Research Proposal

A Task Computing Framework for Taskable Places

Chuong Cong Vo

Department of Computer Science and Computer Engineering
LA TROBE UNIVERSITY

January 22, 2010

Contents

List of Figures

Chapter 1

Introduction

The general trend of computing is progressing towards the vision of *Pervasive Computing* (or *Ubiquitous Computing*) in which technologies are seamlessly embedded everywhere and dissolve in the fabric of everyday life [49]. However, a challenge of adding intelligent technologies to our lives is to "support our activities, complement our skills, and add to our pleasure, convenience, and accomplishments, but not to our stress" [50]. As spaces increasingly embedded with technologies become more complex and sophisticated, everyday users often find themselves spending more time and effort in configuring and instructing these spaces.

A smart environment is "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network"[1].

From the users' perspectives, in order to improve the acceptance and usability of pervasive systems, there should be a solution to relieve the complexity, invisibility, and inconsistency problems in pervasive interactive systems. There are two nominated solutions to these problems: *Context-aware computing* and *Task-driven computing.* Context-aware computing aims to enable computer systems to understand users' situations, thus to provide relevant information, services, and behaviours to the users [51]. *Task-driven computing* is defined as shifting focus to what users want to do (i.e., on the tasks at hand) rather than on the specific means for doing those tasks [52].

In this proposal, I will propose a task-driven framework for development and operation of pervasive interactive systems. These systems are built up from task models. At runtime, based on the user's situation, relevant task are recommended to the user. The models of selected tasks are loaded and guide the user to accomplish the tasks. The aim of this research is to show that the complexity, invisibility, and inconsistency of using pervasive interactive systems can be reduced by the use of the proposed framework. The feasibility of this framework will be illustrated via the development of a prototype system. This system will be evaluated through experiments in simulated scenarios as mentioned in Section 1.5.1.

[1]http://www.smarthome.com

1.1 The basic idea

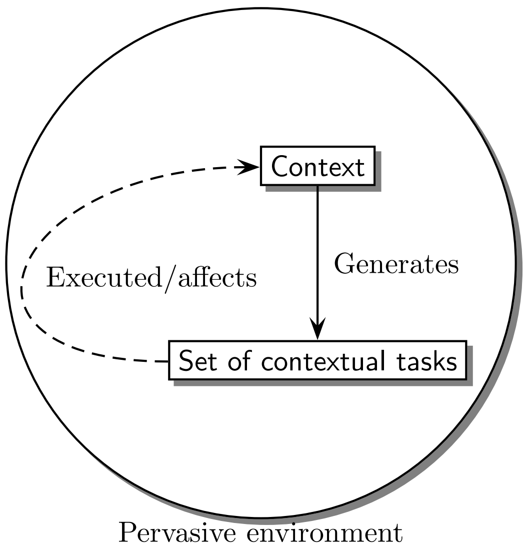


Figure 1.1: Task computing in pervasive computing environments

We examine tasks in continuously immanent movement and change of pervasive environments as shown in Figure 1.1. The context in an pervasive environment can be changed due to external influences of a system or executions of tasks by the system. The tasks, which are called contextual (active) tasks, are selected from the set of available tasks relied on the context. In other works, the context causes the system to expose a set of recommended tasks to its users. The accomplishment of a selected task can lead to change the current context. With the new context, the set of recommended tasks may be changed. In this research, we will model tasks in such this approach. Hereafter, we propose an context-aware architecture supporting for task-oriented systems in pervasive computing environments.

1.2 Future of computing environments

I predict that in the future,

- Mobile devices will continue to evolve and will become more powerful, but limitations on humancomputer interaction will persist. They weave themselves into the fabric of out everyday existence so that they can be accessed by anywhere and anytime [49].
- Computer-supported appliances design will evolve towards cloud computing model. Computer appliances will become inexpensive and widely deployed. In the cloud computing model, devices which users interact with are just interfaces of environments while processing is responsible of powerful computers.
- Emerging context-sensing/sensory technologies will enable more accurate context information acquisition.

- Service discovery will be the cornerstone of any future platforms.
- Wireless technologies are emerging as cable replacements and may potentially serve as system buses in a dispersed system.
- The computing paradigm will shift from humancomputer interaction to human-environment interaction.

1.3 Problems and Approaches

Following, I will present problems existing in pervasive computing environments and approaches to these problems.

1. Nowadays, to accomplish a particular task, a user has to configure themselves available services, devices, and resources. The user has to manage the failures of service invocations. These procedures are not simple for ordinary users. Even for experts, this activity is time and endeavour consumption. In other words, the user is required to at least be an expert or a developer in order to exploit the benefits deriving from the pervasive computing environments. This requirement is a great restriction and a challenge for deployment of pervasive services to the everyday life [53].

   Approach: Pervasive computing systems should recognise user's intentions and then autonomously fill in any technical details that the user left out and assist the user in performing a suitable task [41].

2. Managing pervasive computing environments have been placing too much overhead on users.

   Approach: PCEs are required to place minimal burden on users as users are continually exposed to distractions in PCEs where walking, driving, and other real world activities preoccupy the users [54]. Therefore, software systems in PCEs should be ware of and adapt to context in which they are running. They are required not to overwhelm users by only providing relevant information, services, and supports to their users. The systems should recognise user's tasks in order to provide relevant supports.

1.4 Problems of Pervasive Interactive Systems

User needs:

- Get insight in the digital services the environment offers, i.e. what resources are at hand and what tasks do they support?
- Interact with available resources, e.g. browse the contact list in a mobile phone through the car's head-up display or navigate a picture slide-show displayed on a television using a hand-held device.

- Configure the default and context-specific behaviour of resources, e.g. automatically turn off the lights and turn down the heating when leaving the house.

From users' perspectives, pervasive interactive systems currently exposes at least three problems: *complexity*, *invisibility* and *inconsistency*.

Two major approaches to provide the end-user with an interactive view on the environment's features [4]:

Service-oriented The end-user is shown a list of software services that make up the pervasive computing environment, where she can directly interact with.

Goal-oriented or Task-oriented The pervasive environment is seen as an integrated system where tasks define what the end-user can do within the environment. The end-user is presented with an overview of available tasks while the actual software services that give rise to these tasks are hidden.

Consider for example a scenario that demands for multimedia features such as playing music in a room. A service-oriented view might integrate a 'media' service that exports an integrated user interface to perform all media-related tasks. In contrast, a goal-oriented view might present the end-user with 'play media' and 'create playlist' tasks, which are represented as separate user interfaces to the functionalities of an underlying 'media' service.

To execute a task or even a simple task, the user is required to have knowledge and ability to

- control interfaces of devices involved while each device may have different interface;
- understand devices properties and functions of devices so that the user can exploit completely the features of the devices;
- configuring devices which is significantly difficult;
- choose, customise, config, and perform tasks correctly which is often a time consuming activity.

How we can eliminate these difficulties?

Mark and Su [55] investigated challenges faced by nomadic workers due to their lack of local knowledge of infrastructure. They suggest that until the field of pervasive computing can attain Weiser's vision, most users of ubiquitous computing need infrastructure to be visible.

Weiser's vision of a seamless connection is unfortunately still a dream at this point in time. Indeed, recent articles have begun to question the primacy of invisibility in research on ubiquitous computing [56].

Zimmermann [57] agree that user interfaces of systems in smart environments should meet three requirements: coherence, task-orientation, scalability, and accessibility.

Coherence means a user interface should allow for seamless control across many devices in the environment. Task orientation means a user interface should expose what the user can do which an integrated set of devices rather than how this will be achieved. Scalability means that a user can have a initial system with only a small number of devices. When subsequently adding devices to the environment, the user interface should present additional tasks that are made possible by the new devices. Thus a environment with many devices could incrementally evolve over years. The user should not need a new controller for controlling a new device, and should not have to buy a new set of devices when transitioning to a different controller. Accessibility–a user interface should be accessible to a wide range of users, including older people and people with disabilities.

1.4.1 Complexity of Use

The more technologies and features added to pervasive computing environments, the more resulting complexity they have. This can lead to a usability crisis similar to the current usability crisis in computer-controlled electronic products. Because of their feature-richness, pervasive interactive systems overwhelm users' perception and cognition ability by the overload of services, features, properties, settings, and configurations [53, 58–60]. Moreover, their resulting complexity has exceeded the capacity of current user interface designs for users to operate them intuitively [61]. For example, if the services and devices involved are from different manufacturers or domains, users currently need to learn the different operational details of each device and service to carry out their whole tasks correctly. A challenge is to allow users focusing on their daily life and just use the tools when they need them, without studying their use beforehand and without distracting their focus too much from the everyday activities [62].

Currently, the operation of a typical pervasive system is to show the user a list of all functions/actions (via buttons or menus) and let the user decide himself/herself the combination of functions/actions in order to accomplish his/her intended tasks. In many cases, users have trouble understanding what tasks are supported or how to associate the desired logical actions with physical actions [63]. Consequently, to exploit a pervasive system, users must (1) understand the meanings of features (and perhaps their combinations) provided by the system in order to issue feasible tasks; (2) map their high-level goals of tasks to low-level operational vocabularies of the system; and (3) properly specify constraints for tasks subject to contextual information and available services. These requirements may be beyond ordinary users, as the complexity, the diversity, and the sheer number of devices and services (as well as different combinations of ways they can work together) continually rises.

There is an apparent conflict. As the number of features increases, so too does the desirability of the device. But as the number of features increases, simplicity goes down. Subsequently, even as people buy the devices with extra features, they cry out for simplicity [64].

Pervasive computing environments are complex to interact with due to the dynamic assembly of interaction resources. When the complexity of such an environment is masked by the underlying computing system, end-users are often left with limited or no control over their interactive space. This brings up the need to make users aware of their surroundings.

Two out of three Americans have lost interest in a technology product because it seemed too complex to set up or operate [65].

Remote user interfaces are device-oriented rather than task-oriented [5]. Since there are typically multiple devices from different manufacturers in the home, users end up "juggling" multiple user interfaces, one per device. Each of these user interfaces reflects the functionality of a single device, and does not directly offer cross-device functionality that would be required for more intuitive and task-oriented user interfaces. For example, the task of watching a DVD movie on a typical home theater system involves at least 3 devices that the user needs to set up in the right way for inter-operation: The TV screen must be switched on and accept input from the DVD player; the same holds for the receiver/equalizer; and the DVD player must be switched on and instructed to play the DVD. The fact that users have to operate multiple user interfaces (one per device involved) to achieve a goal is one of the biggest impediments for easy-to-use interfaces in the digital home. This problem becomes worse the more devices are required for a particular task.

An evolving standard on task model representation, CEA-2018 [66], currently being developed, may help to solve this problem for future generations of devices in the digital home.

1.4.2 Invisibility of Features

One challenge to pervasive computing systems is their inherent invisibility from users' awareness. The invisibility is originated from the perfectly and naturally integrating technologies into environments. For example, a table can also be a computer too. Thus, the user tends to be unaware of the presence of functions available to them. Occasional visitors to a particular place are of another example. When arriving for the first time to a particular place, occasional visitors have little or no idea about what the local environment is providing to support their activity. Furthermore, this support has to be self-explainable and quickly learnable, as occasional visitors are not prepared to interact with an unknown system and do not have time to spend understanding and learning how to use new tools [67]. Moreover, because of the rapid advances of technologies, users find difficult understanding what a computing device is or what an artifact is and what it could do for them. They can't reuse it if they don't understand it or don't think they need it.

An inherent challenge with pervasive interactive systems is a user's awareness of the functionality which is relevant to their specific goals and needs. Awareness of functionality is not only important for learning how to accomplish new tasks, but also learning how to better accomplish existing tasks. In a potential "best case scenario", the user works with an expert next to them, who can recommend tasks when appropriate.

TaskRec collects usage data from a pervasive interactive system's user community, and applies recommender system algorithms to generate personalised task recommendations to each user. With TaskRec, we hope to expose users to tasks they are not currently familiar with that will help them use the system more effectively. The recommended tasks are displayed on a master device within the user's vicinity that the user to refer to when convenient. Thus, the system is much more ambient in nature.

1.4.3 Inconsistency of User Interfaces

Much attention has been spent on developing multi-device user interfaces for pervasive interactive applications. While some created user interfaces from scratch to get the best from the devices, others looked for automatic adaptations to reduce the load imposed to the designer. In both cases, resulting user interfaces so inconsistent from each other to the point of compromising usability when performing the same task on many devices [68]. Another reason for inconsistency is that because device manufactures want to reserve their brand identification and product differentiation, hence there is little or no user interface consistency between computing devices with even similar functions [61].

One way to reduce the complexity of a device without sacrificing its features is to design the UI consistently [69]. If a user interface is designed consistently, the user will benefit from what psychologists call *transfer of training*. In other words, learning to do one thing in one context will make it easier to learn how to do similar things in similar contexts [70].

Traditional remote user interfaces are dedicated to one particular target. Infrared-based remote controls shipped with and bound to specific target devices are prevalent in today's homes. This results in a large number of remote controls in the digital home, a real usability problem with a growing number of target devices in the average home [57]. Clearly, traditional infra-red controls do not provide for user interfaces that are reasonably consistent, task oriented and scalable.

"Universal remote controls" are an improvement because they allow for controlling a set of devices with one controller only. This makes for more coherent user interfaces, and enables task-oriented features that span multiple devices. However, universal remote controls typically come with an increase on complexity, especially if user programming is required. Scalability is not guaranteed unless the user interface for the new device can be downloaded from the database of the universal remote control manufacturer [57].

Recently, custom installation environments have been equipped with integrated and sometimes task-oriented user interfaces for consumer electronics products and appliances. The drawback of custom installation is that the design of custom-made user interface is a cumbersome process requiring special skills. It is not really scalable since every time a new device is added, the user interface needs to be revised manually [57].

1.5 Proposal of a Task-Driven Framework

One way to deal with the complexity problem is to simplify their user interfaces as much as possible but without reducing the usage of the systems. When computers are embedded into an environment, the means for interaction between the user and the environment is not just desktop monitors with menus and buttons, keyboards, and mouses, but any interactive devices such as mobile phones, televisions, fridges, chairs, doors, bicycles, washing machines, mirrors... Traditional graphical user interfaces with menus and buttons are no longer suitable for such the emerging interactions. There is a need for a new methodology to design user interfaces of pervasive interactive systems. *Task-based user interfaces* which communicate with the user in terms of "what to do" rather than "how to do it" are seen as potential remedy for the complexity problem in pervasive interactive systems.

For the inconsistency problem, standardisation would appear to be a logical solution. Unfortunately, the standardisation of user interfaces across devices with similar functions has been obstructed because device appearance and operational details are crucial to brand identification and product differentiation [61]. Therefore, application-level standardisation would be more suitable. Task-driven development of pervasive interactive systems can deal with this inconsistency problem. Task models are abstractly specified using a standard language. Once a task model is specified, it can be loaded and executed consistently in different environments for achieving its goal.

To address the invisibility problem, task recommendation would be a solution. The system will base on user's situational context and available functionalities of the environment to recommend relevant tasks for the user. The system will guide the user through steps specified in the task models to accomplish the selected tasks.

My Task-Based Framework is proposed to achieve the objectives mentioned above. The framework supports two phases: design time and run time. At design time, task models are specified using a standard language. At runtime, a runtime system called Task Execution Engine manages and controls the execution of task models. Another runtime system called TaskRec run on devices carried by users. TaskRec acts as the interface between the user, the Task Execution Engine, and interactive computing devices.

1.5.1 Use Case 1

To prove the advantages of the proposed framework, an experimental environment should consist of a number of different computer-controlled devices and should provide a range of different features. Moreover, the operation of some tasks should span across different places to show how the inconsistency problem of task operation can be solved. Let's consider the following scenario which involves places: house, car park and library.

Assume that the house, the car park and the library are installed with a Task Execution Engine. Each is loaded with specific domain task models (i.e., a database of task descriptions) and is connected to various sensors for acquiring contexts from the physical environment.

On a heavy rain and cold early morning in winter, when Bob has stepped out of the house, the Task Execution Engine uses knowledge about his routine and calendar, and asks the TaskRec *on his mobile phone to recommend him the task 'Drive to work'. As Bob selects this task, the Task Execution Engine automatically loads the model of this task and executes it. As a result, the heater and the TV are turned off; his mobile phone is switched to outdoor mode, all calls to his home phone will now be forwarded to his mobile phone...*

At his workplace, while Bob is walking into the university library, the Task Execution Engine of the library asks the TaskRec *to switch his mobile phone to quiet mode (because of library policies) and recommends him two tasks: 'Borrow a book' and 'Book a study-carrel'. He selects the first task. The Task Execution Engine executes this task by guiding him through the borrowing process. Consequently, the book he would like to borrow is located in the local library of the Computer Science department. As he walks into the departmental library, the* TaskRec *recommends him two tasks: 'Borrow a book' and 'Borrow a projector'. He selects the first task. Because he has never borrowed a book here, the borrowing policies as well as underlying technologies are unfamiliar with him. But he still feels very impressive because the Task Execution Engine has guided him through a consistent borrowing procedure.*

At the end of the working day, Bob walks towards his car in the car park. He recognises that the car has been damaged by someone's car but they have gone away. This is the first time he is in this situation. Thanks to the TaskRec *on his mobile phone, he is able to search for relevant tasks for this new situation. The search uses context information provided by both him and the system itself. For a while, the system recommends him the task 'Report a car accident'. Once he selects this task, the system guides him to accomplish it.*

1.5.2 Use Case 2

The following scenario extracted and modified from [39] highlights the challenges of task recommendation in a pervasive computing environment.

Tommy attends a conference to present his paper. He's unfamiliar with the facility holding the conference. Fortunately, he can view a list of conference-related tasks in his smart phone's task recommender. He executes a task to register as a presenter and make his presentation slides public. Tommy also finds a 'Want to know about the conference' task that introduces the conference through a video delivered to his smart phone. This task guides him to a shared high-quality display in the lounge. He moves to the lounge and watches the content on this display. While there, Dana, a student who was recommended a task 'Meet Tommy', approaches to inquire about his research.

During Tommy's presentation, he is recommended for presentation-related tasks such as 'Print a document'. Dana as an audience member is also recommended for the same task. Although executing the same printing task, Tommy's task uses a printer in the same room while Dana's task uses another printer in the next room, which has no job pending. After the presentation, Tommy and Dana meet to discuss *their research in the lounge. Tommy executes the 'Print a document' task to print a document for Dana. This time, a printer in the lounge is used.*

In the first part of this scenario, TaskRec recommended useful tasks for Tommy automatically even though he was unaware of their feasibility. The tasks weren't just unknown to him, but were related to his context. In the second part of the scenario, Tommy and Dana's TaskRecs used contextual information and applied different criteria to recommend the most relevant tasks.

1.5.3 Requirements of the Scenarios

Minimum assumption on mobile devices The mobile device doesn't need to install any specialised software or hardware in order to accomplish recommended tasks in a certain environment.

Automatisation] Automatic discovery of relevant tasks in unfamiliar environments.

Task Composition Automatic composition of tasks based on the availability of devices and services.

User context consideration To provide relevant tasks.

Context-aware user interface generation User interface generation based on user and device capability should be supported.

1.5.4 Expected Research Contribution

The main goal of this research is to address the complexity, invisibility, and inconsistency of the use of pervasive interactive systems in smart spaces. As shown in the scenario mentioned above, our task-driven framework hides the complexity of the system by recommending the user only relevant tasks according to the user's situation (e.g., location information). Then, the user interface will be tailored to the selected tasks to guide him/her through task execution. This differs from a traditional system where all features of the system are shown on its user interface (e.g., hierarchical menus), and the user deals with the decision of selection or combination of features to finish his/her intended tasks.

The use case also shows that the operation of the same task (e.g., 'Borrow a book') can be consistently operated in two different environments. To do so, task models should be specified independently from executing environments.

The task-driven framework will demonstrate the following key features:

- **Task model description**: The research will explore task modelling languages for describing task models. The task model description should provide a mechanism for abstractly specifying task models so that task models are independent of actual services, devices, and resources. It should allow to describe applicability conditions for tasks.

- **Task recommendation and searching**: The framework uses user's location information at various levels (e.g., room, building, area...) and devices near-by for inferring relevant tasks. The user is also able to search for relevant tasks by providing information about a particular situation.

- **A task execution engine**: The framework has a runtime Task Execution Engine which executes tasks selected by users. The engine needs to perform a crucial function that binds abstract services (described in task models) with actual services and devices currently available in the environment. The engine also organises resources and composes available services for a task to be performed.

- **Task-based user interfaces**: The framework can actively help the user learn how to operate tasks via task-based user interfaces.

- **Programmability**: Because task models are developed separately. Different task models can be loaded and removed from the system easily. This makes the pervasive environment programmable. So, the system is not be bogged down by the numerous low-level details of a static set of functionalities, but will allow to configure the environment to the specific requirements of the inhabitants quickly.

Together with design and implementation of the task-driven framework, I will also demonstrate the feasibility of this task-driven framework by developing a prototype system. This system will be evaluated through experiments in simulated scenarios as mentioned in Section 1.5.1.

1.6 Outline of the Research Proposal

The rest of this proposal is organised as follows:

- In Chapter 2, I will discuss the notion of task and task-driven computing research.

- In Chapter 3, I will examine the research direction of task-driven computing and review the related work.

- In Chapter 4, I will describe a detailed design of the Task-Driven Framework. The discussion will cover how the framework describes, manages, and executes task models in pervasive environments.

- In Chapter 5, I will summarise this document and present the preliminary research approach. A feasibility study will be described for the purpose of evaluating the framework. At the end, the milestones of the future research will be presented.

Chapter 2

Background

2.1 What is task computing?

According to [71, 72], in application-, devices-, and service-rich environments, *task computing* allows users to focus on their tasks they want to accomplish rather than how to accomplish them. It has an ability to manage the tasks in runtime by having the capability to suspend from one environment and resume the same task later in a different environment [41].

2.2 Why Tasks, Not Services?

Once overwhelmed by information, services, and devices in the surroundings, a user often asks himself *What can/should I do with what I have in my current situation?*. In particular, the questions a user can ask the system are [61]:

- What task can/should I do next?
- How do I do $\langle task \rangle$?
- When should I do $\langle task \rangle$?
- Why did the system do $\langle task \rangle$?
- What are the inputs/outputs of $\langle task \rangle$? and
- Did $\langle task \rangle$ succeed?

In these questions, $\langle task \rangle$ represents for a particular task, e.g., 'Borrow a book'.

It can be seen that information, services, and devices are only means to accomplish tasks. But what tasks should be done is a key question to be addressed. Task-driven computing aims to provide the answers to these questions.

2.3 Goals of Task-Driven Computing

- To transfer users' focus from the computer to their tasks;
- To minimise human involvement of human-computer interaction;
- To bring computation into the real physical world and to allow people to interact with them in a more natural way: by talking, by moving, pointing, and gesturing; and
- To help a person "forget he/she is using a computer while interacting with one" [73].

2.4 What is a Task?

2.4.1 Definitions of Tasks

There are many definition of a task in the literature (e.g., [12, 41, 58, 61, 63, 74–78]) but there is no common definition of task. In a nutshell, we adopt a definition that "*a task is a goal or objective which is presented by a set of actions performed collaboratively by humans and machines to achieve this goal*".

A task is deferent from a service. A task is a goal or objective while a service is the performance of tasks. Tasks are associated with what the user wants to accomplish, and services with environmental capabilities to complete the tasks [78].

"Task" is not the same as "problem" though some people consider contrary. This statement is justified by the fact that one can say "perform a task" but cannot "perform a problem", which shows their inherent difference. "a task" as "a sequence of problem solving steps". Therefore, a task's name necessarily includes "verbs" representing problem solving activities [79].

2.4.2 Characteristics of Tasks

As identified in the literature, tasks have the following characteristics which should be taken in consideration when developing a task-based system.

- Tasks vary widely in their time extent. Tasks can last minutes, hours or days; some are effectively instantaneous–for example, 'turn off the room lights'; some never finish; and some have a indeterminable time extent;
- Tasks typically involve both human participants–as requesters, beneficiaries, or performers of the tasks–and machines. Some tasks can be performed only by a human being; others can be performed only by a machine; and yet others can be performed by either.
- Tasks can span multiple devices, places, and involve many people, services;

- Tasks can range from a very high abstraction level (such as 'solve a problem') to a concrete, action-oriented level (such as 'select a printer'). Very low-level tasks are closer to the primitive controls of a particular device–for example, 'press the power button on a DVD player'.

- Tasks are often personalised for each user.

2.4.3 What is a Task Model?

A task model or task expression or formalism provides an adequate high-level description of user-oriented tasks [41].

Task models (or task formalisms) are formal descriptions of the activities involved in completing the tasks [41, 61, 63].

The purpose of a task model is to help users (e.g., designers, developers, managers, customers, and experts) understand how activities should be performed. For example, it helps designers identify requirements which should be satisfied to perform tasks effectively. It also helps the system understand how the task are performed. For example, it helps the system identify interaction and presentation techniques best suited to support the task.

The purpose of a task model: A task model (its representation) helps users (designers, developers, managers, customers, experts) understand how activities should be performed. For example, it helps designers identify requirements which should be satisfied to perform tasks effectively [17]. It (its specification) also helps the system understand how activities are performed. For example, It helps the system identify interaction and presentation techniques best suited to support the tasks at hand [17]. Task models can be used in diffident phases of software development. For example, it is a tool for communicating between developers and customers at the analysis phase.

Pervasive Computing Environments (PCEs) are highly dynamic environment regarding to the availability and heterogeneity of users, devices, and services. Therefore, the key requirements of task models for task-based systems in PCEs are:

- to be abstract, i.e., device/service independence; and

- to capture environment's and user's context as pre/post-conditions for task execution.

Task models would be abstract enough to avoid being bound to individual devices. The abstraction of task models allows tasks to be flexibly executed in different environments. So, this can increase the reuse and scalability of task models. A task model should include context information so that the execution of the corresponding task can be tailored to the context of use.

"They describe how high-level tasks can be achieved by using lower-level tasks such as provided by devices and services. Task models are an excellent back-bone for dynamically building seamless user interfaces that cover the functionality of a whole set of devices/services in an environment rather than having one user interface per device/service."

Requirements of task models:

- Models should be abstract;
- Task models should capture environment context and user's cognitive context.
- Task flow must be common to all devices: device independence.
- It must not be too fine-grained such that a step in a subsequent task could be part of the current task without affecting the functionality of the application. For example, all text fields of a search mask which a user can fill in without requiring any additional system interaction must necessarily belong to the same task.
- It must not be too coarse-grained such that there exist two steps within a single task with one step requiring input data that is to be produced as the output of another step within the same task. For example, the search mask and the result list of a search engine query ought to belong to separate tasks.
- Each task should well be displayed on a single view.

2.4.4 Task Modelling

Task modelling is the process of developing and describing task models [61]. The result of this process are descriptions of task models which represent tasks.

The purpose of task modelling is to build a model which describes precisely the relationships among the various tasks identified. These relationships can be of various types, such as temporal and semantic relationships. When we reach tasks which cannot be further decomposed we have basic (atomic or primitive) tasks. In some cases, basic tasks require one single physical action to be performed. It is important that task models are rich in information and flexible so as to capture all the main activities that should be performed to reach the desired goals and the different ways to accomplish them.

2.4.5 Task Computing Framework

A task computing framework is to deal with mapping between tasks and services. Users don't deal directly with services but rather specify high-level tasks to the task computing framework that deals with corresponding low-level services. For example, a lecture hall fitted with a task computing framework can process typical tasks, such as showing presentations and controlling lighting, by completing a set of underlying services [78].

2.4.6 Taskable Spaces

Spaces fitted with a task computing framework are taskable [78].

2.4.7 Task Ontologies

What is an ontology?
"Roughly speaking, ontology is composed of two parts, that is, taxonomy and axioms. Taxonomy is a hierarchical system of concepts and axioms are established rules, principles, or laws among the concepts. Axioms specify the competence of ontology. In other words, a class of the questions to which the answers can be derived from the axiom specifies the competence of the ontology." [80]

Tasks, which are commonly performed at a place or with particular appliances, yields a vocabulary of typical task sets in different settings, called *task ontologies* [78]. A task ontology informs a system of the tasks it should support in making a place taskable. Task ontologies can be built based on places and artifacts. For examples, in a home theater room, a common task would likely be making the room suitable for watching a movie; in a lecture hall, a typical task is such as continue yesterday's lecture; consider windows and drapes–typical operations are open and close; typical operations on lights are turn on and turn off; and if a dimmer is present, darker or brighter.

Task ontology for clinic procedures [81]. It contains a hierarchy of tasks and subtasks for determining cerebrovascular disorders, and other illnesses.

Task ontology for geospatial data retrieval [82]: It is used as a guide for finding relevant geospatial data.

Task ontology for mobile Internet services [83]: It includes tasks such as "buy a ticket for a movie", "move to theme park", "seek for new routes", and "reserve a parking".

Place- and artefact-based task ontology: Task ontology is built on ontology of places and ontology of artefact. An place ontology has identified by [84] such as "at work" is places for thinking or deliberating, speaking to groups or presentations; "at home" is places for maintaining the body, watching the world, or gathering; "in the town" is places for shopping or commemorating; "during travel" is places to visit; "car and other vehicles" are places for transport or temporary accommodation.

Situation-based task ontology: Task ontology is built on situation ontology. Situation ontology is built on time ontology, place ontology, artefact ontology, user ontology, environment condition ontology.

Task ontology [79] is a system of vocabulary for describing problem solving structure of all the existing tasks domain-independently. It is obtained by analysing task structures of real world problems. Design of task ontology is done in order to overcome the shortcomings of generic tasks and half weak methods while preserving their basic philosophies. The ultimate goal of task ontology research includes to provide vocabulary necessary and sufficient for building a model of human problem solving processes. When we view a problem solving process based on such as a sentence of natural language, task ontology is a system of semantic vocabulary for representing meaning of the sentence. The determination of the abstraction level of task ontology requires a close consideration on granularity and generality. A task ontology consists of the following four kinds of concepts:

1. Generic nouns representing objects reflecting their roles appearing in the problem solving process,
2. Generic verbs representing unit activities appearing in the problem solving process,
3. Generic adjectives modifying the objects, and
4. Other concepts-specific to the task.

Chapter 3

A Survey on Task Computing

3.1 Task-Driven Software Development

3.1.1 TERESA[1]

The TERESA tool provides a high-level programming environment for developing tasks. Developers are able to concentrate on the conceptual task model and a corresponding user interface for the task can be automatically generated for different platforms and devices (i.e., the same task can have a different presentation depending on the platform).

3.2 Universal Remote Controls

3.2.1 Logitech Harmony Remote

A Logitech Harmony remote[1] can control many devices at the same time. It offer the user a list of activities such as "Watch TV" and "Listen to Music". Once the user selects an activity, the remote automatically sets up the entertainment system to achieve the selected activity. However, these universal remote controls for the most part still need to be pre-configured with a certain set of activities that can be performed with the pre-known set of devices. They are not able to discover new activities as new devices join the network.

3.2.2 Huddle[2]

The Huddle system lets end-users connect devices together in a user interface centred on content flow diagram supplied by the user.

[1] http://www.logitech.com/index.cfm/remotes/universal_remotes/&cl=au,en

3.2.3 Meta-UI[3, 4]

A meta-user interface (meta-UI) of a pervasive computing environment is the user interface for a point of control to inspect and manipulate the environment's current configuration. A meta-UI provides end-users with an interactive view on a physical or virtual environment. It abstracts a resource's features as executable activities that can be assembled at runtime to reach a common goal.

The main responsibility of the meta-UI is to integrate the functionalities supported by the environment and to present them in an intuitive way to the end-users. It can discover available resources (Services, Tasks, Users and Devices), integrate them in a view and make their software interface accessible to end-users via a user interface.

The meta-UI provides a view on the resources the user can interact with. This view is rendered from information in the underlying model and can be displayed on devices carried by the end-user. As a resource can range from a software service (e.g. weather service) to a physical resource (e.g. digital fridge), the meta-UI provides uniform access to various task-supporting entities. The system can discover the resources in the user's vicinity and present those in a personal, location-based view. While navigating through the meta-UI, the user is presented with the tasks that are supported by a resource.

Meta-STUD [4] is a reference framework that provides a basic set of operations for a meta-UI to support inspection and manipulation of the environment configuration by its end-users. The common resource types of the framework are Services, Tasks, Users and Devices. The framework includes the following operations:

- $Remove(R), Add(R)$: Remove/Add a resource.
- $Share(R), Unshare(R), Share(R, U_1...U_n)$: Share/Unshare a resource (with users).
- $Present(T), Present(T, D)$: Present a task on a device. If no target device is specified, the device running the meta-UI that triggered the operator is considered the target device.
- $Suspend(T), Resume(T)$: Suspend a task and resume it afterwards.
- $Migrate(T, D)$: Migrate a task from one device to another. The task is suspended on the source device and its context is transferred to the target device where the task is resumed.
- $Invite(U, T), Invite(T)$: Invite a user to execute a task. An invite is sent to the user's (default) device. If no user is specified, all users are invited to execute the task.

3.2.4 Universal Remote Console Framework

The Universal Remote Console (URC) framework is currently being finalised as the international standard ISO/IEC 24752. This standard defines an abstract user interface layer called the "user interface socket" and allows the development of

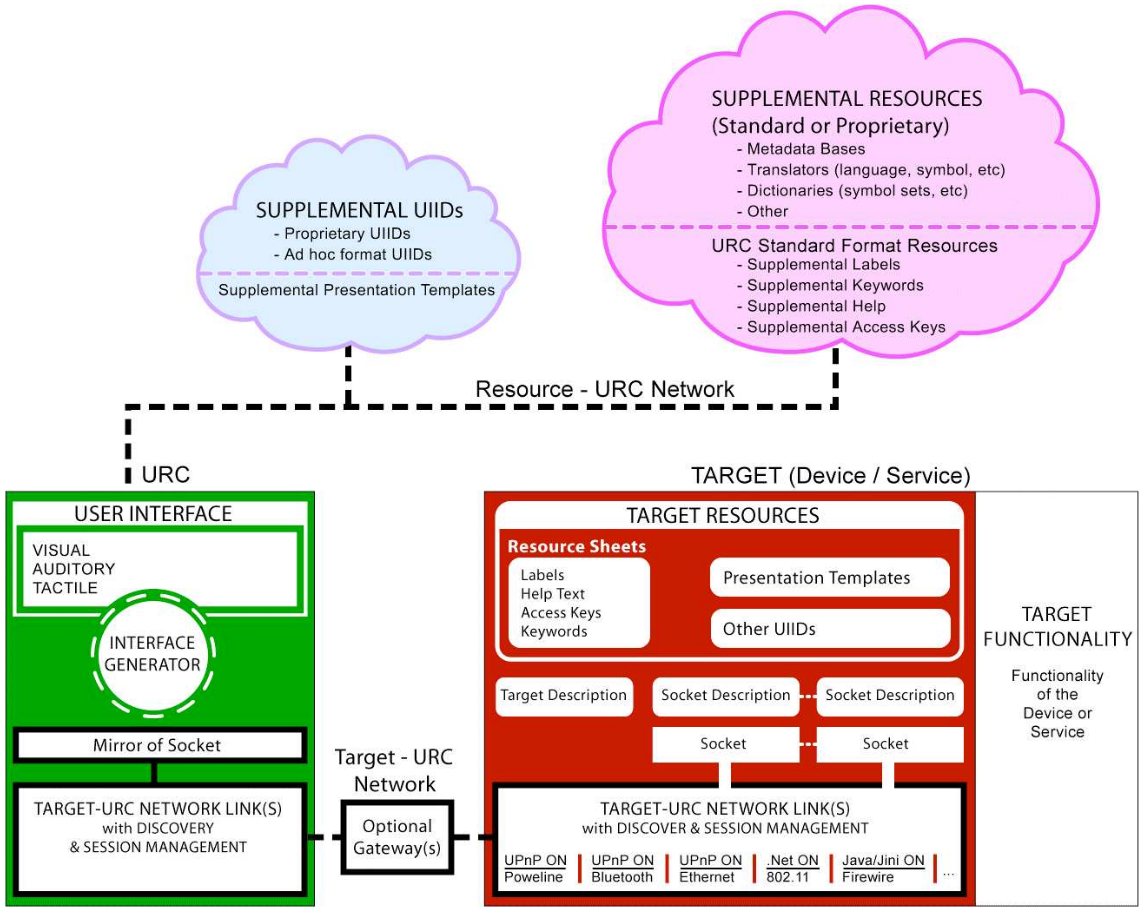


Figure 3.1: Conceptual structure and components of the URC framework

pluggable user interfaces[2]. Controllers (called "Universal Remote Consoles", short URCs) can connect to this Socket, using any kind of user interface that "plugs into the Socket". Controllers can look for and retrieve pluggable user interfaces (called "User Interface Implementation Descriptions", short UIIDs) from "Resource Servers" on the Internet. Figure 3.1 in the appendix illustrates the conceptual model of the URC framework.

Pluggable user interfaces are free to use any modality and user interaction mechanism they want. They can be provided by the manufacturer of a device, by the manufacturer of a particular controller, by companies that specialise in user interface design, or even by user groups.

Coherence of user interfaces is supported by the URC framework in two ways. First, every device describes its "abstract user interface" in the same way, enabling the use of one controller for all devices. Second, pluggable user interfaces can span multiple Sockets (and thus multiple devices), and so one user interface can be built to include the functionality of a whole set of devices.

Task orientation is supported by the URC framework to some degree. The combination of URC and Task Model Description can bring about the full power a task-oriented user interfaces.

[2]http://myurc.org/

Scalability is enabled by the URC framework since every device exposes a Socket that any third party can design a pluggable user interface for. The collection of available pluggable user interfaces is maintained by Resource Servers. Controllers can download suitable pluggable user interfaces from these Resource Servers.

For the home environment, the "Universal Control Hub" architecture[5, 85] represents a solution for the URC framework to be implemented even if the controlled devices and controllers are not URC compatible.

3.2.5 Universal Control Hub[5]

The Universal Control Hub (UCH) is a gateway oriented architecture for implementing the Universal Remote Console framework in the digital home. The main features of the UCH are:

- It acts as a bridge between targets and controllers: each with its own communication and control protocol, that otherwise would be unable to talk to one another.
- Standard-based user interface socket: The UCH is based on the URC framework previously described.
- A variety of user interface protocols: The UCH allows different user interface protocols (DHTML over HTTP, Flash, etc.) to be implemented and used by controllers.
- Globally available resource servers: The UCH can get distributed resources, such as target adaptors, target discovery modules and user interfaces from resource servers.

Figure 3.2 shows the UCH architecture for the URC standard.

3.2.6 I2HOME

The scope of I2HOME[3] is the Intuitive Interaction for Everyone with Home Appliances based on Industry Standards. In this way, I2HOME will make devices and appliances at home more accessible to persons with mild cognitive disabilities and older persons, using a new mainstream user interface standard the Universal Remote Console (URC) standard.

3.2.7 OSCAR [6]

OSCAR provides users with three basic capabilities: (1) browse, select, and control individual devices; (2) connect devices, services, and content together; and (3) create, edit, and invoke reusable service compositions. OSCAR was implemented atop the Obje Framework [86].

[3]http://www.i2home.org/

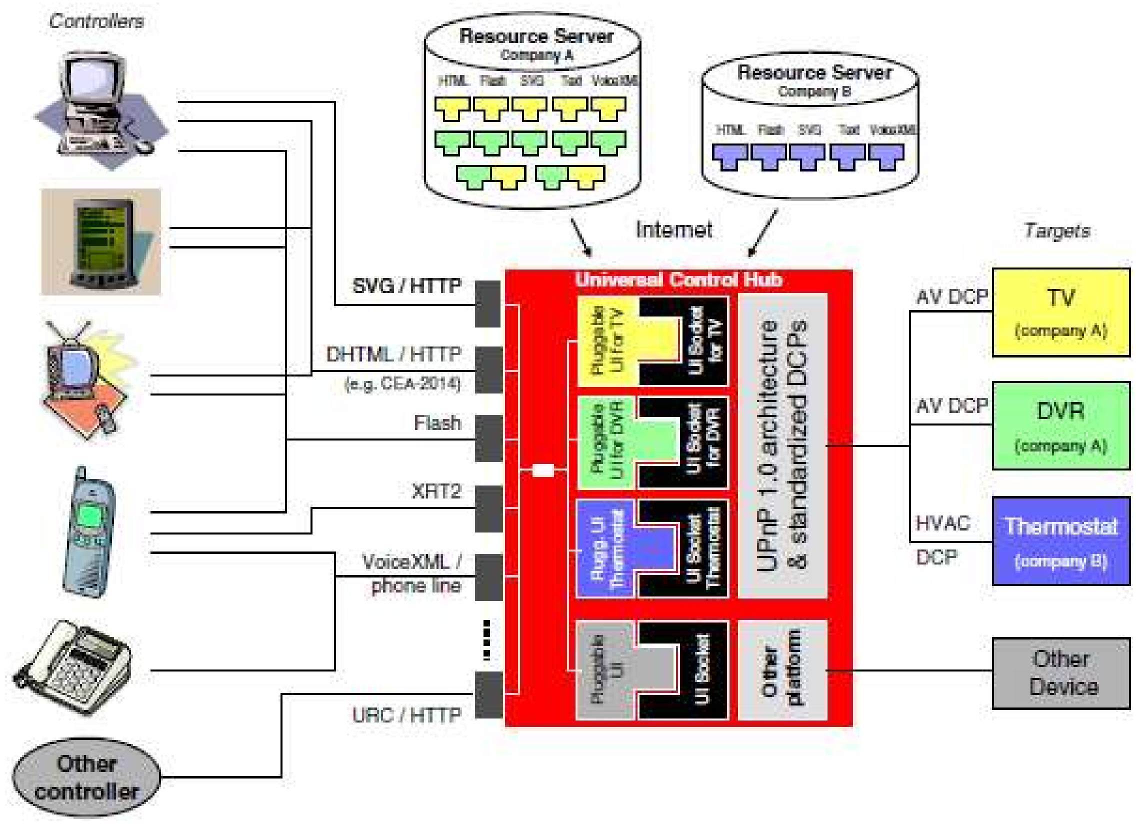


Figure 3.2: UCH architecture

3.3 Task Prediction

3.3.1 Automatic Task Prediction[7]

The authors found that it is possible to predict the next probable task based on history of the users' activity using Markovian models, and that it is easier to predict the next task than the next device being used. They model a task as a set of related commands to devices, and the aim is to group those commands in a meaningful way so that they can be easily used together as one task. The primary disadvantage of the approach is that they are not ware of the changes of the environments. They use a large data set of usage history, so it costs more computation expense and time consumption.

3.3.2 Task Prediction Using Commonsense Reasoning[8]

The authors described the use of commonsense reasoning to suggest tasks or goals automatically. For this, they created a semantic network called EventNet [87], used by Roadie to create a list of possibilities for user goals and plausible candidates for action sequences to accomplish those goals. The plan recogniser uses knowledge mined from the Open-Mind Commonsense Knowledge Base [88], a knowledge base of 800,000 English sentences describing everyday life, contributed by volunteers on the Web. EventNet does not distinguish between event sequences that express causal relations and those that do not. EventNet links can either represent cause-effect relations; express a goal, a sequence of actions, and a result; or simply a contingent observation of temporal occurrence. For example, “I drink coffee” and “I feel alert” are causally related, but “I drink coffee” and “I go to work” are not necessarily causally related, even though “I drink coffee” often occurs before “I go to work” (there may, of course, be a reason why coffee drinking often occurs before work). Their function is to generate candidates for possible goals consistent with a goal the user expresses, or a possible sequence of actions and/or consequences for accomplishing a user-supplied goal.

Note that EventNet may contain links that are patently garbage. In the example, “I walk in the rain” and “I get wet” are plausible continuations of “I observe that it is raining”, but “I paint someone’s house” is not. How did that get there? First, it may have come directly from Open-Mind a malicious or sarcastic user may have typed it in, and we have not verified each and every statement in the knowledge base. Second, it may have come from a misinterpretation by our natural language understanding system, whose capabilities are admittedly limited.

The impact of such garbage statements is kept under control by intersecting the results of retrievals with domain knowledge that comes from plausible operations in the application. That generally causes the garbage to be filtered out. If the system detects the event, “I plug in my guitar”, EventNet continuations may produce “I play music”, “I record music”, and “I start dancing”. However, intersecting the device functions, capabilities and goals with the EventNet results approves playing and recording music, but no function of any device that the system knows about has anything directly to do with dancing. If, on the other hand, the user types a goal, “I would like something to dance to”, it is likely to infer that playing music is appropriate.

3.3.3 Discovery of high-level tasks in the Operating Room [9]

It presents a Markov-based approach for inferring high-level tasks from a set of low-level sensor data. The approach cleans the noisy sensor data using a Bayesian method. Preliminary results on a noise-free dataset of ten surgical procedures show that it is possible to recognise surgical high-level tasks with detection accuracies up to 90%.

3.4 Task-Based/Oriented User Interfaces

A Task-Based User Interface is simply a user interface where navigation is based on tasks, which allows users to concentrate on what they want to do instead of how to do it[89].

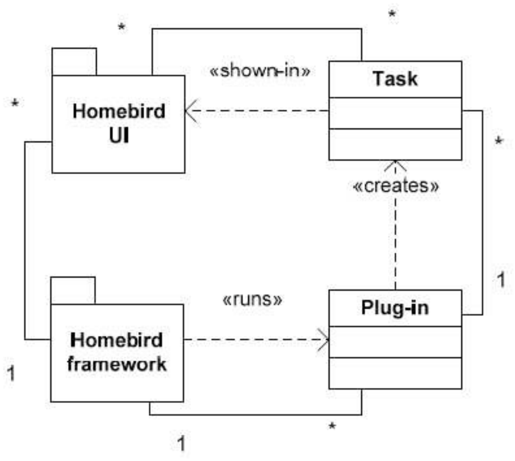


Figure 3.3: Homebird high-level architecture.

3.4.1 A task-oriented interface to a digital library [10]

This paper describes an interface to a heterogeneous digital library. The interface is designed to support user tasks, to smoothly integrate the results of many services, to handle services of widely-varying time scales, to be extensible, and to support sharing and reuse.

3.4.2 Goal-Oriented Interface [8]

They propose to re-orient the interface around the goals of the user, rather than the functions of the device.

3.4.3 Pervasive Menu[11]

A pervasive menu is composed of tasks that are deployed in the environment. In other words, it provides a view on what the user can do in a pervasive environment. The pervasive menu is generated based on the environment model which describes the full context of use of the system as a semantic graph.

3.4.4 Homebird[12]

Homebird is a task-based remote control on a mobile phone. It discovers features of other devices automatically and suggests to the user that certain tasks can be performed together with those devices. A newly discovered device can trigger changes to the set of available tasks.

In Homebird, the logic of tasks is encapsulated in modules called plug-ins. The Homebird system loads these plug-ins when the phone is switched on, and lets them verify applicability conditions in the background (i.e., for deciding whether to display respective tasks for the user). All the plug-ins use the UPnP protocol for communicating with networked devices.

The Homebird architecture is shown in Figure 3.3. The system runs plug-ins. Each plug-in can produce one or more feasible tasks that appear on the Homebird user interface. When a task is selected in the UI, control is handed back to the respective plug-in that can then show its own UI customised for that task.

This approach cuts down the number of steps needed to perform common tasks, and also makes it easier for users to find out what can be done in a particular environment. The system also allows to add new tasks (by adding plug-ins). However, the approach does not consider the need of the user when suggesting tasks (i.e., the system shows all tasks instead of showing only relevant tasks to the user).

This is about devices discovery and shows tasks to users from combining the discovered devices. Not take user context or user intention into account.

Homebird is a demonstration of a task-based user experience on a mobile phone. It discovers features of other devices automatically and suggests to the user that certain tasks can be performed together with those devices. This approach cuts down the number of steps needed to perform common tasks, and also makes it easier for users to find out what can be done in a particular environment. The implementation architecture makes it easy to add new tasks, and they can also have the phone perform actions in the background without user interaction.

3.4.5 Fujitsu's Task Computing

Masuoka et al. [90] introduce a task computing system which discovers devices in a computing environment and identifies the services executable. The system presents the available basic services to the user and allows him to either choose a basic service or compose a complex service using multiple basic services. While this approach minimises the complexity in the underlying middleware, it doesn't provide a good user experience since the user has to spend significant time and effort to understand the services to compose and achieve the desired task.

The system discovers devices in a computing environment and identifies the services executable. The system presents the available basic services to the user and allows him to either choose a basic service or compose a complex service using multiple basic services. While this approach minimises the complexity in the underlying middleware, it doesn't provide a good user experience since the user has to spend significant time and effort to understand the services to compose and achieve the desired task [60].

3.4.6 HP's User-Centric Appliance Aggregation [13]

It presents a user-centric appliance aggregation system which discovers devices and the services available in an ad-hoc environment. The system also identifies the services that could be aggregated and finally suggests the possible service combinations to the user. This approach doesn't provide for how to express user intent/tasks. As a result, this system does not focus on the more complex and user centric goals of our system.

The system that discovers devices and the services available in an ad-hoc environment. The system also identifies the services that could be aggregated and finally suggests the possible service combinations to the user. This approach doesn't provide for how to express user intent/tasks. As a result, this system does not focus on the more complex and user centric goals of our system [60].

3.4.7 InterPlay

Interplay [60]:

- Uses pseudo sentences for capturing user's intent;
- Uses device, user location, and capabilities for task prioritisation;
- Uses device capabilities-based task discovery for task checking/verifying.

A pseudo sentence is a structured set of terms designed to mimic a more complex and grammatically correct sentence. The minimal pseudo sentence representation for a task consists of a verb, a subject (content-type or content) and target device(s). An example of such pseudo sentence would be: "'Play' (verb), 'The Matrix' (subject), 'Living Room DTV' (target device)", which means "Play the DVD 'The Matrix' from the Bedroom DVD Player onto the DVT in the Living room".

Some advantages

- Provide a simple and intuitive means to help capture user's intent;
- Can be understood by users without the need to refer to a manual;
- Capture sufficient detail for mapping tasks to service invocations.

Some limitations

- The users need to explicitly express their intents.

Device, user location and capabilities-based prioritisation:

For example, if a user is using the TV in the Living Room and the available tasks in the home are (#1) "Play Music in the LivingRoom Stereo", (#2) "Play Music in the Bedroom Stereo", (#3) "Play Movie in the LivingRoom TV", it is very likely that the user would want to choose either task #3 or #1, and not choose task #2 because the user is not likely to listen to the music in the bedroom while she is in the living room.

To further process the prioritisation, the device capabilities are examined. In the previous example, the user is using the TV and the primary function of the TV is the watch movie/news, etc. Therefore, it is more likely that the user wants to watch movie than listen to music. The final result of the prioritisation is task #3, task #1, and then task #2.

Moreover, a task that involves devices that are located in same location is highly preferred compared to a task with devices from a different location.

Device capabilities-based task discovery:

Each device has its description specifications. There are two logical types: a device description type and a user task description type. A device description provides richer descriptions about the devices' capabilities and features, and a user task description describes task(s) supported by the device. These descriptions are related so that the task description describes a combination of devices and actions that are required to perform the task using the functionalities included in the device descriptions.

During the task composition process, the functional requirements of a task are checked against the functionalities advertised by the available devices in a home. If for a particular task, the required functionalities are available, then a task object is created. For example, there are "audio/mpeg" contents in a home, and there are devices that can take "audio/mpeg" as their inputs, then "Play Audio" tasks for these devices are created.

3.4.8 Task Management System

Radi and Mayrhofer [91] present a task management system on mobile devices for supporting teamwork and collaboration. The system uses context information for facilitating the creation of tasks in a task management application on mobile devices. The context information is also used to prioritise tasks and to improve the way the tasks being presented to the user. In order to maintain the tasks and their dependencies, the authors develop a task repository which offers functionality for task tracking and later adjustment.

3.4.9 Situation-Aware Task Recommendation

Cheng et al. [40] propose a system which recommends applications based on current situation. A situation is defined by a set of context values that are frequently associated with a pattern of user usages of a device. Context values recorded over a period of time constitute the user usage history which is used to extract latent situations and tasks frequently performed in the individual situations using unsupervised learning technique. At runtime, the system periodically senses the current situation (i.e. the set of current context values) and finds similar situations from the history. The tasks associated with the similar situations are ranked accordingly and recommended to the user.

3.4.10 Task Execution Framework

Ranganathan [24] propose a high-level task specification model and an autonomic task execution framework for ubiquitous computing environments. Some of the key elements of the framework:

1. Ontologies for specifying hierarchies of different kinds of entities and their properties in a standard, reusable way.

2. A novel semantic discovery algorithm based on ontological hierarchies for discovering appropriate resources for performing a certain task.

3. Context-sensitive policies, written as Prolog rules, that are checked while choosing appropriate resources.

4. A multi-dimensional utility function for choosing the "best" values of different task parameters.

3.4.11 Supporting Tasks in a Smart Home

Ranganathan and Campbell [76] propose a solution that allows developing high-level parameterised tasks that support users in their daily lives. The solution includes an autonomic task execution framework that automatically configures and repairs the execution of tasks depending on the current state of the environment, context-sensitive policies, and learned user preferences.

Task Modelling

A task is made up of a number of smaller sub-tasks called activities. Developers first develop basic activities using a high-level programming model called Olympus [24, 92]. These activities include operations such as starting, moving or stopping components, changing the state of devices, services or applications or interacting with the end-user in various ways. Task programs are developed that compose a number of activities into a task that achieves a certain goal. These programs include instructions on how the task execution framework can enhance the performance of the individual activities. The task execution framework decides exactly how the actions are to be performed i.e. which resources (devices, services, applications, locations, etc.) to use or which algorithms to employ. Task programs are written in a high-level manner, in terms of abstract resources or parameters and are not strongly tied into the characteristics of any particular environment. This enhances portability, and the same task can run unchanged in different environments.

In the framework, flexibility of task execution is achieved by parameterising tasks. Each task is associated with one or more parameters that influence how it is performed. The parameters may be devices, services or applications to employ while performing the task or they may be strategies or algorithms to use. When the task is executed, the middleware obtains the values of the different parameters by either asking the end-user or by automatically deducing the best value of the parameter based on constraints specified by the developer, the current state of the environment, context-sensitive policies, and user preferences. It can also recover from failures of one or more actions by using alternative resources.

Their framework writes task programs in C++ or in a scripting language called Lua [93]. Tasks may be initiated either by an end-user or automatically by the framework in response to an event. The framework also decides how best to interact with the end-user automatically (e.g. using a handheld device or a touch-screen or by speech, etc.).

A task is composed of three kinds of activities: *parameter-obtaining*, *state-gathering*, and *world-altering*.

- Parameter-obtaining activities involve getting values of parameters by either asking the end-user or by automatically deducing the best value. The descriptions of these parameters are in a task parameter XML file. In the case of parameters whose values must be deduced automatically, a Discovery Service is queried to get the best value. The Discovery Service has access to different ontologies and policies that specify properties of different entities and have rules that constrain the values of different parameters.
- State-gathering activities involve querying other services or databases for the current state of the environment.
- World-altering activities change the state of the environment by creating, reconfiguring, moving, or destroying other entities like applications and services. World-altering and state-gathering activities are written in the form of C++ functions. These activities are developed using the high-level *operands* and *operators* provided by the Olympus programming model.

The Olympus Programming Model

The Olympus programming model allows developers to write programs that consist of high-level operators and operands. The underlying framework resolves the high-level operators and operands into suitable low-level implementations and entities.

High-level operands in Olympus include services, applications, devices, physical objects, locations, users, and Active Spaces. Each is associated with a hierarchy in the ontology and developers can use any sub-concept of these basic types while programming.

High-level operators in Olympus operate on the high level operands to do one of the following:

1. *Manage the life-cycle of entities*: These include operators for starting, stopping, suspending applications and other entities.
2. *Query or change the state of entities*: For example, the state of a light may be on, off, or dim. High-level operators allow querying or changing the state of the light. Another such high-level operator exists for notifying a user of some information.
3. *Query or change the relationships between entities*: For example, operators exist to query the relationship between a user and a location, or for changing the relationship between a device and an application (e.g., move an application to a different device), etc.

Composing Activities to Form Tasks

The different activities are composed together to create a task program. Task programs are written in C++ or in a scripting language called Lua [93]. The variables in the task program correspond to classes in the ontology and the functions are activities that have been defined using the Olympus high-level programming model. The parameters of the task are declared in the external parameter XML file.

Task programs consist of several primitive activities borrowed from models of workflows like BPEL[4] such as invoking web services or throwing exceptions. The authors claim that the key difference between their task model and other workflow models is that their task models are more dynamic. Particularly, their middleware can configure how the task is executed by picking appropriate values of parameters depending on the current context and the end-user. In BPEL, however, the web services used and the pattern of interaction are pre-specified statically and are difficult to adapt to different situations.

Architecture

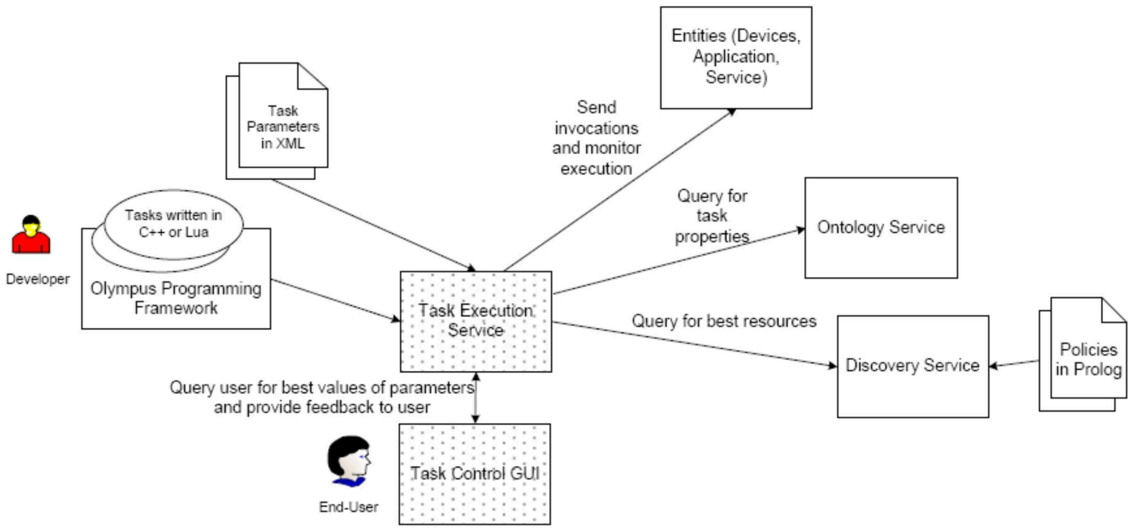

Entities (Devices, Application, Service)
Task Parameters in XML
Send invocations and monitor execution
Query for task properties
Ontology Service
Tasks written in C++ or Lua
Developer
Olympus Programming Framework
Task Execution Service
Query for best resources
Discovery Service
Policies in Prolog
Query user for best values of parameters and provide feedback to user
End-User
Task Control GUI

Figure 3.4: Overall architecture of Olympus framework

Figure 3.4 shows the overall architecture of the Olympus framework for developing and and executing tasks. Developers program tasks with the help of the Olympus programming model. The task programs are sent to a Task Execution Service, which executes the tasks by invoking the appropriate services and applications. The Task Execution Service may interact with end-users through a Task Control GUI to fetch parameter choices, give instructions, and provide feedback regarding the execution of the task. It also fetches possible values of parameters from the Discovery Service, which has information about the current state of the environment and policies in Prolog that help inferring appropriate and best ways of performing different kinds of actions. The Ontology Service maintains ontologies defining different kinds of task parameters, contexts, and other entities.

[4]www.oasis-open.org/committees/wsbpel/

Discussion

The authors have not evaluated their approach in term of usability and efficiency.

3.4.12 Task-Oriented Service Navigation System

Naganuma and Kurakake [30] present a service navigator. The user specifies a general task such as 'go to theme park'. A list of tasks that match this query is presented to the user. The user can select the intended task and then a corresponding detailed tasks are displayed accordingly. In the final step, associated services can be invoked by establishing an Internet connection to actual services. The system employs a task knowledge base which contains semantic descriptions of tasks and links to corresponding services. Although this system enables effective service retrieval, it behaves passive in requiring a user's initial input to trigger the recommendation process.

Fukazawa et al. [31] extend the system by organising tasks according to generic actions (e.g., go, drink, watch...). Once the user inputs a generic action he intends to do, the system suggests detailed tasks associated with this action.

3.4.13 Situation-Aware Task Recommendation

Luther et al. [42] propose a task navigation system extended from the task-oriented service navigation system [30] that can passively recommend tasks and services for users. The extended task navigator enables the delivery of situation-aware recommendations in a proactive way without a need for initial user inputs (as the previous does) to trigger the recommendation process. This is achieved by the integration of a situation engine and a situation-based task filter, meant to expose only those tasks which are relevant for a user in a given situation.

Architecture

Figure 3.5 depicts the overall system architecture. The architecture contains two main parts: the situation engine and the task navigator. The situation engine is to construct, classify, and infer situations according to the world-knowledge encoded in the situation ontology. The task navigator is to detect the most appropriate task nodes within the task ontology by matching the derived situation with the task-specific categories.

The task ontology stores descriptions for abstract as well as concrete tasks and their interrelations. Abstract tasks are annotated with enabling context conditions and concrete tasks are linked to appropriate information services via Uniform Resource Identifiers. Each task node is categorised according to the situation concepts such as 'Business Meeting', defined within the situation ontology.

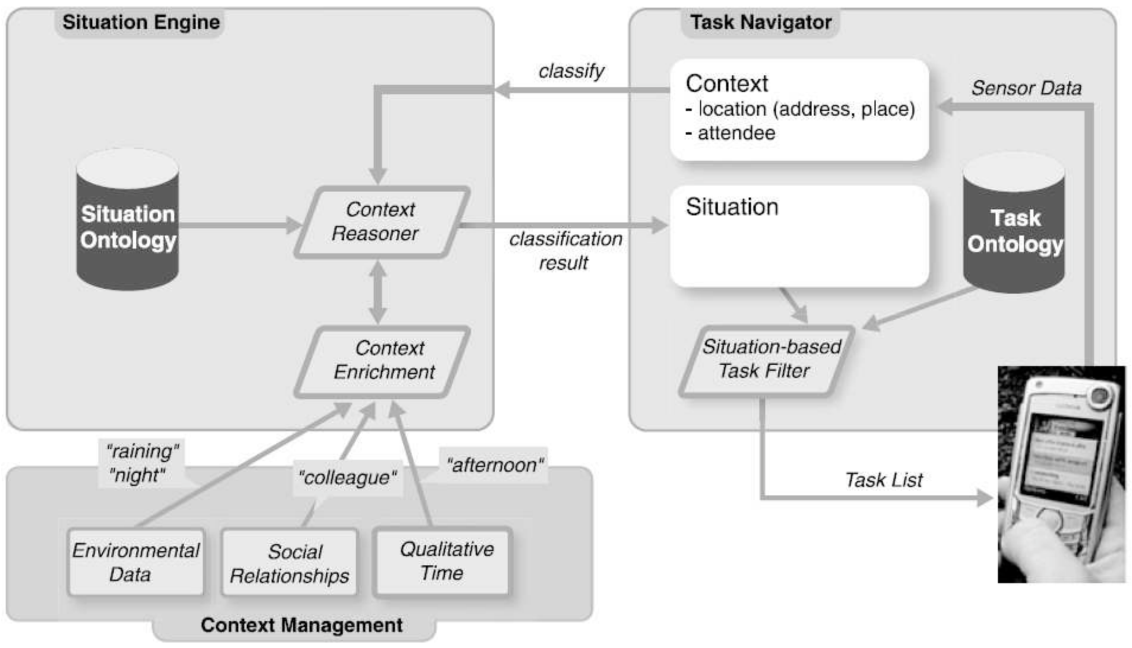


Figure 3.5: Architecture of Situation-Aware Task Recommendation System

Discussion

This system aims to guide users towards best-fitting services. This differs from our approach in which our system aims to recommend tasks, then to execute selected tasks by guiding the user through the task models.

3.4.14 The REACHeS System

Riekki et al. [62] present the REACHeS system that uses mobile terminals, each is attached with an RFID reader, as universal remote controls for the smart world. There are a set of physical icons attached with RFID tags advertising services. When a user wants to use a service, the user touches the icon advertising the desired service with her/his mobile terminal. The requested service replies by creating a user interface into the mobile terminal. As a result, the user then controls the service with the mobile terminal.

Sanchez et al. [94] present the idea of Touch & Compose. It is to assemble applications from the resources that the user has selected manually by touching them with her mobile terminal. Resources (devices, services, files, etc.) are represented with icons attached to real environment objects. RFID tags are placed under the icons; they contain data identifying the resources. The mobile terminal is equipped with an RFID reader. The touched icons are collected in the mobile terminal's resource stack; an application is composed from the resources in the stack either automatically or when requested by the user. Some resources collected from the environment can be stored permanently in the mobile terminal. The stack allows also sharing resources with other users at the environment.

3.4.15 UPnP

UPnP (Universal Plug and Play[5]) provides a standard for describing functional devices and device's services. XML device description holds an optional "presentationURL" tag with a link to a UI description. The device description contains useful metadata and the service descriptions list all "actions" a user can execute on the device, including the "arguments" and their data types. UPnP defines services and how to discover and utilise them. However, UPnP–as a peer-to-peer technology–is inherently device centric. Its services always relate to one specific device, whereas the services the user has in mind may involve an intelligent orchestration of several devices. UPnP descriptions are not compliant with the XML dialect that describes web services (WSDL). The reason is that the web services standard did not exist, when UPnP was designed.

UPnP is an industrial initiative to simplify the interconnection of devices in local networks. UPnP allows to discover devices and also to enumerate the features and services provided by each device. It also defines the communication protocol to invoke the services offered by the UPnP devices. UPnP offers the following features: (1) provides automatic discovery of any devices connected to the home network, (2) allows zero-configuration networking, (3) is independent of the type of devices and networks, (4) uses standard protocols and (5) offers easy system extensibility.

3.4.16 The CARUSO System

Kistler et al. [95, 96] present the UPnP-based CARUSO system which aims to address the following drawbacks of UPnP standard:

- Scalability: For discovery, UPnP uses multicast messages. But IP multicast does not scale very well on large networks.
- High latency: UPnP devices use HTTP/SOAP telegrams for communication. Hence, even turning on a light may result in hundreds of bytes of bidirectional network traffic.
- Access control: UPnP has no concepts of user and session. All devices and services are exposed without restriction. For authentication and security, UPnP relies on the underlying IP network.
- User interface of a device: UPnP does not describe the user interface of a device.
- Device-centric services: UPnP services are device-centric. This is not suitable for smart space where services should act across several devices. Moreover, UPnP is not designed for the Internet while it might be interesting to have services contributed from external providers via the the Internet.

The CARUSO separates the network of Control Points (CPs) from the network of devices (Figure 3.6) by using Control Servers. The servers is to segment the

[5]http://www.upnp.org/standardizeddcps/default.asp

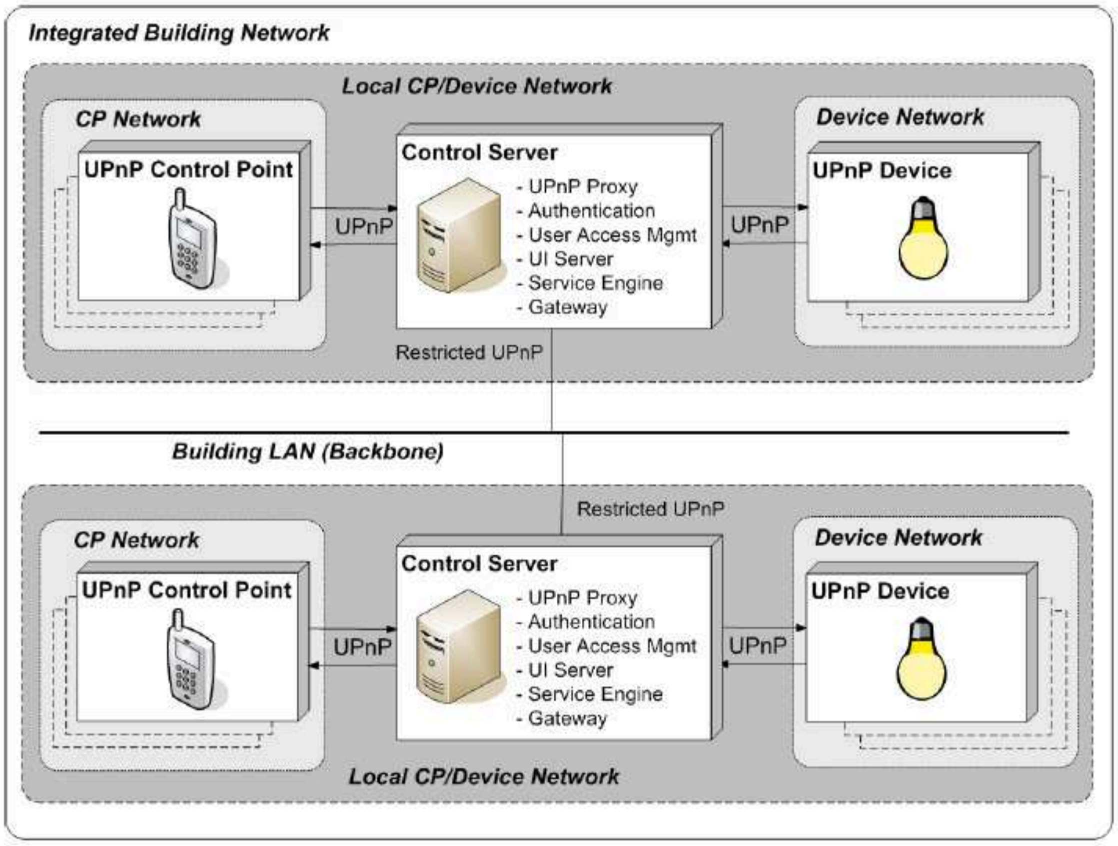


Figure 3.6: Architecture of the CARUSO system

whole network. This can solves the scalability problem and reduce network traffics. The servers have features for security, user interface rendering, and web services handling.

3.4.17 The Useware Markup Language

useML was originally developed for the definition of user interfaces in the field of production environments. useML was restricted to static user interfaces and to single devices or device families only. UDIT [97] is a Graphical Editor for useML. The *Use Model* is formalised through useML. The extended Use Model is the *Room-based Use Model* [98] which includes spatial information in the the Use Model.

3.4.18 SODA

SODA [15] is a standard alliance which is to eliminate the complexity and cost associated with integrating devices into highly distributed enterprise systems. SODA represents devices as a set of software services.

3.4.19 DLNA

DLNA (Digital Living Network Alliance) is a system that allows different devices in the same network to be connected with each other to share content easily and without the need of complex settings. DLNA uses a subset of UPnP-oriented features for exchanging multimedia content. In a DLNA network, there are two types of devices: clients and servers. Servers have contents available to offer, while clients use these contents.

3.4.20 NoTA

NoTA[6], Network on a Terminal Architecture, is an interconnect-centric system architecture for mobile and embedded systems based on SOA concepts and modularity at system level. NoTA is open, doesn't depend on a particular operating system and can be easily ported to different devices and platforms.

A NoTA based system is build upon two main logical elements: nodes and interconnect. A node represents a subsystem inside the device and consists in a set of hardware and software resources. There are two classes of nodes: Service Nodes (SN) and Application Nodes (AN). The communication is based on NoTA Device Interconnect Protocol (DIP).

3.4.21 Multi Access Service Platform (MASP)

The MASP is a runtime architecture which creates user interfaces from a set of models conforming to different meta-models.

3.4.22 DynaMo-AID Development Process

The DynaMo-AID development process is aim to obtain a context-aware user interface. The process consists of the design of several context-sensitive task models (using the ConcurTaskTree notation augmented with context information[4]). After the specification of these models, the supporting tool generates a prototype taking into account the models. Next the user interface can be deployed on the target platform.

3.4.23 Extended DynaMo-AID Development Process

This extension [99] of DynaMo-AID Development Process that uses task modelling for the development of distributed and multi-modal user interfaces. To make it easy for a designer to link modalities to tasks, they constructed an interaction environment ontology describing different modalities and the way these two concepts are related to each other.

[6]http://www.notaworld.org/

3.5 Task-Focused Interfaces

3.5.1 Introduction

According Wikipedia[7], *task-focused interface* is a type of user interface which extends the desktop metaphor of the graphical user interface to make tasks, not files and folders, the primary unit of interaction. Instead of showing entire hierarchies or lists of information, such as the contents document hierarchy, a task-focused interface shows the subset of the content that is relevant to the task-at-hand. This addresses the problem of information overload when dealing with large hierarchies, such as those in software systems or large sets of documents. The task-focused interface is composed of a mechanism which allows the user to specify the Task being worked on, a model of the task context such as a degree-of-interest (DOI) ranking [100], and a focusing mechanism to filter or highlight the relevant documents.

3.5.2 Technology

The primary goal of a task-focused interface is to scope the information shown in a computer application to just that relevant to the user's current task. Based on the user's interactions, each uniquely identifiable element of information available to the user is assigned a degree-of-interest (DOI) ranking. The more frequently and recently a user has interacted with an element of information, the higher the DOI for that element for that task.

The DOI rankings for the information elements can be used within a task-focused interface in four ways. Elements below a certain DOI threshold can be filtered to reduce the number of elements presented. Elements can be ranked according to their DOI; for instance, the elements of highest interest can be shown at the top of a list. The elements can be decorated with colours to indicate ranges of DOI. Finally, the display of structured information elements can be automatically managed based on DOI; for instance, text corresponding to elements with low DOI can be automatically elided.

The DOI value for each information element interacted with as part of a task can be derived from a stored history of interaction events recorded as the user works with the application. This approach requires a user to indicate the start of a task. The collection of all interaction events that take place during a single task is call a "task context".

The Eclipse Mylyn project[8] is an implementation of the Task-Focused Interface. Mylyn filters, sorts, highlights, folds, and manages tree expansion for numerous views within the Eclipse IDE based on the currently active task. Tasktop Pro[9] is a full-featured supported product based on Mylyn with additional productivity features.

[7]http://en.wikipedia.org/wiki/Task-focused_interface
[8]http://www.eclipse.org/mylyn/index.php
[9]http://tasktop.com

3.6 Common User Interface Resource Server for Intelligent Environments

Zimmermann and Wassermann [101] introduce a user interface resource server which supports personalisation, accessibility, context-awareness, openness for 3rd-party contributions, and management of user interfaces. Users can “carry their personal user interface with them”, even when they are on the move and using non-personal interaction devices such as a touch-screen kiosk on the airport, or an advanced universal remote control in a hotel room. The resource server itself could generate personalised (custom-tailored) user interfaces, based on an abstract user interface and user profile data. A resource server infers one ”best-match” pluggable user interface, based on data from the use context (user profile, device profile, situational context) and properties of user interfaces.

3.7 Device Integration

3.7.1 ATLAS[14]

3.7.2 SODA[15]

SODA[10] supports integrating a wide range of physical devices into distributed enterprise systems by representing these devices as a set of software services which can be reused and combined to address changing business priorities. The main goals of SODA can be summarised as follows:

- Provide higher-level abstractions of the physical device.
- Insulate enterprise system developers from the ever-expanding number of device interfaces, protocols, and low-level device-specific operational details.
- Bridge the device hardware and software realms by representing devices as high-level software services which can be dynamically discovered, accessed and controlled.
- Enable multiple rich applications to share the same set of deployed devices.
- Empower IT developers to modify and distribute application-specific business logic at any time without dealing with or affecting device-specific operations.

3.7.3 Ad-hoc Device Ensembles

Hellenschmidt and Kirste [102] propose ad-hoc device ensembles that carry out execution strategies spontaneously and behave like single devices the user can interact with.

[10]http://www.sensorplatform.org/soda/

3.8 Device Description Languages

When compare standards, the readability (automation), succinctness (comprehensibility), and scalability (expansion) of standards are issues of concern. Chen and Helal [103] provides a survey on these standards. It summaries the key differences among the standards with respect to their encoding schemes, design perspectives, and device models, and other criteria.

3.8.1 ECHONET

ECHONET[11] specifies an open system architecture that enables the integration of a variety of home appliances and sensors from multiple vendors. To describe the interface of these devices, ECHONET provides a device specification that explicitly defines their properties and access methods.

While ECHONET device specification is essentially a dictionary of various devices written, it provides little support for automation as plain-text specification is only readable to humans. More importantly, expansion of the device repository or upgrade of hardware would require changes of the specification, which jeopardises its stability.

3.8.2 IEEE 1451 Standard

The IEEE 1451 Standard[12] provides a set of open and network-neutral interfaces for connecting transducers to communication networks. The standard uses TEDS (Transducer Electronic Data Sheet) to specify information such as transducer identification, calibration, correction data, and measurement range. TEDS typically resides in the embedded memory of a transducer. The IEEE 1451 standard uses Interface Definition Language (IDL) to specify operation syntax, which reads like pseudo code and is programming language-independent.

3.8.3 SensorML

SensorML[13] uses XML encodings for describing devices. It aims to integrate complicated sensor systems into a "sensor web", an information infrastructure in which sensors are accessible as Web services. SensorML provides a generic model of measurement and post-measurement transformation, shielding its internal complexity from programmers by defining a uniform process interface.

[11]www.echonet.gr.jp/english/8_kikaku/index.htm
[12]http://ieee1451.nist.gov
[13]www.opengeospatial.org/standards/sensorml

3.8.4 DeviceKit

DeviceKit[14] uses XML encodings for modelling devices. DeviceKit assumes that each device has a transport layer and a connection layer. DeviceKit is an OSGi-based technology. Specifically, it provides an abstract and generic model so that application development for future devices is possible even when hardware-specific information is unknown.

3.8.5 Device Description Language [16]

The Device Description Language–DDL[15], which is a markup language using XML encodings, is capable of describing a variety of devices including sensors, actuators, and more complex devices. DDL assumes devices have no networking capabilities and are connected to applications via sensor platforms. DDL provides a DDL bundle generator which converts a DDL device descriptor document to a service bundle running under OSGi.

3.9 Approaches on Task Modelling

There are many approaches on task modelling as surveyed in [63]. However, Rich [61] points out that the main disadvantage of the traditional approaches is that task models are used only for user interface design at design time, if at all, then discarded. The next disadvantage is the device-dependance of task models. This limits the scalability of task models. Hence, another task model formalism is needed.

Earlier research in the task computing (TC) area has defined task-driven computing [58] and task computing [52], and demonstrated applications in a computer-supported environment. These earlier works simply treated a task as merely binding together a set of relevant computing applications (called virtual services) in a hierarchical or otherwise fashion, with task defined as the top-level virtual service. The fundamental problem with this approach is that it is too application-centric. Since these applications are only a means to carry out a task, they are not suitable to represent the task itself, or to capture user-related context information.

3.9.1 Using natural languages

This approach use performative of the speech act such as "I'd like you to display, given a product, the products references, descriptions, and prices" [104] or verb-object phrases such as "play video", "view on kiosk", "display slide show", "route from A to B", "send message to a user", and "view on projector" [78] to represent high level user's questions.

The limitations of this approach:
Although this approach allow end-users to easily to specify their tasks, the system

[14]www.eclipse.org/ohf/components/soda/index.php

[15]http://www.icta.ufl.edu/atlas/ddl/

must have an ability of understanding natural languages. This is a hard problem, or even impossible.

3.9.2 Low-level task models using ConcurTaskTrees by [17]

The authors use the ConcurTaskTrees notations [105] to model low-level tasks as shown in Figure 3.7. This model has the following features:

- It is a hierarchical and graphical structure. That means a task can be decomposed into subtasks.
- It allows to define temporal relationships between the tasks such as enabling, concurrent performance, choice, parallel composition, disabling, suspend-resume, order independence, independence concurrency, concurrency with information exchange, order independence, and enabling with information passing.
- It describes task allocation by indicating the category of the task including user task, application tasks, interaction task, and abstract task (tasks that have subtasks belonging to different categories).
- It also describes objects including user interface objects and application domain objects which needs for the task performance.
- How tasks exchange information can be defined using this model.
- It allows to specify the properties of a task such as iteration, finite iteration, optional task, and recurrence, preconditions, identifier, name, frequency of use, access rights, and estimated time of performance.
- Moreover, it allows to indicate cooperative tasks where two or more users are involved to perform the tasks. Using this model, a cooperative task is decomposed until the sub-tasks are performed by single users.

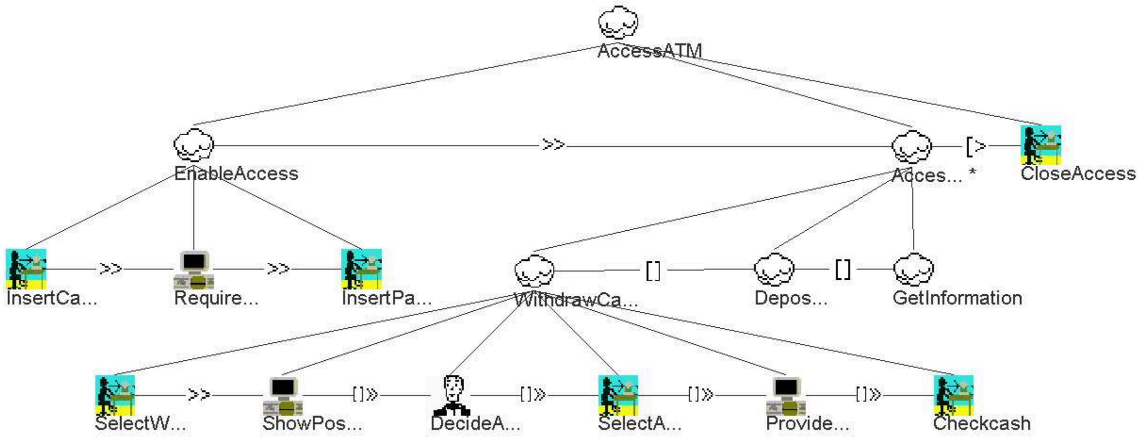


Figure 3.7: An Example of a Task model using ConcurTaskTrees notations

A tool called CTTE is developed to support editing and analysis of task models. The noticeable features of the tool are the follows:

- Editing task models;
- Checking completeness of task models;
- Multiple interactive views of task models;
- Comparing task models;
- Task model simulator which can be used for interactive walkthroughs of developed task model.

The limitations of this approach:

- Task models using this approach only support for designing of applications which are used in statically target environments with static goals and specific devices. It is impossible to make changes to the task model at runtime to adapt the application to specific circumstances.
- Objects associated with tasks are insignificantly abstract so that they are not best suited in dynamic environments.
- This model does not allow the user to interrupt a task and then resume it later in a different environment. However, in highly dynamic environments, users would like the task follow them between different environments.

3.9.3 Executable Task Models by [18]

This is an extension of task models using ConcurTaskTrees (CTT) by Mori [17] to allow dynamic execution of a task model. To do so, firstly, task states and transitions between those states are appended to the CTT task model as shown in Figure 3.8. Secondly, input and output ports are as interfaces of particular tasks which enable information exchange between tasks not directly connected.

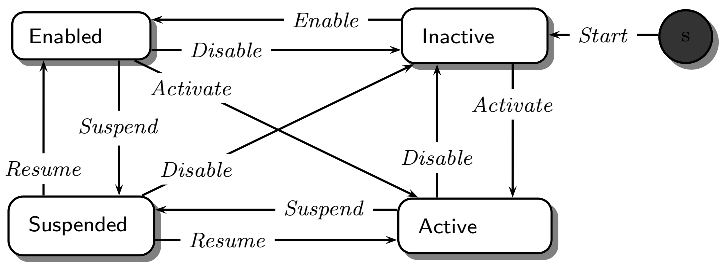


Figure 3.8: Task states and transitions in CTT

3.9.4 Task Graph based modelling and execution in MANETs by [19]

A Task Graph (TG) describes a high-level task. It is a graph with nodes and edges. The nodes represent for abstract devices (or a group of devices) which offers a single service needed by the task while the edges represent for the associations between the nodes. A edge describes orders, properties, conditions, and data communicated among the nodes. A tuple architecture is used for data flows between the nodes.

Execution of a TG (a task) is actually a instantiation of the TG into a MANET. This work present two optimal algorithms (centralised one and distributed one) for instantiation a TG into a MANET. These algorithms allows the system rapidly adapt to the disruption due to the environment changes by dynamic establishment of the segment of the TG affected by this disruption. Context information used in the progress of instantiation is proximity and devices specifications.

The centralised algorithm requires that the user device (where the task is originated) has to execute the algorithm and has the complete knowledge of the network topology at runtime.

The distributed algorithm requires that every device in the network has to store this algorithm and has an ability to execute the algorithm when needed. Moreover, the user device (where the task is originated) needs to execute a specific algorithm. This demands every devices has a copy of this algorithm if it would like to be a user node.

The limitations of this approach:

- This work does not deal with sharing devices and conflict problem due to concurrent task execution sharing the same devices in the environment;
- It does not mention about pre-context and post-context while executing a task. Context information may be useful for the instantiation of a TG into a MANET at runtime;
- The approach does not capture user's feedbacks and user's intents which can be employed in order to provide the best support meeting user's requirements;
- It requires devices involved in a task having an ability for executing the instantiation algorithms and having the complete knowledge of the network topology at runtime. This further requires powerful computation capability of the device. This is opposed to the device constrains.

3.9.5 Contextual ConcurTaskTrees by [20]

This work presents Contextual ConcurTaskTrees. This is an adapted version of the ConcurTaskTrees notation that combines the specification of dynamic context and tasks.

They define two views on context: context of use and context of rendering. The context of use is "all peripheral information relevant to the user's interaction with an application on a certain moment in time". The context of use can be split into two parts, internal and external, depending on what is the subject of the information. The internal context of use contains all information about the platform and the software that is used, all the other information is contained in the external context of use. The context of rendering contains the information, contained in the context of use, but translated so that it can be used directly to realise a user interface.

The work extends the ConcurTaskTrees notation with context tasks, tasks that cause a change in the context of use. These tasks can be performed by the user, the application, an interaction with the application, or by something or someone not directly involved in the interaction.

3.9.6 Using explicit program-like languages

They are able to represent composite tasks. An example of this approach is the *eco* family of languages [106]. This language allow to represent *eco procedures*, for example [78]: "make coffee; turn lights dim; wait for lights; download news; wait for news; show news on tv;". An eco procedure consists of *task statements* and *dependency statements*. A task statement is actually a particular command. A dependency statement specifies that a response from a certain task statement s_1 is a prerequisite for the execution of another task statement s_2.

The limitations of this approach:
Clearly, this approach is an extension of the approach of using natural languages. So, it possesses the same problem of understanding natural languages.

3.9.7 Context-Dependent Task Model [21]

They described a task by a union of the following vocabulary.

Task-ID a unique identifier of a task in a pervasive computing application.

Task-Name a string to distinguish a task and easy to understand for a user.

Condition a set of preconditions, or context information, that must be met before the task can be performed. The condition is specified in the form of parameters.

Priority this field denotes the importance and exigency of a task to further facilitate the execution, suspension and re-scheduling of tasks at runtime. For tasks that have the same priority, their relative importance will be determined by the priority of their respective parent-tasks.

Task-Contract this has two roles. One is to discover necessary resources and services for the task; the other is to organise and guide the steps of executing the task. A task contract can be defined by the following vocabulary.

- **Contract-ID** a unique identifier of a contract.
- **Contract-Name** to distinguish a contract and easy to understand for a user.
- **Parent-Task** to describe the relationship among different tasks, especially the parent-son tasks.

Requirement to express the necessary material and services. The Requirement field is different from Condition field in a task model. Condition depicts a situation surrounding a task (i.e. What), but Requirement describes the resources will be utilised in performing a task (i.e. How).

Procedure this field can contain two different sets of values depending on whether the task is composite or atomic nature. In the case of an atomic task, Procedure field will include a sequence of actions that will be executed by some services associated to either some automatic devices and/or software. Otherwise, this field will contain information of its "leaf" tasks. An example of a Task-Contract is given in Figure 3.9.

```
Contract ID: 1.1.3;
Contract-Name: Watching TV;
Parent-Task: Amusement;
Requirement: Light, Blind, TV, DVD, Hi-Fi speaker;
Procedure:
Begin{
(User sitting down on a sofa in front of TV) –user-initial-event
1. Turn on the light-1 or the blind; –sensor-ready-event
2. Turn on the TV; –TV done event
3. Tune to the favourite channel at the period of time; channel-ready-event
If –user-rejecting-event
Then{
3' Turn on the DVD; –DVD-done-event
4' search for popular movie; –movie-ready-event}
4. Adjust the volume of the Hi-Fi speaker to the style of movie; –Speaker-done-event
}End
```

Figure 3.9: An Example of a Task-contract extracted from [41]

3.9.8 Business Process Execution Language

One of the notable languages for describing tasks as business processes is Business Process Execution Language[16] (BPEL or WS-BPEL). WS-BPEL is an XML-based programming language to describe interactions between business processes and Web Services. A business process is a collection of related, structured activities that achieve a particular goal. The activities of the processes are Web services. Because human and device interactions are not in its consideration, there is a significant gap for many real-world business processes.

3.9.9 CEA-2018: Task Model Description

In CEA-2018, tasks can be defined in terms of subtasks (steps), and the atomic tasks can be "grounded" to actual device functions.

[16]http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

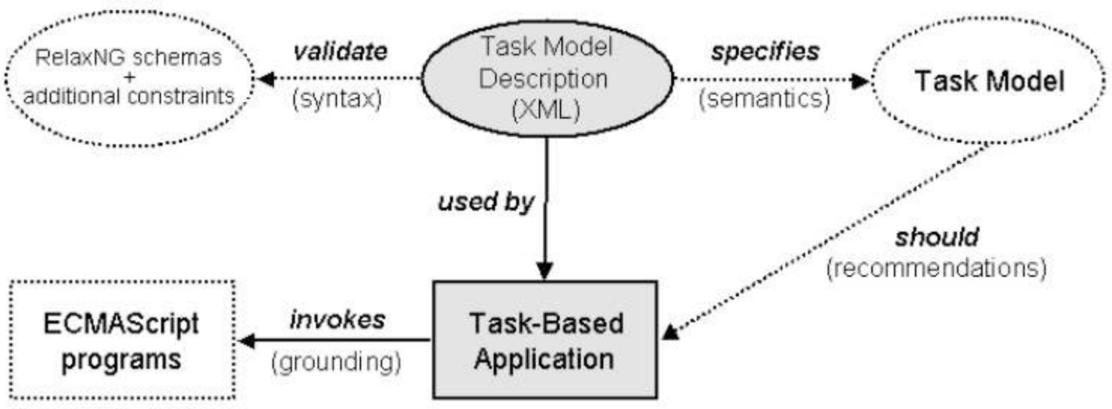


Figure 3.10: Conformance to the CEA-2018 Standard.

CEA-2018 [66] is a standard task modelling formalism which defines the semantics and an XML notation for task models relevant to consumer electronics devices. The standard does not depend on any specific home networking technology or infrastructure. The primary purpose of this standard is to specify the syntax and semantics of an XML document that a system will interpret at runtime to guide the user through the execution of the task.

Figure 3.10 illustrates the conformance approach of this standard. The centre of the diagram is a task model description. A task model description is used by a task-based application. A task model description specifies task classes, and representations of their intent and how high-level tasks can be decomposed into lower-level tasks. A task model description also contains JavaScript programs which ground primitive tasks to devices.

Task Classes and Instances

A task model contains task classes. For example, pressing the power button on a particular DVD player is an example of a task class. Slots (parameters) of this class might include *who* pressed the button, *which* DVD player was involved, and *when* the action took place. Thus, David Smith pressing the power button on the DVD player in his living room at 3:15pm on January 1, 2006 is an example of an instance of this class. A task instance may correspond to an actual event (called *task occurrence*) or it may not be (called *hypothetical task instance*).

Task-Based Applications

Figure 3.11 shows the system architecture that this standard assumes. In a task-based application, the user and system interact primarily in terms of high-level goals, which the system decomposes into primitive actions that are directly supported by the one or more devices involved. Communication between the task-based application and the device(s) is expected to be achieved via some kind of underlying network platform, such as UPnP.

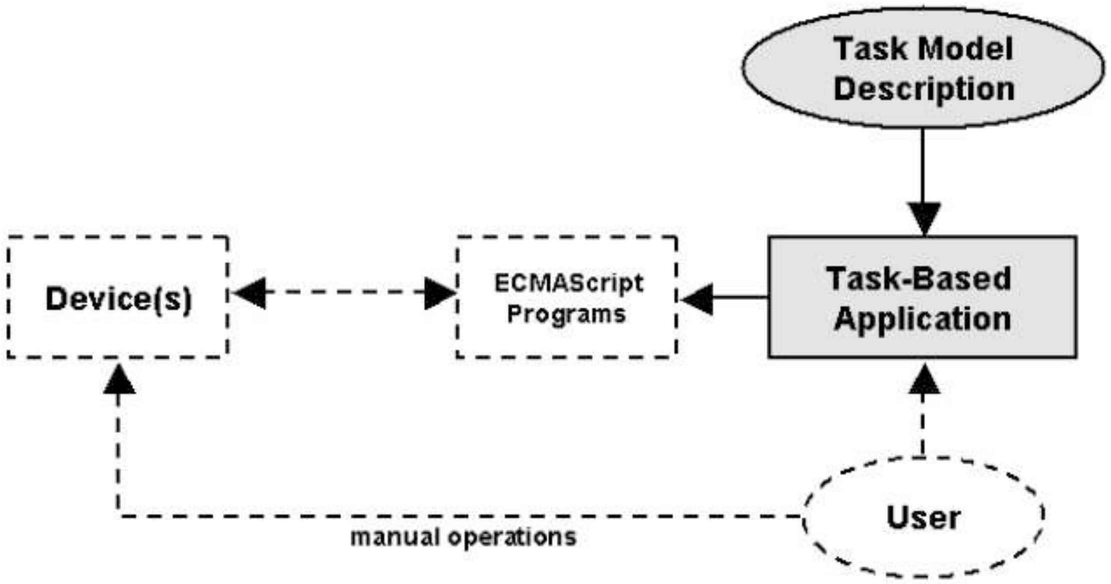


Figure 3.11: Expected architecture for using CEA-2018. ECMAScript is JavaScript.

Task Engines

Figure 3.12 shows a possible refinement of Figure 3.11, in which the task-based application has been implemented in two parts: a generic task engine and an application-specific user interface which uses the task engine.

The basic functions of the task engine are to load the task model description and to maintain a representation of the current status of the user's task instances. The advantage of this approach is that the effort of building a task engine can be amortised over many different task-based applications. The task engine does not interact directly with the user. If the task engine needs input from the user, it sends an interface request to the user interface; how this request is presented to the user depends on the specific user interface, e.g., it may be visual, spoken, etc. Furthermore, the task engine provides an API through which the user interface may access the current task state (task queries) or cause a certain task to be performed (task requests).

Grounding

In order for task models to be useful, applications need to connect (ground) aspects of a task model to the physical world of users and devices. Grounding is a bidirectional connection: applications need to execute tasks to change the state of the world and query the state of the physical world; applications also need to be notified when the state of the world changes due to external agencies, such as the user.

Advantages of CEA-2018

Rich [61] analyses the advantages of CEA-2018 as follows:

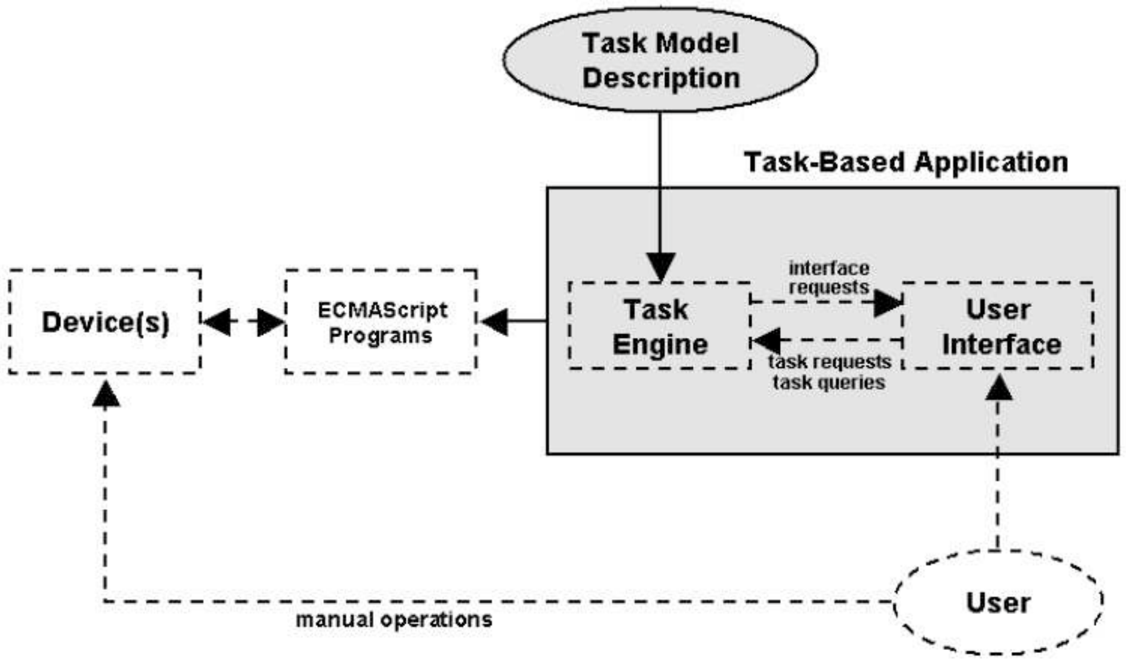


Figure 3.12: Generic Task Engine for CEA-2018.

- CEA-2018 is a standard which makes it possible for systems or devices from different domains and manufacturers to interoperate.
- While traditional task models are used only at design time, CEA-2018 supports runtime execution of task models.
- CEA-2018 tradeoffs between expressive power and computational tractability in task modelling. For example, ConcurTaskTrees [105] is more expressive than CEA-2018 but it is also more computationally expensive to reason about automatically.

Current Questions about CEA-2018

- Given a task model with some tasks already defined, the standard does not allow us to reuse/call these defined tasks within another task models.
- How can two task models be evaluated in terms of their functional similarity so that when one task is failed, the system can offer another similar task (substitution)
- It seems that this standard is only for defining task models which are achieved only by individual devices. How can we use this standard for defining multi-device tasks (i.e., whose accomplishment requires multi-devices) 5. Each task model is grounded to specific devices (via JavaScript code), so it is not abstract enough for the task to be achieved in different environments (support the same types of devices).
- How can the end-user configure task models for their preferences?

- How to allow the end-user in the configuration of the task operations such as start, stop, setup, suspend, resume, share, invite, join, leave.

3.10 Adaptive and Intelligent User Interfaces

3.11 Consistency of User Interfaces/User Interaction

3.12 Task Execution

3.12.1 Task Execution Framework [22]

They propose a framework for estimation of future resource requirements, which would allow the mobile applications to adapt to wearing (due to disassociation and reassociation) of resources. We propose a cognizant object model that would help predict the variation in availability of resources in dynamically changing and heterogeneous environment making the resource discovery protocols context aware. We propose a framework for resource discovery in mobile clients that would help seamless aggregation of resources for execution of the high level tasks in pervasive space.

3.12.2 An Algorithm for Task-based Application Composition [23]

We will present an algorithm for application composition which performs resource allocation in real time according to user requirements. We will present a resource model and a system architecture for dynamic application composition.

In this paper, we will focus on application composition, where the applications are dynamically composed from independent service components running on networking hosts. Composed applications are flexible in adaptation to changing resources and to the user needs, because the components building up the application are dynamically assigned to the resources.

Such applications can be run in different environments with diverse resource availability and they can be replaced and adapted without affecting the entire system. This increases code reusability and manageability of applications.

We will propose a resource model and a new allocation algorithm for application composition. Our algorithm differs from related work because we focus on user needs and take the dynamic applications quality of service (QoS) and resource availability into account. The algorithm is based on evolutionary computing. It optimises application structure, connections and service placement. The goal of the algorithm is to maximise the applications QoS, given a constraint set imposed by the environment and the user.

3.12.3 A Task Execution Framework for Autonomic Ubiquitous Computing [24]

We propose a framework that enables these autonomic ubiquitous computing environments to be self-configuring, self-repairing, and adaptive. The framework allows developers to program ubiquitous computing environments in terms of high-level, parameterised tasks. The values of task parameters may be provided by the end-user or automatically inferred by the framework based on the current state of the environment, context-sensitive policies, and learned user preferences. The framework uses a novel semantic discovery process to infer the best values of parameters. It can also recover from failures of one or more actions by using alternative resources.

When the task is executed, the task execution framework obtains the values of the different parameters in the task by either asking the end-user or by automatically deducing the best value of the parameter. The best value is deduced based on constraints specified by the developer, the current state of the environment, context-sensitive policies, and user preferences. Hence, a task can be executed entirely automatically (where the environment decides all the parameter values), entirely manually (where the end-user chooses all parameters), or anywhere in the space in-between.

The main contributions of the thesis are the proposal of a high-level task specification model and an autonomic task execution framework for ubiquitous computing environments.

Some of the key elements of the framework:

1. Ontologies for specifying hierarchies of different kinds of entities and their properties in a standard, reusable way.
2. A novel semantic discovery algorithm based on ontological hierarchies for discovering appropriate resources for performing a certain task.
3. Context-sensitive policies, written as Prolog rules, that are checked while choosing appropriate resources.
4. A multi-dimensional utility function for choosing the best values of different task parameters.

Chapter 4

A Survey on Context-Aware Recommendation Systems

Recommender systems have become an important approach to help users deal with information overload and provide personalised suggestions and have been successfully applied in both industry and academia. Recommender systems support users by identifying interesting products and services, when the number and diversity of choices outstrips the user's capability of making good decisions. One of the most promising recommending technologies is collaborative filtering [107, 108]. Essentially a nearest-neighbour method is applied to a user's ratings, and provides the user with recommendations based on how her likes and dislikes relate to a large user community.

Little research has been conducted to help users learn and explore a complicated pervasive interactive system using a recommender system. Typical approaches to proactively introducing functionality to a user include "Tip of the day", and "Did you know" [109], but these are often irrelevant to the user and are presented in a decontextualised way [110].

Personalised recommendation service aims to provide products, content, and services tailored to individuals, satisfying their needs in a given context based on knowledge of their preferences and behaviour [111]. The personalised services are usually realised by the form of recommender systems. Recommender systems appeared as an independent research field in the mid-1990s [111]. They help users deal with information overload by providing personalised recommendations related to products, content, and services, usually accomplished by the use of personal profile information and item attributes. In the past decade, most works focused on modifying algorithms for greater effectiveness and correct recommendations [112]. They used methods from disciplines such as human-computer interaction, statistics, data mining, machine learning, and information retrieval [111]. Recommender systems can be classified into three types according to how recommendations are made [111]:

Content-based Recommendation It recommends items to users that are similar to those they preferred previously. The analysis of similarity is based on the items attributes.

Collaborative Recommendation It recommends items to users according to the item ratings of other people who have characteristics similar to their own. The analysis of similarity is based on the users tastes and preferences.

Hybrid Recommendation It is a combination of content-based and collaborative recommendations.

Traditionally, recommender systems usually compute the similarity using two-dimensional user-item information. They failed to take into consideration contextual information which might affect users' decision making behaviour, such as time, location, companions, weather, and so on. Including human-in-context information as one system design factor is necessary for producing more accurate recommendations.

Adomavicius and Tuzhilin proposed a multidimensional approach to incorporate contextual information into the design of recommender systems [112]. They also proposed a multidimensional rating estimation method based on the reduction-based approach, and tested their methods on a movie recommendation application that took time, place, and companion contextual information into consideration. Here, recommendations are generated using only the ratings made in the same context of the target prediction. However, in fact, it is rarely the same context occurs in the future but instead the similar context. The disadvantage of that method is the increase of data sparsity.

Alternatively, Yap et al. [113] exploit a different way of incorporating contextual information and tries to improve prediction accuracy using a Content Based (CB) approach. The authors model the context as additional descriptive features of the user and build a Bayesian Network to make a prediction. They increase the accuracy even with noisy and incomplete contextual information.

4.1 Media Recommendation

4.1.1 xPod [25]

xPod keeps track of the music a user is listening along with their mood and activities, and uses machine-learning algorithms for recommending music based on the user's current activity.

4.1.2 CoMeR [26]

The CoMeR system uses a hybrid approach comprising a Bayesian classifier and a rule based method to recommend media on mobile phones. The Naive Bayes classifier is to determine an item's relevance to the situation context and the rule based scheme is to check the presentation suitability of a media item against device-capability context.

4.1.3 SCAMREF [27]

Context is classified into three categories: **Preference Context**, the context about user's taste or interests for media content, e.g. user requirements, user preference; **Situation Context**, the context about a user's spatio-temporal and social situation, e.g. location, time; and **Capability Context**, the context of physical running infrastructure, e.g. terminal capability, network condition.

They define user preference, terminal capability, location, time, etc., as context dimensions, and define modality, format, frame rate, frame size, score (similar to rating), etc., as QoS (Quality of Service) dimensions, which constitute the recommendation output.

Let $CD_1, CD_2, \ldots, CD_{N-1}$ be context dimensions, $QD_1, QD_2, \ldots, QD_M$ be output QoS dimensions, the recommendation model is defined as:

$$R : MediaItem \times CD_1 \times CD_2 \times \ldots \times CD_{N-1} \to QD_1 \times QD_2 \times \ldots \times QD_M.$$

The recommendation process consists of four steps:

1. They model both the media item and preference context as vectors. The cosine value of the angle between the two vectors is adopted as similarity measure between media item and preference context. The larger the similarity is, the more relevant between the media item and preference context.

2. They group the values of each situation context dimension into classes. For example, a user's home location can be divided into three classes: Living room, Bed room, and Dining room; social activities into four classes: At party, At date, Accompanying with parents, and Alone. They evaluate the probability of a media item belonging to a class of a context dimension or a combined situation context, e.g. how much probability of the movie *Gone With the Wind* is viewed by the user in *Bed room*, $P(\textit{Bed room}|\textit{Gone With the Wind})$. Suppose $C_1, C_2, \ldots, C_j, \ldots, C_k$ are k classes of situation context considered, the probability of media item $\overrightarrow{x}$ belonging to class C_j, that is, $P(C_j|\overrightarrow{x})$, can be calculated through statistical analysis of user viewing history. Given a class C_j, only the media items that have a high degree of $P(C_j|\overrightarrow{x})$ would be recommended.

3. The modality, format, frame rate, frame size, etc., of the recommended item must satisfy the capability context. They use rule-base approach to infer appropriate form from capability context.

4. The recommendation output consists of two parts: appropriate form and score. The score is composed of the similarity between a media item and the preference context and the probability of the media item belonging to the situation context $P(C_j|\overrightarrow{x})$. They use a weighted linear combination of these two sub-scores to calculate the overall score as:

$$Score = W_p * Similarity + W_s * P(C_j|\overrightarrow{x}),$$

   where W_p and W_s are weighting factors reflecting the relative importance of preference context and situation context.

Although the evaluation of this approach is rubbish but the method may be reasonable.

4.2 Context-Aware Information/Content Provisioning

4.2.1 Contextually Aware Information Delivery[28]

The authors adopt a semantic model for context sensitive message delivery. They model task, domain, location and devices using semantic language. The use of semantic based language gives their system inferencing capability, which is useful to understand the user's task context in a logical manner.

4.2.2 Context-Aware Content Provisioning [29]

The approach provides the right educational content in the right form to the right student, based on a variety of contexts and QoS requirements. They use knowledge-based semantic recommendation to determine which content the user really wants and needs to learn. Then They apply fuzzy logic theory and dynamic QoS mapping to determine the appropriate presentation according to the user's QoS requirements and device/network capability.

They designed three ontologies: a context ontology, a learning content ontology, and a domain ontology. The context ontology depicts the content already mastered by the student, along with his or her learning goals, available learning time, location, learning style, and interests. It also describes the hardware/software characteristics and network condition of the student's client devices. The learning content ontology defines educational content properties as well as the relationships between them. The relation *hasPrerequisite* describes content dependency information–that is, content required for study before learning the target content. The domain ontology is to integrate existing consensus domain ontologies such as computer science, mathematics, and chemistry. The domain ontologies are organised as a hierarchy to reflect the topic classification.

The content recommendation procedure consists of four steps:

Calculating Semantic Relevance Rank content according to how much it satisfies the student's context. The semantic relevance between the student's goal and content is the ranking criteria. Semantic relevance is calculated via the following steps:

1. Map the student's learning goal to the domain ontology.
2. Locate the content's subject in the domain ontology.
3. Estimate the conceptual proximity between the mapped element and the content's subject node. The conceptual proximity $S(e_1, e_2)$ is defined

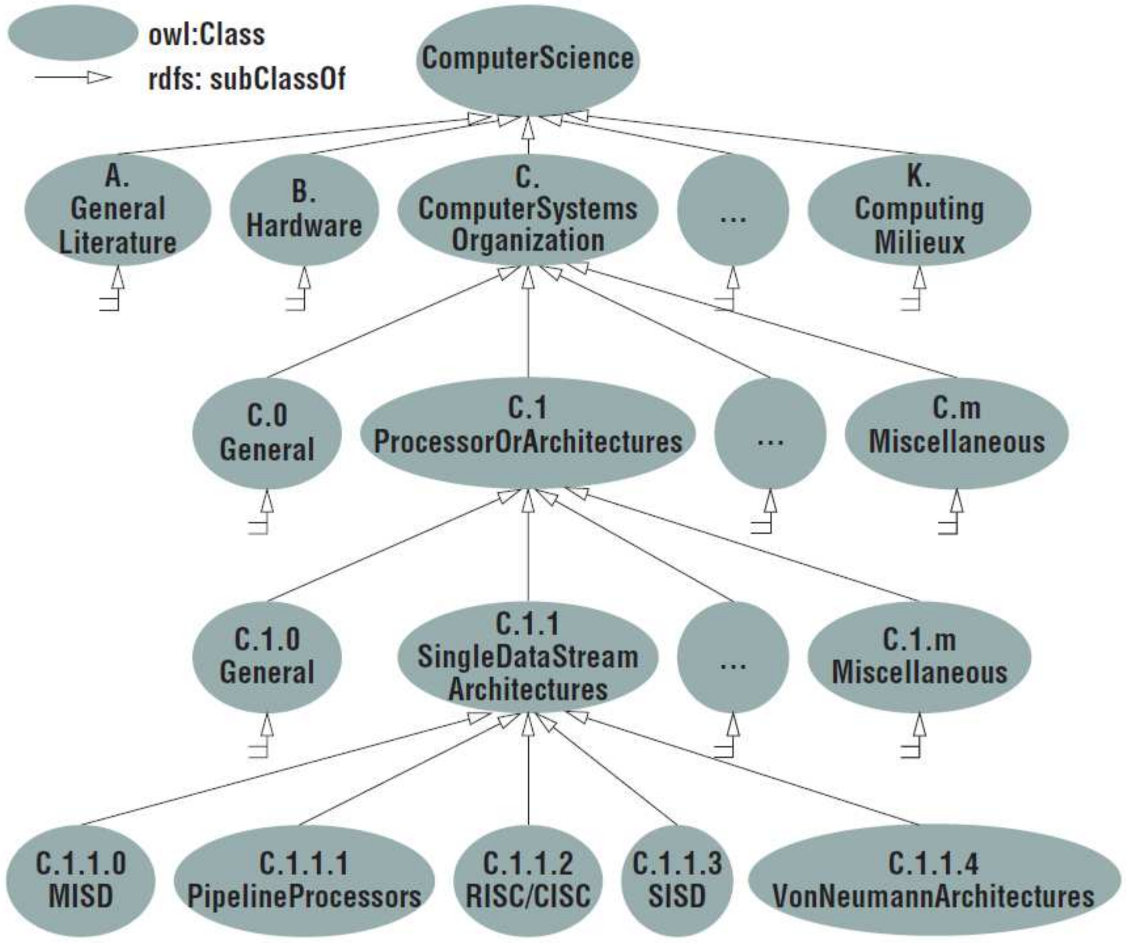


Figure 4.1: Computer Science Domain Ontology

according to the following rules (e_1 and e_2 are two elements in the hierarchical domain ontology):
Rule 1: The conceptual proximity is always a positive number.
Rule 2: The conceptual proximity has the property of symmetry–that is, $S(e_1, e_2) = S(e_2, e_1)$.
Rule 3: If e_1 is the same as e_2, then $S(e_1, e_2) = Dep(e_1)/M$. M denotes the total depth of the domain ontology hierarchy; $Dep(e)$ is the depth of node e in the hierarchy (the root node always has the least depth, say, 1).
Rule 4: If e_1 is the ancestor or descendant node of e_2, then $S(e_1, e_2) = Dep(e)/M$, where

$$e = \begin{cases} e_1 & \text{if } e_1 \text{ is the ancestor node of } e_2, \\ e_2 & \text{if } e_1 \text{ is the descendant node of } e_2. \end{cases}$$

Rule 5: If e_1 is different from e_2 and there is no ancestor/descendant relationship between them, then $S(e_1, e_2) = Dep(LCA(e_1, e_2))/M$. $LCA(x, y)$ means the least common ancestor node for nodes x and y.

Figure 4.1 shows the computer science domain ontology.
$M = 5$;

$LCA(MISD, SISD) = SingleDataStreamArchitecture$;
$Dep(LCA(MISD, SISD)) = 4$; hence,
$S(MISD, SISD) = Dep(LCA(MISD, SISD)) = 4/5 = 0.8$.
It is intuitive that two subjects with more detailed contents and closer ancestors are more relevant to each other–for example, two subjects under "SingleDataStreamArchitecture" are known to be more relevant than two subjects under "ProcessorOrArchitecture".

Refined Recommendations Students can refine recommendation results according to the specialty or difficulty of contents.

Specialty If the recommendation contains few items and the student wants more generalised content, the system can provide all contents whose subject is one level higher than the LCA in the hierarchy. Similarly, if the recommendation includes many items and the student wants more specialised ones, the system can return those contents whose subject is one level lower than the LCA in the hierarchy.

Difficulty Students can refine the recommendation by choosing easier or more difficult contents through the *hasDifficulty* property applying to each content. Each content segment is assigned a difficulty level when authored, such as "very easy", "easy", "medium", "difficult", and "very difficult".

Generating Learning Paths Learning paths are to guide the learning process and suggest prerequisites that a student must complete before tackling the target content. When the student selects an item from the recommendation list, the system generates a learning path that connects prerequisite contents with the target content. It does this by recursively adding prerequisite content until the path reaches the content that has no prerequisites, and then it prunes the path based on the student's prior knowledge. The *hasPrerequisite* relation of a particular content provides the prerequisite course information.

Augmenting Recommendations Recommendation Augmentation is references to examples, exercises, quizzes, and examinations related to the main course the student is studying. It does this by aggregating the course contents through "*hasExample*", "*hasExercise*", "*hasQuiz*", and "*hasExamination*".

4.3 Service/Application Recommendation

4.3.1 Domain-, place-, and generic task-based methods [30, 31]

Tasks are categorised based on domain ontology, place ontology, and generic task ontology. Generic tasks are actions such as watch, view, drink, and so on. The users needs to provide information about where they currently are (e.g. at home), what the action (abstract task) they want to do (e.g. watch), what the object on which they will action (e.g. movie), and where they want the task happen (e.g. theatre). For example, "I am at home, I want to watch movie at a theatre".

The approaches requires that domains and generic tasks are pre-defined. Moreover, the selected tasks is assumed to be feasible. The users need to explicitly express their intents. The approach does not resolve with the issue of task feasibility and the user's situations.

4.3.2 Context-aware service discovery[32]

Matching context requirements of user tasks against context requirements of services. Semantic-aware service discovery is based on the matching algorithm proposed by [114].

4.3.3 Task-Oriented Navigation of Services [33]

In the task-oriented service navigator, the users seek for services by specifying a task they are involved, for example, "Move to station X", "Draw cash to buy a ticket", "Get on the next bus". The services which are associated which a task are offered to the users. Tasks are organised in a task ontology.

4.3.4 Gain-based Selection of Media Services [34]

Gain refers to the extent a media service is satisfying to a user in a particular context. The gain is computed by adopting user's context, profile, interaction history, and the reputation of a service. The computed gain is used in conjunction with the cost of using a service (e.g., subscription and energy consumption cost) to derive the service selection mechanism. A combination of greedy and dynamic programming based solution is adopted to obtain a set of services that would maximise the user's overall gain in the ambient environment by minimising the cost constraint.

The objective is to dynamically compute the gain from the media services in different contexts and to obtain a subset of services such that the overall gain of a user is maximised subject to the total cost constraint specified by the user.

User's context A particular context can be defined in terms of the user's location (where), the time of presence (when), the current activity of the user (what), the companion of the user (with whom) and the mood of the user (psychological status).

User's profile The user's profile stores some static user-specific information (e.g., sex, age), as well as their preferences for different media-related attributes (e.g. movie genre, actor, actress, preferred news types, sources, singer, and subject preference). The user's media-related preference attributes is a set of ⟨*media type, attribute, score*⟩ tuple, which is called AMP. The media type in AMP refers to the type of media, for example, movie, music, and news feed. The attribute refers to the metadata of the particular media type. The score refers how much a user likes the media service corresponding to the attribute's data item. For example, if ⟨*movie, genre, score*⟩ refers to a movie attribute that has two data items as ⟨*movie, action*, 70%⟩ and ⟨*movie, comedy,* 30%⟩, this

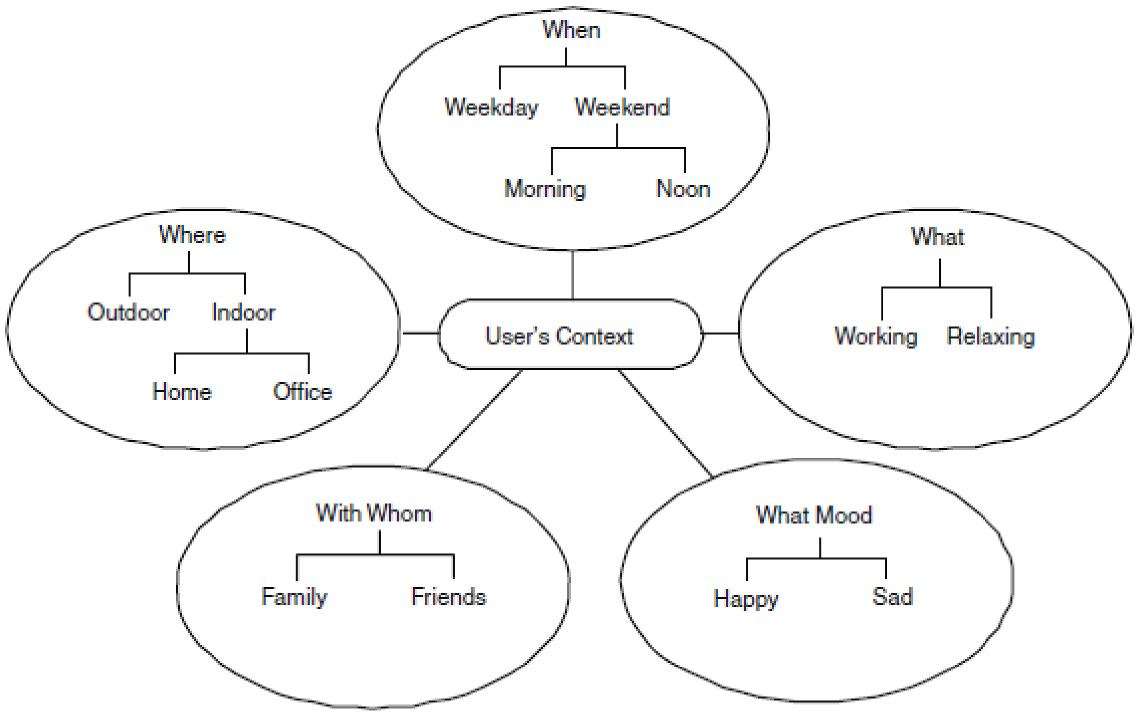


Figure 4.2: The different context parameters

reflects the fact that the user likes action movie more than that of comedy movie. These preferences can be either explicitly provided by the user or implicitly collected by the system. During the system initialisation phase, the user may choose to provide few entries of these preference attributes while the system can later use their interaction history to automatically update the initial scores provided by the user.

Interaction history It is used to update the scores of the data items for different attributes in the AMP. Additionally, it is used to obtain different patterns of service usage. For example, frequent service sets can be obtained from the historical data [115]. The frequent service sets may provide the recurrent patterns of service usage information where two or more services co-occur together frequently.

Media reputation It often refers to how good or bad a service is in terms of content, delivery and other factors.

But finally, this paper is rubbish.

4.3.5 Location-Aware Service Selection Mechanism [35]

In this approach, the geographical area of a target environment is divided among several service domains, where a set of services can be bounded with a service domain, such as specific library services while within a library. Service domains can be overlapping. The service selection mechanism is based on considering similarity,

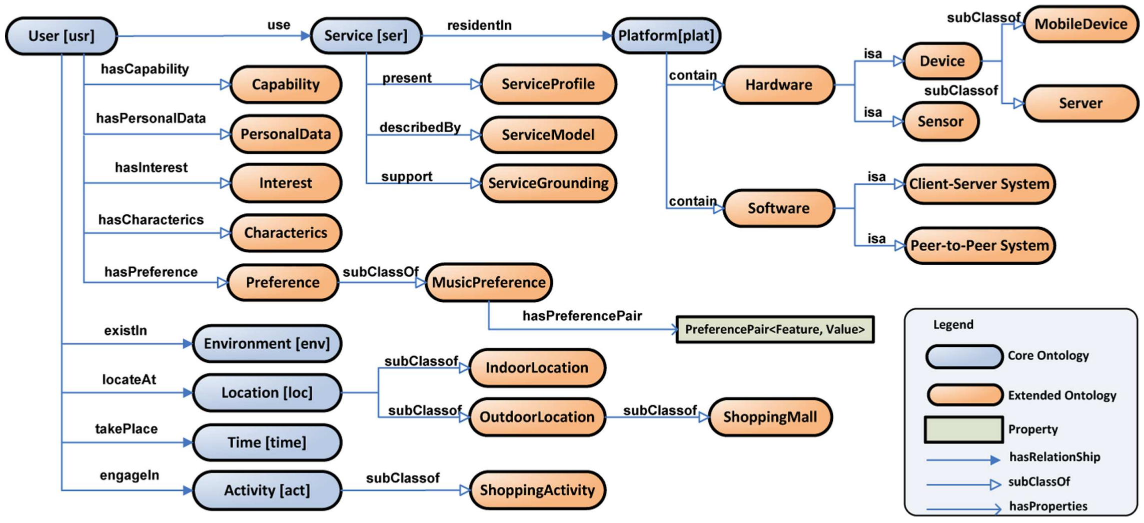


Figure 4.3: A partial context ontology

precedence, and restrictions among the services and on defining some aggregation rules.

4.3.6 MoBe [36]

MoBe allows the most relevant applications are selected by matching context and application descriptors. The applications, called MoBeLets, reside on the MoBe MoBeLet Server. Each MoBeLet is described by a MoBeLet descriptor that holds information related to the application (e.g., the type of task carried out, information about the minimal CPU/memory requirements, the kind of needed peripherals/communication media).

4.3.7 Spontaneous Service Provision [37]

The authors are based on the similarity degrees between the current user profiles and situation contexts with services' contexts to rank services. For computing similarity, they use Pearson's correlation coefficient.

Context Modelling The authors build an ontology-based context model using Resource Description Framework and Web Ontology Language. In the model, the context ontology is divided into two sets: core context ontology for general conceptual entities in smart environment and extended context ontology for domain-specific environment. The core context ontology investigate seven basic concepts of *User, Location, Time, Activity, Service, Environment, and Platform*. Figure 4.3 shows a partial context ontology.

They define context (including user profile, situation context, and service) as a n-dimension vector: $C = (c_1, c_2, \ldots, c_n)$, where $c_i, (i \in 1..n)$ is quantified as a context attribute (e.g., Activity, Location) ranging from -1 to 1. The similarity of context C_1 and C_2 is defined as,

$$Similarity(C_1, C_2) = \frac{C1 \cdot C2}{||C1|| \times ||C2||} = \frac{\sum_{i=1}^{n} \alpha_i \beta_i}{\sqrt{\sum_{i=1}^{n} \alpha^2 \sum_{i=1}^{n} \beta^2}},$$

where $C_1 = (\alpha_i), C_2 = (\beta_i), (i \in 1..n)$.

Service Description The service description includes a important set of *Dependency* attributes. Each *Dependency* has two properties *Feature* and *Value*. The former is to describe a context attribute, and the latter one is to measure the semantics relevance degree of that context attribute to the service.

This work has not provided an evaluation the system usability.

4.3.8 CASUP [38]

CASUP can provide users with personalised services using context history. Each instance in context history database consists of user' profile, high-level context and the service selected by the user in the given context (e.g., Smith, Male, 25, Dinner Context, Family restaurant Service). Context history is used to extract user preferences using classification such as decision tree algorithm. Association rules representing the relationship among the services or service sequences are extracted for recommending the next service. They apply the Apriori algorithm [116] to locate association rules.

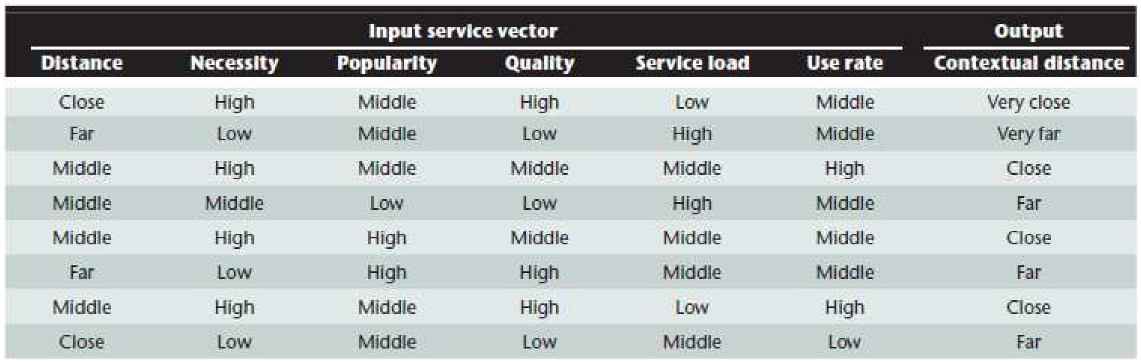
4.3.9 Personalised Service Discovery [39]

This aims to provide mobile users only services that fit their preferences and are appropriate to their context. Their framework was based on Virtual Personal Space (VPS)–a virtual administrative domain of services managed for each user.

The framework operates as follows. Services automatically send their advertisements to an appropriate directory (how to discover an appropriate directory?). The advertisements carry services' contextual attributes such as name, category, physical position, popularity, quality, service load, and required services. Directories can propagate queries to adjacent directories if they don't have appropriate services for the queries.

When a user moves into a new place, a service crawler automatically finds available directories and retrieves service advertisements. Then the services that suit the user's context and preference are added into the VPS. The services that do not suit the user's context and preference are dropped from the VPS. When a user inputs a service query, the system first searches the VPS. If it fails to find any appropriate services, it sends the query to a local service directory.

To find personalised services, they employed the adaptive-network-based fuzzy inference system [117]. Each service is represented by a service vector that includes location, distance, necessity, popularity, quality, service load, and user rate. These parameters are directly obtained from the service advertisement or calculated using



Input service vector						Output
Distance	Necessity	Popularity	Quality	Service load	Use rate	Contextual distance
Close	High	Middle	High	Low	Middle	Very close
Far	Low	Middle	Low	High	Middle	Very far
Middle	High	Middle	Middle	Middle	High	Close
Middle	Middle	Low	Low	High	Middle	Far
Middle	High	High	Middle	Middle	Middle	Close
Far	Low	High	High	Middle	Middle	Far
Middle	High	Middle	High	Low	High	Close
Close	Low	Middle	Low	Middle	Low	Far

Table 4.1: Fuzzy-variable-based rules

service information and user context. Necessity is the number of services in the VPS that require the service. Use rate represents how often the user employs this class of services.

The service vector is used to calculate a value denoting the contextual distance, which describes the service's contextual proximity to the user. The system uses a set of fuzzy rules, as Table 4.1 shows, and makes a decision by applying these rules on a service vector.

To accommodate the differences of users' preferences, the system uses feedbacks to reflect user preferences. When the user employs a suggested service, the feedback is *immediate*; if the service is already included in the VPS, the feedback is *positive*; if the service isn't included in the VPS, the feedback is *negative*; if the service is not used, the feedback is *negative*. The learning affects each fuzzy variable's membership function. The system can learn the fuzzy meanings as it repeatedly performs personalisation and receives feedbacks. Thus, the system starts with a set of general rules defined by system developers, but it gradually reflects the user's personal preferences.

To evaluate, they compared their system to other management models such as the location model, the quality model, the least recently used model, the rule-based model (which manages services that satisfy the rules). They compared their hit ratios. In the their model, 70% of the discovery queries found an appropriate service, but other models had only 30% to 50% hit ratios.

4.3.10 Situation-Aware Applications Recommendation on Mobile Devices[40]

Cheng et al. [40] use unsupervised learning (Minimum-Sum Squared Residue Co-clustering [118]) to extract patterns from user usage history. A pattern contains a situation and applications frequently performed in the situation. The system periodically senses the user's current situation, finds similar situations it has learned from the history, ranks the applications typically performed in the similar situations, and recommends applications by their ranks. Situation similarities are identified by computing the Euclidean distances between the current situation and the situation part of every co-cluster centroid. The application part of the centroids of the identified similar situations are then ranked for recommendation.

This approach using the unsupervised learning technique, specifically co-clustering, to derive latent situation-based patterns from usage logs of user interactions with the device and environments and use the patterns for task and communication mode recommendations.

Some advantages

- No need for predefined situations;
- No need for user-defined profiles;
- Do not require users to proactively train the system;
- Able to adapt to user habit changes;
- Accounts for many context variables.

Definition of "situation" Situation is a set of relevant context values that are frequently associated with a pattern of user usages of a mobile device [40].

Patterns of usage Patterns of usage are patterns from user usage history. The history contains interactions between user and his/her mobile device along with the context in which the interactions occurred. A pattern contains a latent situation and tasks frequently performed in the situation.

The operation of the system

1. Periodically recognises the user's current situation;
2. Finds similar situations it has learned from the history;
3. Ranks the tasks typically performed in the similar situations;
4. Recommends matching tasks by their ranks when the user asks for recommendations.

Some limitations

- Requires the user usage history data. Hence, for the first time of use, the system may behave inappropriately because it has not enough history data for recognising situations.
- Memory restriction may limit the usability of the system which requires a large of history data.
- Do not verify the feasibility of the tasks.

4.4 Task Recommendation

4.4.1 Why do we need task recommendation systems?

- Help users to find services which they may not know in advances;
- Help users to carry out their tasks effectively, automatically, or semi-automatically thanks to support from the computation-embedded surroundings;
- Suggest users tasks they are intent to do and help them to do the selected tasks.

4.4.2 Context-dependent task discovery [41]

The authors introduces a concept of active task which is exactly determined by the current context. A task is described by a 5-tuple

$$\langle Task{-}ID, Task{-}Name, Condition, Priority, Task{-}Contract \rangle$$

In order to discover an active task in a particular context, an active task discovery mechanism is proposed. The idea is to match the Conditions of individual tasks T with the current context values using Condition's similarity:

$$dis\Big(T(c), T'(c)\Big) = \sum_{j=1}^{n} w_j * dis\Big(v(c_j), v'(c_j)\Big)$$

where $c_j, j = \overrightarrow{1..n}$ is a context attribute of the Condition and $v(c_j)$ is its expected value while $v'(c_j)$ is its current value. w_j is the *attribute weight* of c_j where $\sum_{j=1}^{n} w_j = 1$. The attribute weights are explicitly specified in task descriptions. And

$$dis\Big(v(c_j), v'(c_j)\Big) = \frac{|v(c_j) - v'(c_j)|}{dom(c_j)}$$

where $dom(c_j)$ means the maximal difference of two values $v(c_j)$ and $v'(c_j)$.

The range of $dis\Big(T(c), T'(c)\Big)$ is $[0, 1]$, a value of zero means perfect match and 1 meaning complete mismatch.

Some limitations

- Fixedly assigning task priority may be unappropriate in PCEs because the priority of tasks is often dynamic time by time and user by user depending on user's intention. Moreover, user's intention may change over time depending on their situation.
- The work provides a method for negotiation between condition of individual task and context information. But they do not mention about discovery of concurrent tasks. In fact, there may be multiple tasks needed to be concurrently performed in a context in which they may sharing some resources.

- In reality, when the Condition's similarity is often not a zero value, how the system should behave to help the user instead of saying the system cannot do the tasks. Moreover, in the case that the Condition's similarity is ideally a zero value (e.g. perfectly matching), but the task does not meet the user's intention, how the system trade-off between the relevance and the possibility of tasks?

4.4.3 Homebird [12]

Homebird can discover features of other devices automatically and suggests to the user that certain tasks can be performed together with those devices. The set of available tasks can be triggered to change by arbitrary events–for example, a newly discovered network device. The logic of tasks is encapsulated in modules called plug-ins that can be written separately. Homebird automatically loads plug-ins found in the environment. When a task is selected, the control is handed back to the respective plug-in that can then show its own UI customised for that task. All the plug-ins use the UPnP protocol for communicating with other networked devices. the user study shows that users wanted to be able to customise which tasks appear.

4.4.4 Situation-Based Task Recommendation [42, 43]

The system reasons about a user's current situation based on a predefined situation ontology. Tasks which are associated with the inferred situation (found in a pre-specified task ontology) are retrieved. Then, the corresponding services that may be helpful for the retrieved task are recommended.

This work introduces a situation-sensitive task navigation system which expose only those tasks that are relevant to user's inferred situation. To do so, situational reasoning, which applies classification-based inference to qualitative context elements, is integrated in to the system. High-level qualitative context elements are formulated in the Web Ontology Language (OWL).

The system operates as follows. The current situation is inferred from the current context information and the situation ontology (see an example in Figure 4.4). A list of abstract tasks inferred from this situation using ontology-based task categorisation is shown to the user. Now, the user can select their desired task, then a corresponding sub-task list is displayed. Repeatedly, in a final step, associated services can be invoked to carry out the selected task.

The task ontology is hence required. A part of the task ontology is shown in Figure 4.5. Tasks are categorised according to the high-level situation concepts such as '*Business_meeting*', defined within the situation ontology. The enabling context conditions are encoded as corresponding OWL-S service profiles.

Some limitations

- Requires that the situation ontology and the situation ontology-based task ontology are pre-defined. However, what defines each situation and what

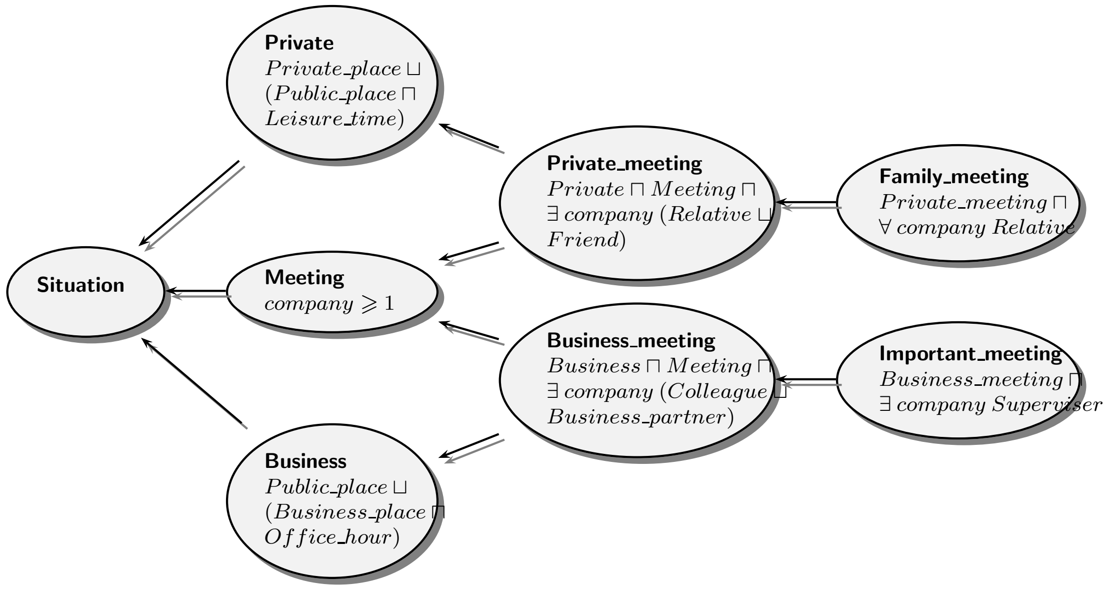

Private
$Private\_place \sqcup (Public\_place \sqcap Leisure\_time)$
Private_meeting
$Private \sqcap Meeting \sqcap \exists\, company\, (Relative \sqcup Friend)$
Family_meeting
$Private\_meeting \sqcap \forall\, company\, Relative$
Situation
Meeting
$company \geqslant 1$
Business_meeting
$Business \sqcap Meeting \sqcap \exists\, company\, (Colleague \sqcup Business\_partner)$
Important_meeting
$Business\_meeting \sqcap \exists\, company\, Superviser$
Business
$Public\_place \sqcup (Business\_place \sqcap Office\_hour)$

Figure 4.4: Situation ontology fragment

services are preferred in the inferred situation not only vary from user to user but also change over time. Therefore, this assumptions are impractical for ordinary consumers [40].

- Do not verify the feasibility of the tasks. It may assume that all required services and resources are somehow available for the tasks to be completed. However, in an ever-changed and dynamic pervasive environments, this assumption is not suitable.

4.5 Activity Recommendation

Chen [119] presents a context-aware collaborative filtering system which could recommend activities customised for a user for the given context (e.g., weather, location, and travelling companion(s), based on what other people like him/her have done in a similar context. To incorporate context into the recommendation process, the approach weights the current context of the active user against the context of each rating with respect to their relevance in order to locate ratings given in a similar context. One major problem of this approach is the availability of ratings in comparable contexts. The sparseness of ratings is an issue in collaborative filtering in general and further aggravated when integrating context.

4.5.1 Personalised Daily-Life Activity Recommendation [44]

By using a flexible concept hierarchy and a dynamic clustering method, the authors provide a recommendation service highly related to the users' context, based on the multidimensional recommendation model. Users can request for activity recommendations by providing their personal profile data and contextual information

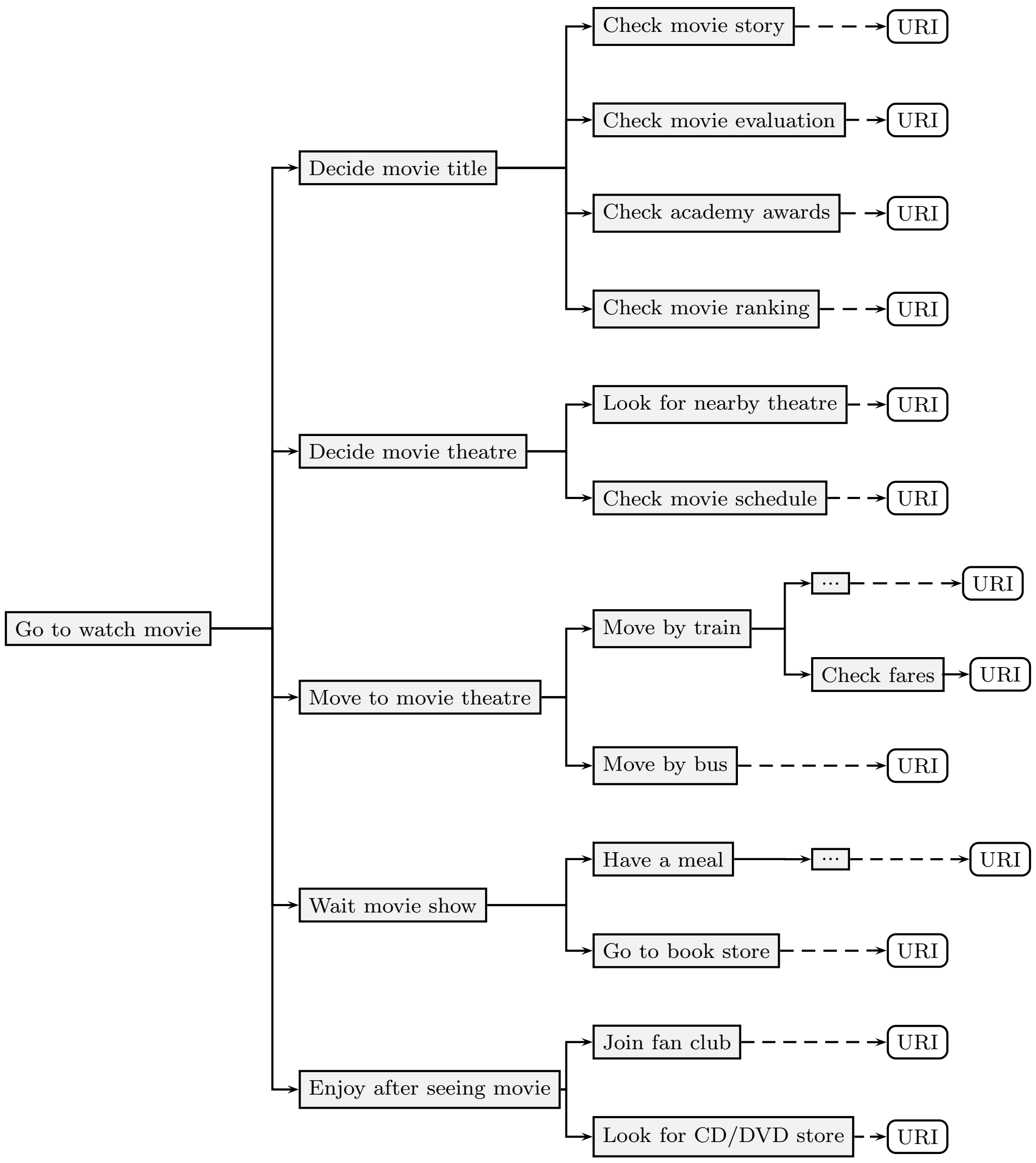


Figure 4.5: Task ontology fragment

through access devices. Name, age, gender, single/married, location, and some registered information are used as users' original profile data. Users are dynamically clustered based on contextual information, before making activity recommendations to users. At the same time, users can rate the recommendations and help to modify the accuracy of recommendations.

The approach uses the multidimensional model (MD model) proposed by Adomavicius and Tuzhilin [112] to store the information related to user, activity, and context factors, where each factor can be represented as a concept hierarchy. The MD model extends the concept of data warehousing and OLAP application in databases. This approach uses time, location, weather, and companions as the contextual information dimensions, and the recommendation space is defined as:

$$S = User \times Activity \times Time \times Location \times Weather \times Companion$$

In the MD model, a dimension D_i is the Cartesian product of attributes and can

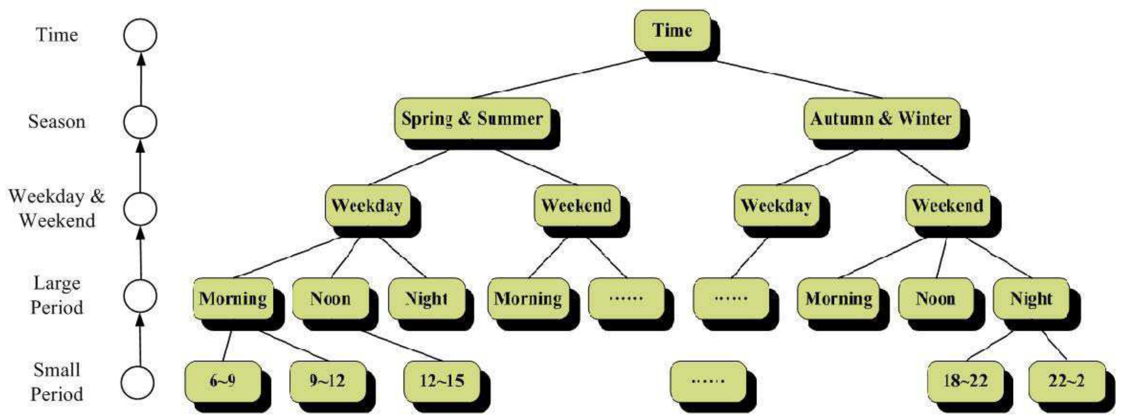


Figure 4.6: *Time* Concept Hierarchy

be expressed as $D_i \subseteq A_{i1} \times A_{i2} \times \ldots \times A_{ij}$. Each attribute A_{ij} defines a set of attribute values of one particular dimension. For example, the *User* dimension is defined as: $User \subseteq Name \times Age \times Gender \times IsMarried$. Similarly, the *Location* dimension is defined as: $Location \subseteq Country \times City \times Place$. For each dimension, the attributes can be represented as a concept hierarchy which consists of several levels of concepts. The top-down view of a concept hierarchy is organised from generalisation to specialisation; i.e., the higher the layer, the more generalised the layer. Take *Time* for example, its concept hierarchy can be expressed as Figure 4.6.

Given dimensions: $D_1, D_2, \ldots, D_n$, ratings are the *Ratings* domain which represents the set of all possible rating values under the recommendation space $D_1 \times D_2 \times \ldots \times D_n$. The rating function R is defined as: $R : D_1 \times D_2 \times \ldots \times D_n \rightarrow Ratings$. Based on the recommendation space $User \times Activity \times Time \times Location \times Weather \times Companion$, the rating prediction function $R(u, a, t, l, w, c)$ specifies how much user u likes activity a, accompanied by c at location l and time t under weather w, where $u \in User, a \in Activity, t \in Time, l \in Location, w \in Weather$, and $c \in Companion$. The ratings are stored in a multidimensional cube and the recommendation problem is to select the maximum or top-N ratings of $R(u, a, t, l, w, c)$.

The computation of recommendations grows exponentially with the number of dimensions. The reduction-based approach can reduce the multidimensional recommendation space to the traditional two-dimensional recommendation space by fixing the values of context dimensions, and improve the scalability problem [112]. Assume that $R^D_{User \times Activity \times Time} : U \times A \times T \rightarrow Ratings$ is a three-dimensional rating estimation function supporting *Time* and D contains the user-specified rating records (*user, activity, time, rating*). It can be expressed as a two-dimensional rating estimation function:

$$\forall (u, a, t) \in U \times A \times T, R^D_{User \times Activity \times Time}(u, a, t) = R^{D[Time=t](User,Activity,Rating)}_{User \times Activity}(u, a),$$

where $D[Time = t](User, Activity, Rating)$ is a set of rating records by selecting *Time* dimension which has value t and keeping the values of *User* and *Activity* dimensions.

Another problem is rating estimation that $D[Time = t](User, Activity, Rating)$ may not contain enough ratings for recommendation computation. This approach uses the rating aggregations of time segment S_t that expresses the superset of the time t when insufficient ratings are found in a given time value t. The rating of $R(u, a, t)$ is expressed as:

$$R^{D}_{User\times Activity\times Time}(u,a,t) = R^{D[Time\in S_t](User,Activity,AGGR(Rating))}_{User\times Activity}(u,a),$$

where $AGGR(rating)$ is the rating aggregations of time segment S_t. For example, the rating prediction function for **weekend afternoon** might be presented as the formula:

$$R^{D}_{User\times Activity\times Time}(u,a,t) = R^{D[Time\in \mathbf{weekend}](User,Activity,AGGR(Rating))}_{User\times Activity}(u,a).$$

For evaluating the quality of the recommender system, they use predictive accuracy metrics to examine the prediction accuracy of recommendations. Predictive accuracy metrics are usually used to evaluate the system by comparing the recommender system's predicted ratings against the actual user ratings. Generally, Mean Absolute Error (MAE) is a frequently used measure for calculating the average absolute difference between a predicted rating and the actual rating. In addition, Normalised Mean Absolute Error (NMAE) represents the normalisation of MAE which can balance the range of rating values, and can be used to compare the prediction results from different data sets. According to related research, the predictive accuracy of a recommender system will be acceptable when the value of NMAE is below 18%. The approach uses the AllButOne method [120] as the data set selection strategy.

4.6 Recommending Mobile Applications

The recommender system [121] recommends mobile applications to users derived from what other users have installed and rated positively in a similar context (location, currently used type of device, etc.). When making a recommendation, the system retrieves the current user position, determines POIs in the vicinity and generates a recommendation based on this context information. The approach uses collaborative filtering to rank found items according to user ratings of applications in a second step. User ratings are collected implicitly by automatically recording when a user installs an application. The ratings are stored together with context information (time, location, used device etc.) to capture the situation when a rating was made.

4.7 Others

CityVoyager [122] can find and recommend shops that match each user's preferences. The procedure for finding shops that match user preferences is based on a place learning algorithm that can detect users' frequented shops. They use the unavailability in 5 minutes of GPS signals as evidence that the user has gone indoors.

Gas Stations Recommendation [123] uses a hybrid, multidimensional recommender system which takes driver preferences (user-specified), ratings of other users, the current location, and fuel level of a car into account. The approach first filters items based on preferences and context, and takes ratings of other users and additional information into account.

4.8 Best Recommendation for the whole group

4.8.1 Let's Browse [45]

Let's Browse recommends web pages to a group of users who are browsing the web.

4.8.2 MusicFX [46]

MusicFX used in a fitness center to adjust the selection of background music to best suit the preferences of people working out at any given time. A special feature found in this system is that a group is composed by people who happen to be in the place at the same time. MusicFX uses explicit preferences of all participants to make a music selection that will be listen by everyone who is present. In this case, the group is composed by strangers rather than family members or friends.

4.8.3 Intrigue [47]

Intrigue recommends attractions and itineraries by taking into account preferences of heterogeneous groups of tourists (such as families with children) and explains the recommendations by addressing the requirements of the group members. Attractions are separately ranked by first partitioning a user group into a number of homogeneous subgroups with the same characteristics. Then each subgroup may fit one or more stereotypes and the subgroups are combined to obtain the overall preference, in terms of which attractions to visit for the whole group.

4.8.4 Travel Group Recommendation [48]

The recommender system performs the travel group recommendation task based on the formalism of distributed constraint optimisation problem.

Chapter 5

Proposal: Task Computing Framework

Pervasive computing systems can offer a plenty of features for users but also overwhelm them by the complexity and inconsistency in terms of how to operate these systems. This leads to reduction of the scalability and acceptance of new pervasive systems for everyday users. In order to address these issues, the user-centric approach to developing and operating pervasive systems appears to be a potential solution. This research develops a task-based framework which aims to eliminate the complexity and inconsistency of using pervasive systems. The ultimate goal is to provide the user a task-driven unified user interfaces based on user's situational context and needs. In this chapter, I will describe the design of the proposed task-driven framework.

In the next section, we will present our proposed architecture. Indeed, we will describe how the set of tasks to be generated based on the current context; how to express these tasks on user interfaces; and how a task to be accomplished.

5.1 The proposed architecture

5.1.1 Overview

5.1.2 Context-aware task selection

5.2 Objectives of the Framework

The overall design objective of the proposed task-driven framework is to reduce the complexity and inconsistency of using pervasive systems. The design of this framework will demonstrate the following key features:

- Allow task models to be abstractly described at design time;
- Base on user's situational context (such as location, devices near by), tasks are recommended for the user;

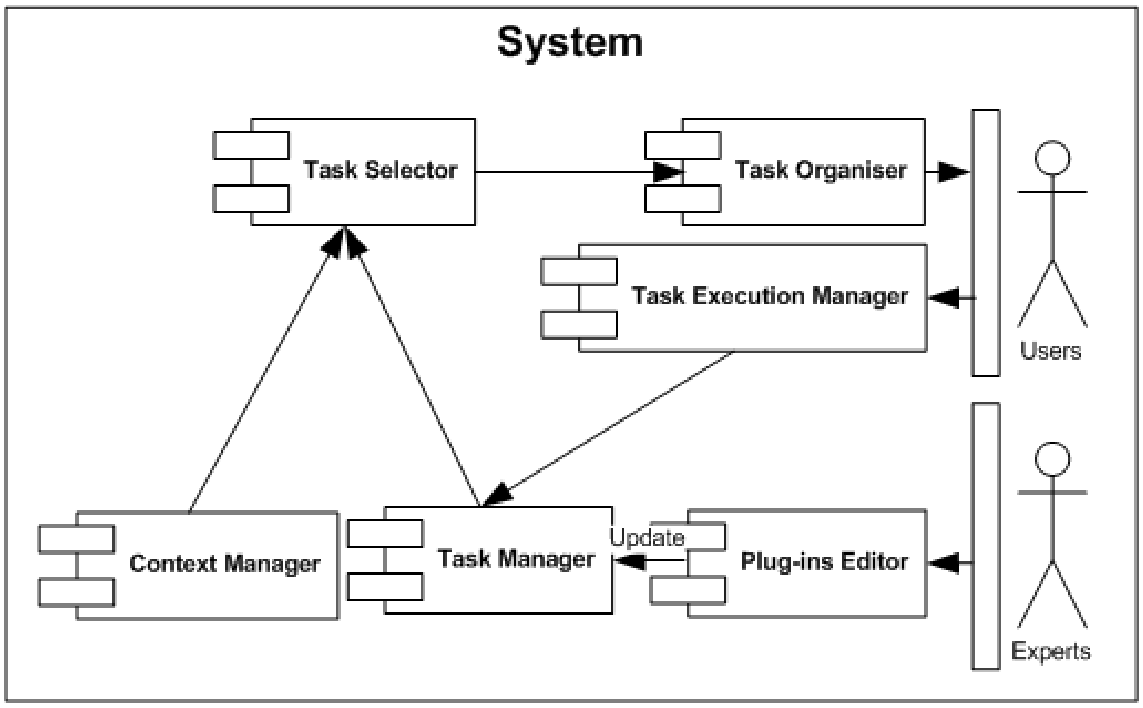


Figure 5.1: A proposed architecture of context-aware task selection

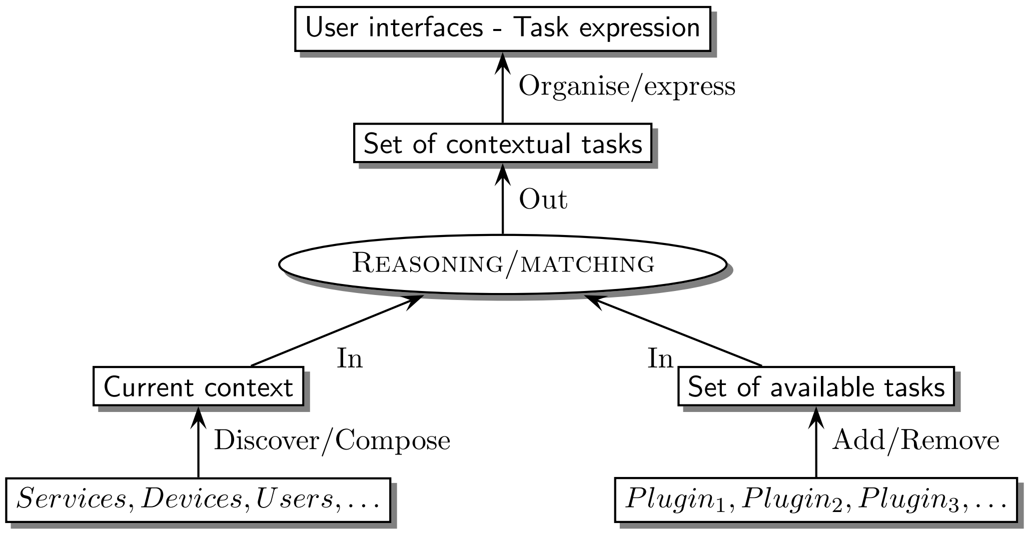


Figure 5.2: Diagram of the algorithm for context-aware task selection

- Allow users to search for tasks which are not listed in the recommendation;
- On the user's selection of tasks, the model of the selected task will be loaded and executed;
- Allow the currently executing task to be suspended and resumed on another environment if the conditions for resumed are satisfied;
- Able to manage available resources and services.

5.3 How to Describe Task Models?

I propose to use ANSI/CEA-2018 [66] for task model descriptions. I will extent this standard so that it can capture context information and support for searching task models.

5.4 How to Get User's Situational Context?

I will examine several context management systems and choose the most suitable one for this framework. I focus on contextual information of location and devices near by the user. So, a context management system would be accepted if it can provide user's current location information and available devices in the current environment.

5.5 The Design of the Framework

The framework should include the following modules:

Task Modelling Tool This tool provides an graphical user interface for the user to describe task models;

TaskRec This is a software system which is able to run on devices carried by the user. It recommends the user relevant tasks as well as allows the users to search for tasks by providing some searching conditions.

Task Execution Engine This important module is responsive for loading and executing selected tasks.

Context Management System This system is to acquire context of the user and the environment.

Resource, Device, and Service Management System This module discover available resources, devices, and services in the current environment.

In Figure ???, the proposed framework is shown. Here, the rectangular boxes represent system components and each arrow indicates information flow.

Context Providers track changes of contextual variables. They contact appropriate context services and stores all context states in a local database. When requested by the **Context Manager**, they provide information about known contextual variables for a user or a task of a specific time point.

Context Manager performs all the reasoning related to the context. It determines if a contextual variable is important for a task prediction, removes noisy context data and makes predictions for missing contextual variables.

User Model/User Profile Manager represents user information in the system. A user is modelled with his/her preferences. In collaborative filtering approach, this is a vector of task ratings.

Task Model/Task Manager represents task information in the system. It captures relevant knowledge in the application domain.

Model Adapter is responsible for integrating contextual data into the prediction algorithm. It takes information provided by the **Context Manager** and enhances the representation of the user/task model with context. This adds relevant contextual variables to the user/task model.

Recommendation Engine takes the enhanced data model and generates a list of rating predictions.

Explanation Engine takes the recommendations and provides the explanations for each of them. It could also use the **Context Manager** to find out needed information to motivate a recommendation because of the particular contextual conditions. The feedback of the user is recorded and is used to influence the **Model Adapter**.

Active situation: It is a situation within which the active user currently presents. It is also called current situation.

5.6 Task Repository

A task repository contains a set of task model descriptions. There are many strategies for building task repositories. They include *domain-based*, *place-based*, *device-based*, and *situation-based*. For example, common tasks in a home may be 'prepare food', 'listen to music', 'watch movies', and 'lock or unlock doors'. Common tasks in a conference room may include 'display slideshows' and 'give speeches'. Common tasks in car park may be 'find a parking space' and 'report a car accident'. Common tasks in a library may be 'book a study-carrel' and 'borrow a book'.

5.7 Resource Management

5.7.1 Environment Model

A pervasive environment arises from a set of connected resources such as an office space. From this set of resources, we consider three types of resources [4]: users, devices, and services.

Device We consider a resource to be a device if it runs a Resource Manager: a middleware software that supports a predefined set of events and operations to query and configure the environment.

Service A service provides specific functionalities (application logic) for interacting with a hardware resource (e.g. an interaction resource) or a computer program. Besides, it can offer embedded user interfaces that leverage its application logic.

User A user corresponds to a human who can interact with the surroundings through tasks and services.

The Resource Manager integrates the software services on a device and the hardware resources attached to the device such as a keyboard or display panel and publishes them in the pervasive environment as services. Devices can fulfill different roles: they can run in the background and host resources for hardware resources that have no computing power (e.g. a residential gateway, hosting the software services for the room lights) or act as interaction devices (e.g. personal devices carried by the end-user such as a laptop or a smart phone).

The Resource Manager queries the available services in the pervasive environment. Each service runs on a device and can be shared amongst other devices from where it can be accessed remotely through a proxy interface (described in some interface description languages (IDLs), e.g. WSDL[1]) or an end-user interface embedded in the service (described in some user interface description languages (UIDLs), e.g. HTML or XML User Interface Language[2]) or realised as mobile code (e.g. an approach adopted by Jini[3]).

The selection or generation of an appropriate user interface for a task may use existing user interface toolkits, modalities (graphical, speech) and adaptation strategies to present the interface. This can be realised by providing appropriate groundings for the IDL and UIDL descriptions.

5.7.2 Functional Requirements

First, the system needs to be aware of changes that occur in the environment's configuration and reflect these in its view. Therefore, it monitors the environment model in order to get notified of events that are triggered when resources enter or leave the environment. Apart from updating its view, the system may proactively propose rewiring strategies, e.g. if a new device becomes available that is better suited to execute a task.

Second, a basic set of operators is required to interact with resources:

- Present a task on a device by means of a user interface. A compatible user interface is distributed and rendered on the target device, e.g. a graphical user interface or a speech-based interface. If no target device is specified, the device running TaskRec that triggered the operator is considered the target device.
- Suspend a task and resume it afterwards. The state of the task and/or the user interface presenting it is stored until the task is resumed.
- Migrate a task from one device to another. The task is suspended on the source device and its context is transferred to the target device where the task is resumed.

[1]http://www.w3.org/TR/wsdl/
[2]http://www.mozilla.org/projects/xul/
[3]http://java.sun.com/products/jini/

- Invite a user to execute a task, for instance a task that is associated with a collaborative application. An invite is sent to the user's (default) device which is extracted from the environment model.

If we consider tasks as the building blocks of a pervasive application, end-users must be able to start and manage them (present task, suspend task, resume task). Besides, in order to exploit the heterogeneous nature of a pervasive computing environment, end-users must be able to traverse tasks to those devices best suited for executing the task (migrate a task to a device) and collaborate with other users (invite a user to execute a task).

5.7.3 Integrate and Configure Resources

Plug and play is often considered the default strategy to integrate devices in a pervasive environment. If a device is turned on or brought into a pervasive environment, it announces its availability and the list of services it supports in a broadcast message over the network. The Resource Manager takes care of the discovery of computing devices and builds up an environment model at runtime.

To avoid an explosion of resources, spatial information could be taken into account to display only those services and tasks in the user's vicinity.

Information about the coupling of tasks and their supported presentations and software dependencies is stored in 'groundings'. A grounding - the concept is adopted from OWL-S - dictates how a resource (e.g. a web service) can be accessed.

5.7.4 Case Study

improve the everyday life of a family by addressing vital aspects such as home care and safety, comfort, entertainment, etc. Mainly, this case study provides the following services (which were shown in Fig. 4):

- Multimedia management: allows inhabitants to store, manage and reproduce multimedia archives.
- Intelligent lighting management: controls the lighting according to both user presence and light intensity.
- Security management: when activated, if it detects presence inside or detects that a door or window has been opened, the system goes off the alarm, starts to record and sends a warning to users.
- Heating and Air conditioning management: keeps the temperature optimum in the room where the user is, keeps a temperature close to optimum in the locations where the user can go and puts the Heating and Air conditioning in saving energy mode in the rest of the house.
- Blind Management: allows inhabitants to control the blinds of the home.

5.8 Design Considerations for Task Recommendations

5.8.1 User Interface Considerations

Unobtrusive The user interface of task recommendations should stay out of the user's way. We avoid a system that pops-up or forces the user to respond to the recommendation before continuing to work, since this type of system could be frustrating [124].

Self Paced The user should be able to act on the recommendations when it is convenient for them.

5.8.2 Recommender System Considerations

Novel Recommendations The recommendations should be tasks that the user is unfamiliar with or unknown to.

Relevant Recommendations The recommended tasks also need to be relevant to the user's situation. This could mean that the task is relevant immediately, or relevant at some point in the future given the type of work the user does.

Global and Opportunistic Suggestions The system should be able to provide global suggestions. However, the system could also have some knowledge about what the user is doing at the current moment so it is able to highlight suggestions which may be particularly relevant in the current context.

Support Different User Communities The recommender system should be able to base its recommendations on different collections of users. Users may want to see recommendations generated from known expert users, a group of co-workers, or the entire user community of the pervasive interactive system.

The combination of novel and relevant recommendations leads to a two-dimensional space. We consider a good recommendation to be a task that is both relevant and novel to the user. A poor recommendation is a task that is not relevant to the user. An unnecessary recommendation is a task which is relevant to the user, but the user was already familiar with. Unnecessary recommendations can actually be helpful in improving the user's confidence in the system, but this is very dependent on the user's expectations. If the expectation is that the system will be suggesting "new" tasks which may be relevant, tasks with which the user is already familiar may be seen as poor suggestions.

5.9 Task Recommendation System

We now describe TaskRec, a new system that provides personalised task recommendations using collaborative filtering algorithms. The general idea is to first compare a user's task frequencies to the entire user population. Our system then generates a top 10 list (although the list size could vary) of recommendations for that user. This top 10 list is presented within the user interface on the master device that the user can refer to when convenient.

5.9.1 Target Application

TaskRec is implemented within a smart house, a widely used pervasive interactive environment. A smart house would be an excellent environment to work with since it not only has hundreds of tasks, but also numerous domains of usages. While our work is implemented within a smart house, the concepts map to any pervasive interactive systems where feature awareness may be an issue, and its usage varies across users.

5.9.2 Task Repository/Task Database

TaskRec requires usage data for its users to provide personalised tasks. In a smart house, how to collect task accomplishment histories? A tuple can be as {User, Task, Situation}...

5.9.3 The "Ratings"

Typical recommender systems depend on a rating system for the items which it recommends. For example, a recommender system for movies may base its recommendations on the number of stars that user's have assigned to various titles. These ratings can be used to find similar users, identify similar items, and ultimately, make recommendations based on what it predicts would be highly rated by a user. Unfortunately, in our domain, no such explicit rating system exists. Instead, we implicitly base a user's "rating" for any task on the frequency for which that task is executed. Our collaborative filtering algorithm then predicts how the user would "rate" the tasks which they do not execute. In other words, we take a user's observed task-frequency table as input, and produce an expected task-frequency table as output.

We explored two of the most commonly used collaborative filtering techniques: user-based [125] and item-based [126]. Both of the algorithms discussed have two inputs: the task history for each user in the community, and the task history for the user we are generating a recommendation, which we refer to as the active user.

5.9.4 User-Based Collaborative Filtering

User-based collaborative filtering generates recommendations for an active user based on the group of individuals from the community that he/she is most similar to. The algorithm averages this group's task frequencies, to generate an expected task-frequency table for the active user. The algorithm details are described below.

Defining Task Vectors

For user-based collaborative filtering, we require a method to measure the similarity between two users. A common approach for doing this is to first define a representative vector for each user, and then compare the vectors. A basic method is to define the task vector V_j such that each element, $V_j(i)$, contains the frequency for which the user u_j has executed the task t_i. A limitation of using this approach is that in general, a small number of tasks will be frequently executed by almost everyone [111]. Thus, when comparing the vectors, each pair of users will tend to have high similarity because they will all share these popular high frequency tasks.

We need to suppress the overriding influence of tasks that are being executed frequently and by many users. Document retrieval algorithms actually face a similar challenge. For example, an Internet search engine should not consider two webpages similar because they both share high frequencies of the words "a", "to", and "the". Such systems use a "term frequency inverse document frequency" (*tf-idf*) technique to determine how important a word is to a particular document in a collection. For our purposes, we adapt this technique into a task frequency, inverse user frequency (*tf-iuf*) weighting function, by considering how important a task is to a particular user within a community. To do so, we first take the task frequency (*tf*) to give a measure of the importance of the task t_i to the particular user u_j.

$$\mathit{tf}_{ij} = \frac{n_{ij}}{\sum_k n_{kj}},$$

where n_{ij} is the number of occurrences of the considered task of user u_j, and the denominator is the number of occurrences of all tasks of user u_j.

The inverse user frequency (iuf), a measure of the general importance of the task, is based on the percentage of total users who execute it:

$$\mathit{iuf}_i = \log \frac{|S|}{|\{u_j : t_i \in u_j\}|},$$

where $|S|$ is the total number of users in the community; $|\{u_j : t_i \in u_j\}|$ is the number of users who execute t_i.

With those two metrics, we can compute the *tf-iuf* as

$$\mathit{tf\text{-}iuf}_{ij} = \alpha \cdot \mathit{tf}_{ij} \cdot \mathit{iuf}_i,$$

where α is a tuning parameter.

A high weight in *tfiuf* is obtained when a task is executed frequently by a particular user, but is executed by a relatively small portion of the overall population. For each user u_j, we populate the task vector V_j such that each element, $V_j(i)$, contains the *tf-iuf* value for each task t_i, and use these vectors to compute user similarity.

Finding Similar Users

As with many traditional recommender systems, we measure the similarity between users by calculating the cosine of the angle between the users' vectors. In our case, we use the task vectors, as described above. Considering two users u_A and u_B with task vectors V_A and V_B,

$$\mathit{similarity}(u_A, u_B) = \cos(\theta_{V_A,V_B}) = \frac{V_A \cdot V_B}{||V_A|| * ||V_B||}.$$

Thus, when similarity is near 0, the vectors V_A and V_B are substantially orthogonal (and the users are determined to be not very similar) and when similarity is close to 1 they are nearly collinear (and the users are then determined to be quite similar). As can be seen in Figure **??**, using the cosine works nicely with our rating system based on frequencies, since it does not take into account the total number of times a user has executed a task, but only its frequency.

We compare the active user to all other user in the community, to find the n most similar users, where n is another tuning parameter.

Calculating Expected Frequencies

To calculate an expected frequency for each task, we take a weighted average of the task frequencies for the active user's n similar users. We define the expected frequency, ef_{ij} , for task t_i and user u_j:

$$\mathit{ef}_{ij} = \sum_{k=1}^{n} w_{jk} \mathit{tf}_{ik},$$

where w_{jk} is any weighting function (which can be tuned) and tf_{ik} is the frequency of task t_i and user u_k.

Removing Previously Used Tasks

Once we create a list of all the task frequencies, we remove any tasks which the user has been observed to execute, preventing no known tasks from being suggested.

Returning Top 10 List

The final step is to sort the remaining tasks by their expected frequencies. The highest 10 tasks will appear in the user's recommendation list.

5.9.5 Item-Based Collaborative Filtering

Rather than matching users based on their task accomplishment, our item-based collaborative filtering algorithm matches the active user's tasks to similar tasks. The steps of the algorithms are described below.

Defining User Vectors

We first define a vector V_i for each task t_i in the n dimensional user-space. Similar to user-based approach, each element, $V_i(j)$, contains the *tf-iuf* value for each user u_j.

Building a Task-to-Task Similarity Matrix

Next, we generate a task-to-task similarity matrix, M, where each element m_{ik} is defined for each pair of tasks i and k as:

$$m_{ik} = \cos(\theta_{V_i,V_k}).$$

Creating an "Active List"

For the active user, u_j, we create an "active list" L_j, which contains all of the tasks that the active user has used:

$$L_j = \{t_i | tf_{ij} > 0\}.$$

Find Similar Unexecuted Tasks

Next, we define a similarity score, s_i, for each task t_i which is not in the active user's active list:

$$s_i = average(m_{ik}, \forall t_k \in L_j).$$

Generating Top 10 List

The last step is to sort the unused tasks by their similarity scores s_i, and to provide the top ten tasks in the user's recommendation list.

5.9.6 Domain-Specific Rules

The above techniques work without any specific knowledge about the application domain. This could lead to some poor recommendations which should be avoided. Thus, we created two types of rules to inject some basic domain knowledge into the system.

Upgrades ($A \nRightarrow B$)

An upgrade is a situation where if you execute task A, there is no need for you to execute task B. For example, if a user executes "Make tea", we would not recommend the "Make coffee" task, since it is a less efficient mechanism to activate the similar task.

Equivalencies ($A \nLeftrightarrow B$)

We consider two tasks to be "equivalent" when it makes sense for a user to execute one of the two tasks, but not both. For example, ...

5.10 Simulation and Evaluation

Evaluating context-aware recommender systems is difficult in principle, because every recommendation is only valid in a particular context. In addition, standard recommender evaluation methods do not account for context information. Therefore, the methods like mean average errors of precision and recall metrics [127] cannot easily be applied. Therefore, we decided to implement a simulation environment in order to test the approach with real world data.

So far, we haven't had the resources to conduct real-world usage studies to confirm our simulation results outside the laboratory. We hope to conduct such studies in the future as a means to increase our simulation's accuracy and to obtain feedback on how we could adjust the system to work better for real-world users. We are currently constructing a realistic test-bed and developing a task execution engine for smart phones in cooperation with commercial vendors. Our goal is to begin helping mobile phone users find relevant tasks according to their preferences and contexts in the near future.

5.10.1 Off-line Algorithm Evaluation

Here, we present an automated method to evaluate our recommendation algorithms using our existing off-line data. Although off-line evaluation cannot replace online evaluation, it is a necessary and important step to tune the algorithms and verify our design decisions before the recommender system is deployed to real users.

The development of the algorithm was a challenging task since we required a metric that would indicate if a recommended task, which had never been observed, would be relevant/useful to a user. To do so, we developed a new k-tail evaluation where we use the first part of a user's task history as a training set, and the rest of the history (the most recently executed tasks) as a testing set.

Consider a user u_j with a series of tasks S. k-tail evaluation divides this task sequence into a training sequence S_{train} and a testing sequence S_{test}, so that there are k unique tasks in S_{train} which are not in S_{test}. For example, the task sequence in Figure **??** is a 2-tail series since there are two tasks, ABC and DEF, which have never appeared in the training set.

To evaluate an algorithm, we find the average number of tasks which are in both the user u_j's recommendation list R_j, and his/her testing set $S_{test,j}$. We define the evaluation result of k-tail as hit_k:

$$hit_k = \frac{\sum_{j=1}^{n} |R_j \cap S_{test,j}|}{n},$$

where n is the size of user community.

Two algorithms were evaluated using the k-tail method. The task sequence of each user is divided into a training set and a k-tail. Figure **??** shows that when $k = 1$, the item-based algorithm predicts the next new task correctly for x users.

5.10.2 On-line Algorithm Evaluation

While our off-line evaluation showed promise for our new techniques, the results may not be fully indicative of how our algorithms would work in practice. As such, we conducted an online "live" study with real users. We collected data for a set of real users, generated personalised recommendations for each of them, and had them evaluate their recommendations in a web-based survey.

Participants

We recruited n users (x female, y male) of our Staff Common Room (SCR) at our department to participate in the study. To be considered for the study, users were required to use the SCR a minimum of 20 hours per week. Participants were aged m to l and worked in varying fields including maths, computer sciences, physics, and education, coming from varying countries.

5.11 Implementation

It is difficult to capture raw context data or sensor data due to the constraints of tools and time. The researches for inference of high-level context such as users' current activity from raw context or sensor data also have been carried out by many researchers. So, we assume that the high-level situations is already inferred in this research.

Roughly speaking, the task recommendation problem (TRP) is an optimisation problem its solution is the most relevant task offered to the user (or a group of users) located in a given environment. The most relevant task maximally meets/satisfy the user's intention while it should trade-off between its feasibility, autonomy, and relevance with the user intention.

Notations

We define that the environment is determined by a sequences of its context which is captured over the time of the existence of the environment. We denote the related notations as shown in Table 5.1.

Following, we formally define the concepts and functions which used in our approach of solving the problem of task recommendation.

E	A given pervasive computing environment. The current state of E is determined by its current context
c	The current context of the environment, $c \in E$
$\mathcal{T}$	The task space which contains all tasks in the environment
t	A candidate task, $t \in \mathcal{T}$
$\mathcal{I}$	The user's intentions
i	The user's current intention, $i \in \mathcal{I}$

Table 5.1: Notations in the task recommendation problem

Feasibility function

The purpose of this function is to find out what is missing in order to accomplish the task. If the task is selected to be performed, the system will suggest him what needs to be available for the task to be successfully performed.

Definition 1 (*Feasibility function*)**.** *The feasibility function, f_F, computes the feasibility degree of a task, $t \in \mathcal{T}$, in a given context $c \in \mathcal{C}$,*

$$f_F : \mathcal{T} \times E \longrightarrow [0, 1].$$

For example, $f_F(t, c) = 80\%$, that is, the current environment, c, meets 80% of the requirements to achieve the task, t. If the user wants to perform the task, he needs to rearrange or setup the environment to fulfill the 20% left.

Challenges

- We need to build the task space. The requirements (pre-conditions) of each task in the task space needs to be specified.
- The capabilities of the environment (such as available resources) needs to be captured and modelled. The capabilities are derived from the current context information which is sensed and provided by a context management system.
- A mechanism for measuring the feasibility degree between the task requirements and the environment capabilities.

Autonomy function

The purpose of this function is to determine the automatic parts and the manual parts in the performance of a task in a given environment. This result will be used for ranking tasks in order of priority.

Definition 2 (*Autonomy function*)**.** *The autonomy function (f_A) measures the autonomy degree of the task performance provided by the environment (E) at the current context c,*

$$f_A : \mathcal{T} \times E \longrightarrow [0, 1].$$

For example, $f_A(t, c) = 90\%$, that is, 90% of the performance of the task, t, is automatically carried out by the capabilities of the current environment, c. Hence, the user needs to manually complete the 10% left.

Challenges

- Each task in the task space consists of set of actions. The requirements of these actions needs to be specified. The requirements should describe which actions need to be carried out by only automatic computation, which actions are only carried out by human, which actions need computer-human interaction, and which actions cab be done by either.
- There is a need for a method to measure the autonomy degree of the task performance based on the task specification and the capabilities of a given environment.

Relevance function

The purpose of this function is to measure how much a task can fulfill the user's needs. Its result is then employed for prioritising tasks.

Definition 3 (*Relevance function*)**.** *The relevance function (f_R) measures the relevance degree of a task (t) with the user's current intention (i),*

$$f_R : \mathcal{T} \times \mathcal{I} \longrightarrow [0, 1].$$

Challenges

- A task has its objectives and its influences on the environment. These should be primarily considered in measurement of the relevance of a task with the user's current intention. Therefore, task's objectives and its influences on the environment need to be specified.
- Task objectives are matched with user's intention to calculate the relevance degree mentioned above. Capturing, modelling, and inferring user's intention are great challenges remaining.
- Given user's intention and task objectives, we need a component for figuring out the most relevant task with which the user's intention best match.

The priority of a task

To give a method for solving the task recommendation problem, we make the following assumptions:

1. Users prefer the most relevant task to other tasks in any case;

2. If there are many tasks which have the same relevance degrees, the more autonomic the task performance is, the more preferred the task is;

3. For the tasks, the autonomy degrees of their performances are the same, then the task which is more feasible is more prioritised.

Formally, we define the priority of a task is as follow.

Definition 4. *Let t and t' be two tasks in the task space, $\mathcal{T}$, and let $f_P(t)$ and $f_P(t')$ be the priorities of these tasks. We say that*

$$f_P(t) \geq f_P(t') \quad \Leftrightarrow \quad \left[\begin{array}{ll} f_R(t) \geq f_R(t') & \textit{or} \\ f_R(t) = f_R(t') \wedge f_A(t) \geq f_A(t') & \textit{or} \\ f_R(t) = f_R(t') \wedge f_A(t) = f_A(t') \wedge f_F(t) \geq f_F(t'). & \end{array} \right.$$

In the above formula, the relation '$\geq$' stands for "equivalent to or greater than".

Definition of task recommendation problem

We are now going to formally define the task recommendation problem.

Definition 5 (*Task recommendation problem – TRP*)**.** *Given a triplet $\langle c, i, \mathcal{T} \rangle$, where $c \in E, i \in \mathcal{I}$, find a task t (so called* the optimal solution or the most relevant task*), $t \in \mathcal{T}$, such that*

$$f_P(t) = \max \Big\{ f_P(t') \mid t' \in \mathcal{T} \Big\}.$$

5.11.1 System architectural decomposition

The main purpose of the task recommendation system described here is to solve the task recommendation problem. That is, the system has to figure out the most relevant task for the user. This task, if the user accept the task to be performed, will be an input of a *Task Execution Management System.* In this work, we adopt the previous solutions of such the Task Execution Management System.

Architecture of Task Recommendation System

As indicated in Figure 5.3, there are three sub-systems making up the task recommendation system. The general purpose of these sub-systems is to provide information (e.g. context information, user's intention, task specification) for further processes. There are four processes which are responsible for finding the most relevant task based on the information provided by the three sub-systems.

Context Management System

- Manage context information of the environment and infer the current capabilities of the environment;

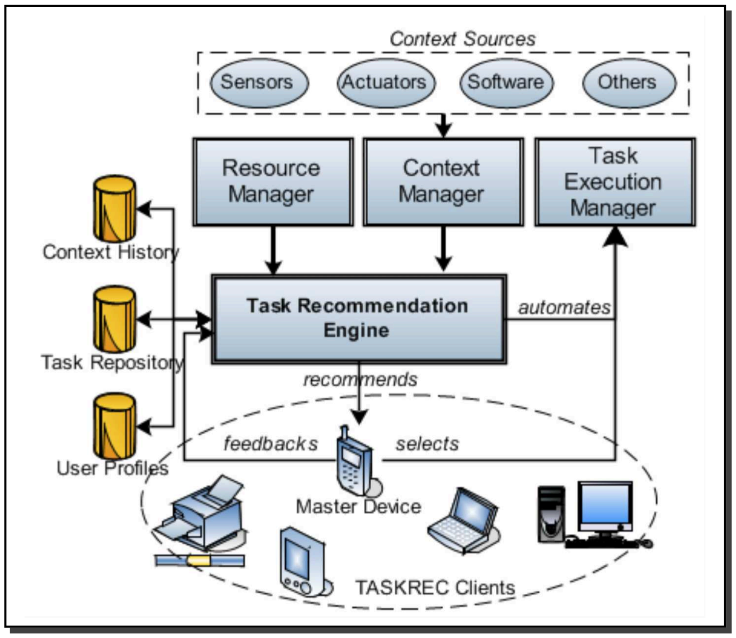


Figure 5.3: Architecture of Task Recommendation System

- Provide information of environment capabilities to other processes when needed;
- Throw an event called "*Context_Change*" every time the context has been changed. Other processes can listen to the Context_Change event to re-execute their functions which are related to the current context.

Task Management System

- Manage specifications of tasks;
- Provide task specifications to other processes when needed;
- Throw an event called "*Task_Change*" every time a task specification has been changed. Other processes can listen to the Task_Change event to re-execute their functions which are using this task specification.

User Intention Inference System

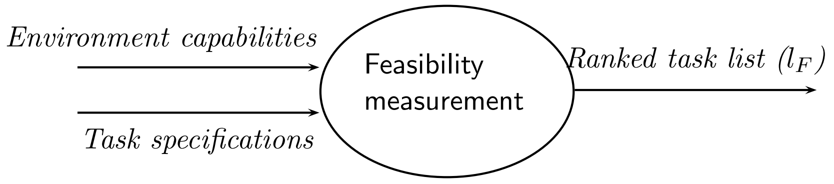
- Manage users' intentions;
- Provide users' intentions to other processes when needed;
- Throw an event called "*Intention_Change*" every time the user's intention has been changed. Other processes can listen to the Intention_Change event to re-execute their functions related to this user's intention.

Feasibility measurement process

Function

- Compute the feasibility degree of each task in the task space;
- Ranking tasks based on their feasibility;
- Provide the ranked list of tasks to the task prioritisation process.

Operation



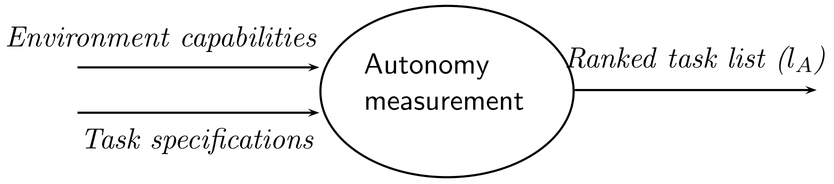
The process takes the current capabilities of the environment and the task specifications as its two inputs. The output is the ranked list of tasks.

Autonomy measurement

Function

- Compute the autonomy degree of each task in the task space;
- Ranking tasks based on their autonomy degree;
- Provide the ranked list of tasks to the task prioritisation process.

Operation



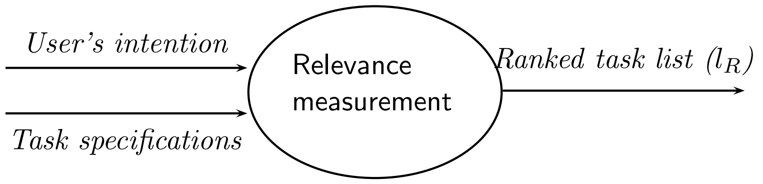
The process takes the current capabilities of the environment and the task specifications as its two inputs. The output is the ranked list of tasks.

Relevance measurement

Function

- Compute the relevance degree of each task in the task space;
- Ranking tasks based on their relevance degree;
- Provide the ranked list of tasks to the task prioritisation process.

Operation



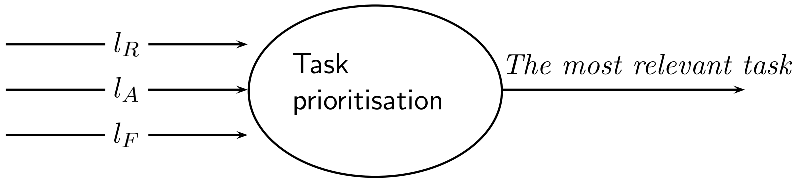
The process takes the user's intention and the task specifications as its two inputs. The output is the ranked list of tasks.

Task prioritisation

Function

- Compute the priority of each task in the task space;
- Ranking tasks based on their priority;
- Provide the most prioritised task (e.g. the most relevant task) to the user.

Operation



The process accepts three inputs including l_R, l_A, and l_F which are desired from the three previous processes. The output is the most relevant task.

Context-aware task generation:

In order to model our proposed architecture, firstly, we formalise two key elements in our architecture which are context and task.

5.11.2 Definition of context

Definition 6 (*Context*)**.** *Let $\mathcal{C}_{curr}$ be current context, $\mathcal{C}_{curr}$ is a set of attribute-value pairs. Each attribute-value pair expresses a certain aspect of the context at the current time or in the relevant past (i.e., context history).*

For example, in our scenario, `Context #4` would be expressed as follows:

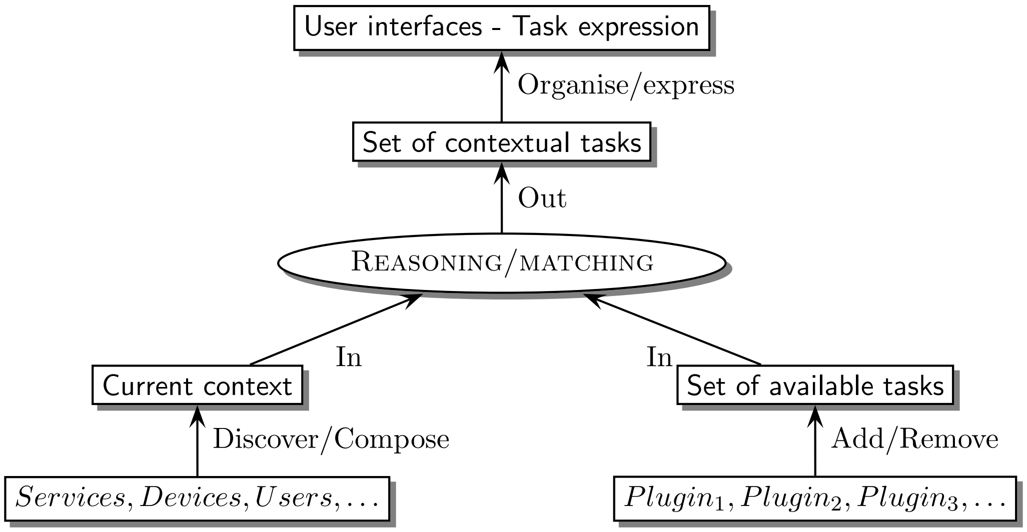


Figure 5.4: Diagram of the algorithm for context-aware task selection

$$\begin{aligned}
\mathcal{C}_{curr} = \{\ & \texttt{Location} && = \text{“}\texttt{Home}\text{”},\\
& \texttt{Day of week} && = \text{“}\texttt{Monday}\text{”},\\
& \texttt{Time} && = \text{“}\texttt{Early morning}\text{”},\\
& \texttt{User} && = \text{“}\texttt{Husband}\text{”},\\
& \texttt{Next activity} && = \text{“}\texttt{Appointment with friend}\text{”},\\
& \texttt{Weather} && = \text{“}\texttt{Bad}\text{”},\\
& \texttt{Breakfast} && = \text{“}\texttt{Finished}\text{”},\\
& \ldots\}
\end{aligned}$$

5.11.3 Definition of tasks

Tasks are specified in *plugins*. We mainly examine a task in the combination of its important parts including *required contexts*, *impacts* on the context, and *execution plan* of the task. There are other properties of a task such as name, objective, identification, and type of a task. We are going to describe them in the following subsections.

Required contexts of a task

A required context (denoted by c_{req}) of a task is seen as the condition of the task exposed to the user. It addresses two problems. Firstly, it ensures that the task being offered to the user is the appropriate task to the current context and the user intention. Secondly, it guarantees that the task can be successfully accomplished.

A required context of a task is a set of attribute-value pairs. For example, one of the required contexts of the task “`turn the light off`” can be expressed as

$$c_{req} = \{\texttt{user} = \text{“}\texttt{empty}\text{”}, \texttt{light} = \text{“}\texttt{on}\text{”}\}.$$

Naturally, one task may have many required contexts which satisfy it. For example, there may be another required context of the task "`turn the light off`", it can be

$$c_{req} = \{\texttt{user} = \text{``}\texttt{someone}\text{''}, \texttt{light} = \text{``}\texttt{on}\text{''}, \texttt{user activity} = \text{``}\texttt{sleeping}\text{''}\}.$$

Impact on context of a task

The execution of a task would affect on the context in which the task is carried out. Hence, the impact on context of a task (denoted by c_{imp}) described in the plan can help the system to predict the context after the task is accomplished. The context prediction can be used to offer further appropriate tasks to users. Similarly, the impact on context of a task is expressed by attribute-value pairs. For example, the impact on context of a task "turn light off" is ⟨light = "off"⟩.

Statuses of a task

In order to consistently manage the execution of a task, we need to formalise statuses of the task. As shown in Figure 5.5, at a particular point of time, a task should be in one of the following statuses: Ready, Pending, In progress, Successful, Aborted, and Failed.

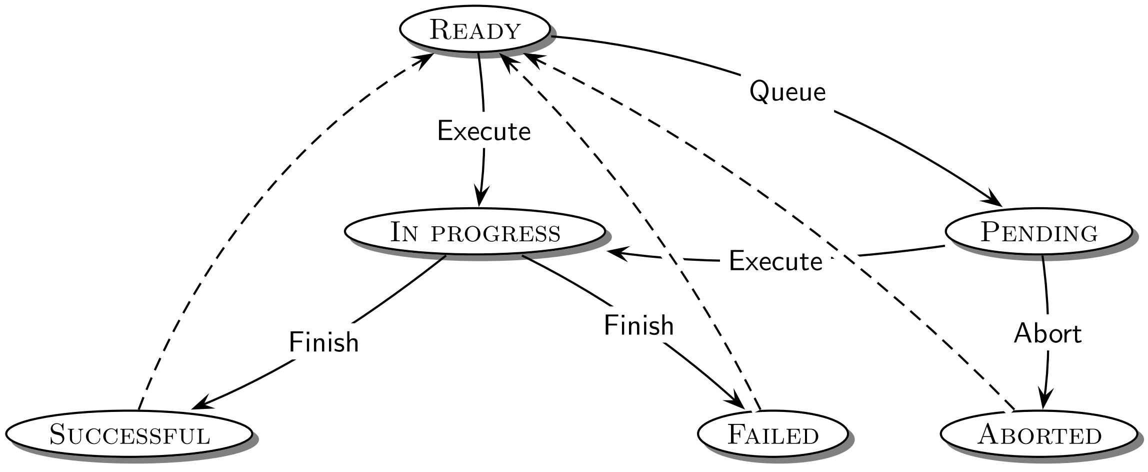


Figure 5.5: The status circle of a task

The status circle of a task starts from Ready status when the task is ready to be carried out. In the during of its execution, the task is In progress status. That is, other invocations of this task may be pended (i.e., Pending status) until the task get into Ready status again. If a task is aborted from the queue of pending tasks, its status will get into Aborted. A aborted task will be ready for re-execution. After the execution, the task will be in one of the following statuses. It is Successful if the execution is successfully completed. Otherwise, the task status is Failed. In both these statuses, the service then is changed to Ready status for re-execution.

Execution plan of a task

For a task to be accomplished, the system must know how to carried out it. A execution plan of a task described in, e.g. a script file or a program, is for this purpose. For example. a execution plan can specify what the sub-tasks to be executed and their execution orders are.

One task may be completed by many ways to obtain its goal. Therefore, a task has a set of execution plans, called $\mathcal{P}$, one of which can be chosen to be executed by the system when the task is selected to accomplished. If a plan is unsuccessfully executed, the system still has a chance to choose another plan to execute. It is also important that the system should have a mechanism to identify the best execution plan in the set of plans to do. Obviously, a task must have at least one execution plan, i.e. $\mathcal{P} \neq \emptyset$. We will specify execution plans in details in sections of the next chapter. Following, we give the definition of a task in our viewpoint.

Definition 7 (*Task*). *A task, denoted t, is a tuple of four $\langle \mathcal{C}_{req}, c_{imp}, \mathcal{P}, st_{curr} \rangle$, where*

- *$\mathcal{C}_{req}$ is the set of required context of t,*
- *c_{imp} is the impact context of t,*
- *$\mathcal{P}, \mathcal{P} \neq \emptyset$ is the set of execution plans of t, and*
- *st_{curr} is the current status of t.*

At a specific context and conditions of a pervasive computing environment, it is a question that what tasks can be done manually or automatically? What is a factor of context and conditions: user's location, time, who, available services, available resources, user's preferences,...

Let C is all factors of possible context and conditions, C^* is the set of all subsets of C, and $C_{curr} \in C^*$ is the current factors. Additionally, let T is a universe of the possible (well-defined) tasks, T^* is the set of all subsets of T, and $T_{curr} \in T^*$ is a set of tasks which can be done in the current context and conditions. We construct a mapping (function) f as follows:

$$\begin{aligned} f : C^* &\longrightarrow T^* \\ C_{curr} &\longmapsto T_{curr} = f(C_{curr}) \end{aligned}$$

Problems: How to model C (ontology, reasoning)? How to get C_{curr} (censoring, perception, service and resource discovery)? how to describe T (workflow, semantic, ontology)? What is function f and how does it works (algorithms)?

5.11.4 Plurality of task execution

There would be many differently possible execution paths which can be used to accomplish the same task. We call E to be a set of such these execution paths and call e^i is the one in the set E. We have

$$E = \{e^1, e^2, ..., e^m\} \quad m \geq 1 \tag{5.1}$$

where

$$e^k = Execution(s_1^k, s_2^k, ..., s_n^k), \quad n \geq 1; \quad 1 \leq k \leq m \tag{5.2}$$

In order to accomplish a task, an execution e^k can be selectively chosen. If the execution fail, there are some strategies to accomplish the task successfully:

- Try again the service which has caused the failure; or
- Go back to the nearest service on the execution path from which the service is successfully executed and there is a linking service to another execution; or
- Choose another execution path from the beginning.

There are some issues arising here relating to

- how E can be formation,
- what e^i to be choose to execute so that it is the most optimal,
- how e^i can be evaluated optimal or worst and what evaluation conditions are, and
- how to deal with failures of task execution.

In the following sections, we will try to carry out the solutions to these issues.

5.11.5 Graphical representation of task execution

We are intend to use graph theory to deal with the problems of the plurality of task execution. So, we employ the graph representation for our approach to represent task execution paths. As shown in the Figure 5.6, which illustrates an example of two possible execution paths for the same task, each black soiled node symbolises for a service while each soiled arrow linking two nodes symbolises for the order of the corresponding services in the chosen execution path. Whilst, the dash arrows are an alternative solution for accomplishing the same task. It can be very useful in the case that the current execution path is unsuccessfully completed.

5.11.6 Task hierarchy

Definition 8 (*Subtask*)**.** *Subtask*

5.11.7 Mapping tasks to services

Underlying service invocations is done via a translator from eco to BPEL programs and then an engine executes these BPEL programs, invoking the services and managing failures [78].

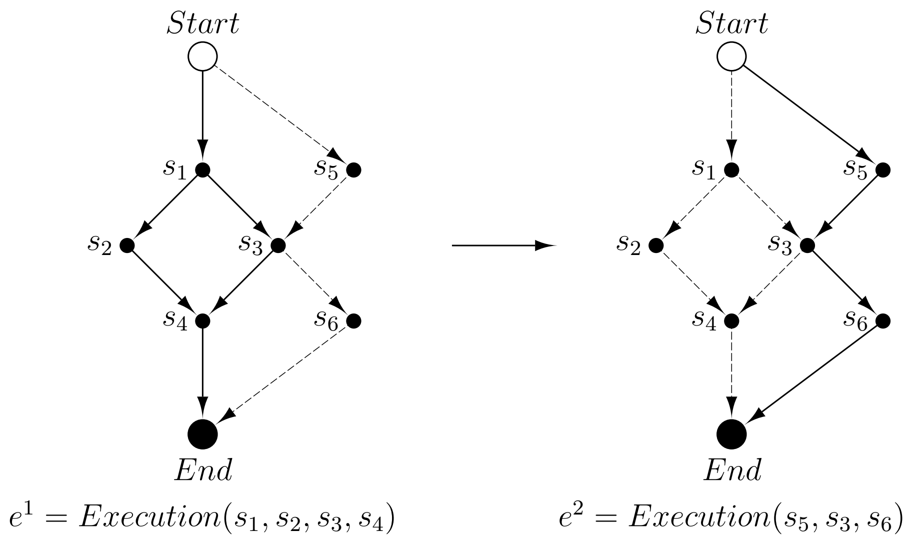


Figure 5.6: Two possible execution paths for the same task

5.11.8 Task Execution

For an atomic task, the procedure field within the task contract will contain one or more discrete action steps. These steps specify the sequential actions to be taken during task execution, along with a number of discrete events that may happen during the execution.

5.11.9 Contextual application or context-aware system

We define a *contextual application* be a dynamic set of contextual tasks. That is, the functionalities of the application can be dynamic depending on the context. The dynamic has two meanings: the dynamic of the order, the amount, and the availability of the functionalities; and the dynamic of the behaviours of the functionalities.

Context-aware systems are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account [128].

The history of context-aware systems started when Want et al. [129] introduced their Active Badge Location System which is considered to be one of the first context-aware applications [128].

Sub-tasks and underlying service of a task

A task can be classified as a *atomic task* or *composite task*.

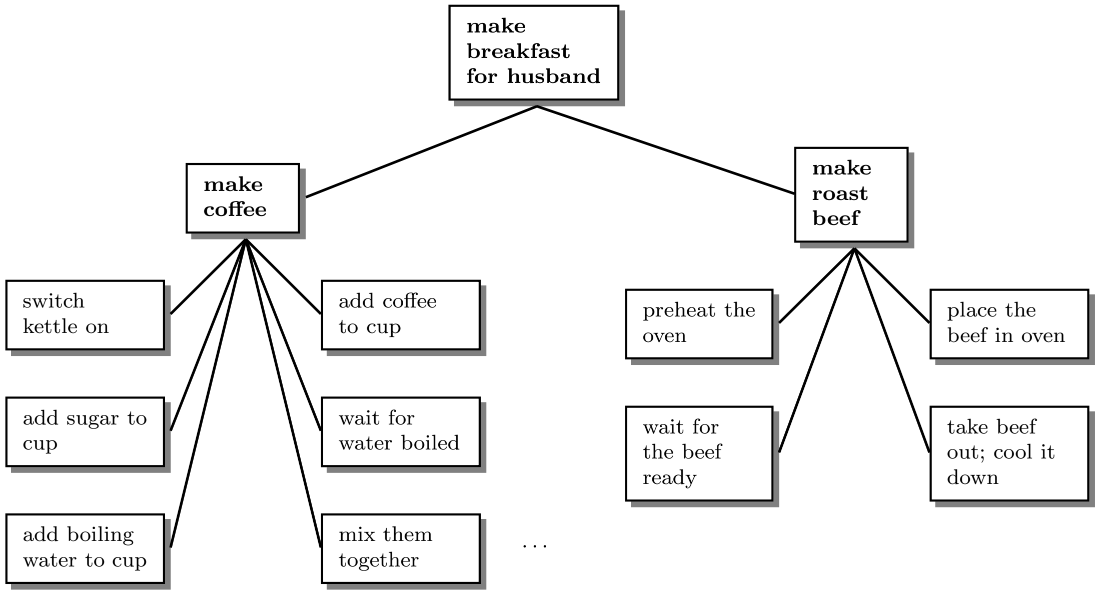


Figure 5.7: An example of task decomposition tree (TDT)

Accomplishment of a composite task needs one or more other tasks (referred as *sub-tasks*) to be accomplished. For example as illustrated in Figure 5.7, the task "`make breakfast for husband`" has its sub-tasks as "`make coffee`" and "`make roast beef`" while the task "`make coffee`" consists of some sub-tasks such as "`switch the kettle on`", "`add coffee to the coffee cup`", "`add some sugar to the cup`", "`wait for the water boiled`", "`add the boiling water to the cup`", and "`mix them together`". In this case, we call the task "`make coffee`" *depends on* its sub-tasks. We denote the set of sub-tasks of a task is $\mathcal{T}_{sub}$.

An atomic task has no sub-tasks. It can be independently completed by directly mapping to one underlying service. For example, the task "`switch the kettle on`" can be seen as an atomic task which invokes a service to turn the kettle on.

Priority of a task

pr denotes the importance and exigency of a task to further facilitate the execution, suspension and re-scheduling of tasks at runtime. For tasks that have the same priority their relative importance will be determined by the priority of their respective parent-tasks [41].

Task Decomposition Tree

A full decomposition of a task into tasks, which will all be carried out to complete the initial task, is called a *Task Decomposition Tree* (TDT). The Figure 5.7 is an example of the graphical representation of a TDT. We have the following concepts:

- The node representing the task "`make breakfast for husband`" is called the *root task*. A TDT has only one root task. Moreover, this task must not appear somewhere else in the tree for avoiding loops.

- The sub-tasks of a task are called *sibling tasks*. It is absolute that sibling tasks must be pairwise distinguished.
- The task "`make coffee`" is called the *parent task* of the task "`switch the kettle on`".
- The sub-tasks of a task are also called its *children tasks*.

The edges connecting between a parent task and its children tasks represent the dependence of the parent task on the children tasks.

Dependence of a task on its sub-tasks

There is an practical situation where a task t' is a sub-task of a task t but t' is optional to be completed. That is, in some cases, a user accepts the result of t without t' to be accomplished. In many cases, a optional sub-task is used to make the overall result of its root task smoother or better. Therefore, we have two kinds of dependence between a task and its sub-tasks. They are *optional dependence* and *compulsory dependence*.

In this section, a proposed architecture of contextual service composition as depicted in Figure **??** will be described. This architecture will be employed to build the systems which would address the issues mentioned so far. The idea is to divide the architecture into two levels: the task level and the service level. This division will support for modularisation of the system because there are two key series of problems regarding to *service computing* and *task computing* which the architecture needs to consider about.

The task level mainly focuses on designing user interfaces which meaningfully express the functionalities of the contextual services to the users. The task level also address the problems of getting task commands from the users. The service level concerns with the problems of e.g. service discovery, service composition, and context gathering.

Dividing the architecture into two levels because of [41]:

1. Task definition models human preferences and requirements better than service-orientation models adopted in earlier works;
2. Separation of tasks and services would allow for greater flexibility of changing the tasks without changing the services and vice-verse;
3. It hides the complexity of composting embedded services in pervasive environment from the users.

The system operates in two phases. The first phase is to create contextual tasks. The second phase is to execute contextual tasks. The execution of a task is dependent on the current context while the current context determines the tasks available to be performed.

5.11.10 Automatic task creation

The purpose of this phase is to produce a list of contextual tasks which can be chosen to accomplish by a user. As defined so far, a contextual task is a semantic expression of a contextual service. In other words, there is a bijection from the set of contextual services to the set of contextual tasks. The algorithm of this phase is shown in Figure 5.8.

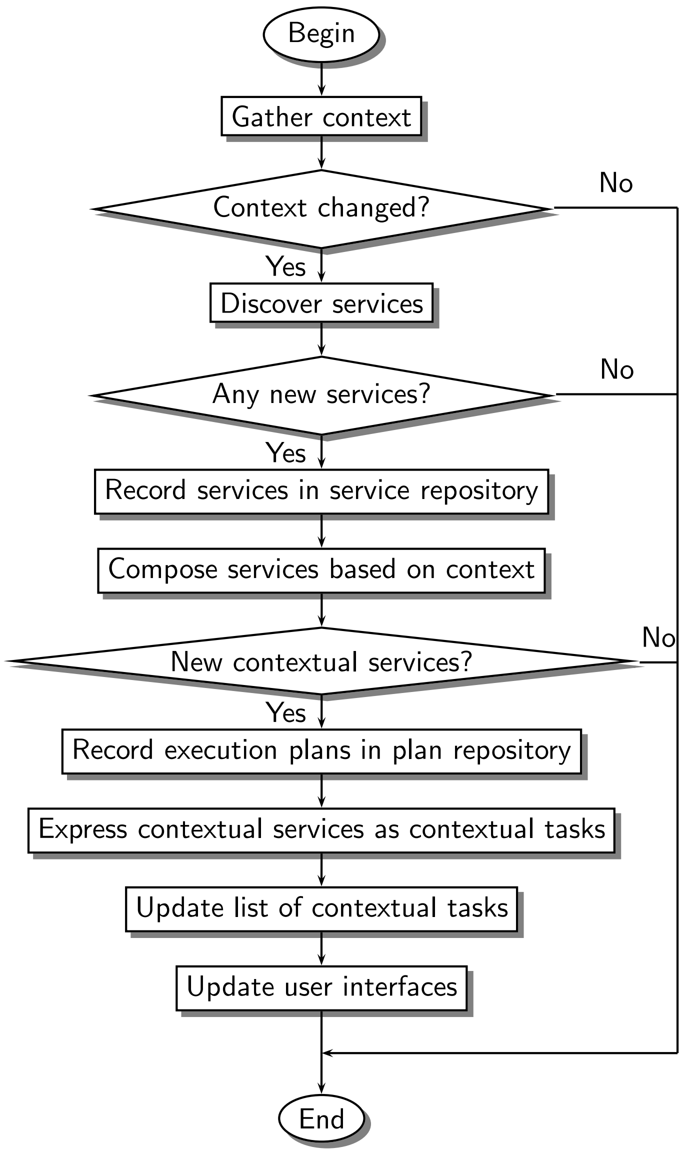

Begin
Gather context
Context changed?
No
Yes
Discover services
Any new services?
No
Yes
Record services in service repository
Compose services based on context
New contextual services?
No
Yes
Record execution plans in plan repository
Express contextual services as contextual tasks
Update list of contextual tasks
Update user interfaces
End

Figure 5.8: The algorithm of the contextual task creation

Every specified interval, the system automatically gathers the current context to detect whether it has been changed. If the context has been changes, then the process of service composition is invoked. Firstly, base services are discovered and the new set of discovered services is compared to the previous set of services to find the new ones. Next, the list of contextual services is produced by the service composition component based on the context and the set of base services. Then, using the description of base services, the ontology of the context, the list of contextual tasks corresponding to the contextual services is updated. Finally, the user interfaces will be updated so that a user can browse the list of tasks or choose to perform a specified task. What happen when a task is selected to accomplish will be described in the second phase.

It is important to consider the structure and the expression of contextual tasks so that the user can quickly find the wanted tasks. The following strategy should be taken into account:

- Organising tasks follows the hierarchal structure. The structure can be based on the ontologies of context such as places, roles of the user, devices, and time. This issue will be investigated in this research.
- The task can be structured as a task list which is ordered based on the priority levels of tasks. The priority level are determined relied on the context, users' preferences, usage histories, habits, and routines.
- There is a method of task structure in which a user can specify a task by providing a command in forms of e.g., natural language, speech, thoughts, body language, eye-lights.
- The system can ask a user a few simple questions so that it can identify the task or reduce the number of tasks enabling the user quickly find out the task the user would like to accomplish.

As described above, we do not care the concepts of *composite task* or *compound tasks*. Actually, subtasks will not appear in this proposed architecture. Any tasks are *atomic tasks* (or *primitive tasks*). However, a contextual service may have its sub-services and/or dependant services. Reason for this is that the system can concentrate on solving the complexity of service execution only. Moreover, it can be proved that all contextual tasks can be carried out without subtasks.

5.11.11 Automatic or user initiated task execution

The algorithm of the second phase is shown in Figure 5.9. Firstly, from the user interface, a user inputs a task command. How the user specify a task would depend how the task list is structured. Then, if the system identifies the task, it will ask the user for the parameters required by the task. Otherwise, the user needs to modify the task command so that that system can recognise it. Next, the system will validate the parameters required for executing the contextual service associated with the

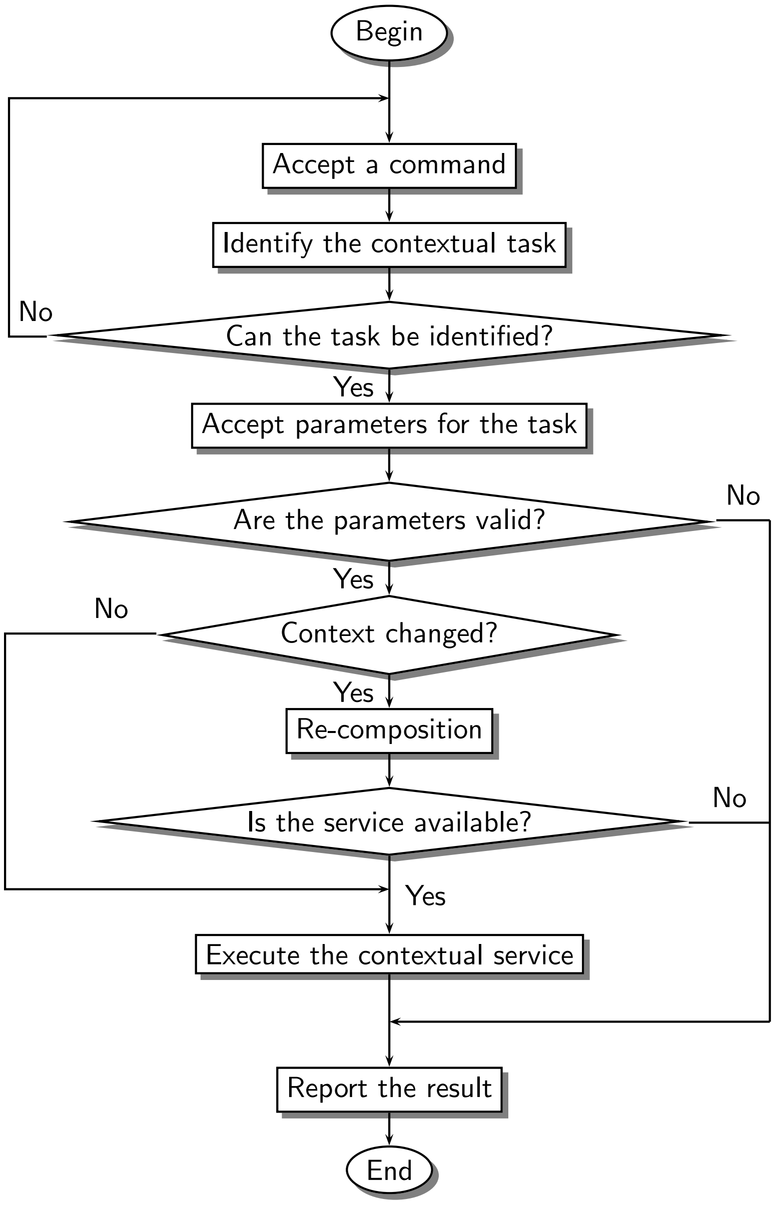

Begin
Accept a command
Identify the contextual task
Can the task be identified?
No
Yes
Accept parameters for the task
Are the parameters valid?
No
Yes
Context changed?
No
Yes
Re-composition
Is the service available?
No
Yes
Execute the contextual service
Report the result
End

Figure 5.9: The algorithm of the contextual task execution

specified task. If the parameters are valid, the system then double-checks on the context to determine whether the context has been changed or not. This procedure is to ensure that the context is remaining appropriate to the chosen task. In the case that the parameters provided are invalid, an appropriate report will be informed to the user so that he/she can modify the parameters. Coming back to the case of checking on the context for changes, if the context is still unchanged, the system will execute the contextual service. Otherwise, the service must be re-composed. After the re-composition, if the service is still available in the new context, it will be executed. If not, an appropriate report will be informed to the user. Finally, the system will give out the result of the service execution (e.g., successful, failed messages, or something has been done).

In order to concretise these algorithms of the two phases above, the research will formally model and specify the concepts which are includes in the proposed architecture. In the next section, I will give the formal definitions of context, a contextual task, a contextual service, an execution plan of a service.

5.11.12 A proposal of a task-driven operation

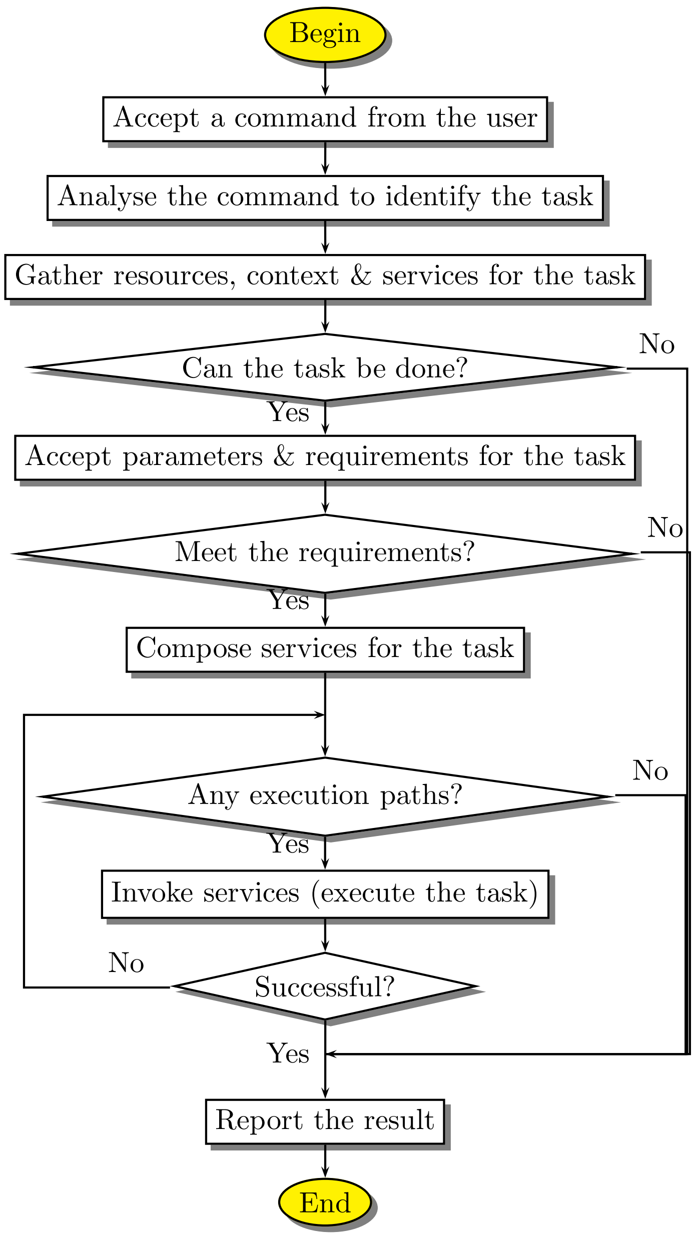

Begin
Accept a command from the user
Analyse the command to identify the task
Gather resources, context & services for the task
Can the task be done?
No
Yes
Accept parameters & requirements for the task
Meet the requirements?
No
Yes
Compose services for the task
Any execution paths?
No
Yes
Invoke services (execute the task)
No
Successful?
Yes
Report the result
End

Figure 5.10: A proposal of the operation of a task-driven system

5.11.13 Command acceptance

1. Typing the command phrase (command lines).

2. Using graphical user interfaces including touching or clicking the command in the list of available commands on the screen.
3. Using speeches.
4. Commands are dynamically automatic created when the current context changes.
5. Thoughts in mind with supports by special devices attached on the body.
6. Eye-beam or eye-wink.
7. Gestures [78]

hierarchical task graphs (task decomposition and levels of service utilization) phrases sequence of subtasks (task statements (concurrently, parallel) and dependency statements (explicit prerequisite))

Thay vi a library of pre-programmed hierarchical task graphs or customise an existing graph for their purposes [78], hierarchical task graphs co the duoc tao ra tu dong dua tren thoi quen su dung cua nguoi dung ma he thong hoc duoc, dua vao ngu canh hien thoi biet duoc de recommand cho nguoi dung nhung tac vu uu tien nhat. Lam the nao de lam duoc dieu nay???

Hoi nguoi dung paramaters hoac cung co the gathering papamaters tu moi truong xung quanh.

discovering the relevant services –¿ composing them –¿ invoking

Chapter 6

Research Plan

In this chapter, I will describe the research plan to complete the proposed framework as my PhD dissertation. First, I will discuss critical technologies that are essential for implementing the task-based framework. Second, I will describe the method to evaluate the feasibility of the framework. Third, I will present the milestones of the future research.

6.1 State of the Work

This section will introduce the state of the work carried out so far. The previous work consists of early and partial designs and implementations of the framework proposed in Chapter 3 (Section 5.5), a case study application, and a context simulator.

6.1.1 Implementation

...

I use Java Micro Edition for implementing TaskRec which can be run on most mobile phones. Java Enterprise Edition can be used to implement the Task Execution Engine, the Context Management System, and the Resource, Device, and Service Management System because it supports well for Service-Oriented Architectures.

6.1.2 A Case Study Application

In order to show that the complexity and inconsistency of using of pervasive systems can be greatly reduced by the use of the proposed task-driven framework, I evaluate the feasibility of using the framework in a prototype system. This prototype systems is developed based on the scenario described in Section 1.5.1.

...

6.2 Future Work

Implementation, Evaluation, Task Execution,...

6.3 Milestones

The proposed research work is expected to be completed in 19 months from 9/2009 to 3/2011. The work will be divided into several stages as follows:

Sep. 2009: Complete research proposal, PerCom 2010 due on 28/9

Oct. 2009: Complete survey on task computing, Pervasive 2010 due on 16/10, PerCom workshop 2010 due on 18/10,

Nov. 2009: Implementation

Dec. 2009: Implementation

Jan. 2010: Implementation, Pervasive workshop due on 1/2/2010

Feb. 2010: Implementation

Mar. 2010: Implementation, Ubicomp due on 30/3/2010

Apr. 2010: Implementation

May 2010: Implementation

Jun. 2010: Implementation, Ubicomp workshop due on 30/6/2010

Jul. 2010: Implementation

Aug. 2010: Conduct a series of experiments based on the prototyped system

Sep. 2010: Analyse data

Oct. 2010: Write a journal paper

Nov. 2010: Write the PhD dissertation

Dec. 2010: Write the PhD dissertation

Jan. 2011: Write the PhD dissertation

Feb. 2011: Write the PhD dissertation

Mar. 2011: Write the PhD dissertation

6.4 Notes

Note:

- Different users under different situations should see different, task based user interfaces of the system.
- In general, the user is neither interested in which devices are involved nor what sequence of actions to execute on them. She just wants to get the job done.
- One could think of services reaching from simple weather forecasts to intelligently control blinds.
- Embedded devices invade us.
- What resources are at hand and what tasks do they (PCEs) support?
- multi-modal interaction: visual, acoustic, and haptic input and output
- To reduce the usage complexity of user interfaces and improve their usability. The increasing complexity due to technological development will be reduced by using a model-based approach for the generation of user interfaces. The core model of a model-based approach focusing on user-centred development is often the task model of a user interface.
- Smart environments bring together multiple users, (interaction) resources and services. This creates complex and unpredictable interactive computing environments that are hard to understand. Users thus have difficulties to build up their mental model of such interactive systems.
- Users immersed in a pervasive environment should become aware of the tasks they can execute using the resources present in the environment.
- Users should be able to manage the tasks they are involved in, i.e. (re)configure them or switch between tasks.

Bibliography

[1] Giulio Mori, Fabio Paterno, and Carmen Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520, 2004. ISSN 0098-5589.

[2] Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. Uniform: automatically generating consistent remote control user interfaces. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 611–620, New York, NY, USA, 2006. ACM. ISBN 1-59593-372-7. doi: http://doi.acm.org/10.1145/1124772.1124865.

[3] Joelle Coutaz. Meta-user interfaces for ambient spaces: Can model-driven-engineering help? In Margaret H. Burnett, Gregor Engels, Brad A. Myers, and Gregg Rothermel, editors, *End-User Software Engineering*, number 07081 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[4] Geert Vanderhulst, Daniel Schreiber, Kris Luyten, Max Muhlhauser, and Karin Coninx. Edit, inspect and connect your surroundings: a reference framework for meta-uis. In *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 167–176, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-600-7.

[5] Gottfried Zimmermann and Gregg C. Vanderheiden. The universal control hub: An open platform for remote user interfaces in the digital home. In *HCI (2)*, pages 1040–1049, 2007.

[6] Mark W. Newman, Ame Elliott, and Trevor F. Smith. Providing an integrated user experience of networked media, devices, and services through end-user composition. In *Pervasive '08: Proceedings of the 6th International Conference on Pervasive Computing*, pages 213–227, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-79575-9.

[7] Charles L. Isbell, Jr., Olufisayo Omojokun, and Jeffrey S. Pierce. From devices to tasks: automatic task prediction for personalized appliance control. *Personal Ubiquitous Comput.*, 8(3-4):146–153, 2004. ISSN 1617-4909.

[8] Henry Lieberman and José Espinosa. A goal-oriented interface to consumer electronics using planning and commonsense reasoning. *Know.-Based Syst.*, 20(6):592–606, 2007. ISSN 0950-7051.

[9] L. Bouarfa, P.P. Jonker, and J. Dankelman. Discovery of high-level tasks in the operating room. *Journal of Biomedical Informatics*, 2010. doi: doi: 10.1016/j.jbi.2010.01.004.

[10] Steve B. Cousins. A task-oriented interface to a digital library. In *CHI '96: Conference companion on Human factors in computing systems*, pages 103–104, New York, NY, USA, 1996. ACM. ISBN 0-89791-832-0.

[11] Geert Vanderhulst, Kris Luyten, and Karin Coninx. Rewire: Creating interactive pervasive systems that cope with changing environments by rewiring. In *Proceedings of the 4th IET International Conference on Intelligent Environments (IE'08)*, pages 1–8, 2008.

[12] O. Rantapuska and M. Lähteenmäki. Homebird–task-based user experience for home networks and smart spaces. In *PERMID 2008*, 2008.

[13] Rajnish Kumar, Vahe Poladian, Ira Greenberg, Alan Messer, and Dejan Milojicic. User-centric appliance aggregation. Technical report, HP Labs, 2002.

[14] J. King, R. Bose, S. Pickles, A. Helal, and H. Yang. Atlas: A service-oriented sensor platform. In *Proceedings of the 1st IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, Tampa, November 2006.

[15] Scott de Deugd, Randy Carroll, Kevin Kelly, Bill Millett, and Jeffrey Ricker. SODA: Service oriented device architecture. *IEEE Pervasive Computing*, 5 (3):94–96, c3, 2006. ISSN 1536-1268.

[16] Chao Chen and Abdelsalam (Sumi) Helal. Device integration in soda using the device description language. In *SAINT '09: Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet*, pages 100–106, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3700-9.

[17] Giulio Mori, Fabio Paternò, and Carmen Santoro. Ctte: support for developing and analyzing task models for interactive system design. *IEEE Trans. Softw. Eng.*, 28(8):797–813, 2002. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2002.1027801.

[18] Tobias Klug and Jussi Kangasharju. Executable task models. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, pages 119–122, New York, NY, USA, 2005. ACM. ISBN 1-59593-220-8. doi: http://doi.acm.org/10.1145/1122935.1122958.

[19] Brithwish Pasu, Wang Ke, and Thomas D.C. Little. A novel approach for execution of distributed tasks on mobile ad hoc networks. Technical report, Boston University, 2001.

[20] J. Van den Bergh and K. Coninx. Contextual concurtasktrees: integrating dynamic contexts in task based design. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 13–17, 2004.

[21] Hongbo Ni, Xingshe Zhou, Zhiwen Yu, and Kejian Miao. Owl-based context-dependent task modeling and deducing. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 846–851, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2847-3.

[22] K. Kalapriya, S. K. Nandy, Deepti Srinivasan, R. Uma Maheshwari, and V. Satish. A framework for resource discovery in pervasive computing for mobile aware task execution. In *Proceedings of the 1st conference on Computing frontiers*, pages 70–77, Ischia, Italy, 2004. ACM.

[23] Davidyuk O, Ceberio J, and Riekki J. An algorithm for task-based application composition. In *Proc. 11th IASTED Int. Conference on Software Engineering and Applications (SEA07)*, pages 465–472, Cambridge, Massachusetts, USA, November 2007.

[24] A Ranganathan. *A Task Execution Framework for Autonomic Ubiquitous Computing*. PhD thesis, University of Illinois at Urbana-Champaign, 2005.

[25] S. Dornbush, K. Fisher, K. McKay, A. Prikhodko, and Z. Segall. Xpod – a human activity and emotion aware mobile music player. In *Mobile Technology, Applications and Systems, 2005 2nd International Conference on*, pages 1–6, 2005.

[26] Zhiwen Yu, Xingshe Zhou, Daqing Zhang, Chung-Yau Chin, Xiaohang Wang, and Ji Men. Supporting context-aware media recommendations for smart phones. *Pervasive Computing, IEEE*, 5(3):68–75, 2006. ISSN 1536-1268.

[27] Daqing Zhang and Zhiwen Yu. Spontaneous and context-aware media recommendation in heterogeneous spaces. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 267–271, 2007.

[28] I. Millard, D. D. Roure, and N. Shadbolt. Contextually aware information delivery in pervasive computing environments. In *The international conference on location and context-awareness*, pages 189–197, 2005.

[29] Zhiwen Yu, Yuichi Nakamura, Daqing Zhang, Shoji Kajita, and Kenji Mase. Content provisioning for ubiquitous learning. *IEEE Pervasive Computing*, 7 (4):62–70, 2008. ISSN 1536-1268.

[30] Takefumi Naganuma and Shoji Kurakake. Task knowledge based retrieval for service relevant to mobile user's activity. In *The Semantic Web – ISWC 2005*, pages 959–973, 2005.

[31] Y. Fukazawa, T. Naganuma, K. Fujii, and S. Kurakake. Service navigation system enhanced by place- and task-based categorization. In *MobiSys 2006*, 2006.

[32] Sonia Ben Mokhtar, Nikolaos Georgantas, and Valrie Issarny. Ad hoc composition of user tasks in pervasive computing environments. *Software Composition*, pages 31–46, 2005.

[33] Munehiko Sasajima, Yoshinobu Kitamura, Takefumi Naganuma, Kunihiro Fujii, Shoji Kurakake, and Riichiro Mizoguchi. Task ontology-based modeling framework for navigation of mobile internet services. In *IMSA'07: IASTED European Conference on Proceedings of the IASTED European Conference*, pages 47–55, Anaheim, CA, USA, 2007. ACTA Press.

[34] M. Anwar Hossain, Pradeep K. Atrey, and Abdulmotaleb El Saddik. Gain-based selection of ambient media services in pervasive environments. *Mob. Netw. Appl.*, 13(6):599–613, 2008. ISSN 1383-469X.

[35] Seng Wai Loke, Shonali Krishnaswamy, and Thin Thin Naing. Service domains for ambient services: concept and experimentation. *Mob. Netw. Appl.*, 10(4): 395–404, 2005. ISSN 1383-469X.

[36] P. Coppola, V. Della Mea, L. Di Gaspero, S. Mizzaro, I. Scagnetto, A. Selva, L. Vassena, and P. Zandegiacomo Rizio. Information filtering and retrieving of context-aware applications within the mobe framework. In *International Workshop on Context-Based Information Retrieval (CIR-05), CEUR Workshop Proceedings*, 2005.

[37] W. Qin, D. Zhang, Y. Shi, and K. Du. Combining user profiles and situation contexts for spontaneous service provision in smart assistive environments. In *UIC '08: Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, pages 187–200, Berlin, Heidelberg, 2008. Springer-Verlag.

[38] Jongyi Hong, Eui-Ho Suh, Junyoung Kim, and SuYeon Kim. Context-aware system for proactive personalized service based on context history. *Expert Syst. Appl.*, 36(4):7448–7457, 2009. ISSN 0957-4174.

[39] Kyung-Lang Park, Uram H. Yoon, and Shin-Dug Kim. Personalized service discovery in ubiquitous computing environments. *IEEE Pervasive Computing*, 8(1):58–65, 2009. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2009.12.

[40] D. Cheng, H. Song, H. Cho, S. Jeong, S. Kalasapur, and A. Messer. Mobile situation-aware task recommendation application. In *The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*, 2008.

[41] H. Ni, X. Zhou, D. Zhang, and N. Heng. Context-dependent task computing in pervasive environment. *Ubiquitous Computing Systems*, pages 119–128, 2006.

[42] Marko Luther, Yusuke Fukazawa, Matthias Wagner, and Shoji Kurakake. Situational reasoning for task-oriented mobile service recommendation. *The Knowledge Engineering Review*, 23(1):7–19, 2008.

[43] Norbert Weissenberg, Rüdiger Gartmann, and Agnés Voisard. An ontology-based approach to personalized situation-aware mobile service supply. *Geoinformatica*, 10(1):55–90, 2006. ISSN 1384-6175.

[44] Chen-Ya Wang, Yueh-Hsun Wu, and Seng-Cho T. Chou. Toward a ubiquitous personalized daily-life activity recommendation service with contextual information: A services science perspective. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 107, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 0-7695-3075-8.

[45] Henry Lieberman, Neil W. Van Dyke, and Adrian S. Vivacqua. Let's browse: a collaborative web browsing agent. In *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces*, pages 65–68, New York, NY, USA, 1999. ACM. ISBN 1-58113-098-8.

[46] Joseph E. McCarthy and Theodore D. Anagnost. Musicfx: an arbiter of group preferences for computer supported collaborative workouts. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, page 348, New York, NY, USA, 2000. ACM. ISBN 1-58113-222-0.

[47] L. Ardissono, A. Goy, G. Petrone, M. Segnan, and P. Torasso. Intrigue: Personalized recommendation of tourist attractions for desktop and handset devices. *Applied Artificial Intelligence: Special Issue on Artificial Intelligence for Cultural Heritage and Digital Libraries*, 17(8-9):687–714, 2003.

[48] Fabiana Lorenzi, Fernando Santos, Paulo R. Ferreira, Jr., and Ana L. Bazzan. Optimising preferences within groups: A case study on travel recommendation. In *SBIA '08: Proceedings of the 19th Brazilian Symposium on Artificial Intelligence*, pages 103–112, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88189-6.

[49] M. Weiser. The computer for the 21^{st} century. *Scientific American*, 3(265): 94–104, 1991.

[50] D. A. Norman. *The Design of Future Things*. Basic Books, 2007. ISBN 0465002277.

[51] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, February 2001.

[52] R. Masuoka, B. Parsia, and Y. Labrou. Task computing – the semantic web meets pervasive computing. *The SemanticWeb - ISWC 2003*, pages 866–881, 2003.

[53] D. Garlan, D.P. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: toward distraction-free pervasive computing. *Pervasive Computing, IEEE*, 1 (2):22–31, Apr-Jun 2002. ISSN 1536-1268. doi: 10.1109/MPRV.2002.1012334.

[54] Donna Griffin and Dirk Pesch. Context in pervasive environments. *Ambient Intelligence with Microsystems*, 18:187–203, 2009.

[55] G. Mark and N.M. Su. Making infrastructure visible for nomadic work. *Pervasive and Mobile Computing*, 2010. doi: doi:10.1016/j.pmcj.2009.12.004.

[56] Peter Tolmie, James Pycock, Tim Diggins, Allan MacLean, and Alain Karsenty. Unremarkable computing. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406, New York, NY, USA, 2002. ACM. ISBN 1-58113-453-3.

[57] G. Zimmermann. Open user interface standards - towards coherent, task-oriented and scalable user interfaces in home environments. In *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, pages 36–39, 2007.

[58] Z. Wang and D. Garlan. Task-driven computing. Technical report, School of Computer Science, Carnegie Mellon University, 2000.

[59] G. Fischer. Articulating the task at hand and making information relevant to it. *Human-Computer Interaction*, 16(2):243–256, 2001.

[60] A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K. H. Yi. InterPlay: A middleware for seamless device integration and task orchestration in a networked home. In *PERCOM'06*, pages 296–307. IEEE Computer Society, 2006. ISBN 0-7695-2518-0.

[61] Charles Rich. Building task-based user interfaces with ANSI/CEA-2018. *Computer*, 42(8):20–27, 2009. ISSN 0018-9162.

[62] Jukka Riekki, Ivan Sanchez, and Mikko Pyykkönen. Universal remote control for the smart world. In *UIC '08: Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, pages 563–577, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-69292-8.

[63] F. Paternò. *Handbook of Software Engineering & Knowledge Engineering*, chapter Task Models in Interactive Software Systems. World Scientific Publishing Co., 2001.

[64] Donald A. Norman. The way i see it simplicity is not the answer. *interactions*, 15(5):45–46, 2008. ISSN 1072-5520.

[65] P. Zeven. Do people need the gizmos were selling? Online, December 2006. URL `http://news.cnet.com/Do%20people%20need%20the%20gizmos%20were%20selling/2010-10` CNET News.

[66] Consumer Electronics Assoc. Task model description (CE Task 1.0), ANSI/CEA-2018, Mar. 2008. URL `http://ce.org/cea-2018`.

[67] H. Pinto. *An activity-centered ubiquitous computing framework for supporting occasional human activities in public places*. PhD thesis, University of Minho, 2008.

[68] Rodrigo de Oliveira and Heloísa Vieira da Rocha. *Human-Computer Interaction*, chapter Multi-Device Design in Contexts of Interchange and Task Migration, pages 75–100. I-Tech Education and Publishing, Vienna, 2008.

[69] Dong San Kim and Wan Chul Yoon. A method for consistent design of user interaction with multifunction devices. In *HCD 09: Proceedings of the 1st International Conference on Human Centered Design*, pages 202–211, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02805-2.

[70] A Monk. Noddy's guide to consistency. *Interfaces*, 45:4–7, 2000.

[71] R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin. Ontology-enabled pervasive computing applications. *IEEE Intelligent Systems*, 18(5):68–72, 2003. ISSN 1541-1672. doi: http://doi.ieeecomputersociety.org/10.1109/MIS.2003.1234773.

[72] Zhexuan Song, Ryusuke Masuoka, Jonathan Agre, and Yannis Labrou. Task computing for ubiquitous multimedia services. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 257–262, College Park, Maryland, 2004. ACM.

[73] Michael H. Coen. Design principles for intelligent environments. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 547–554, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7.

[74] Lawrence D. Bergman, Tatiana Kichkaylo, Guruduth Banavar, and Jeremy B. Sussman. Pervasive application development and the wysiwyg pitfall. In *EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pages 157–172, London, UK, 2001. Springer-Verlag. ISBN 3-540-43044-X.

[75] Patrick Sauter, Gabriel Vögler, Günther Specht, and Thomas Flor. A model–view–controller extension for pervasive multi-client user interfaces. *Personal Ubiquitous Comput.*, 9(2):100–107, 2005.

[76] Anand Ranganathan and Roy H. Campbell. Supporting tasks in a programmable smart home. In *In ICOST 2005: 3rd International Conference On Smart Homes and Health Telematic – From Smart Homes to Smart Care*, pages 3–10, Magog, Canada, 2005.

[77] Heidi Harley. *The Cambridge Encyclopedia of Linguistics*. Cambridge University Press, 2007.

[78] Seng W. Loke. Building taskable spaces over ubiquitous services. *IEEE Pervasive Computing*, 8(4):72–78, 2009. ISSN 1536-1268.

[79] et al. Mizoguchi, R. Task ontology for reuse of problem solving knowledge. In *Proc. of KB & KS '95*, pages 46–59. The Netherlands, 1995.

[80] M. Ikeda, K. Seta, O. Kakusho, and Mizoguchi R. Task ontology: ontology for building conceptual problem solving models. In *Proceedings of ECAI98 Workshop on Applications of ontologies and problem-solving model*, pages 126–133, 1998.

[81] Anand Kumar, Paolo Ciccarese, Barry Smith, and Matteo Piazza. Context-based task ontologies for clinical guidelines. In Domenico M. Pisanelli, editor, *Ontologies in Medicine: Proceedings of the Workshop on Medical Ontologies*, pages 81–94, Rome, October 2003. IOS Press.

[82] Nancy Wiegand and Cassandra Garca. A task-based ontology approach to automate geospatial data retrieval. *Transactions in GIS*, 11(3):355–376, 2007.

[83] M. Sasajima, Y. Kitamura, T. Naganuma, S. Kurakake, and R. Mizoguchi. Task ontology-based framework for modeling users' activities for mobile service navigation. In *Proc. of Posters and Demos of ESWC 2006*, pages 71–72, Budva, Montenegro, June 2006.

[84] M. McCullough. *Digital Ground: Architecture, Pervasive Computing, and Environmental Knowing.* MIT Press, 2004.

[85] G. Zimmermann, G. Vanderheiden, and C. Rich. Universal control hub & task-based user interfaces. Online, 2006. URL `http://myurc.org/publications/2006-Univ-Ctrl-Hub.php`. URC Consortium.

[86] Rebecca E. Grinter, W. Keith Edwards, Mark W. Newman, and Nicolas Ducheneaut. The work to make a home network work. In *ECSCW'05: Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work*, pages 469–488, New York, NY, USA, 2005. Springer-Verlag New York, Inc. ISBN 978-1402040221.

[87] José H. Espinosa and Henry Lieberman. Eventnet: Inferring temporal relations between commonsense events. In *MICAI*, pages 61–69, 2005.

[88] P.. Singh. The public acquisition of commonsense knowledge. In *Proceedings of AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access*, Palo Alto, CA, 2002. AAAI.

[89] Lieve Laporte, Peter Eyckerman, and Bieke Zaman. Designing a mobile task based ui for tourists. In *MobileHCI '09: Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–2, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-281-8.

[90] R. Masuoka, Y. Labrou, and Z. Song. Semantic web and ubiquitous computing - task computing as an example. *AIS SIGSEMIS Bulletin*, 1(3):21–24, October 2004.

[91] Harald Radi and Rene Mayrhofer. Towards alternative user interfaces for capturing and managing tasks with mobile devices. In *MoMM '08: Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 272–275, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-269-6.

[92] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R.H.Campbell, and D. Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *IEEE PerCom 2005*, Kauai Island, Hawaii, 2005.

[93] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua—an extensible extension language. *Softw. Pract. Exper.*, 26(6): 635–652, 1996. ISSN 0038-0644.

[94] Ivan Sanchez, Jukka Riekki, and Mikko Pyykkonen. Touch & compose: Physical user interface for application composition in smart environments. In *International Workshop on Near Field Communication*, pages 61–66, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[95] R. Kistler, S. Knauth, D. Kaslin, and A. Klapproth. Caruso - towards a context-sensitive architecture for unified supervision and control. In *IEEE International Conference on Emerging Technologies & Factory Automation*, pages 1445–1448, 2007.

[96] R. Kistler, S. Knauth, and A. Klapproth. Upnp in integrated home- and building networks. In *IEEE International Workshop on Factory Communication Systems*, Dresden, 2008.

[97] Gerrit Meixner, Marc Seissler, and Marcel Nahler. Udit - a graphical editor for task models. In *Fourth International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2009)*, Florida, USA, Feb. 2009.

[98] Kai Breiner, Daniel Görlich, Oliver Maschino, Gerrit Meixner, and Detlef Zühlke. Run-time adaptation of a universal user interface for ambient intelligent production environments. In Julie A. Jacko, editor, *HCI (4)*, volume 5613 of *Lecture Notes in Computer Science*, pages 663–672. Springer, 2009.

[99] Tim Clerckx, Chris Vandervelpen, and Karin Coninx. Task-based design and runtime support for multimodal user interface distribution. In *Engineering Interactive Systems: EIS 2007 Joint Working Conferences*, pages 89–105, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-92697-9.

[100] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. Comput.-Hum. Interact.*, 1(2): 126–160, 1994. ISSN 1073-0516. doi: http://doi.acm.org/10.1145/180171.180173.

[101] Gottfried Zimmermann and Benjamin Wassermann. Why we need a user interface resource server for intelligent environments. In *Workshops Proceedings of the 5th International Conference on Intelligent Environments*, 2009.

[102] M. Hellenschmidt and T. Kirste. Dynamite - the approach to ubiquitous computing within dynamic ad-hoc device ensembles. *Computer Graphik Topics*, 15(6):24–25, 2003.

[103] Chao Chen and Sumi Helal. Sifting through the jungle of sensor standards. *IEEE Pervasive Computing*, 7(4):84–88, 2008. ISSN 1536-1268.

[104] N. Sabouret. A model of requests about actions for active components in the semantic web. In *Proc. STAIRS 2002*, pages 11–20, 2002.

[105] Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications.* Springer-Verlag, London, UK, 1999. ISBN 1852331550.

[106] G. Butler, S.W. Loke, and S. Ling. Device ecology workflows with semantics: Formalizing automation scripts for ubiquitous computing. Online, 2007. URL `http://homepage.cs.latrobe.edu.au/sloke/papers/eco.pdf`.

[107] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1.

[108] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating "word of mouth". In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1.

[109] D. Owen. *User-Centered System Design, New Perspectives on Human-Computer Interaction*, chapter Answers first, then questions, pages 361–375. Lawrence Erlbaum, USA, 1986.

[110] Gerhard Fischer. User modeling in human–computer interaction. *User Modeling and User-Adapted Interaction*, 11(1-2):65–86, 2001. ISSN 0924-1868.

[111] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005. ISSN 1041-4347.

[112] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, 2005. ISSN 1046-8188.

[113] Ghim-Eng Yap, Ah-Hwee Tan, and Hwee-Hwa Pang. Discovering and exploiting causal dependencies for robust mobile context-aware recommenders. *IEEE Trans. on Knowl. and Data Eng.*, 19(7):977–992, 2007. ISSN 1041-4347.

[114] Massimo Paolucci, Takahiro Kawamura, Terry Payne, and Katia Sycara. Semantic matching of web services capabilities. *The Semantic Web – ISWC 2002*, pages 333–347, 2002.

[115] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM. ISBN 0-89791-592-5.

[116] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th international conference on very large database*, pages 115–121, 1994.

[117] J.R. Jang. Anfis: Adaptive-network-based fuzzy inference system. *IEEE Trans. Systems, Man, and Cybernetics*, 23(3):665–685, 1993.

[118] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. Minimum sum-squared residue coclustering of gene expression data. In *Proceedings of the Fourth SIAM International Conference on Data Mining (SDM)*, pages 114–125, April 2004.

[119] Annie Chen. Context-aware collaborative filtering system: Predicting the user's preference in the ubiquitous computing environment. *Location- and Context-Awareness*, pages 244–253, 2005.

[120] Sunghoon Cho, Moohun Lee, Changbok Jang, and Euiin Choi. Multidimensional filtering approach based on contextual information. In *ICHIT '06: Proceedings of the 2006 International Conference on Hybrid Information Technology*, pages 497–504, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2674-8.

[121] W. Woerndl, C. Schueller, and R. Wojtech. A hybrid recommender system for context-aware recommendations of mobile applications. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 871–878, 2007.

[122] Yuichiro Takeuchi and Masanori Sugimoto. A user-adaptive city guide system with an unobtrusive navigation interface. *Personal Ubiquitous Comput.*, 13(2):119–132, 2009. ISSN 1617-4909. doi: http://dx.doi.org/10.1007/s00779-007-0192-x.

[123] Wolfgang Woerndl, Michele Brocco, and Robert Eigner. Context-aware recommender systems in mobile scenarios. *International Journal of Information Technology and Web Engineering*, 4(1):67–85, 2009.

[124] James Fogarty, Scott E. Hudson, Christopher G. Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C. Lee, and Jie Yang. Predicting human interruptibility with sensors. *ACM Trans. Comput.-Hum. Interact.*, 12(1): 119–146, 2005. ISSN 1073-0516.

[125] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, New York, NY, USA, 1994. ACM. ISBN 0-89791-689-1.

[126] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167, New York, NY, USA, 2000. ACM. ISBN 1-58113-272-7.

[127] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/963770.963772.

[128] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[129] Roy Want, Andy Hopper, Veronica Falco, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.