* will be allowed to use these resources. Since these resources are required by the Java class, it implies that the specified Java class will be allowed to use by the customers only. When the SCA runtime will be instantiating the task, it will match the policies and allow only those users who are "customers", otherwise, the non-customer user will not be allowed to use the component on this device. An alternative component on the same device or some other device, with different policy allowing for non-customer users, will be selected by the runtime.

4.5.3 Service Collocation in SCA

In the previous chapter, we introduced the notion of *service collocation*, i.e., when one service needs to be collocated with another service on the device. For example, in the example chat application, let us assume that the application architect requires that the `VideoDisplay` service and the `VideoCaptureService` reference must be executed on the same device: they cannot communicate on network protocols. Similarly, `AudioOutput` service and the `AudioCaptureService` references cannot communicate across devices and must be executed on the same device.

To accommodate service collocation in SCA, we introduce a new attribute called `@collocation` in the service definition tag. The `@collocation` attribute of a service specifies the list of other services or references which should collocate with the given service. The listing below shows how the AudioOutputService of the `AudioProcessor` component specifies its collocation with the AudioCaptureService of the `Microphone` component.

Figure 4.3 (a) shows the example chat application with additional notations to represent the proposed extensions to SCA. As it can be observed, the diamond symbol attached to a service or a reference signifies the service or reference's requirement for a capability. A triangle on a wire indicates the collocation dependency between the services enclosed by the corresponding components joined by the wire.

4.5.4 SCA Application Composition as Graph

So far we have described how user task can be described in SCA along with capability requirements and collocation dependencies. We also described how an SCA application

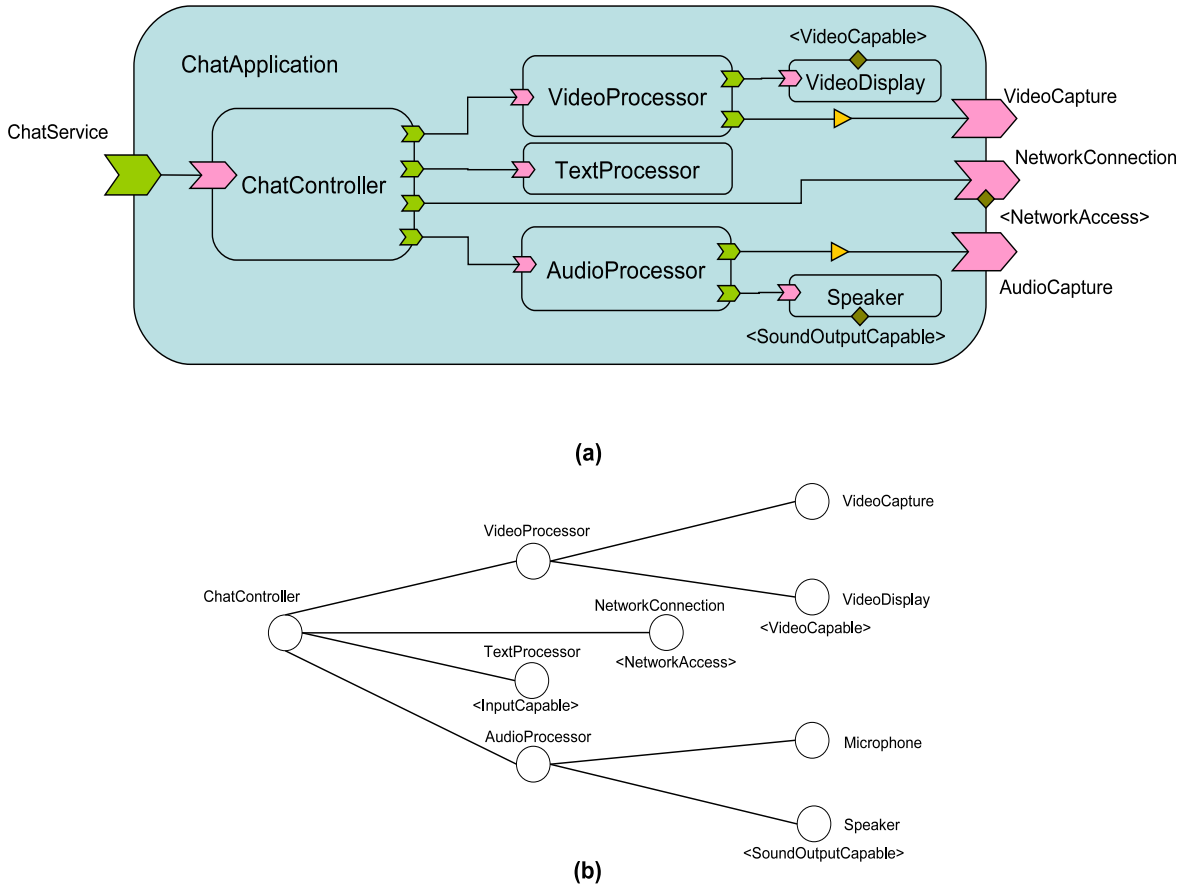

Figure 4.3: Example Chat application (a) SCA composite (b) graph representation

can be represented as a graph. In our previous works (Belaïd et al., 2009a)(Belaïd et al., 2009b), we had used the tree structure for representing SCA composition where the root of the tree was the outermost composite, however, considering the wiring between intermediate components in a composite, it is more suitable to use a general graph instead of tree. One important thing to consider for SCA based applications is that they also have references. These references may be provided by services residing locally or on a remote device. To accommodate the distributed nature of references, we model them similar to services and represent by nodes in the graph. Figure 4.3 (b) shows the graph representation of the chat application.

4.6 Capabilities and Preferences Description in SCA

Earlier, in Section 3.2, we explained that being a W3C standard, CC/PP was widely used for capabilities description of devices by various vendors and, hence, we would adopt the CC/PP model for device description in our approach. However, CC/PP is based on RDF and the importance of RDF could only be realised if we were to use semantic matching. Since our task model is described in SCA, it is more appealing to use SCA for device description as well. In fact, our extended CC/PP model (see figure 3.3) consists of the root component that references hardware, software, and network, etc. component and each one of these also references other components, this structure can be better explained with the recursive nature of SCA assembly model. The advantage of using SCA, instead of CC/PP, for device description is compact description resulting in efficient processing, as well as allowing us to use the same programming API (parser) for processing device description as well as the user task.

User Preferences Description

In Section 3.2.1, we identified that W3C has proposed CC/PP for expression of device capabilities and user preferences. However, in practical situations, it has only been used for device capabilities only (e.g., as in (Indulska et al., 2003),(F. et al., 2006) and (Li and Wang, 2006)). This is because the CC/PP specifications have not defined particular attributes related to user preferences and there is no way to differentiate between whether a given attribute stands for a user preference or a device capability. Apart from CC/PP, there is no standard for user preferences description, to the best of our knowledge.

This left us no other choice except to define our own description language for preference specification. One possibility was to extend CC/PP with additional annotations to differentiate between user preferences and device attributes. However, by using this approach, we would still not be able to express the different *importance levels* that we proposed to use for a given attribute, using CP-nets.

Since CC/PP is based on Resource Description Framework (RDF), one possibility was to use pure RDF description for preferences description. The base element of the RDF model is a triple: a resource (called subject) is linked to another resource (called object) through a relation using third resource (called predicate). We can use the same triple to represent user preferences. For example, figure 4.4 (a) shows the diagrammatic representation of a RDF triple as (*Subject, Predicate, Object*), and figure 4.4 (b) shows how a user preference can be modelled as a triple (*Attribute, Preference, Value*). Figure 4.4

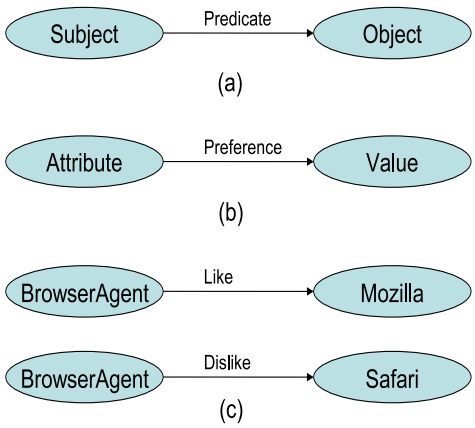

Figure 4.4: (a) An RDF Triple (b) Representing Preferences in RDF (c) Example Preference Specification Using RDF Triple

(c) shows how the RDF triple can be used for two instances of user preferences. Here, we have shown how for a single attribute, we can have multiple values using RDF. However, knowing the parsing complexity of RDF documents and its unusual verbosity, we decided to use a light-weight solution based on SCA. The purpose was to achieve both parsing efficiency and compactness of presentation.

Since user preferences are about device capabilities, we were also inclined to re-use our CC/PP model for user preferences. Thus, our final solution was to describe the CC/PP model in SCA with additional annotations for user preferences.

The listing below shows how user preferences can be specified for some attributes in the SoftwareExtensions components of our extended CC/PP model in SCA.

```
<component name="SoftwareExtensions">
    <property name="Calendar" type="userpref">
        <preference value="Thunderbird" level="Like"/>
        <preference value="MS Outlook" level="Somewhat Like"/>
</property>
    <property name="VideoViewer" type="userpref"/>
        <preference value="VLC" level="Like"/>
        <preference value="Media Player" level="Somewhat Like"/>
        <preference value="QuickTime" level="Dislike"/>
</property>
        . . .
</component>
```

4.7 User Task Composition Using MATCH-UP

In this section, we present MATCH-UP: our Middleware for Ad hoc Task Composition in Heterogeneous environments considering User Preferences. MATCH-UP provides a comprehensive solution for *ad hoc* user task composition. MATCH-UP considers all the

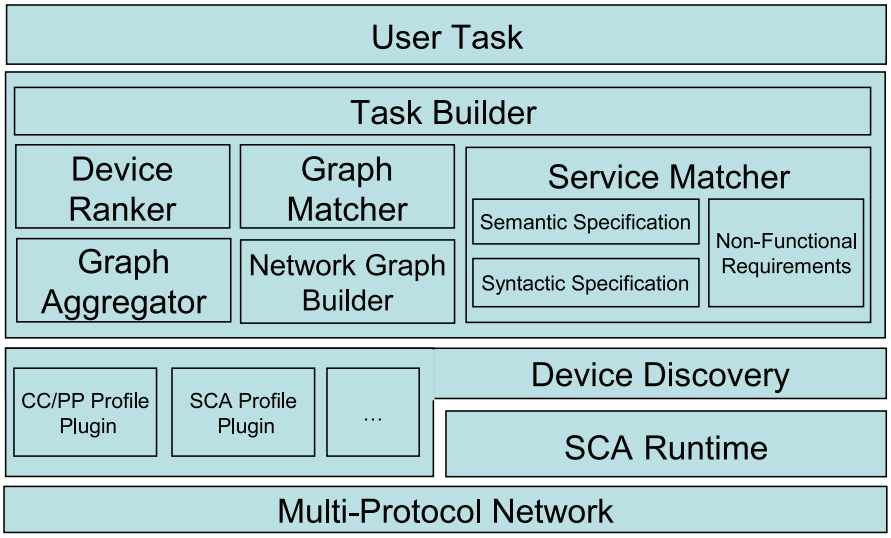

Figure 4.5: MATCH-UP Architecture

requirements we identified in Section 1.1 by integrating the modelling and algorithmic aspects of our approach, discussed so far.

In the remaining of this chapter, we describe the architecture and functionality of MATCH-UP as well as a prototype implementation and its evaluations in the next few sections.

4.7.1 MATCH-UP Architecture

Figure 4.5 shows MATCH-UP architecture. The architecture builds up on the underlying multi-protocol network consisting of heterogeneous service discovery and network access protocols. The network access protocols are taken care of the underlying SCA runtime, which can, possibly, delegate this task to some other available infrastructure as discussed in Section 3.9.1. The device (and service) discovery protocols are dealt with by the Device Discovery component of our middleware. Due to the heterogeneity of device profile description languages, the Device Discovery component relies on additional plugins for processing individual profiles. This will be discussed in Section 4.9.1.

The architectural that coordinates the user task composition is the Task Builder. It uses the services of other architectural components, namely, Device Ranker, Service Matcher, Network Graph Builder, Graph Aggregator and Graph Matcher to carry out various functionalities, which are described in the next subsection.

4.7.2 MATCH-UP functionality

When a user wants to execute a task, he sends the task to the MATCH-UP middleware through some suitable user interface. The Task Builder accepts the user task description and, after coordinating a number of activities, composes the concrete user task. The activities are depicted in the collaboration diagram shown in figure 4.6 and are carried out as following.

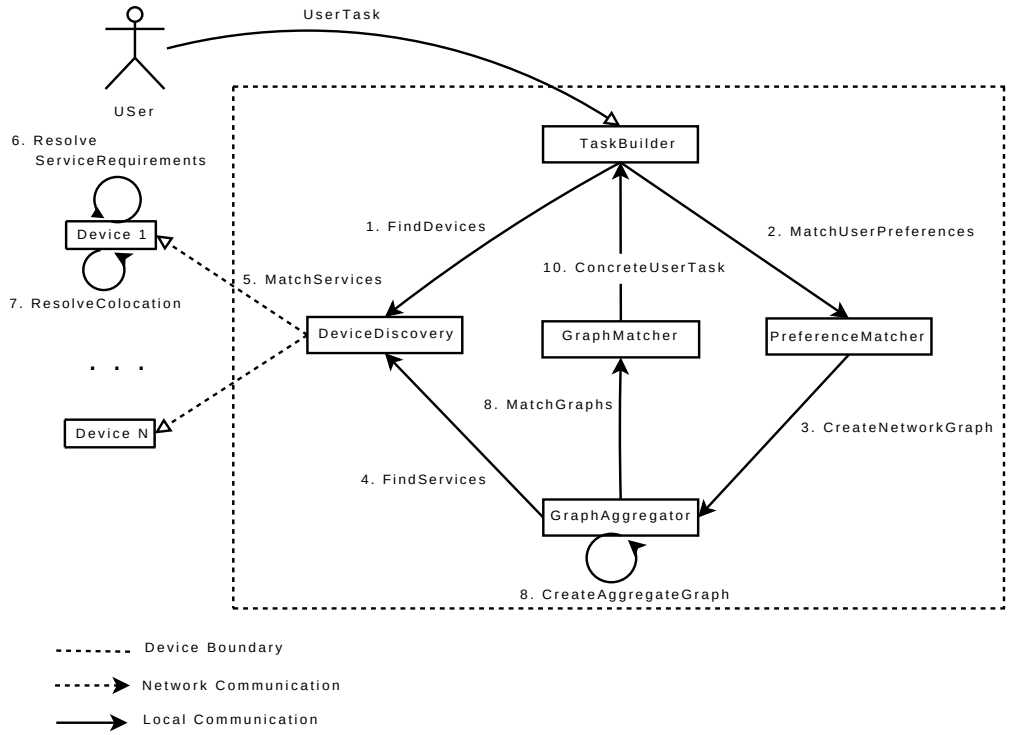

Figure 4.6: The collaboration diagram showing activities involved in the user task composition process

User Task Analysis

When the Task Builder receives the description of the user task, it analyzes the individual services in the task and transforms it into a graph structure. The Task Builder then queries the Device Discovery component for the available devices in the environment. This is shown as activity 1 in figure 4.6.

Device Discovery

The Device Discovery component has either a list of all the available devices in the environment or it sends a device discovery request simultaneously in all the networks in which it can communicate to know about the available devices. In the first case, we assume that each device sends regular messages in the network to notify other devices about its status. The status can be of several forms such as availability or unavailability due to device's mobility or due to usage by other users present in the environment. Each device upon receiving a notification message, stores the sender device's status and refreshes it each time when it receives a new notification from the device after regular intervals. This way each device keeps itself up to date about the other devices in the pervasive environment.

Once the Device Discovery knows about the peer devices in the environments, it disseminates the user task description to all of them. This description contains the services defined by the task along with their requirements. No user preferences are sent to the peer devices.

Service Matching

Each peer device, upon receiving the task description, passes the task to the Service Matcher, which performs three steps:

Requirement matching for each of the service in the task, it compares the service requirements (if any) with the capabilities provided by the device. If any of the requirements of a service are not met by the device, the service is not fit for the device and it is removed from the list of services in the task to be selected on this device.

Component matching for those services in the user task, which have passed the first step, the Service Matcher finds the matching concrete components on the device repository. Again, any service for which no corresponding concrete component can be found is removed from the list of services in the task.

Service collocation the Service Matcher ensures that all service collocation requirements are met. If a service has collocation dependency on some other services in the task, they must also have passed through the previous steps; otherwise, if for a given service, its collocation cannot be found (or does not fit) on the device, the service must be removed from the list of services in the task.

Service matching is a filtering step. A service failing any of the previous steps is filtered out for use on the given device. The list of selected services is sent back to the

requesting device along with metadata concerning the matched components and their properties.

The user device receives response from all the devices in the network. However, some devices would have responded without any matching component, because of filtering of all of the services during the matching process described above. Any such device is not considered further by the Device Discovery. All other devices which have responded with a match for at least one service in the user task are forwarded to the Task Builder. For a given service in the user task, the Task Builder may have received several matching components from different devices. The Task Builder then considers user preferences for selection of the most suitable components.

User Preferences Matching

The user has specified their preferences previously (possibly, different preferences are specified at different times) and their preferences are stored on their device. When executing a task, they can also specify additional task-specific preferences just before sending the task to MATCH-UP.

User preferences are used during selection of devices, which ultimately results in the selection of final components for the user task. When the Task Builder analyses the user task, it also identifies the task specific user preferences and compares them with the existing user preferences. If it happens to be that similar user preferences existed before, the Task Builder[4] overrides the existing preferences by the newer ones, for the task at hand. The Preference Matcher then compares user preferences with individual device capabilities for all devices found by the Device Discovery component previously. The matching between user preferences and device capabilities is done similar to the procedure described in the PreferenceTree algorithm in Section 3.6.6. This results in ranking of all the devices according to device value calculated on the basis of user preferences.

Device Elimination

Devices are ranked so that the device higher up in the rank is more promising for the user — according to their preferences — as compared to the ones with lower ranking. Since the devices are ranked on the basis of their device values, we choose a certain real-valued threshold level below which all the devices are discarded. The assumption is that the devices above the threshold will be sufficient to contain the components required by the services in the user task. Hence, any device with a value below this threshold will not be needed for component selection. This will result in less number of devices to consider for selection of components.

Since the device value varies from device to device and even for the same device it may vary from one user to another due to difference in their preferences, we cannot determine a specific value for the threshold. Instead, we can use a statistical technique to find the threshold as discussed in Section 4.9.

4. Another entity, the Preference Builder is responsible for user preferences related issues, however, for simplicity of discussion we have referred here to the Task Builder.

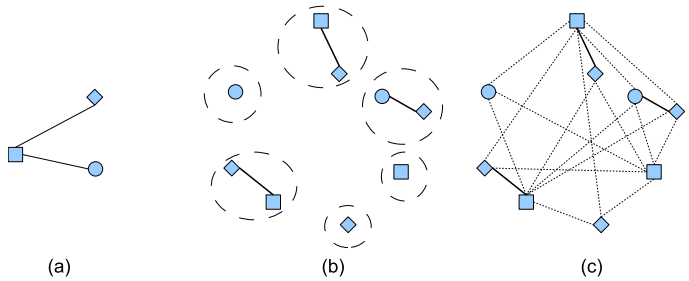

Figure 4.7: (a) User Task Graph (b) Different Devices with Available Components (c) Aggregate Graph

Network Graph Creation

When a selected number of devices is chosen based on user preferences, they are connected together in the form of a graph. The graph is created on the basis of protocol compatibility between devices, i.e., two devices are connected if they can communicate with each other using some common access protocol. The reason is to define a valid device configuration so that when a device is selected for a particular component corresponding to a service in the task, it should be able to communicate with components on peer devices if other services in the task are selected to be executed on that device.

Aggregate Graph Creation

By creating the network graph, we have ensured that the devices be connected in a valid configuration. However, we are still not sure about how the components on these devices will be connected together (or bound, in programming terminology) to compose a valid user task. Thus, an aggregate graph of consisting of components is created. This graph is a transformation of the network graph: a node in the network graph, representing a device, is replaced by the nodes representing components on the device and each component node is connected to all other component nodes, on the same device or on the adjacent devices, preserving the relation between the corresponding services in the user task graph. The aggregate graph consists of all those components which are found on the selected devices.

Figure 4.7 illustrates the creation of an aggregate graph for a user task with three components. Figure 4.7 (a) shows the user task. Figure 4.7 (b) shows different components required by the task available on various devices and figure 4.7 (c) shows the aggregate graph created from these components.

Graph Matching

Our aim is to map the user task graph, consisting of services, on to the aggregate graph, consisting of components which implement those services, such that there may exist more than one component in the aggregate graph corresponding to a service in the user task graph. This mapping should give us a sub-graph of the aggregate graph that is isomorphic to the user task graph. That is, the components in the sub-graph will be

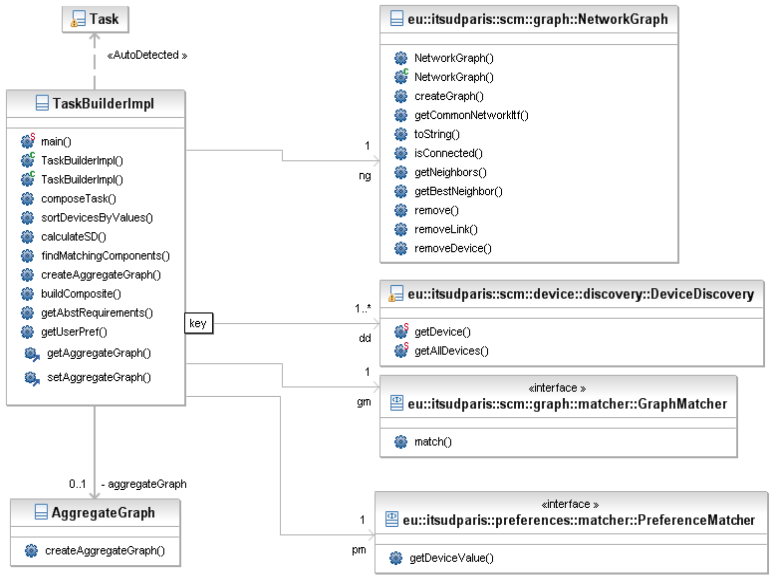

Figure 4.8: UML class diagram showing the Java classes corresponding to our architecture

connected if and only if they are connected in the user task graph for their respective services.

The mapping between the user task graph and the aggregate graph is done using the ComposeTask algorithm described in Section 3.8.2. This mapping will determine the components distributed on various devices (or possibly on a single device) that can be instantiated and bound together for execution of the user task.

4.8 Prototype Implementation

We have implemented a prototype of our composition system in Java based on the architecture defined above. The UML class diagram of figure 4.8 shows the important Java classes of our implementation. The current prototype is based on simulations: all the above mentioned classes are executed on a single machine. The Device Discovery is already equipped with a list of "known" devices. The user preferences are defined in advance and instead of providing the specifications as a TCP-net, they are taken for a fixed set of attributes with randomly selected importance levels. So no conditional selections are implied in the preferences. This makes the preference specification much similar to what we described in our quantitative preference model (see Section 3.5), but instead of variables taking only a single value, multiple values can be specified simultaneously with varying importance levels. Some evaluation results of the prototype implementation are discussed in Section 4.10.

To test our system in a working environment, consisting of real devices, our plans are to incorporate our implementation into one of the existing SCA runtimes. Since some of the functionalities of our framework will depend upon the underlying runtime, we are particularly interested in selecting one that will allow the dynamic binding of components,

remote communication between the components as well as notifying our framework of the events taking place in the underlying network.

Currently, there are more than a dozen commercial or open-source implementations of SCA available[5] in the market. Two notable, light-weight, open-source implementations that may satisfy our requirements are the FraSCAti component framework[6] and the Newton component model[7]. FraSCAti is based on Java runtime and uses the Fractal component model[8] to support dynamic binding of SCA components. Newton defines a SCA component model on top of the OSGi service model. Selection of one of these will require detail investigation into their properties and flexibility in response to our requirements. We can then deploy our middleware on top of the selected runtime as a service.

Apart from deployment of MATCH-UP on top of an SCA runtime, as a single service, we are also looking into the possibility of distributing our middleware functionality in the environment. For example, instead of loading all the components of MATCH-UP on every device, we may assume that some components (such as only Task Builder and Device Builder) must be available on each device as a minimal requirement. The rest of the components will be available on some of the devices only.

This also has practical implications. Considering the resource limitations of small, hand-held devices and the fact that SCA runtime is not yet available for such devices, only the distributed deployment of the middleware will be feasible in many circumstances.

4.9 Discussion

In this section, we discuss some of the issues that were not explained above.

4.9.1 Interoperability of Device Profiles

When a device needs to know about the capabilities of some other device, it requests the device profile from the latter. Considering the heterogeneity of devices and the various device profile description mechanism available (discussed in Section 3.2), the requestor device may receive peer devices' profiles in different description languages. This raises the issue of interoperability among various device profile descriptions.

To solve the interoperability issues among different device profiles, we have defined a Common Device Description Structure (CDDS) for processing device capabilities. That is, while a device may use any language for its profile description (e.g., SCA, CC/PP, FIPA, OWL, etc.), a translation of the language into a common description language, CDDS, is performed on the requestor device. Thus, any type of device description is first translated into CDDS before processing. This translation is performed by the Device Discovery component of the architecture with the help of various plug-ins available to it. If a device description is encountered for which there is no plug-in, the Device Discovery will not be able to process it and, thus, the device will be discarded. The purpose of CDDS is to allow better interoperability and to rely on a single programming interface

5. http://osoa.org/display/Main/Implementation+Examples+and+Tools
6. http://frascati.ow2.org/
7. http://newton.codecauldron.org
8. fractal.ow2.org/

for accessing device capabilities, independent of the underlying description language used by the device. It can be any of the existing language, but as described in Section 4.6, we have used SCA based description for its representation.

4.9.2 Threshold for Device Elimination

When all the selected devices are ranked, we also calculate the average device value simultaneously and use it as a threshold, i.e., for a given device value which falls below the average device value, the device is eliminated. However, this threshold is quite optimistic and the devices above the threshold may not contain all the required components. Thus, we may need to loosen our threshold value. Assuming that the ranking of the devices results in a normal distribution, we may bring down the threshold by one standard deviation [9] to consider even more devices.

The important assumption here is that the devices will be ranked as a normal distribution. This is a very strong assumption and considering the hardware and software heterogeneity of devices, the distribution may likely be a random one; however, since user preferences are same for all the devices, i.e., all devices are ranked using same set of criteria, our experimental results showed that depending upon the user preferences, a quasi-normal distribution was noticed most of time, with a little skew in the mean value.

This does not always guarantee that the selected threshold value will work in most of the circumstances. We believe that further investigation into this problem is needed to select the optimal threshold value: either a fixed, pre-defined value or one that is determined dynamically based on the device capabilities and the distribution of components on them. However, we do not treat this problem as a part of this thesis and leave it for future work.

4.10 Evaluation

We have evaluated several aspects of our implementation. However, since there is no existing system which considers user preferences for the task composition, as we do here, we cannot compare our results with other approaches. Thus, we present here only a few results to demonstrate the efficiency and performance of our implementation.

Since the number and types of devices in pervasive environments can be dynamic as well as the number of components on them, we cannot determine a particular scenario for evaluation of our experiments. Thus, we begin our experiments with a few assumptions. First, the number of services in the user task, the number of devices in the environment and the number of components on these devices cannot be very large. Normally, a user task may not contain more than a few components. Similarly, the number of simultaneous devices available to a user in a pervasive environment may not be very large in most of the cases. Also, the number of components on different devices may be different and there will be no correlation between the components on these devices. Thus, we assume a random distribution of components on up to 10 devices. Based on these assumptions, we run the following experiments. These experiments have been run on a DELL Latitude

9. One standard deviation above or below the mean value is about 34.1%. Since we are considering all the devices above the mean value, using the new threshold about 84% of devices will be included.

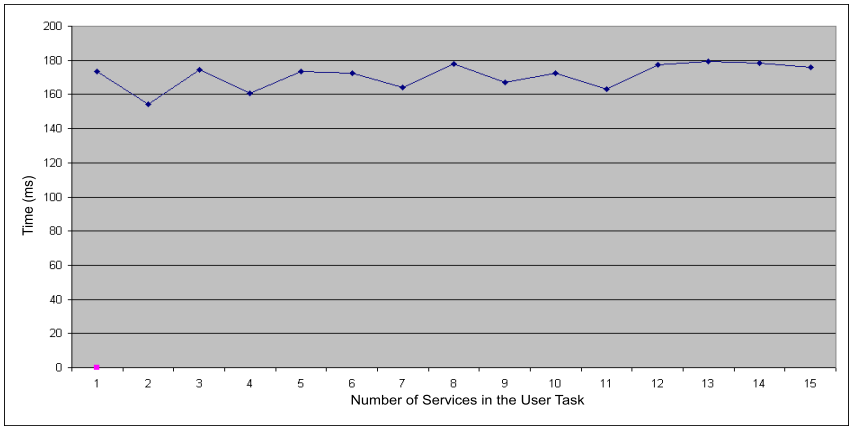

Figure 4.9: Complexity of the user task: increasing the number of services in the user task from 1 to 15

D810 laptop with 1.7 GHz Intel processor and 1 GB of RAM. All times were recorded as average of 10 runs of simulations.

Complexity of the user task

One aspect of the evaluation was to see the scalability of our approach in terms of the services in the user task. In the first experiment, we wanted to evaluate the time required to compose user task with varying number of services. Figure 4.9 shows the resulting graph for a user task when it contains from 1 to 15 different services. This includes the time to create the abstract user task graph from a parsed description, creation of the network and aggregate graphs, as well as matching of the abstract and aggregate graphs. As it can be seen, the variation between the times required to compose the tasks does not vary much in all the cases. The reason is that for all the cases, since the number of devices and components is the same, the time required to create the aggregate graph does not change very much. Hence, we can conclude that increasing the number of services does not have significant affect on the performance of the composition process. It will mainly depend on the type and number of devices and the components available on them.

Size of the aggregate graph

Figure 4.10 explains how, by adding service requirements and user preferences, the complexity of the aggregate graph can be reduced. These experiments have been run on the same setup as explained above; however, the composition process is carried out for three different cases. In the first case, an aggregate graph was created for the user task

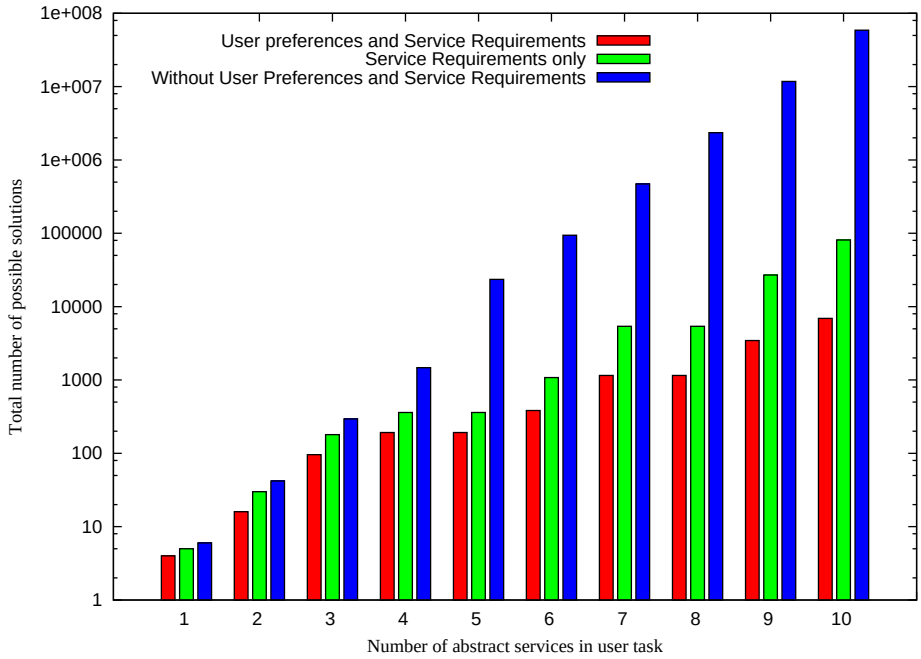

Figure 4.10: Comparison of task composition solutions: without and with considering user preferences and service requirements

without considering the user preferences and service requirements. In the second case, only service requirements were considered, while in the last case, both user preferences and service requirements were considered.

The complexity of the aggregate graph is determined in terms of the possible number of solutions by different combination of components on various devices. For example, when there is only one service in the user task the number of solutions is 4 when considering user preferences and service requirements, 5 when considering service requirements only and 6 when not considering any of them. This is because after considering service requirement some of the devices (e.g., 1 device in this case) may be eliminated. Similarly, after considering user preferences, any device that does not conform will be eliminated (again 1 devices in the example). Since the number of components in the aggregate graph increases exponentially with respect to the number of services in the abstract graph, the graph has been drawn to log scale on y-axis. It can be found that even for 10 services in the abstract graph, there are 10 times more number of solutions when considering only service requirements compared to considering both service requirements and user preferences simultaneously. Similarly, there are 1000 times less number of solutions in the aggregate graph when considering service requirements as compared to the case when they are not considered.

Thus, we can conclude that by incorporating service requirements and user preferences in the user task, the size of the aggregate graph can be reduced. However, since we are considering the most promising device for component selection with the respect to user preferences, followed by the next one and so on, we do not need to consider all the solutions. In fact, as described in algorithm 3, the matching process is fast because a

sorting is first done on the components according to device values. As soon as the first matching pair of components on one or more devices is found, it is considered in the final solution. The matching process is stopped when the components for all the services in the task are found.

4.11 Concluding Remarks

In this chapter, we described how Service Component Architecture can be used for describing *ad hoc* user tasks. We also pointed out some limitations of SCA in this regard and proposed extensions of SCA, without disturbing the SCA specifications. We proposed semantic annotation that can be attached with SCA elements so that they can be matched semantically: a feature which is not supported by the SCA specifications. We also proposed to use service collocation as an extension to SCA for specifying that a service may require to collocate with another service on a device. Finally, using the SCA policy framework, we were able to show how service capabilities can be represented as policies in SCA.

We then explained the architecture and functionality of our proposed middleware and provided some implementation details along with evaluation results. In the next chapter, we discuss how our middleware can be used, as a special case, for continuity of sessions across devices.

Chapter 5

Seamless Session Continuity Across Devices

5.1 Introduction

In the previous chapters, we discussed how a user task, defined in terms of abstract services, can be resolved dynamically into components available in the environment at the time of its instantiation and then we proposed our middleware for management of all issues related to user task instantiation. The important aspect of the task resolution was to consider user preferences vis-à-vis device capabilities. Suppose that when the user task is being executed on the selected set of components on different existing devices, a new, better device appears in the environment. The user should be able to choose the better components on the new device and execute his task with the newly selected set of components. Since the user has already executed his task partially (say, he has seen some portion of the movie), it would not be desirable to re-execute the task from the beginning. Ideally:

- The user should be able to continue his task from the point where he left previously.
- The disruption happening due to the reselection of new components should be negligibly minimal for the user
- and the selection of the new components should be done with minimal user intervention and only when the some better device is found.

Building on the concepts developed in previous chapters, this chapter treats the mobility issue that arises frequently in pervasive environments.

5.1.1 Motivation

With the abundance of devices and services, a user will use a particular device merely due to their own preferences. While these devices may offer many similar services, the capabilities of each device may differ from another in terms of its processing power, available memory, display size and battery power etc. These capabilities can be viewed by the user as the quality of service provided by the device. A user may start using one device for utilising a service but will change to another device a few minutes later due to the inadequate quality of service provided by the first device, e.g., low quality video display due to low battery power. However, the user would like to access transparently and

seamlessly his ongoing multimedia session when he changes his device. The user would also desire to transfer the session with minimum loss and with minimum involvement on his side.

Before going into the details of our contribution to tackle the above mentioned issues, first we need to categorize the problem space and then specify our position in it. This will be done in Section 5.1.2. Then in Section 5.2.1, we explain how similar problem is handled in the well-known platforms of Gaia and Aura and what are their limitations, followed by some recent approaches that have also tried to address this issue in Section 5.2. In Section 5.3, we propose our approach to handle session continuity in pervasive environments and in Section 5.4.3 we leverage our approach by incorporating user preferences in the process, thus, always keeping the user in the loop.

5.1.2 Types of Mobility

(Schulzrinne and Wedlund, 2000) has defined four types of mobilities when it is related to user session management:

1. Terminal Mobility allows a device to move between different IP subnets, while continuing to be reachable for incoming requests and maintaining sessions across subnet changes. Simple terminal mobility requires only DHCP and dynamic DNS, but this allows only being able to be reached for *new sessions* after subnet changes.

2. Personal Mobility allows addressing a single user located at different terminals by the same logical address. For example, a user may want to be reachable via a traditional PSTN phone, a PC and a wireless device at different times, while the caller may use only one logical identifier to be redirected to the current access method used by the callee.

3. Service Mobility allows users to maintain access to their services even while moving or changing devices and network service providers. For example, even if a user changes their preferred device, they will likely want to maintain their speed dial lists, address books, call logs, media preferences, buddy lists, etc.

4. Session Mobility allows a user to maintain a media session when changing from one terminal to another. For example, a caller may want continue a session begun on a mobile device on the desktop PC when entering her office.

While all of the above mentioned mobility types are desirable, in the context of this thesis we are interested in the last type of mobility: session mobility. The reason is that the focus of our work is about task composition in the *pervasive environment*, where the user movement is usually localised in a small area. This is different from terminal or personal mobility, e.g., in which we assume that user's movement can be considered in sufficiently large area involving subnet changes, etc. Works as such are mostly considered as hand-off issues in cellular networks and are out-of-scope for this thesis.

When considering session mobility, we assume that the user has already set up a multimedia session on one of his devices and then he wants to change his device in favour of another one. This requires the *seamless transfer* of user session from the former device to the latter. By *seamless* we mean that there is minimal or no disruption in the user

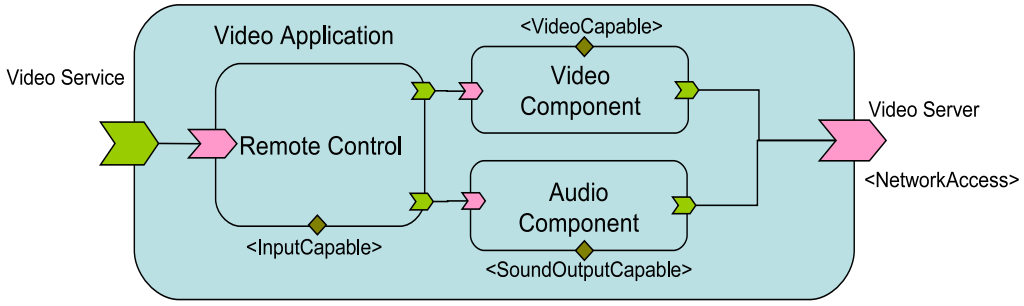

Figure 5.1: Example Video Application in SCA

session during the transfer process, at least from the user's point of view. Since this process leads to continuity of the same session across devices, we will term this process as *Session Continuity* across devices and will use it interchangeable with the term session mobility.

The approach we present here for seamless session continuity is based on Universal Plug-n-Play (UPnP). Its constraint include the fact that the visibility of a UPnP-enabled device is limited into the LAN/WAN in which the device is found but as we discussed above, when considering a pervasive environment where all devices are in user's reach, this is not a draw back to limit our purpose. On the other hand, its advantage is that the protocol is used as a standard protocol across thousands of devices from small, hand-held devices to powerful computers and other home appliances and, hence, is available widely. UPnP will be discussed in more detail in Section 5.3.

Figure 5.1 shows an example video application described in SCA. This application consists of three independent components, so they all can be executed on different devices each. Thus, we will show in this chapter, how the user session can be continued when different components are selected during the execution of the user application, due to appearance of better devices in the pervasive environment.

5.2 Existing Approaches for Seamless Session Continuity

Before presenting our approach for achieving seamless session continuity in a home network using UPnP, first the process of session migration in Gaia and Aura is explained and contrasted with our approach followed by the work carried out under the Internet Suspend/Resume project. Then, some recent approaches for seamless session continuity are discussed.

5.2.1 Session Migration/Continuity in Gaia and Aura

The Gaia application framework (introduced in Section 2.2.2) provides support for both inter and intra-space mobility. Intra-space mobility is implemented as a library that interacts with the Middleware Operating System to create and terminate components, and with the Coordinator to attach and detach new and terminated components. For example, moving a Presentation requires creating a new instance of the Presentation, attaching it to the application via the Coordinator, and terminating the original instance.

Inter-space mobility is implemented by a service (Mobility Service) that reuses the application management suspension and resumption methods. The service interacts with the Middleware Operating System to detect people leaving and entering the space. When a user leaves, the service obtains a list of associated applications and suspends them. Then, when the user enters an Active Space, the service resumes the suspended applications.

In the Aura project (introduced in Section 2.2.1, this process can be elaborated as migration of the user task from one device to another or as the continuity of the same task on the user device if reconfiguration of component happens due to change in QoS, etc. In both these cases, the state of the executing task is stored in Aura file space and after successful reconfiguration it is retrieved and associated with the newly executing task.

Both Gaia and Aura support suspension and resumption of sessions when the user moves from one pervasive environment (active space) to another. In both of these cases, a reconfiguration of components takes place after which the task is re-initialized. However, while in Aura the state of the task is preserved, in Gaia, no state is maintained and when the user moves to another active space, their task is re-initialised as a new session. In addition, Aura also supports the notion of session continuity by allowing to continue the task with the same state after its reconfiguration. The problem, however, is that Aura depends on the centralized file space for saving/retrieving state of the user task. This become a bottleneck for *ad hoc* user tasks, which can be instantiated on-the-fly, but which do not require any a priori knowledge of a central entity as required for the case of Aura.

5.2.2 The Internet Suspend/Resume Project

To support session continuity across a variety of hardware and software platforms, the Internet Suspend/Resume (ISR) project at Carnegie Mellon University (Satyanarayanan et al., 2007) cuts the tight binding between PC state and PC hardware. By layering a virtual machine on distributed storage, ISR lets the VM encapsulate execution and user customization state; distributed storage then transports that state across space and time. The motivation behind ISR is that the falling hardware costs will someday eliminate the need to carry computing environments in portable computers. Instead, ISR will deliver an exact replica of the last check-pointed state of a user's entire computing environment (including the operating system, applications, files, and customizations), on demand over the Internet to hardware located nearby.

The vision of the project is to enable the usage of a user's preferred OS and applications on demand. The approach relies on a fixed infrastructure for supporting mobile users, relieving mobile users from the need of carrying computing equipment or the planning of what resources might be needed. Their approach is to capture and serialize both runtime state of the OS running on the machine and any user-perceived state or preferences. This state might then be reinstated at another terminal, thus providing mobility both over space and time. Heterogeneity is handled through the use of a virtual machine. Regarding utilization of resources found in the surroundings of a user, there is no distribution. State management is employed at the level of the OS. The ISR project is implemented on top of the Linux kernel and is available for free download [1].

1. At http://isr.cmu.edu/

Although ISR can be successfully deployed on Linux kernels, it is not yet known whether it can also be deployed on small devices with Linux kernels having limited capabilities. If it cannot be executed on small devices, its utilisation would be much restricted vis-à-vis device heterogeneity in pervasive environments. Another important issue is the dependence on the distributed storage facilities and their availability in the underlying infrastructure, which is not always possible in the pervasive environments.

Thus, as the reader can observe, while the above cited approaches provide the possibility to the user to resume the task after reconfiguration or state saving as part of a check-point, they do not support *seamless session continuity of ad hoc user tasks across devices in the pervasive environment*. In the following sub-sections, we describe some of the recent approaches for seamless session continuity in pervasive environments.

5.2.3 IP-based Approaches

IPv4 and IPv6 do not have direct support for session mobility, so solutions have been invented for this purpose, as in Migrate (Snoeren and Balakrishnan, 2000), which has proposed a service migration approach based on IP. However, an IP-layer approach is not sufficient for session mobility for many reasons. First, an IP-layer approach is not suitable for handling application-level information that is considered for deciding session mobility issues, such as the device capabilities and resources-availabilities as well as the states of applications. Second, an IP-layer approach imposes a restriction on node movement, because such an approach assumes the existence of certain fixed nodes. Finally, implementing an IP-layer approach requires modifications to the OS and dependence on it.

5.2.4 Socket-based Approaches

Some approaches have used socket level manipulation for session mobility. (Ohta et al., 2003) utilizes Virtual Sockets, on top of the real ones, in order to resume transmissions after a session hand-off. Both (Kaneko et al., 2003)(Ohta et al., 2003) have added a middleware under application layer to maintain the transport information (IP address and port) of the media sessions. The problem with such solutions is installation of the additional middleware which is not desirable, or even applicable, for many devices in the home network.

5.2.5 Proxy-based Approaches

(Takasugi et al., 2003) has proposed a solution for seamless service continuity by introducing a S-proxy (seamless proxy) on each terminal. An application uses one of the available services in the network of S-proxies. The data exchange between applications takes place via the S-proxies, which lie above the transport layer. The S-proxy model itself consists of four layers. The S-session layer, which is the topmost layer, links a given service with another service. The S-connection layer is for communication between applications. The S-path layer manages the path for data exchange between applications involving two or more S-proxies. The S-link layer communicates with the underlying TCP/IP layer. When a user changes one device for another, the node address and the proxy ID changes.

This translates to changes in S-path and S-link layers without changing the S-session layer, maintaining the same session.

(Bellavista et al., 2003) has proposed a proxy-based middleware for service continuity in mobile ad-hoc networks. The middleware is based on application-transparent proxies that act as decoupling intermediaries between the moving clients and servers. The proxy role is assigned dynamically in a completely decentralized way. Proxies exploit code mobility to install only when and where needed. In order to hide the localisation issues, all service messages are automatically and transparently sent through the proxy, acting as a bridge between the clients and the servers. Instead of querying a server directly, the clients communicate with the proxy which communicates with any of the available servers. If a server goes out, the proxy searches for a new server and continues the communication with the new server. For the client, this change of server is transparent. This approach is based on agent-based communications and involves dynamic code installation.

In both of these approaches, manual installation of the proxies is required, i.e. a nomadic user, with no knowledge of the proxies, cannot take advantage of such solutions. The drawback of proxy-based solutions is that for each possible application the system requires an application-specific module to handle session mobility. Also, the user data flow between two calling parties must always traverse the proxy, regardless of whether session migration is desired or not, introducing triangular routing.

5.2.6 SIP-Based Approaches

The Session Initiation Protocol (SIP) (Rosenberg et al., 2002), an IETF standard, is an application-layer signaling protocol for creating, modifying, and terminating sessions with one or more participants. Recently, it has been used for session mobility across devices in different ways (Schulzrinne and Wedlund, 2000). The "cleaner approach", according to authors, is as follows. Bob is in a session with Alice on his mobile phone (bob@mobile) and he wants to move the session to a fixed host (bob@fixed). Here, bob@mobile simply sends a REFER request to Alice, indicating that she should contact bob@fixed. Alice then negotiates a session with bob@fixed using the SIP INVITE call (Alternatively, Bob can also send a SIP REFER request to bob@fixed, asking her to invite Alice). Bob's session with mobile phone is terminated afterwards.

(Kaddour et al., 2006) proposes an enhanced procedure for session mobility using SIP. The architecture defines an Access Portal which contains the core components for user identification and authentication, presence management, service enabling, session management, and user profiling. The architecture supports both direct session mobility and session transfer through an intermediary host. The sessions can also be suspended and resumed using different devices. SIP has also been used for splitting session over multiple devices (Chen et al., 2007)(Shacham et al., 2005). SIP-based approaches are not suitable for home networks as SIP does not support generic multimedia control functionalities such as play, pause, etc.

Existing work related to use of UPnP for multimedia purposes in home networks focuses on multimedia delivery and adaptation (Liao et al., 2007)(Sung et al., 2006) and, to the best of our knowledge, the only work that treats continuity of session using UPnP was proposed by us (Mukhtar et al., 2007b)(Mukhtar et al., 2009a).

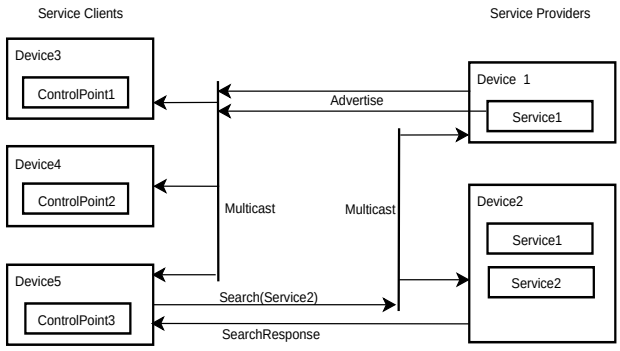

Figure 5.2: UPnP devices, control points and services

5.3 Session Continuity Using UPnP

All of the above described approaches support session continuity at different levels of abstractions. The IP- and socket-based approaches are applicable at network level. The proxy-based approaches are applicable mostly at the middleware level, while the SIP-based approaches apply at the application level. The advantage of the network level and middleware level approaches is that they are application-independent. However, on the downside they require the lower levels to be aware of changes at application level to cope for changes giving rise to the session continuity. SIP-based approaches have the disadvantage that SIP requires the availability of server for taking care of device registration and de-registration and is not suitable for ad hoc environments.

To overcome all of these limitations, we have used UPnP for session continuity. The advantage of UPnP is that is well-suited for pervasive environments, it does not require the presence of any centralised entity and a UPnP-enabled device does not require any configuration for participating in the environment. Thus, UPnP provides an ideal solution for session continuity in pervasive environments.

The approach we have taken for session continuity is based on the UPnP A/V Architecture (UPnP Forum, 2006), which describes the communication protocols for an A/V source device and an A/V rendering device to share multimedia contents in a UPnP-enabled environment. The existing UPnP A/V Architecture specifications do not describe any mechanisms for session continuity. However, taking advantage of the fact that the specifications are broad, device-independent, and support a wide range of messaging between different A/V entities, they can be easily applied for session continuity purpose.

Another important aspect is the splitting of session across devices. Assuming that a certain user session can be modelled as a combination of two or more channels (such as audio, video, and text), UPnP A/V architecture can be used to split the session in its constituent channels. The principles for session splitting are same as that for session continuity.

In Section 5.3.1 we discuss UPnP and in Section 5.3.2 the UPnP A/V Architecture.

5.3.1 Universal Plug-n-Play

The scope of UPnP is large enough to be suitable for a truly pervasive environment in many existing as well as new and exciting scenarios including home automation, printing

and imaging, audio/video entertainment, and comprising devices such as kitchen appliances, handheld devices like PDA's and mobile phones, as well as the existing computing environment. Advantages of UPnP are that it is built on top of TCP/UDP and uses SOAP messages (Simple Object Access Protocol. SOAP v1.2 Specifications,) over HTTP for communication. Hence, it is independent of any operating system and programming environment and can be deployed without modifying the existing systems.

With UPnP, a device can dynamically join a network, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices. A device can be a UPnP server by offering services, or a UPnP client by requiring services, or both at the same time.

Figure 5.2 shows a network of UPnP-enabled devices. Each device exposes a set of services and may embed other devices exposing yet some other services. Whenever a device is brought into a network, it announces its services using multi-cast packets in the network. The device can also provide a presentation page which can be accessed by a user via a browser application to see the device's properties and services. Apart from this, each device is able to query other devices for their capabilities and to execute actions on it. A UPnP device can subscribe to services provided by another UPnP device.

5.3.2 UPnP A/V Architecture

The UPnP A/V architecture (UPnP Forum, 2006) defines three entities: a Media Server, a Media Renderer, and a Control Point, which are used together for controlling and sharing multimedia contents across various devices regardless of the device, content format or transfer protocol. All three entities are considered as if they were independent devices on the network, but the A/V Architecture supports arbitrary combinations of these entities within a single physical device (e.g. a device having both the Media Render and Control Point). The relationship between the devices is shown in fig. 5.3.

Media Server

The Media Server is used to locate the multimedia content that is available to Media Renderers. Media Servers include a wide variety of devices such as VCRs, DVD players, satellite/cable receivers, TV tuners, radio tuners, CD players, audio tape players, MP3 players, PCs, etc. The Media Server contains 1) a `ContentDirectory` service which provides a set of actions that allow the Control Point to enumerate the content that the Server can provide to Renderers, 2) a `ConnectionManager` service to manage the A/V connections associated with a particular device and 3) an optional `AVTransport` service that is used by the Control Point to control the playback of the content that is associated with the specified A/V transport.

Media Renderer

The Media Renderer is used to render (e.g. display and/or listen to) the contents provided by the Media Server via network. Examples include a wide variety of devices such as TVs, stereos, speakers, hand-held audio players, etc. The Media Renderer, like Media Server, has a `ConnectionManager` service and an optional `AVTransport` service to control the flow of the content (e.g. Stop, Pause, Seek, etc), but instead of the `ContentDirectory`

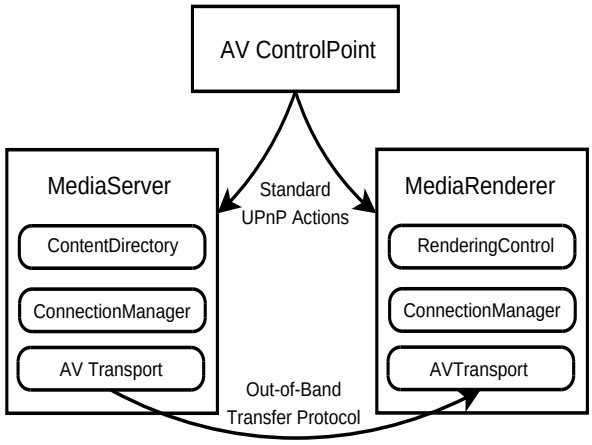

Figure 5.3: UPnP playback architecture

service it includes a `RenderingControl` service. This service provides a set of actions that allow the Control Point to control how the Media Renderer renders content by modifying characteristics such as brightness, contrast, volume, mute, etc.

Control Point

The Control Point is the only component that initiates UPnP actions, usually in response to user interaction with the Control Point's UI. The Control Point requests to configure the Media Server and Media Renderer so that the desired content flows from the Media Server to the Media Renderer, using one of the transfer protocols and data formats that are supported by both the Media Server and Media Renderer. The Control Point is capable of controlling the flow of the content by invoking various `AVTransport` actions such as Stop, Pause, FF, REW, Skip, Scan, etc. Additionally, the Control Point is also able to control the various rendering characteristics on the Renderer device such as brightness, contrast, volume, balance, etc. Example of Control Point is a remote control for TV, VCR, CD-Player etc. In the next section, we will show how a Control Point is used to initiate the transfer of session across devices.

While the UPnP A/V Architecture specifies UPnP as a protocol for controlling devices, it does not specify any data format or protocol for actual transfer of contents between the devices. This can be any of the standard formats (such as MPEG2, MPEG4, JPEG, MP3, Windows Media Architecture (WMA), Bitmaps (BMP), etc.) and protocols (such as RTP, HTTP GET/PUT/POST, TCP/IP, etc.) or even a vendor-specific format/protocol.

5.4 Session Continuity and Splitting Using UPnP A/V Architecture

Our research team has also treated the issue of session continuity in the context of Session Initiation Protocol (SIP) (Kaddour et al., 2006). While SIP is becoming a significant protocol in telecom services, it is not designed to be used in home network appliances.

Being a signaling protocol, SIP does not support controlling devices for functions such as play, pause, etc., unless it is accompanied by a gateway such as the Open Services Gateway Initiative (OSGi)(OSGI, 1999) (see, for example (Brown et al., 2006)(Latvakoski et al., 2002)). UPnP, on the other hand, has already been supported by many home and office devices such as printers, cameras, and media players and has also been introduced in various mobile phones recently. That is why a UPnP-based approach for session continuity is more appealing and feasible for home networks.

5.4.1 Push vs. Pull Transfer Protocols

The UPnP A/V architecture supports both isochronous-push transfer protocols (e.g., IEC61883/ IEEE1394) and asynchronous-pull transfer protocols (e.g. HTTP GET). In the first case, the underlying transfer mechanism provides real-time content transfer between the Media Server and Media Renderer. This allows the Media Renderer to provide the user with smooth rendering of the content without implementing a read-ahead buffer. The pull transfer protocols, on the other hand, do not provide real-time guarantees and a read-ahead buffer is required at Media Renderer. Algorithms for both these approaches do not differ significantly and can be found in the specifications (UPnP Forum, 2006).

Figure 5.4 shows the interaction diagram for session continuity across devices using UPnP and a push transfer protocol. While the UPnP A/V Architecture is designed to support arbitrary transfer protocols and data formats, we assume that each of the Media Server and Media Renderer devices is able to provide the same communication protocol for the content exchange as well as a standard data format that is understandable and render-able by both the renderers properly. This is an important and valid assumption to consider because, otherwise, the devices will not be able to share multimedia contents. The coordination between these devices takes place as follows:

1. Discover A/V devices initially only the Media Server and the Media Renderer1 are present in the local environment. The Control Point discovers both the devices using SSDP (Simple Service Discovery Protocol), used by UPnP for discovery of devices. The user is able to interact with both the devices using the Control Point's UI.

2. Locate desired content the user searches for the desired content on Media Server using `Browse()` or `Search()` actions. These actions also return the transfer protocol and data format of the media content supported by the server.

3. Get renderer's supported protocol/format the Control Point uses Media Renderer's `GetProtocolInfo()` action to get a list of transfer protocols and data formats supported by the Media Renderer.

4. Choose matching protocols/formats the protocol/format information returned by the Media Server for the desired content item is matched with the protocol/format information returned by the Media Renderer's `GetProtocolInfo()` action. The Control Point selects a transfer protocol and data format, which are supported by both the Media Server and Media Renderer.

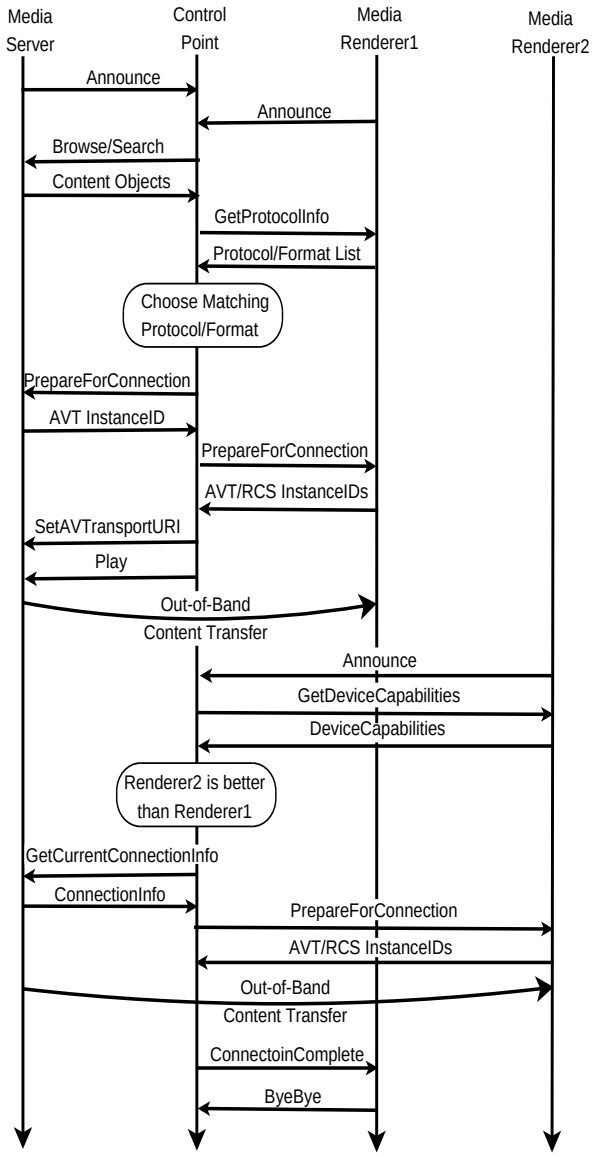

Figure 5.4: Transfer of session between two Media Renderers

5. Configure server/renderer the `PrepareForConnection()` action on each device specifies that a connection is about to be made using the specified transfer protocol and data format that was previously selected. This methods takes four arguments as input: 1)`RemoteProtocolInfo` that identifies the protocol, network, and data format used to transfer the content, 2)`PeerConnectionManager` identifies the `ConnectionManager` service on the remote side of the connection, 3)`PeerConnectionID` that identifies the specific connection on that `ConnectionManager` service; this information allows a Control Point to link a connection on Media Server to the corresponding connection on Media Renderer, 4)`Direction` specifies "output" for the Media Server and "input" for the Media Renderer. The Media Server returns an `AVTransport` InstanceID to control the flow of the content (e.g. Play, Stop, Pause, Seek, etc). The Media Renderer returns a `RenderingControl` InstanceID that is used by the Control Point to control the Rendering characteristics of the content.

6. Start content transfer the Control Point invokes the `SetAVTransportURI()` on Media Server to identify the previously selected content item. Upon the invocation of `Play()` method, the content transfer between the Media Server and the Media Renderer begins. The protocol used is the one identified previously. The user can then use the Control Point to adjust attributes of the Server or Renderer during playback process.

7. Announcement of the Renderer2 device the user notices the availability of another device, Renderer2, on his Control Point. This happens when the user turns on the new device or when he moves to a nearby environment where the new device is detected by the Control Point. The user interrogates the new device's capabilities and finds it to be better than Renderer1. The user decides to transfer his session to Renderer2.

8. Transfer the active connection the Control Point invokes `GetCurrentConnectionInfo()` on Renderer1 to obtain the current connection attributes, which contain the ConnectionID of the Media Server among other attributes. The Control Point then invokes `PrepareForConnection()` action on Renderer2. This action takes as arguments the `ConnectionID` of the Media Server and the `Direction` of media (specified as "Input"). Renderer2 has now the same set of attributes as used by Renderer1 for its connection with the Server, and the transfer of content begins between the Server and Renderer2. At this point, the Control Point has `InstanceIDs` of both the Renderers.

9. Connection completion the Control Point invokes `ConnectionComplete()` action on both the Media Server and Renderer1. This action is used to inform the devices that the specified connection, which was previously allocated by `PrepareForConnection()`, is no longer needed. Any resources that were allocated for that connection during `PrepareForConnection()` are freed by the device. The Media Server removes Renderer1 from its active connections list. Renderer1 also frees the resources used by its connection with the Media Server. Renderer1 can be, optionally, turned off by the user which then sends *ByeBye* message to Control Point indicating its removal from the Control Point's active devices list. The user now only communicates with Renderer2 and Media Server for the rest of his session.

Session continuity using pull transfer protocol can be carried out similarly. For example, the Control Point will invoke the `SetAVTransport` and `Play` actions on the Media Renderers instead of the Media Server.

5.4.2 Session Splitting

Session splitting can be achieved using a procedure similar to session continuity. For example, in figure 5.4, when a new device is announced and a need is felt for split of a session, e.g., based on user's initiative, the `GetCurrentConnectionInfo()` action can be invoked to know about the session attributes. Depending upon the implementation of the content transfer protocol, the various channels of the session can be determined and it can be decided which channel to transfer from one device to another. Thus, only selective channels can be transferred from one device to another resulting in splitting of the session across devices.

5.4.3 Session Continuity Based on User Preferences

So far in this chapter, we have discussed how UPnP can be used for session continuity of multimedia applications, as shown in the sequence diagram of fig. 5.4. The diagram shows the session transfer is initiated after it is found that Renderer2 is better than Renderer1 (and user agrees to initiate the session transfer), however, it does not show *how* to determine which renderer is better.

As one would expect, this decision is made on the basis of user preferences parameters as proposed in (Mukhtar et al., 2009b). This can be achieved by updating the preference tree to reflect the changes as new devices appear. For example, in Section 3.6.6, we explained how a preference tree can be created on the basis of available devices in the pervasive environment. Suppose that once the tree is created, a new better device is announced in the environment. Instead of re-creating the tree to accommodate the changes, the newly discovered device is added into the existing tree, thus, requiring only changes in the selected branches, which are affected by the addition. This is illustrated with the help of an example scenario described in Section 5.5.

5.5 Example Scenario

Consider that a user named Bob wants to watch a movie from his collection of movies in his portable hard disk. However, right now the children are watching cartoon on the desktop and his wife is using his laptop for checking her emails, so he uses his mobile phone to watch the movie in another room of his house. The mobile phone is receiving the audio/video stream from the external hard disk via an *ad hoc* wireless connection. A few minutes later, the children leave to play outside and leave the desktop. The video application on Bob's mobile phone notifies him about the availability of the desktop and suggests him that since screen of the desktop is better than screen of the mobile phone; he can now watch the rest of the movie on the desktop. In addition, it also suggests that based on his preferences, he can use the mobile phone for controlling the audio/video instead of using desktop's control buttons. While he is heading to the other room for desktop, he accepts the notification to redirect the audio/video stream emanating from

Table 5.1: Bob's preferences for device capabilities

Variable	Preference value
NetworkAccess	Yes ↦ Very important No ↦ Avoid
AccessCharges	Yes ↦ Dislike No ↦ Like
OSName	Linux ↦ Important Windows ↦ Somewhat like
ScreenResolution	OSName↦Linux⇒ ScreenResolution ↦ ClosestMatch(1280x800) OSName↦Windows⇒ ScreenResolution ↦ Maximum()
VideoPlayer	OSName↦Linux⇒ VLC ↦ Important OSName↦Windows⇒ MediaPlayer ↦ Important
InputType	TouchScreen ↦ Important Voice-command ↦ Slightly Important
Constraints	AccessCharges ≻ OSName AccessCharges=Yes ⇒ SecuritySupport ↦ Very Important

Table 5.2: Device Capabilities

Capabilities	SP	DT	LT
NetworkAccess	Yes	Yes	Yes
MediaPlayer	Real	VLC/Quicktime	Real/WMP
ScreenResolution (wxh)	640x360	1280x800	1600x1200
InputType	Keypad/Voice	Mouse/Keyboard	Mouse/Keyboard

the disk to the desktop with the help of a convenient user interface provided by the application. However, the playback of audio/video is still controlled by the mobile phone.

To illustrate how user preferences can be used in this scenario, consider some of Bob's preferences that can be specified as following and are summarized in table 5.1:

1. Since the movies are stored in the portable hard disk attached to the home network, to access them a device must have network access capability. This criterion should be absolutely met, otherwise the device cannot be used for watching movie. This can be specified as $NetworkAccess = \{Yes \mapsto Very\ Important, No \mapsto Avoid\}$.
2. Bob is also selective about the media player: he likes the freely available VLC media player because of its rich set of feature; however, if it is not available then he would prefer Real player to the rest. So he assigns $VideoPlayer = \{VLC \mapsto Important, Real \mapsto SomewhatLike\}$.
3. Since watching movies on a big screen is more appealing than watching them on smaller screens, Bob assigns $ScreenResolution \mapsto Optimum()$.
4. Finally, to interact with a device with a pointing device is more interesting than using a keyboard. Thus, Bob specifies $InputType = \{PointingDevice \mapsto Important, Keyboard \mapsto Like\}$.
5. Bob also specifies an additional condition: if two devices have the same final weights for given preferences, he would prefer a device that has a better media player rather than a device that has a higher screen resolution, i.e., $MediaPlayer \succ ScreenResolution$.

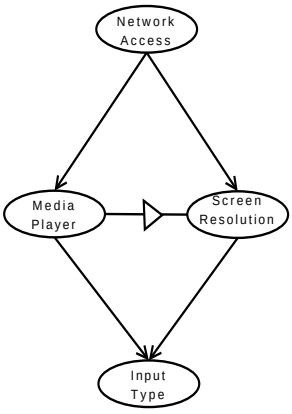

Figure 5.5: The TCP-net for Bob's preferences

Figure 5.5 shows the TCP-net obtained based on Bob's preferences shown in table 5.1. Note that the different types of edges in the graph do not correspond to the original definition of TCP-nets; for example, the arrows do not represent the dependencies but they represent the order in which the preferences are specified. The edge with a triangular symbol represent the relative importance between the *MediaPlayer* and *ScreenResolution* attributes as specified by the constraint in table 5.1.

The capabilities of smartphone (SP), desktop (DT) and laptop (LT), present in Bob's home are shown in table 5.2. Given both Bob's preferences and the devices' capabilities, we are now able to construct the preference tree. In the example scenario, when the desktop becomes available and is compared with Bob's preferences, the resulted tree is shown in fig. 5.6(a). Comparing it with the TCP-net of fig. 5.5, the first node of the tree is the *NetworkAccess* feature after which the *MediaPlayer* feature is evaluated because it is more important than the *ScreenResolution* feature according to the TCP-net. Note that the comparison is made only between those devices which have network access. Had there been any other devices without network access, they would have been avoided for selection in the very beginning due to user preference $NetworkAccess = No \mapsto Avoid$ of table 5.1. Alongside each node, we have also specified the value of each device as it is evaluated according to its capabilities. At each next node, the device value is updated according to a feature's value and its preference weight assigned by the user. At the bottom of the tree, the cumulated device values are shown in the rectangles. As shown, DT has higher device value and, hence, shown to Bob as a better device for Media Renderer. The *Optimum*() function was defined in Section 3.6.5.

Now let's suppose that in the above scenario, when Bob's wife finishes her work on the laptop she leaves it. When the laptop becomes available, its capabilities are compared with existing devices as shown in fig. 5.6(b). But since the device value of laptop is not greater than that of desktop, it is not chosen and the session continues as it is.

5.6 Implementation

For the purpose of verification, we have implemented a test-bed which allows users to transfer session from one device to another. To support streaming on Media Server and

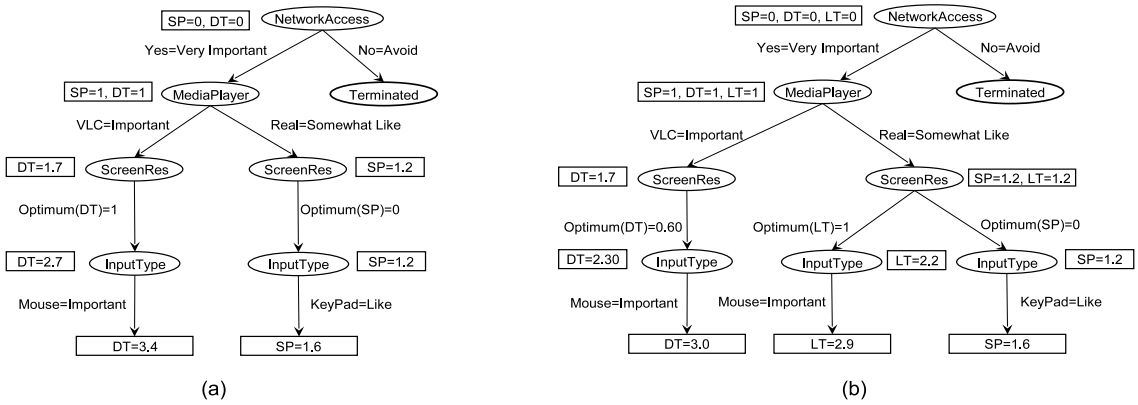

Figure 5.6: The Preference Tree for Bob's preferences for device capabilities (a) before and (b) after discovery of the laptop

Media Renderer, we have used the Java Media Framework (Sun Microsystems, 2001). For supporting UPnP on the devices, we have used CyberMediaGarage (CyberGarage, 2005) for Java, which is a reference implementation of UPnP A/V Architecture. These API's are freely available for download as open-source software. Apart from this, a Control Point has been developed for controlling the Media Server and Media Renderer as well as for carrying out the session transfer as discussed in Section 5.4. The Control Point is light-weight and can be executed on small devices, like PDA, however, it has not yet been tested on small devices.

RTP/UDP is used as the format/protocol between the Media Server and Media Renderers. The target renderer for a RTP stream is specified as (IP address, port) pair, which is added to Media Server's active connections list. The user chooses source and target renderer devices using the Control Point and triggers the session transfer action. The Control Point then carries out the actions on each device as mentioned in figure 5.4. Before the session transfer, Media Server has only Renderer1 in the active connections list. During the transfer of session from Renderer1 to Renderer2, Media Server adds Renderer2 to its active connection list. However, as soon as Renderer2 requests to start receiving the stream, Renderer1 is removed from the connections list and its session is terminated. This helps prevent any loss of packets on the renderer's side during session transfer.

In case the Media Server does not support more than one active connection at a time, it is possible that some packets are lost due to the time required to terminate the old connection and establish a new one. In such a case, either a server-side buffer will be required to solve the problem or a suspend-resume method can be used to temporarily pause the content transfer while the new connection is being established, and resuming it when the connection is completed.

There are a total of 15 classes in our implementation with a total size of 156 kb. In addition the size of the CyberLink Media Garage API is about 257 KB and that of JMF executable jar files is about 2.4 MB.

Figure 5.7 shows the execution of our application, describing the UPnP Receiver, Transmitter and Control Point components. The Remote Control has detected the audio/video receivers and transmitter available in the environment as shown in the devices

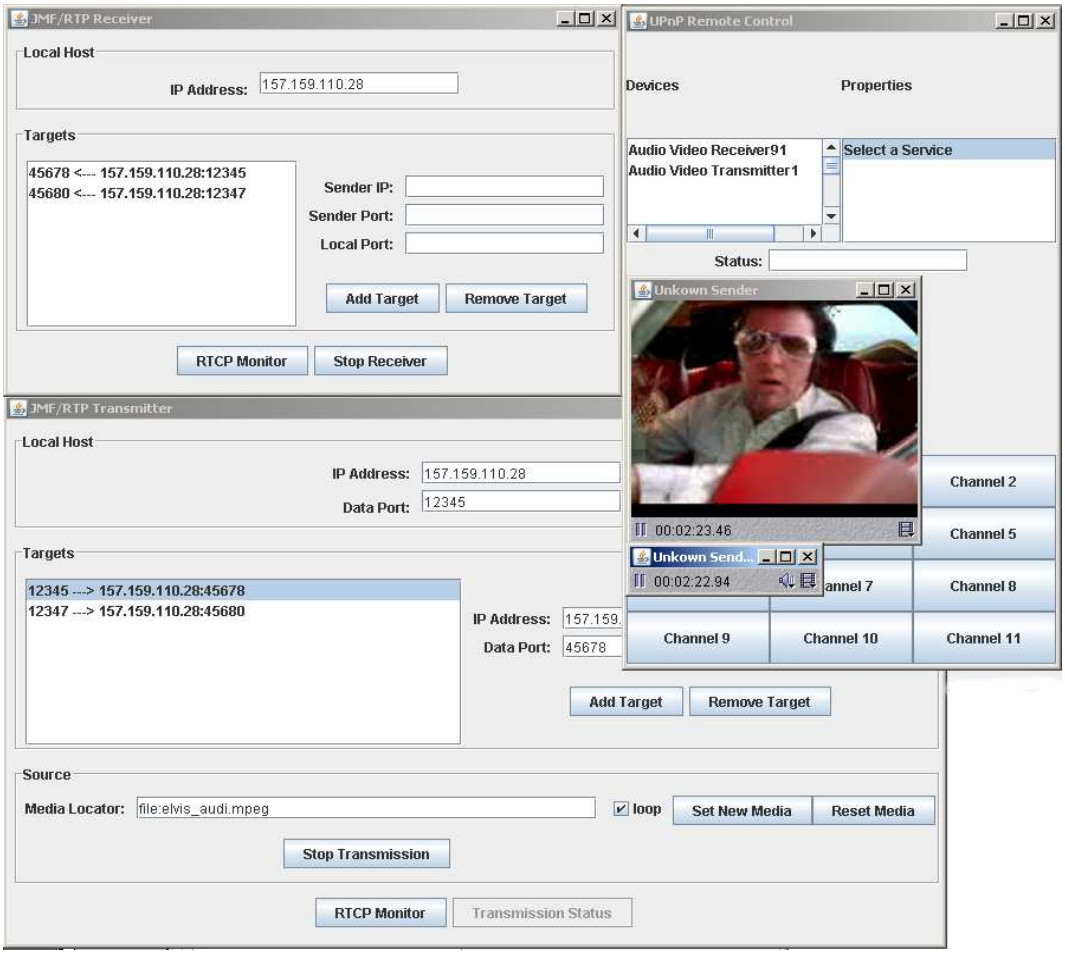

Figure 5.7: Screen Capture Showing User Session Before Transfer of Session

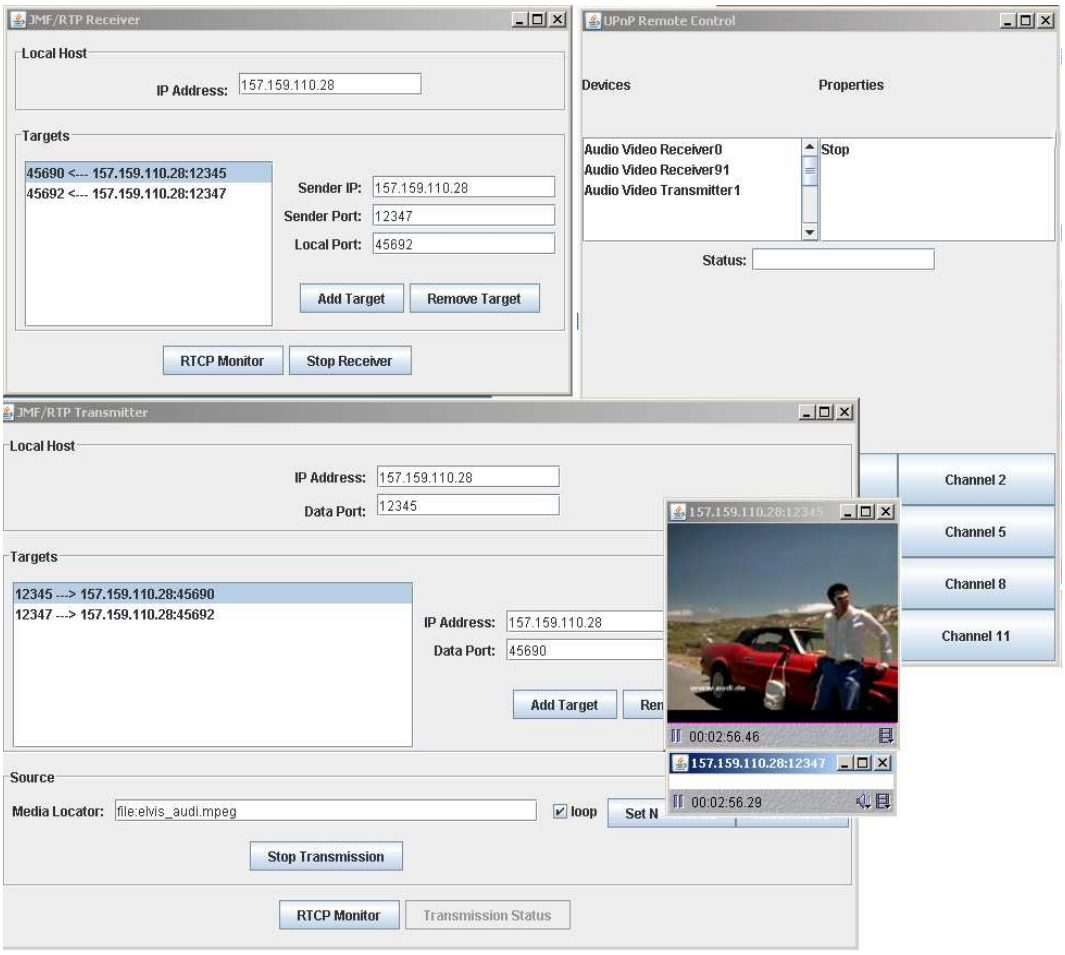

Figure 5.8: Screen Capture Showing User Session After Transfer of Session to a New Device

list. As can be seen the audio and video are shown in different windows because they are implemented using different components. They are processed separately, and that is why, as one can observe, the clocks on these windows are not synchronised exactly. However, this lag of synchronisation was caused due to initialisation issues of Java related to each component. The processed audio and video are in complete synchronisation and the user feels no noticeable difference.

Figure 5.7 shows the availability of another receiver in the user environment, as can be seen in the devices list on the Control Point. The user executes the transfer of session action on the remote control and the session is transferred to the new receiver. This can be verified by the changed port number in the new receiver.

Our implementation currently does not consider any user preferences. In the future, we will add this capability into the session continuity module of our middleware.

5.7 Concluding Remarks

In this chapter, we presented our approach for seamless session continuity across devices in pervasive environments using UPnP. The approach was illustrated for a particular type of user task (multimedia applications) and the limitations of UPnP were also discussed. However, since UPnP is an extensible protocol, it can also be used for session continuity for any type of user task once it has been defined what constitutes a session for the task.

Chapter 6

Conclusions

As the number of devices around us is increasing exponentially, a typical user may have access to several such computational devices at the same time. This has led to the creation of pervasive environments where devices present in a particular location can communicate with one another. This has also given rise to the concept of service-oriented pervasive environments where devices share their hardware and software resources with each other in the form of services.

One particular research trend is to engineer such applications that take full advantage of the devices and resources available in such environments. Such applications are designed on the principles of Service Oriented Architecture (SOA), which emphasizes on the usage of services as building blocks for applications. Such applications are constructed using service provided by small, manageable, and independent components available on different devices in the pervasive environments. These applications are more flexible as they able to adapt themselves according to services available in the environment.

Further enhancement to this trend is brought about by modelling daily user tasks as a coalition of services. The premise is that user tasks can be identified as consisting of several activities, each one of which can be represented as a service. The objective is then to find the components in the environment, which offer these services, for realization of the user task. The advantage is that using this approach tasks can be realized on dynamically, the fly in terms of the components available to the user in the current pervasive environment. As an example, when a user who is watching a movie on their mobile device, with limited resources such as screen size and battery power, enters a pervasive environment and there they find a large display screen, the user would prefer to use the display screen instead of the mobile device for watching rest of the movie.

There are many sub-problems and challenges within the broad goal of dynamic user task composition that must be solved and the objective of this thesis is to propose a framework that solves most, if not all, of such challenges. These challenges include the heterogeneity of environment in terms of hardware, software, and networking technologies, the methodology to use for selection of components, the engineering of the user task as abstraction of services, to hide the heterogeneity aspects of the environment, and the consideration of user preferences for particular devices and components.

6.1 Contributions

There are a number of existing approaches that have proposed various solutions for most of these challenges. In order to distinguish our work from that of the others, we have clearly identified several requirements and have tried to propose a unified solution as our answer to these requirements. Our important contributions in this regard can be summarised in the following.

We consider a user task as an abstract description of requirements for hardware and software components, which are then sought in the pervasive environments. However, existing systems have not considered or solved several issues that we have tried to achieve in this thesis.

First, when selecting components for realization of the user task, different users may have different preferences related to devices and the components when executing the same task. In our proposed solution, we consider user preferences in priority. When selecting components, we select those devices in priority which satisfy maximum of the user preferences. Moreover, user preferences can be positive (liking) or negative (dislike); our solution considers both types of preferences and, based on a formal model, tries to achieves a solution that is the best compromise between the two.

Second, when looking for the components required by the user task in the environment, we also consider device and user capabilities. Most of the existing approaches assume that the devices in the environment have equal capabilities (hardware, software, etc.) and are equally accessible to the users. This assumption cannot be held true in pervasive environments where different devices such as PDA's, laptops, and desktop PC's have different capabilities and resources and will have different resource usage and access policies for various users. In our solution, we have proposed to specify explicitly the resources required by the services in the user task. This way, when the user task is realised, the resources required by it are to used to guide the selection of appropriate devices, which have, or allow the usage of, the required resources.

Third, most of the existing approaches assume that all the devices in the pervasive environment can communicate using a protocol that is understood by all. However, similar to the hardware and software heterogeneity, we also have heterogeneity of communication protocols such as Bluetooth, IrDA, Wi-Fi, WiMax, etc. as well as the wired network. It will be very rare that all the devices will understand one common protocol, e.g., Wi-Fi. Our proposed solution considers such network heterogeneity and when realising the user task, components are selected only on those devices which can communicate among them.

Fourth, most of the existing frameworks develop the user task using a description mechanism that is either proprietary or are not based on open standards and they build up only on a particular architectural paradigm. This leads to architectural and description interoperability issues. In our proposed solution models user task is based on the Service Component Architecture (SCA) which is a realisation of the Service Oriented Architecture (SOA). SCA is suitable for pervasive environments because using SCA we define the user task in terms of interoperable services for which the components can be implemented in a number of technologies in a protocol-independent manner, thus providing the solution to various interoperability problems.

To address these issues, we have proposed an approach for user task composition as following: first of all, we model device capabilities and resources and propose them

as an extension of the existing W3C standard called Composite Capability/Preference Profile (CC/PP). The users can then provide their preferences for devices and components based on the same model. However, for preference elicitation and for modelling various preferences related issues (likes, dislikes, etc.) that cannot be modelled using CC/PP, we propose a preference model that is qualitative from user's point of view, and which is then translated into a quantitative model for ranking of devices and components using a proposed algorithm. The user task also defines its requirements for device capabilities so that they can be executed only on proper devices having the required capabilities.

In order to consider network heterogeneity, and to select the most promising components for the preferred devices for the user, we model both the user task and the underlying network, consisting of devices and the components available on them, as graphs. This ensures the compatibility of devices in terms of communication protocols and the mapping of user task on to distributed components using a proposed matching algorithm.

The unified solution to all these problems has been presented in the form of our middleware called MATCH-UP that stands for Middleware for *ad hoc* user Task composition in Heterogeneous environments considering User Preference. We described the architectural details and functionality of MATCH-UP and also presented some evaluation results.

Finally, one of the initial objectives of this thesis was to propose a solution for continuity of sessions across devices. This will allow the users to continue their task from one device to another to support user mobility and for better ergonomics, etc. In this regard, we also proposed a solution for user session continuity across devices using UPnP. Our current approach is limited as it is validated against only a particular protocol, UPnP, and only for specific types of user tasks: multimedia applications. However, an important aspect of this approach is the consideration of user preferences for device selection, which is not supported by existing approaches.

6.2 Future Work

Through out this thesis, we have referred to some aspects of our work that still need to be treated. We consider them as future work and outline them here. Some of these aspects can be treated in short term due to their simplicity while others require sufficient time for investigation before they can be solved, and they will be treated in long term.

Among the short term future work, the most important aspect to consider is the usability of our approach when our middleware will be deployed on devices in a truly pervasive environment. Suitably, we would like to test our framework on top of an existing SCA runtime. Our current prototype implementation, for which we provided the evaluations as well, is based on simulations. However, since the implementation has been done on the basis of MATCH-UP architecture and special care has been taken to define the programming level interfaces between the architectural components, it will require very few changes to adapt accordingly. For this purpose, we are planning to port our framework on top an existing SCA runtime. Two obvious choices are the Frascati runtime, which is based on the Fractal component model, and the Newton runtime, which is based on the OSGi platform.

One of our objectives was to be able to realise user tasks partially, if it cannot be instantiated entirely. That is, if some of the services in the user task cannot be found in

the environment, or if resource requirements for some services cannot be satisfied, then the user task will be realised only using the services available at hand. Part of the solution to this problem can be managed by the underlying component runtime. For example, in OSGi component model, services can be executed if not all of the dependencies are solved. However, we must also consider this concept at the user task level. For example, the task architect should be able to specify the components which are optional so that the task can be executed without them, if they are not found.

Another short term objective is to incorporate semantic matching in our approach. This can be done both at user task description level for selection of components, as well as at the level of user preferences and device capabilities, for selection of devices for the user. Although, we have already proposed how SCA can be enriched with semantic description, in Section 4.5.1, we have not considered them during the implementation of our prototype. Similarly, semantic matching can be used for device selection by using approaches similar to (Bandara et al., 2008). Moreover, we are also working towards the unification of both the hardware and software components in the same SCA meta-model to allow consideration of the hardware resources even more precisely.

As long term objectives, we also have several proposals to consider. First, our proposed solution for session continuity is applicable only to multimedia applications and considers only UPnP-enabled devices. We would like to extend this approach so that it is applicable to different types of applications in general, in a variety of situations. This means that the application should be adaptable and, hence, we should consider adaptation as our next step. This also leads to issues related to re-composition the user task graph. For example, to determine if re-composition of the whole composition graph is needed or only of those parts of the graph which are affected.

Finally, in this thesis, we have assumed that the user task has been previously defined as an assembly of services. However, this will not always be the case; the users should also be able to compose their task dynamically, with little or no effort, using the services available in the environment. Thus, the resources and services in the environment can be used by the users even if they do not have any pre-defined task for that.

Bibliography

Akkiraju, R. and Sapkota, B. (2006). Semantic annotations for WSDL. Technical report, W3C. (cf. http://www.w3.org/TR/sawsdl-guide/).

Bandara, A., Payne, T., Roure, D., and Clemo, G. (2004). An ontological framework for semantic description of devices. In *Semantic Web Conference, 2004. International.*

Bandara, A., Payne, T., Roure, D. D., Gibbins, N., and Lewis, T. (2008). A pragmatic approach for the semantic description and matching of pervasive resources. *Advances in Grid and Pervasive Computing*, pages 434–446.

Belaïd, D., Mukhtar, H., and Ozanne, A. (2009a). Service composition based on functional and non-functional descriptions in SCA. In *AT4WS '09: Proceedings of The 1st International Workshop on Advanced Techniques for Web Services in conjunction with the 11th International Conference on Enterprise Information Systems (ICEIS 2009).*

Belaïd, D., Mukhtar, H., Ozanne, A., and Tata, S. (2009b). Dynamic component selection for sca applications. In *Software Services for e-Business and e-Society*, pages 272–286. Springer Boston.

Bellavista, P., Corradi, A., and Magistretti, E. (2003). Proxy-based middleware for service continuity in mobile ad hoc networks. In Armano, G., Paoli, F. D., Omicini, A., and Vargiu, E., editors, *WOA 2003: Dagli Oggetti agli Agenti. 4th AI*IA/TABOO Joint Workshop "From Objects to Agents": Intelligent Systems and Pervasive Computing, 10-11 September 2003, Villasimius, CA, Italy*, pages 1–8. Pitagora Editrice Bologna.

Ben Mokhtar, S. (2007). *Semantic Middleware for Ubiquitous Services.* PhD thesis, PhD Thesis. University of Paris 6, Paris, France.

Ben Mokhtar, S., Georgantas, N., and Issarny, V. (2007). Cocoa: Conversation-based service composition in pervasive computing environments with qos support. *J. Syst. Softw.*, 80(12):1941–1955.

Ben Mokhtar, S., Kaul, A., Georgantas, N., and Issarny, V. (2006). Semantic-based context-aware service discovery in pervasive-computing environments. In *in Proc. of IEEE Workshop on Service Integration in Pervasive Environments (SIPE), In conjunction with IEEE International Conference on Pervasive Services (ICPS), 2006.*

Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., and Poole, D. (2003). CP-Nets: A tool for representing and reasoning with conditional Ceteris Paribus preference statements. *Journal of Artificial Intelligence Research (JAIR).*

Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., and Poole, D. (2004). Preference-based constrained optimization with cp-nets. *Computational Intelligence*, 20(2):137–157.

Boutilier, C., Brafman, R., Hoos, H., and Poole, D. (1999). Reasoning with conditional ceteris paribus preference statements. In *Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 71–80. Morgan Kaufmann.

Bowman, D. A., Kruijf, E., LaViola, J. J., and Poupyrev, I. (2005). *3D user interfaces: theory and practice*. Addison-Wesley.

Brafman, R. and Domshlak, C. (2002). Introducing variable importance tradeoffs into cp-nets. In *In Proceedings of UAI-02*, pages 69–76. Morgan Kaufmann.

Brown, A., Kolberg, M., Bushmitch, D., Lomako, G., and Ma, M. (2006). A SIP-based OSGi device communication service for mobile personal area networks. In *Consumer Communications and Networking Conference, 2006. CCNC 2006. 2006 3rd IEEE*, volume 1, pages 502–508.

Chakraborty, D., Joshi, A., Finin, T., and Yesha, Y. (2005). Service composition for mobile environments. *Mobile Network Applications*, 10(4):435–451.

Chakraborty, D., Yesha, Y., and Joshi, A. (2004). A distributed service composition protocol for pervasive environments. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, volume 4, pages 2575–2580.

Chen, M., Peng, C., and Hwang, R. (2007). Ssip: Split a sip session over multiple devices. *Comput. Stand. Interfaces*, 29(5):531–545.

CyberGarage (2005). Cyber media garage: Java implementation v1.2. http://www.cybergarage.org/net/cmgate/java/index.html.

Davidyuk, O., Selek, I., Duran, J. I., and Riekki, J. (2008). Algorithms for composing pervasive applications. *International Journal of Software Engineering and Its Applications*, 2(2):71–94.

(ECMA), E. C. M. A. (2005). High Rate Ultra Wideband PHY and MAC Standard. ECMA-368 Int. Standards, 1st Edition, Dec 2005.

Edwards, M. (2007). Policy Framework White Paper. OSOA white paper.

F., J. V., Casanova, M. A., Rubinsztejn, H. K., and Endler, M. (2006). An ontology based on the cc/pp framework to support content adaptation in context-aware systems. In *Proc. of WOMSDE 2006, in conjunction with SBES, Florianópolis*, pages 1–10.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616.

Fujii, K. and Suda, T. (2009). Semantics-based context-aware dynamic service composition. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–31.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the theory of NP-Completeness*. Freeman company.

Greenwood, D. and Ghizzioli, R. (2008). Autonomic communication with rascal hybrid connectivity management. *Advanced Autonomic Networking and Communication*, pages 63–80.

Greenwood, D., Ghizzioli, R., and Calisti, M. (2008). Hybrid seamless mobility supporting pervasive service collaboration. In *UBICOMM '08: Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 445–450, Washington, DC, USA. IEEE Computer Society.

Handte, M., Herrmann, K., Schiele, G., and Becker, C. (2007). Supporting pluggable configuration algorithms in pcom. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 472–476.

IEEE (2009). IEEE 802.11n-2009. IEEE Standards Association. Online: http://standards.ieee.org/prod-serv/80211n.html.

Indulska, J., Robinson, R., Rakotonirainy, A., and Henricksen, K. (2003). Experiences in using cc/pp in context-aware systems. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 247–261. Springer-Verlag.

Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan, F., Chibout, R., Levy, N., and Talamona, A. (2005). Developing Ambient Intelligence Systems: A Solution based on Web Services. *Automated Software Engineering*, 12(1):101–137.

Kaddour, M., Belaïd, D., and Bernard, G. (2006). Sip-based multimedia service continuity across various hosts and networks. In *MediaWiN 2006: 1st Workshop on multiMedia Applications over Wireless Networks*.

Kalasapur, S., Kumar, M., and Shirazi, B. (2007). Dynamic service composition in pervasive computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):907–918.

Kaneko, K., Morikawa, H., and Aoyama, T. (2003). Session layer mobility support for 3c everywhere environments. In *6th International Symposium on Wireless Personal Multimedia Communications, WPMC 2003*.

Kiss, C. (2007). Composite capability/preference profiles (cc/pp): Structure and vocabularies 2.0. w3c working draft 30 april 2007. http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430/.

Klyne, G. (1999). Protocol-independent Content Negotiation Framework. RFC 2703.

Klyne, G., Reynolds, F., C.Woodrow, Ohto, H., Hjelm, J., Butler, M. H., and Tran, L. (2004). Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/. W3C Recommendation.

Kumar, M., Shirazi, B. A., Das, S. K., Sung, B. Y., Levine, D., and Singhal, M. (2003). PICO: a middleware framework for pervasive computing. *IEEE Pervasive Computing*, 2(3):72–79.

Latvakoski, J., Paakkonen, P., Pakkala, D., Tikkala, A., Remes, J., and Valitalo, P. (2002). Interaction of all IP mobile internet devices with networked appliances in a residential home. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 717–722.

Li, H. and Wang, H. (2006). A method of service description and discovery in pervasive computing environments. In *Pervasive Computing and Applications, 2006 1st International Symposium on*, pages 604–607, Urumqi,.

Liao, W. S., Huang, Y. J., and Hu, C. L. (2007). Mobile media content sharing in upnp-based home network environment. *saint-w*, 0:52.

Lonnfors, M. and Kiss, K. (2008). Session Initiation Protocol (SIP) User Agent Capability Extension to Presence Information Data Format (PIDF). RFC 5196.

Lyons, K., Want, R., Munday, D., He, J., Sud, S., Rosario, B., and Pering, T. (2009). Context-aware composition. In *HotMobile '09: Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, New York, NY, USA. ACM.

Mukhtar, H., Belaïd, D., and Bernard, G. (2007a). A graph-based approach for ad hoc task composition considering user preferences and device capabilities. In *Workshop on Service Discovery and Composition in Ubiquitous and Pervasive Environments*, New Orleans, LA, USA.

Mukhtar, H., Belaïd, D., and Bernard, G. (2007b). Session mobility of multimedia applications in home networks using UPnP. In *Multitopic Conference, 2007. INMIC 2007. IEEE International*, pages 1–6.

Mukhtar, H., Belaïd, D., and Bernard, G. (2008a). A model for resource specification in mobile services. In *SIPE '08: Proceedings of the 3rd international workshop on Services integration in pervasive environments*, pages 37–42, New York, NY, USA. ACM.

Mukhtar, H., Belaïd, D., and Bernard, G. (2008b). A policy-based approach for resource specification in small devices. In *UBICOMM '08: Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 239–244, Washington, DC, USA. IEEE Computer Society.

Mukhtar, H., Belaïd, D., and Bernard, G. (2009a). A quantitative model for user preferences based on qualitative specifications. In *ICPS '09: Proceedings of the 2009 international conference on Pervasive services*, pages 179–188, New York, NY, USA. ACM.

Mukhtar, H., Belaïd, D., and Bernard, G. (2009b). Session continuity and splitting of multimedia applications using qualitative user preferences. In *Mobilitty'09 : ACM International Conference on Mobility 2009*, Nice, France.

Mukhtar, H., Belaïd, D., and Bernard, G. (2009c). User preferences-based automatic device selection for multimedia user tasks in pervasive environments. In *ICNS '09: Proceedings of the 2009 Fifth International Conference on Networking and Services*, pages 43–48, Washington, DC, USA. IEEE Computer Society.

Ohta, K., Yoshikawa, T., Nakagawa, T., Isoda, Y., and Kurakake, S. (2003). Adaptive terminal milddleware for session mobility. *Distributed Commputing Systems Workshop, Proceedings*.

Ohto, H. and Hjelm, J. (1999). Cc/pp exchange protocol. http://www.w3.org/1999/06/NOTE-CCPPexchange-19990624. W3C Note, World Wide Web Consortium (W3C).

Open Mobile Alliance (OMA) (2001). User agent profile (uaprof) specifications. http://www.openmobilealliance.org/.

Open SOA Collaboration (2005). Sca policy framework v1.00 specifications. http://www.osoa.org/.

Open SOA Collaboration (2008). Service component architecture (sca): Sca assembly model v1.00 specifications. http://www.osoa.org/.

OSGI (1999). Open services gateway initiative. http://www.osgi.org.

Papazoglou, M. P. (2003). Service-oriented computing: concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12.

Passani, L. and Trasatti, A. Wurfl - wireless wireless universal resource file. Open source project. http://wurfl.sourceforge.net. Last Accessed on 20th August, 2009.

Pering, T., Want, R., Rosario, B., Sud, S., and Lyons, K. (2009). Enabling pervasive collaboration with platform composition. In *Pervasive '09: Proceedings of the 7th International Conference on Pervasive Computing*, pages 184–201, Berlin, Heidelberg. Springer-Verlag.

Perttunen, M., Jurmu, M., and Riekki, J. (2007). A qos model for task-based service composition. In *Proc. 4th International Workshop on Managing Ubiquitous Communications and Services*, pages 11–30.

Poladian, V., Sousa, J. P., Garlan, D., and Shaw, M. (2004). Dynamic configuration of resource-aware services. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 604–613, Washington, DC, USA. IEEE Computer Society.

Preuveneers, D. and Berbers, Y. (2005). Automated context-driven composition of pervasive services to alleviate non-functional concerns. In *2nd International Workshop on Software Aspects of Context (IWSAC'05)*, pages 28–38.

Ranganathan, A. and Campbell, R. H. (2004). Pervasive autonomic computing based on planning. In *The IEEE International Conference on Autonomic Computing (ICAC 2004)*, pages 28–38.

Raverdy, P., Riva, O., de La Chapelle, A., Chibout, R., and Issarny, V. (2006a). Efficient context-aware service discovery in multi-protocol pervasive environments. *2006. MDM 2006. 7th International Conference on Mobile Data Management*, pages 3–3.

Raverdy, P.-G., Issarny, V., Chibout, R., and de La Chapelle, A. (2006b). A multi-protocol approach to service discovery and access in pervasive environments. In *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on*, pages 1–9, San Jose, CA,.

Román, M. and Campbell, R. H. (2003). A middleware-based application framework for active space applications. In *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 433–454, New York, NY, USA. Springer-Verlag New York, Inc.

Rosenberg, J., Schooler, E., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schulzrinne, H. (2002). Internet rfc 3261. SIP: Session initiation protocol.

Rosenberg, J., Schulzrinne, H., and Kyzivat, P. (2004). Indicating User Agent Capabilities in the Session Initiation Protocol (SIP). RFC 3840.

Satyanarayanan, M., Gilbert, B., Toups, M., Tolia, N., Surie, A., O'Hallaron, D. R., Wolbach, A., Harkes, J., Perrig, A., Farber, D. J., Kozuch, M. A., Helfrich, C. J., Nath, P., and Lagar-Cavilla, H. A. (2007). Pervasive personal computing in an internet suspend/resume system. *IEEE Internet Computing*, 11(2):16–25.

Schulzrinne, H. and Wedlund, E. (2000). Application-layer mobility using sip. *SIGMOBILE Mob. Comput. Commun. Rev.*, 4(3):47–57.

Shacham, R., Schulzrinne, S., Thakolsri, S., and Kellerer, W. (2005). The virtual device: expanding wireless communication services through service discovery and session mobility. In *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, volume 4, pages 73–81.

Simple Object Access Protocol. SOAP v1.2 Specifications. http://www.w3.org/TR/soap/.

Snoeren, A. and Balakrishnan, H. (2000). An End-to-End Approach to Host Mobility. In *6th ACM MOBICOM*, Boston, MA.

Sousa, J. and Garlan, D. (2003). The aura software architecture: an infrastructure for ubiquitous computing. Technical report, Carnegie Mellon Technical Report, CMU-CS-03-183.

Sousa, J. P. and Garlan, D. (2002). Aura: an architectural framework for user mobility in ubiquitous computing environments. In *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 29–43, Deventer, The Netherlands, The Netherlands. Kluwer, B.V.

Sun Microsystems (2001). Jmf. sun java media framework api. http://java.sun.com/products/java-media/jmf/.

Sung, J., Kim, D., Song, H., Kim, J., Lim, S. Y., and Choi, J. S. (2006). Upnp based intelligent multimedia service architecture for digital home network. *seus-wccia*, 0:157–162.

Takasugi, K., Nakamura, M., Tanaka, S., and Kubota, M. (2003). Seamless service platform for following a user's movement in a dynamic network environment. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 71–78.

UPnP Forum (2006). Av architecture specification v1.0. http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v1-20020622.pdf.

Valiente, G. (2007). Efficient algorithms on trees and graphs with unique node labels. In *Applied Graph Theory in Computer Vision and Pattern Recognition*, pages 137–149.

W3C (2003). OWL: Web Ontology Language. URL: http://www.w3.org/TR/2003/CR-owl-features-20030818/.

Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M., and Light, J. (2002). The personal server: Changing the way we think about ubiquitous computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*.

Want, R., Pering, T., Sud, S., and Rosario, B. (2008). Dynamic composable computing. In *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 17–21, New York, NY, USA. ACM.

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11.

WS-Policy, W. (2007). Web services policy (ws-policy) framework. http://www.w3.org/TR/ws-policy/.

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327.