Towards a Task-Oriented Framework for Smart Spaces

Chuong C. Vo, Torab Torabi, and Seng W. Loke
Department of Computer Science & Computer Engineering, La Trobe University
e-mail: ccvo@students.latrobe.edu.au, t.torabi@latrobe.edu.au, s.loke@latrobe.edu.au

Abstract. In smart spaces such as smart office and workplaces, users are surrounded by hundreds of networked devices and services which often vanish into the background. Therefore, the users are often unaware of possible tasks which they can achieve within a given physical space. Moreover, the user does not want to dig in a long list of individual services and devices for functionalities which are required to accomplish a task at hand. To enable doing more with less in everyday life, we envision a future intelligent computing where the users should not handle directly functionalities provided by individual services and devices but rather high-level tasks, e.g. 'watch a movie' or 'borrow a book'. In this paper, we present our vision of a task computing framework which is to deal with development, deployment, discovery, recommendation, and execution of high-level tasks within smart spaces. We outline a research agenda aiming to realise the proposed framework.

Keywords. Task-Oriented Computing, Smart Spaces, Task Recommendation

1. Introduction

The vision of pervasive computing [1] is emerged from seamlessly integrating technologies into the fabric of everyday life. Smart technologies will be everywhere providing users with functionality and simplifying user's everyday tasks[1] which are beyond the tasks supported by a single computational device. As smart spaces are increasingly embedded with hundreds of services and devices, everyday users often find themselves spending more time and effort in understanding, configuring, and exploiting these spaces. Task-oriented computing [2] appears to be a promising paradigm to address this problem as it shifts focus to what users want to do (i.e., on the tasks at hand) rather than on the specific means for doing those tasks [3]. In other words, users perceive the tasks but not the devices and/or services that enable the accomplishment of these tasks.

There are several approaches such as [4,5] that provide models for interacting with smart spaces. However, the solutions they propose are rather application-centric or service-centric while our approach is task-centric. In this paper, we present the task computing framework, a framework designed to provide support for users (in executing their everyday tasks) and developers (in developing and deploying tasks) in smart spaces. We call a smart space fitted with a task computing framework a taskable space [6]. A task-

[1]A task is a goal or objective presented by a set of actions performed collaboratively by humans and/or machines to achieve its goal.

able space can recommend possible, relevant tasks for the users based on contextual information related to the physical space (both physical and virtual) and the users. A taskable space hides the complexity of underlying technologies and computational processes. The users interact with the space to accomplish their intended tasks using their personal devices such as mobile phones.

The rest of the paper is organised as follow. Section 2 presents requirements for the proposed framework. We elaborate the proposed framework in Section 3. We decide to incorporate related work and research problems in every sub-sections representing components of our framework. Section 4 describes an early implementation of a system developed based on the framework. Finally, a conclusion is given in Section 5.

2. Requirements

This section presents requirements we consider essential for the cornerstones of the proposed framework. The first three requirements are from users' expectation while the final requirement is from designers' expectation.

2.1. Task Recommendation & Execution

Users in smart spaces are surrounded by hundreds of services and devices including sensors, actuators, appliances, and computers. Currently, in order for the users to exploit these resources, they must not only be aware of the resources' existence – the capabilities, availability, and cardinality of these resources but also understand configurations of these resources to accomplish a particular task. In a smart seminar room, for example, the user needs to be aware of existing devices (e.g., displays, projectors, printers, desktop and computers) and services (e.g., light control, temperature control, window blind control, audio control, slide-show manager, and video conference). This requirement is impossible for ordinary users, especially for those who are unfamiliar with the space. In contrast, a taskable space assists users in being aware of what tasks they can do within the current space by discovering and recommending possible, relevant tasks for the users. For example, a presenter, who is unfamiliar with the smart seminar room, can quickly get aware of tasks he can do such as 'Register as a presenter', 'Publish presentation slides', 'Print a document', 'Dim the lights', and 'Make a slide-show'. These tasks are dynamically recommended on e.g. his/her mobile phone when he/she asks for recommendation. Upon selecting a task, the user will be guided through the task accomplishment on the mobile phone. The phone is only the interface for the user to interact with the space while all computational processes are hosted by the space.

To avoid the user digging in a long list of recommended tasks for a desired task, not all of available tasks but only relevant tasks are recommended for the user. This is done by taking into consideration the user's current context such as location, devices nearby, activity (e.g. presenting at a seminar), and user's pointing gesture (i.e., the user points the phone at a particular appliance).

The taskable space allows navigation both forward and backward during task execution if applicable, and allows task execution to be paused and resumed (perhaps at different locations) as needed. Moreover, depending on the interaction styles (e.g. pen-based interaction, touch-based interaction, and speech-based interaction) of devices the user is using, the taskable space automatically generates appropriate user interfaces.

2.2. Task-Oriented User Interfaces

The traditional paradigms of user interfaces (e.g., device-oriented user interfaces and application-oriented user interfaces) require users to know which devices and services to use before they can achieve a certain task. This philosophy is no longer suitable in smart spaces where services can be "invisible" from users' awareness. For example, a user may not be aware of that a physical table can be used as a display and keyboard to compose and send emails. To address this issue, a taskable space provides users a view of the entire environment from a task-oriented perspective. Specifically, given an unfamiliar space, a user does not have to know which devices and/or services to use before he/she can do a particular task.

2.3. Mobility of Tasks

The users' mobility across or within taskable spaces require task executions to be mobile (perhaps following the user). There are two types of mobility: intra-space mobility and inter-space mobility [4]. Intra-space mobility is related to the migration of executions of sub-tasks among different devices within a taskable space and is the result of ad hoc availability of devices and services (e.g., because of the user mobility). Intra-space mobility allows executions of sub-tasks to move among different devices according to e.g. users' locations and the interaction device at hand. Inter-space mobility concerns moving task executions across different spaces. A taskable space enables the migration of all settings from the previous task session to a new space. Accordingly, the user interface is adapted to current interaction devices and available services within the new space. If the currently performed task/sub-task fails (because of such as failure of a device/service or unavailability of a required service/device), the taskable space quickly offers an alternative.

2.4. Creation & Management of Task Models

Task models are formal descriptions which describe how high-level tasks can be achieved by using lower-level tasks. Task models are an excellent back-bone for dynamically generating task-oriented user interfaces that cover the functionality of a whole set of devices/services in a space rather than having one user interface per device/service [7]. Task models should be abstract enough to avoid being bound to specific devices/services. This allows task models to be flexibly executed in different spaces.

There is a need for a task model description language and an editor for creating task models based on this language. Once task models are created, there is a need of a mechanism for effectively advertising, searching, and retrieving task models over a network. This requires an architecture for publishing and managing task models which is similar to the Universal Description, Discovery and Integration registry (UDDI) for businesses worldwide to list themselves on the Internet.

3. Components of the framework

To fulfil the requirements above, we propose a task computing framework that models functionality of a taskable space as a collection of possible tasks. The framework exploits

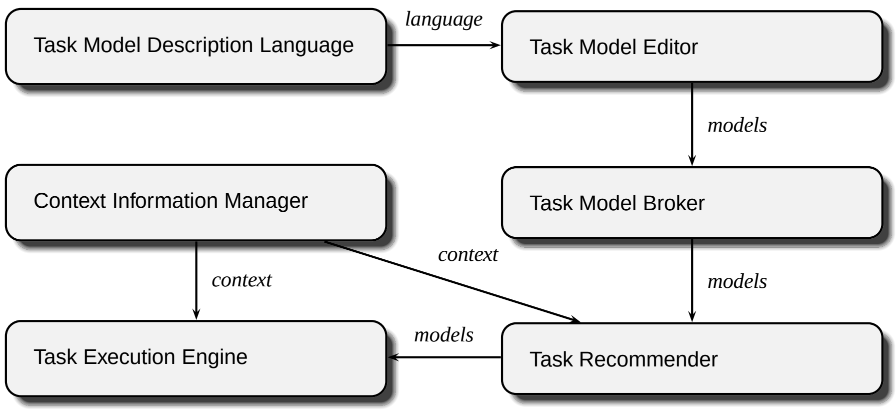

Figure 1. The component architecture for the proposed framework and the relationships of components. Developers use Task Model Editor to develop task models which are described in a standard Task Model Description Language. These task models are registered with Task Model Broker. Task Recommender recommends tasks for users based on context information provided by Context Information Manager. Task Execution Engine executes task models of selected tasks taking into consideration context information.

resources (devices, services, and task models) present in a taskable space, recommends relevant task for users, executes selected tasks, guides users through task executions, and provides tools for task model creation and management.

The framework consists of six components: *Task Model Description Language*, *Task Model Editor*, *Task Model Broker*, *Task Recommender*, *Task Execution Engine*, and *Context Information Manager*. Figure 1 illustrates the component architecture for the framework and the mutual relationships of components.

3.1. Task Model Description Language

A task model description language consists of a set of symbols and a set of grammar structures, rules, and schemas which are used to describe task models. Some challenges need to be addressed when designing and choosing a task model description language for the framework:

- Standardisation to enable unify and inter-operation,
- Supporting semantic descriptions/annotations,
- Supporting runtime execution of task models, and
- Technology-independent.

There are several approaches on task modelling. The main disadvantage of these approaches is that task models are not executable (i.e., they are used only for designing user interfaces at design time, then discarded) and rather device-dependence [8]. This limits the scalability of task models. Our framework uses the ANSI/CEA-2018 standard [9] as a task model description language because it well addresses the challenges above. However, our experiences revealed that there is a need for an extension of this standard that allows queries of context information to be integrated into task models. This is because some parameters required for executing a task can be automatically filled by context information (e.g., location) which is provided by Context Information Manager.

3.2. Task Model Editor

A task model editor supports for developing and describing task models conforming to a task model description language. It also supports visualisation, validation, and simulation of task models. ConcurTaskTrees [10] is a graphical editor that is widely used for specifying task models using diagrammatic notations. This tool aims to only support for designing user interfaces at design-time. The TERESA [11] tool based on ConcurTaskTrees goes a step further by automatically generating a corresponding user interfaces of a task models for different platforms and devices (i.e., the same task models can have a different presentation depending on the platform or device). The disadvantage of editors based on ConcurTaskTrees is that task models are tightly coupled with specific applications. Being too application-specific will limit scalability of task models in different taskable spaces.

Currently, there is no such an editor for authoring task models based on the ANSI/CEA-2018 standard (perhaps because this standard has just been approved recently). However, we believe that developers are developing editors for this standard and they will be available soon in the future.

3.3. Task Model Broker

A task model broker is a repository of task models. It publishes and adversities task models which have been registered with the broker. There are a number of research challenges on management of tasks models. The broker should provide effective mechanisms for searching and matching task models. Task model searching is to find appropriate task models. Task matching is a process that finds similar task models (e.g., a failed task can be substituted by a similar task). Managing a large repository of task models is also challenged which require more research attention on indexing techniques and query answering solutions.

3.4. Task Recommender

Imagine that a taskable space (e.g., a university campus or a smart seminar room) contains hundreds of controllable devices and computational services. Individuals and different combinations among them can support users with hundreds of high-level tasks. This list of possible tasks is highly ever-changing because of the change of the space resulting from unforeseen adding, removing, and upgrading devices and services. Moreover, because technologies blend well into the space, users may not recognise advanced features beyond common usages of normal objects. Therefore, there is a need for a task recommender system that suggests users for possible, relevant tasks they can do within a particular space at a certain time.

A task recommender dynamically discovers possible tasks based on available devices and services in the space. Then, in collaborating with a context information manager (as discussed below), the task recommender reduces the number of tasks which are relevant to users' current situations. Towards this vision, Chuong C. Vo and his colleagues [12] propose a conceptual task recommendation system which takes task ontologies (e.g. place/device-specific task ontologies), environment's capability (i.e. devices' functionalities), users' situations, and users' preferences into consideration for discovering relevant tasks. This system is needed to be further implemented and evaluated in the combination with our proposed framework.

3.5. *Task Execution Engine*

The runtime task execution engine is to execute task models. It guides the user through task execution and deals with how should the system inform the user of the progress of task completion, failures, and, if necessary, involve the user in the task execution. The engine also enables managing sessions of task executions and avoids the user having to start the session over again from the beginning when resuming the task.

Anand Ranganathan [13] propose a task execution framework which can automatically infer the best values for task parameters based on the current state of the space, context-sensitive policies, and learned user preferences. It can also recover from failures of one or more actions by using alternative resources. Similarly, the authors in [14] propose a method for estimation of future resource requirements when executing a task, which allows task execution to adapt to the variation in availability of resources in the space. While these frameworks mainly focus on executing tasks, their solutions to resource allocation problems can be incorporated into our framework.

Recently, a reference implementation of an engine for executing task models which are specified using the ANSI/CEA-2018 standard has been developed [8]. The current problem of this implementation is its command-line user interface which is not convenient for users when they need to provide inputs during task execution, especially on devices with small screens such as mobile phone or PDA. We are extending this implementation based on a client/server architecture to allow the user remotely executing tasks through their mobile devices.

3.6. *Context Information Manager*

Context information manager is an essential component for any intelligent systems today. Context is any information that can be used to characterise the situation of a user [15]. The Context Toolkit [15] is a well-known toolkit supporting development of context-aware applications. However, the management of context information using this tool is very application-centric (e.g. the configuration for context provision/retrieval has to be set up explicitly for each application). This is not suitable for task-oriented computing where context information could be used by any tasks and shared among multiple components of the framework.

4. Current Implementation

We have implemented the initial framework and an application called TASKREC based on this framework. TASKREC is a web page running on any devices which have a web browser. The web page consists of Java scripts and uses Ajax technology to retrieve and render the dynamic list of recommended tasks for the user to select. The task list is frequently updated by the task recommender hosted on the server. The task recommender always listens to the change of the user's context (in this implementation, we only consider user's location and user's pointing gestures as user context). We use multi-model location technologies for inferring user's location including GPS, Bluetooth, RFID, and Cricket[2].

[2]http://cricket.csail.mit.edu/

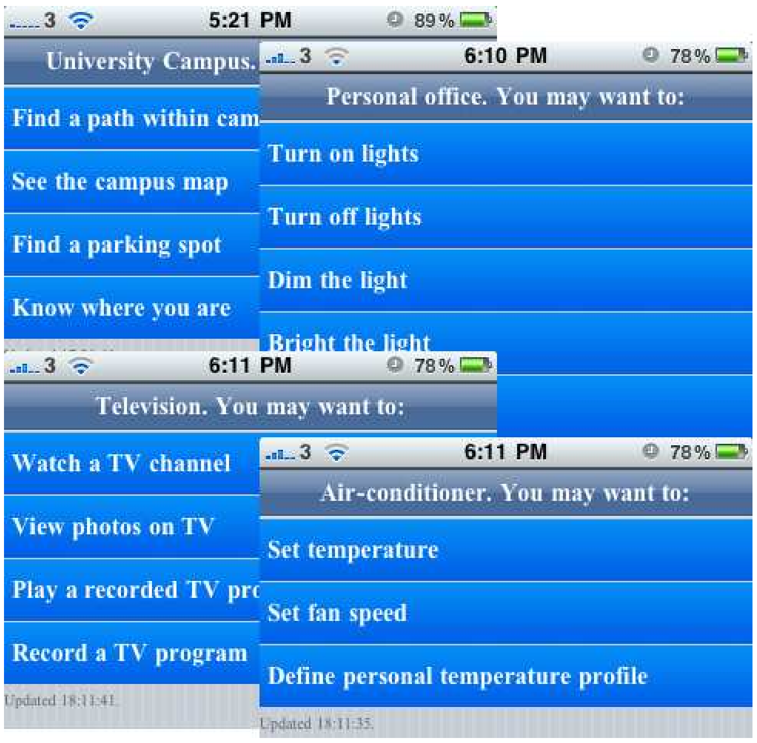

Figure 2. Location-based task interfaces for smart spaces.

We observed the changes of the task list in four scenarios (see Figure 2): the user enters the University campus, the user enters his personal office, the user points his phone at the television in his office, and the user points his phone at the air-conditioner. Currently, the tasks for controlling of lights and fan have been implemented using an X10-based HomeAutomation Kit though any other technologies are possible to be incorporated into our framework as long as the designers are provided URLs of services for controlling devices.

5. Conclusion

This paper has analysed requirements and challenges towards a task-oriented computing framework for smart spaces. We have presented the proposed framework consisting of six components. We have discussed related work and outlined future research directions to realising this framework including:

- Languages for describing task models,
- Tools for creating task models,
- Mechanisms for effectively publishing and retrieving task models,
- Context-aware task recommendation, and
- Engines for executing task models.

We wish to promote an active research community of task-oriented computing to develop more task-oriented applications for smart spaces. Meanwhile, smart spaces (e.g., university campuses, houses, and public places) will inevitably become commonplace. This gives us great opportunities to develop and experiment our task-oriented applications.

Acknowledgment

The research was carried out whilst the first author was supported by La Trobe University Tuition Fee Remission & Postgraduate Research Scholarships.

References

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

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

[3] R. Masuoka *et al*. Task computing–the semantic web meets pervasive computing. In *ISWC 2003*, 2003.

[4] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[5] 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.

[6] S.W. Loke. Building taskable spaces over ubiquitous services. *IEEE Pervasive Computing*, 8(4):72–78, 2009.

[7] 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.

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

[9] Consumer Electronics Assoc. Task model description (CE Task 1.0), ANSI/CEA-2018, Mar. 2008.

[10] F. Paternò, C. Mancini, and S. Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In *INTERACT '97*, pages 362–369, London, UK, 1997. Chapman & Hall, Ltd.

[11] G. Mori, F. Paternò, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520, 2004.

[12] C.C. Vo, T. Torabi, and S.W. Loke. Towards context-aware task recommendation. In *ICPCA-09*, Taiwan, 2009.

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

[14] 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 *CF '04: Proceedings of the 1st conference on Computing frontiers*, pages 70–77, New York, NY, USA, 2004. ACM.

[15] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.