

# TASKCOM: from Task Models to Task-Oriented User Interactions with Smart Environments

Chuong C. Vo\*, Torab Torabi, Seng W. Loke

*Department of Computer Science & Computer Engineering, La Trobe University, Australia*

---

## Abstract

The current computing interaction paradigm is mostly application-centric and/or document-centric, which has been dominantly employed for desktop environments. In smart environments such as smart homes and smart offices, this paradigm might no longer be suitable for occupants whose attention is often preoccupied with daily activities. In this paper, we propose a task-oriented computing paradigm for users to interact with smart environments. This paradigm allows users to focus on their tasks rather than low level mental concepts such as device features, applications, and services. This paradigm also supports collaborations (i.e., multiple users can collaborate on the same task) and context-aware task recommendation. In this paper, we present our design and implementation of a task-oriented computing framework (TASKCOM) for this paradigm.

*Keywords:* Task-Oriented Interactions, Task-Oriented Computing, Task Recommendation

---

## 1. Introduction

A smart environment [22] 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.” There is a trend in smart environment technologies that electric appliances often come with a mobile application that the users can install on their mobile devices (e.g., smartphones and tablets) to control the appliances remotely. For example, several models of Samsung air conditioners can be controlled by the Smart Air Conditioner application<sup>1</sup>. Several models of LG washers and dryers can be controlled by the LG Smart Laundry & DW application<sup>2</sup>. Even a single bulb such as Bluetooth Bulb<sup>3</sup> or a milk jug<sup>4</sup> also have their smartphone applications. This current model of user interactions with smart environments is oriented around applications (so called the “application-centric interaction paradigm” [2, 17]).

---

\*Corresponding author

*Email addresses:* [c.vo@latrobe.edu.au](mailto:c.vo@latrobe.edu.au) (Chuong C. Vo), [t.torabi@latrobe.edu.au](mailto:t.torabi@latrobe.edu.au) (Torab Torabi), [s.loke@latrobe.edu.au](mailto:s.loke@latrobe.edu.au) (Seng W. Loke)

<sup>1</sup><http://www.samsungapps.com/topApps/topAppsDetail.as?productId=G00005321268>

<sup>2</sup><https://play.google.com/store/apps/details?id=com.lg.apps.lglaundry>

<sup>3</sup><http://bluetoothbulb.com/>

<sup>4</sup><http://www.teehanlax.com/labs/do-we-have-milk/>



Figure 1: List of application icons on mobile devices. Left to right: iPhone4, Galaxy Nexus, and Nokia-N9.

The application-centric interaction has been a dominant paradigm in desktop computing for more than forty years. It is based on the traditional assumption that the users are sitting in front of a personal desktop computer with a mouse and a keyboard as the input methods. However, this assumption is not always valid in smart environments where the users are expected to be mobile; their tasks are expected to be short-term, event-driven, frequently interrupted, and involve ad-hoc collaboration; and mobile devices can accept touch, voice, and gesture input methods. Indeed, from the user's perspective, the application-centric paradigm has the following potential issues for user interaction with smart environments:

**It does not scale well with the increasing number of appliances [19].** If the user needs to control tens of different appliances (perhaps some of them are not always relevant to the user, especially occasionally visiting public smart environments), he/she might need to install that number of applications<sup>5</sup> on his/her mobile device. If the user has an extra mobile device or has just changed his/her mobile device, he/she would need to repeat the same number of application installations. Consequently, the increase in the number of appliances would result in an expanding collection of application icons on the user's mobile devices. In fact, current mobile devices present to the users a potentially "infinite" number of inanimate application icons on the home screens as shown in Figure 1.

From users' perspectives, the application-centric paradigm is a very low level of abstraction:

---

<sup>5</sup>In this paper, an application means a mobile application which is used to control an appliance. Because the mappings between appliances and applications in the context of our paper are one-to-one, "appliance" and "applications" are interchangeable in this paper.

applications and individual appliances, rather than a high level of abstraction; e.g., user tasks [1, 19]. Indeed, to accomplish a task which involves controlling one or more appliances in a smart environment via a mobile device, the user must mentally (1) determine what appliances should be used for the task; (2) split the task into sub-tasks in order to map them to the appliances; determine the applications which control these appliances; (3) map the applications to application icons; and (4) locate the application icons on the mobile device's screen. This process is complex and demanding of cognition.

**It does not scale well with the increasing number of appliances' features [6, 16].** Most applications are not directed towards the users' tasks, but rather designed to present all their functions on a graphical user interface with icons, tabs, menus, lists, buttons, and dialog boxes. An application with rich functions might have tens of menu options which rarely take into account contextual information such as the task that the user is currently accomplishing. Because the user interface does not automatically adapt according to the user task, the users must discover, find, and combine functions together to accomplish their intended tasks. This can be difficult and time-consuming [10].

**It does not cope well with frequent changes [19].** A smart environment can rapidly configure by adding, removing, or upgrading appliances. Accordingly, to carry out tasks, users must incorporate these changes (e.g., install, update, or uninstall the impacted applications). This is demanding of attention and unacceptable, especially for occasional visitors who come in to a smart environment for a short time and then leave.

**It does not cope well with tasks involving multiple applications [9, 10].** Current mobile applications mostly operate in isolation from others. They have no knowledge about the existence of other applications. They rarely automatically and implicitly collaborate with each other. For example, it is rare to see one application automatically discover and use a feature (e.g., turn on/off a light) which is provided by another application. Therefore, the user must manually combine the uses of the applications to fulfil a task. Indeed, the user must manually split the task into sub-tasks and map them to the right applications, or even to the right functions of the applications, and switch between the applications during the execution of the task. For example, a smart room has a light and a window drape each of which can be remotely controlled by a different application on a smartphone. If the user's task is to brighten the room with more natural light from the outside, he/she firstly needs to split this task into two sub-tasks and then map them to the right applications. In particular, one sub-task is to dim the light which is mapped to the light control application and the other sub-task is to open the drape which is mapped to the window drape control application. Next, the user runs the light control application to adjust the light, then runs the window drape control application to adjust the drape. He/she may need to switch between the two

applications several times to achieve the preferred level of the room's brightness.

Moreover, inter-application data exchange is only achieved manually and indirectly through the use of clipboard or a memory card. While the clipboard currently only supports plain-text data, the use of a memory card requires that two applications previously agree on the structure of the data they will exchange. For example, unless direct inter-application data exchange is supported, it is impossible for an application to directly consume RFID data which is just read by another application.

**It does not support tasks involving multiple users [1].** Tasks may need collaboration among multiple users across smart environments. For example, Bob is a student who wants to borrow a book from a library remotely via his mobile device with help from his friend who is currently located in the library. Although there is a mobile library application which allows him to borrow books by using his mobile device's camera and NFC to scan books' barcodes and to read student cards' codes, it is still impossible for him to accomplish this particular collaboration task unless the library application is specifically designed to support such kind of collaboration.

**It does not support composition, reuse, and personalisation of tasks.** Users may like to repeat a similar or exactly same task at different times, compose two different tasks into a new task, or personalise a task. For example, given an application for switching a light on/off and another application for opening and closing a window of a room, the user may wish to compose two new tasks based on the existing basic tasks. One of the task will turn on the light and open the window while the other will turn off the light and close the window. Next, the user may wish to reuse this same task composition for controlling the light and window in his/her office. Finally, the user's colleague may wish to have similar task compositions but in the reverse working order. The current application-centric paradigm does not support these needs.

**It does not support resuming a task on different mobile devices.** Consider a situation that a user is currently accomplishing a task on a tablet which is running out of power. The user wishes to suspend the task execution on the tablet and resume it on a smartphone. This need is difficult to achieve with the current application-centric paradigm unless the application is specifically designed to use a cloud service that stores the task's state on a server.

In recognition of these issues and user needs, many companies like Microsoft, Apple, Samsung and Google are searching for alternatives. They are examining metaphors for input methods such as voice, touch, and gesture. As a result, a number of commercial systems have recently been developed to better support mobile users in accomplishing their tasks on mobile devices.

In particular, Apple’s Siri<sup>6</sup>, Google’s Google Now<sup>7</sup>, Samsung’s S Voice<sup>8</sup>, and Microsoft’s Tellme<sup>9</sup> are initial attempts which allow the users to tell the mobile devices several tasks they want to accomplish. Such systems are becoming an important factor for competition in the mobile industry. Although these systems have a capability of speech recognition (mostly English) and massive knowledge bases (e.g., the Wolfram Alpha knowledge base), currently they only reach the stage where they can only answer several information questions or execute phone-related typical tasks such as sending messages, making phone calls, or setting reminders. Most of these tasks are single mobile device-supported. Clearly, these systems do not address all of the shortcomings and user needs aforementioned.

As we shall discuss in more detail later in Section 5, the research community has also developed several systems that aim to help users in accomplishing their tasks in smart environments. However, these system only support some of the requirements we mentioned previously. Notably, these systems often heavily rely on the capability of context recognition (e.g., the recognition of location and user activity). Because techniques for context recognition often require a training phase and streaming data which are very domain-specific, such techniques cannot be deployed widely and generally. In fact, as far as is known, not many of such systems have been available in our everyday life.

In this paper, we propose and develop the paradigm of task-oriented computing for smart environments. The aim of task-oriented computing is to allows users to focus on their tasks directly rather than individual appliances and applications. It also aims to reduce the users’ cognitive load in accomplishing their tasks using appliances in smart environments. To realise the task-oriented computing paradigm, we have designed and implemented a task-oriented computing framework, we call TASKCOM. As we show in detail later, TASKCOM has the following advantages from the end-user’s perspectives.

**It can scale with the increasing number of appliances.** TASKCOM provides a uniform interface for users to access features which are provided by appliances in a smart environment. In TASKCOM, tasks are the fundamental elements. TASKCOM operates on top of a service-oriented technology. From users’ perspective, there are no applications (of course, we are not trying to remove/replace all applications unless all tasks are achievable without applications). Therefore, application-related activities such as finding, downloading, (un)installing, updating applications are no longer necessary. Other manual activities are also not needed such as determining what appliances should be used for the task, splitting the task into sub-tasks in order to map them to the appliances, mapping the applications to the appliances, mapping

---

<sup>6</sup><http://www.apple.com/iphone/features/siri.html>

<sup>7</sup><http://www.google.com/landing/now/>

<sup>8</sup><http://www.samsung.com/global/galaxys3/feature.html#svoice>

<sup>9</sup><http://www.microsoft.com/en-us/tellme/>

the applications to application icons, and locating the application icons on the mobile device's screen. Features of appliances and applications are provided as services each of which will automatically be called if the execution of a task requires a feature which is provided by that service. For example, if the user's task is to adjust a light's brightness, he/she just tells TASKCOM something like "adjust light", then the system would show a user interface with a slider that the user can change its value to adjust the light's brightness.

**It can scale with the increasing number of appliances' features.** When a smart environment is added with many appliances, each of which may be very rich in features, the number of possible tasks could be exponential. To deal with this situation, TASKCOM allows the users to quickly express their intended tasks via a universal search dialog using either voice or gesture. TASKCOM can also recommend users only relevant tasks based on their context.

**It can cope with frequent changes of smart environments.** With TASKCOM, the users do not need to be aware of the (un)availability of appliances. All changes will be reflected via the availability of tasks which the users can execute. If a change of the smart environment results in the (un)availability of a task, this task will be added/removed from the list of possible tasks. For example, if a light has been removed, then the task of turning the light off will no longer be available from the task list.

**It can cope with tasks involving multiple appliances.** In TASKCOM, appliances expose their functionalities as services which are composed into task models. This means that a task can compose and coordinate functionalities of different appliances. For example, if the user's task is to brighten a room, he/she would only need to tell the system "brighten room", the system will show the task "adjust room lighting" in the task list. Once the user selects the task, the system shows a user interface where he/she can adjust the light and the window drape at the same time.

**It can support multiple user tasks.** TASKCOM allows multiple users to collaborate on the same task. For example, during an execution of a task which has multiple sub-tasks, a user can ask other users to finish some of the sub-tasks, or a user can invite other users to remotely observe the accomplishment of the task.

**It can support for composition and personalisation of tasks.** In TASKCOM, tasks are specified as compositions of sub-tasks, services, or manual actions (i.e., instructions). Therefore, task specifications can be reused and personalised.

**It can support resuming a task on different mobile devices.** The user can suspend a task and resume it on another mobile device.

The rest of this paper is organised as follows. The next section presents the background of task computing with scenarios and concepts. Then in Section 3, we present the technical design

and implementation of our proposed task-oriented computing framework for user interaction with smart environments. We discuss our approach in Section 4, followed by related work in Section 5. The final section concludes the paper.

## 2. Task-Oriented Computing

The notion of task computing<sup>10</sup> has been receiving increasing research interest [1, 4, 7, 8, 11, 16, 19], especially since the emergence of pervasive computing [21]. The idea of task computing is that future software systems should allow users to interact with computers in the form of high-level tasks while freeing them from low-level configuration activities. In other words, task computing tries to shift users' focus towards what they want to do, and away from the specific means for doing those tasks [8]. Tasks are the first class objects in such systems, which we call "task-oriented systems".

### 2.1. Scenarios

The following user scenarios demonstrate how task-oriented computing supports mobile users in smart environments. The first scenario describes a task of a student who wants to remotely borrow a book from a smart library on a smartphone. This scenario illustrates how task-oriented computing supports the student with task execution and collaboration. The second scenario illustrates the concepts of context-aware task recommendation and task-oriented interaction with a smart environment. Note that, we consider a simple environment which consists of one user, one mobile device, and one cloud service as a smart environment.

#### **Scenario 1:**

*Bob is a newly arrived postgraduate international student. He currently is at home and wants to remotely borrow a book from the university library with help from his friend, Alice, who is located in the library at the time of borrowing. He tells his smartphone 'borrow book'. The smartphone, which is connected to a task-oriented system provided by his university, shows the "borrow book from library" task on the screen and asks him to confirm his intended task. Once Bob confirms his task, the system loads the task model specification which is specifically defined for students to borrow books from the library by themselves over mobile devices. This task has a sequence of sub-tasks such as searching for a book in the database, locating the shelf where the book is placed, and checking out the book. The task of checking out the book further has two sub-tasks: scanning a student card and scanning the book's barcode. Each of the sub-tasks corresponds to a user interface that provides instructions and allows Bob to supply necessary information for the task.*

*When Bob has found the location of the book, he shares this task to Alice, so she knows where the book is and picks up the book from the shelf for Bob. When Bob is checking out the book, he*

---

<sup>10</sup>The similar terms are task-driven computing [19] and activity-based computing [1].

scans his student card by the use of his smartphone's NFC reader. Then he again shares the task with Alice so she can help him scan the book's barcode using her smartphone's camera. Once Alice's completed scanning the book's barcode, the task is brought back to Bob so he can confirm and finish the borrowing.

### **Scenario 2:**

At the university, Bob is provided with a smart personal office which has computer-controllable appliances such as a light, a TV, a heater, a window drape, and a coffee maker. Bob does not need to install any applications for controlling these appliances because the university's task-oriented system provides him a set of possible tasks for controlling those appliances. To not overwhelm Bob with too many tasks, the system uses his context (e.g., location) to recommend him only relevant tasks. For example, when Bob is in his office, the list of recommended tasks includes print documents, brighten the room, darken the room, make room warm, and watch TV. When Bob is nearby or pointing to the TV, the system recommends him tasks which can be done with the TV; e.g., watch TV, change channel, and adjust volume. Moreover, Bob can quickly navigate to a task on his mobile device by the use of a global search dialog that accepts advanced input methods such as voice and gesture.

## *2.2. Concepts*

The scenarios above illustrate several concepts in our paradigm on task-oriented computing, which are supported by our framework.

**Tasks.** We consider a task as a goal (which is expressed by a human-understandable phrase) that a user wants to achieve. The user can accomplish a task by either manually operating appliances or remotely controlling the appliances via a user interface on a mobile device (e.g., smartphone or tablet). The remote controlling of appliances is achieved by invocation of software services. Formally, in our framework, a task can be a basic action on an appliance, an invocation of a service, a sequence of sub-tasks (i.e., the order is significant), or a set of sub-tasks (i.e., the order is insignificant).

A task may involve multiple appliances, and/or multiple users across multiple smart environments. A user may at any time suspend an on-going task and start a new one or resume a task that has been suspended previously. Tasks may be planned ahead to be triggered to start based on a criteria (i.e., applicable context) such as time and location. They can be handed over to another person, or they can be shared to enable collaboration.

**Task models.** A task model [13] is an execution model or a routine of a task that describes how the task should be performed to reach the goal of the task. It may include models of sub-tasks and constraints.



**Task model specifications.** A specification of a task model<sup>11</sup> is an actual document which is written in a standard language (e.g., an XML-based language).

**Services.** TASKCOM operates on top of a service-oriented computing infrastructure which exposes everything computer-embedded (including applications on mobile devices such as map or calendar applications) as services.

**Devices.** Devices include mobile devices (e.g., smartphones and tablets which are used by users to interact with their tasks) and any other appliances and digital devices which are controllable via services.

**Users.** A user can be an executor and/or a collaborator of a task.

**Recommended tasks.** The tasks which are relevant to the user's current context.

**Capturing user's intended tasks.** This is a capability of a task-oriented system to infer users' intention of tasks which they want to accomplish. For example, a task-oriented system has the capability of speech recognition that allows users to express their intended tasks by voice in natural language.

**Task-oriented user interface.** TASKCOM provides a uniform interface for users to access functionalities and features provided by appliances in smart environments. The interface is re-oriented around tasks, rather than a list of buttons (e.g., on remote controls for appliances) or application icons (e.g., on most current smartphones, see Figure 1). This interface also allows users to quickly express their intended tasks (e.g., using voice), and provides them with step-by-step guidance to accomplish selected tasks.

### 3. The Task-Oriented Computing Framework

The TASKCOM framework is an implementation of our concept of task-oriented computing. The main goal of this framework is to provide a runtime platform and a model for the development and deployment of task-oriented systems. A task-oriented system is an entire solution; it is not merely a mobile application running on mobile devices. The runtime platform manages task model specifications, task instances (e.g., execution status of tasks), users, context, service invocations, and automatic generation of task-oriented user interfaces. The programming model helps programmers and administrators specify task models and deploy their task-oriented systems.

In this section, we first present the architecture of the framework. We then present task modelling, task recommendation, users' intention capturing, task execution, task resumption, and task collaboration.

---

<sup>11</sup>Some authors call it a task model description [16].

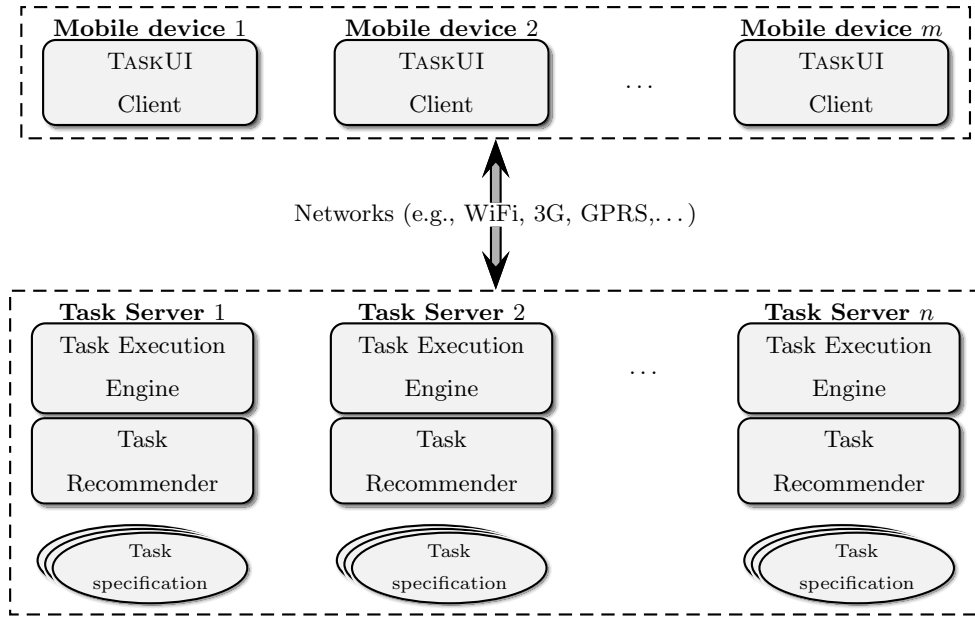


Figure 2: Conceptual component and deployment architecture.

### 3.1. Architecture

Figure 2 illustrates the conceptual component and deployment architecture of the framework. The architecture consists of task servers and TASKUI. There can be an unlimited number of task servers which manage task model specifications and task instances. For example, in Scenario 2, the University has its own task server that allows the staff and students to accomplish several relevant tasks. A smart meeting room may also have its own task server which manages task model specifications for controlling appliances within the room. A user may also have a personal task server that manages his/her personalised task model specifications.

TASKUI is a mobile client-side software which runs on mobile devices (e.g., smartphones) which acts as an interface for users to interact with their tasks (see Figure 4 & 7 for examples). TASKUI generates user interfaces for tasks based on specifications of their models. TASKUI can connect to multiple task servers concurrently, allowing users to access tasks hosted on different task servers at the same time. Users can add or remove a particular task server on demand. They can also manually switch to particular task servers or allow TASKUI to do it automatically based on their context. TASKUI provides a task-oriented user interface which presents the user with relevant tasks, instructions for accomplishing selected tasks, and other task features (e.g., next, back, cancel, skip, and share). TASKUI reserves the look and feel of different mobile platforms because it reuses native user interface elements which are provided by that platform.

The Task Recommender handles the real-time recommendation of tasks for a user based on the context. For example, when a user changes from his/her car “environment” to his/her home “environment”, the home relevant tasks are recommended while the car-relevant tasks are hidden

(or have lower priority when shown in the task list). Then, when the user is pointing<sup>12</sup> his/her mobile device at the heater, the heater-relevant tasks are recommended for him/her.

The Task Execution Engine is the runtime engine which manages registered users and the life-cycle of task instances (e.g., instantiation, suspension, resumption, collaboration, and termination).

### 3.2. Task modelling

In TASKCOM, task models must be specified in a standard, machine-readable format which we call *task model specifications* or *task specification* for short. Task specifications are primary data which are managed and processed by the task servers. They are created by users, developers or administrators. A task specification is an XML document which specifies the task properties, task decomposition (i.e., steps to execute the task), inputs, outputs, required services, conditions, and user interface representation of the task and its sub-tasks. We've designed a basic RNC<sup>13</sup> schema for specifying task models.<sup>14</sup> Figure 3 shows the current version of this schema.

In a task specification, the `optional` attribute specifies if the task is optional. The user can only skip a task if it is specified as optional; the default value is `false`. The `shareable` attribute specifies if the task is shareable. The user can only share a task with other users if it is specified as shareable; its default value is `false`. The `ui` element specifies the user interface representation of the task. The TASKUI client software uses this information to generate the user interface for the task on a mobile device. Basic user interface elements, which are supported by our current schema, include map, text, image, selection, slider, list, and input. `Listener` is a special user interface element which listens for any changes made on other user interface elements by the user then responds to the changes by invoking a specified service. For example, when the user changes the value of a slider on the user interface of an “adjust TV volume” task, the slider’s listener will invoke a corresponding service which then results in changing the volume accordingly. Figure 4 shows a specification of an “adjust TV volume” task and its corresponding user interface generated by TASKUI on an Android mobile device.

#### 3.2.1. Task Composition and Decomposition

Task composition lets developers and users create new task specifications by combining existing task specifications and ordering them to best suit their requirements. Task decomposition is the reverse of task composition. In TASKCOM, task models are hierarchical. A model of a task is a composition of other task models (called *sub-tasks*). In turn, each of these sub-tasks is decomposed further unless the sub-task is an “action” (which is a manual operation on an actual appliance) or a service call. The representation of a task model is a tree (we call it a “task tree”) where the

---

<sup>12</sup>Where pointing could mean the use of compass and location reasoning, the mobile camera with image recognition (in the augmented reality style), RFID technology, NFC technology, ultrasound or infrared technology.

<sup>13</sup><http://relaxng.org/>

<sup>14</sup>The latest RNC schema and samples of task specifications can be found at <https://github.com/ccvo/taskcom/>

```

default namespace = "http://homepage.cs.latrobe.edu.au/ccvo/task"
namespace xsd = "http://www.w3.org/2001/XMLSchema"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = Task | IncludedTask | Service | If

Task = element task {
  attribute id { xsd:ID }?,
  [ a:defaultValue="false" ] attribute optional { xsd:boolean }?,
  [ a:defaultValue="false" ] attribute shareable { xsd:boolean }?,
  Title?,
  UI,
  ((Task | IncludedTask | Service | If)*)*
}

Title = element title { text }
UI = element ui { MapView*, TextView*, Image*, Select*, Input*, Slider*, List*, Listener* }

TextView = element textview { attribute text { text } }
Image = element img { attribute url { xsd:anyURI } }
Select = element select { attribute name { xsd:QName }, attribute value { text }?, Option* }
Input = element input { attribute type { DataType }, attribute name { xsd:QName },
  attribute value { text }? }
Option = element option { attribute value { text }, attribute text { text } }
MapView = element mapview { attribute address { text }? }
Slider = element slider { attribute name { xsd:QName }, attribute value { xsd:integer }?,
  attribute min { xsd:integer }, attribute max { xsd:integer },
  attribute step { xsd:integer }, attribute listener { xsd:QName }? }
List = element listview { attribute name { xsd:QName }, attribute value { text }?,
  attribute listener { xsd:QName }?, Option* }
Listener = element listener { attribute id {xsd:ID}?, attribute url { xsd:anyURI }, Arguments? }

Service = element service { attribute id { xsd:ID }?, attribute url { xsd:anyURI }?, Arguments? }
Arguments = element args { Argument* }
Argument = element arg { attribute name { xsd:QName }?, attribute value { text } }

If = element if { attribute id {xsd:ID}?, attribute condition { text }, Then, Else? }
Then = element then { (Task | IncludedTask)+ }*
Else = element otherwise { (Task | IncludedTask)+ }*

IncludedTask = element include { attribute id { xsd:ID }?, attribute taskid { xsd:ID } }

DataType = "barcode" | "address" | "datetime" | "phone" | "string" | "boolean" | "int" | "double"

```

Figure 3: The basic RNC schema for task specifications.

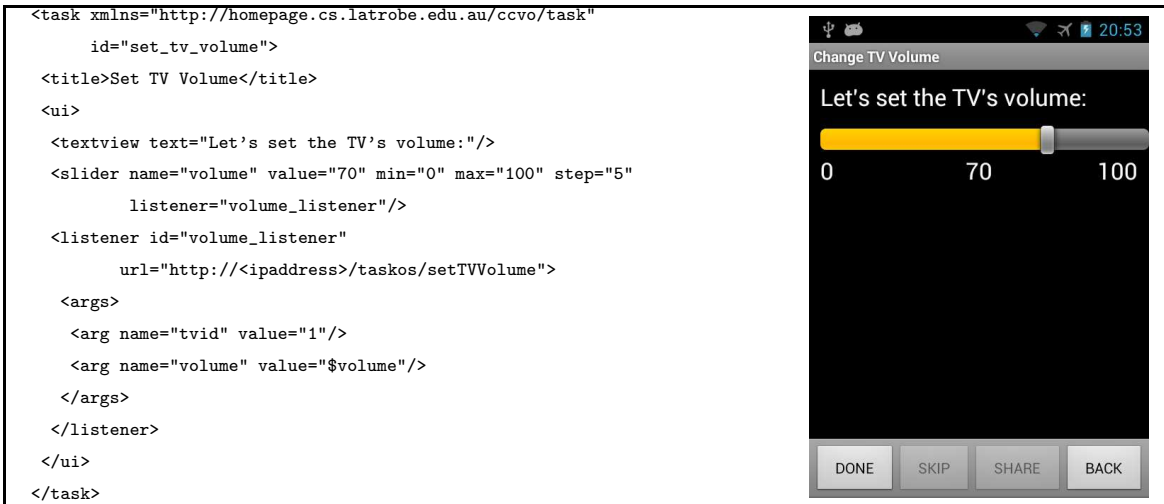


Figure 4: A specification for the “adjust TV volume” task and its corresponding user interface generated by TASKUI on an Android mobile device.

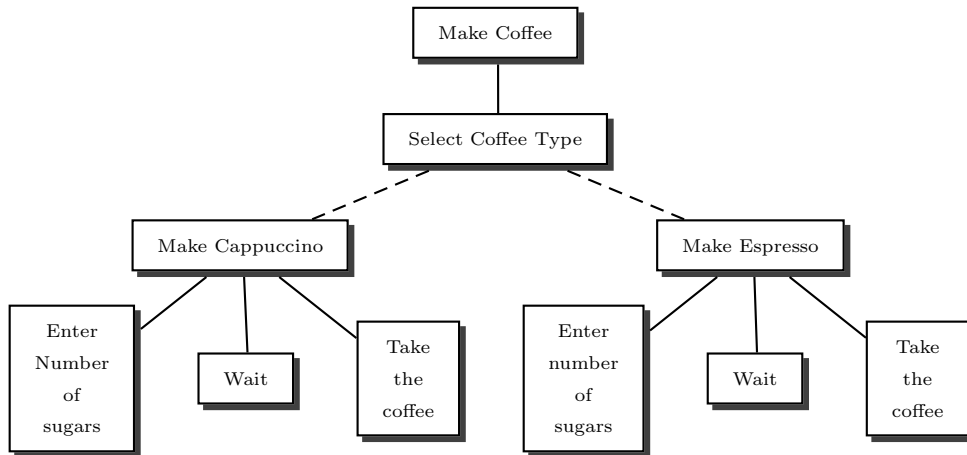


Figure 5: A graphical presentation of the “make coffee” task specification in Figure 6. Dotted lines indicate decomposition choices.

root is the task itself, the nodes represent the sub-tasks, and the edges represent the composition relations between these tasks.

Figure 5 contains a graphical presentation of a “make coffee” task model for coffee machines. This task model has a composition choice which allows the user to select the type of coffee he/she would like to make. Depending on the user’s selection, a “make cappuccino” task or a “make espresso” task will be actually executed next. As shown in Figure 6, a composition choice is represented by an *If-Then-Otherwise* element which consists of a conditional expression. This expression will be evaluated at runtime by the task engine in order to determine an appropriate sub-task to be executed next. The “make cappuccino” task and the “make espresso” task are referenced within this specification by the use of *include* elements.

```

<task xmlns="http://homepage.cs.latrobe.edu.au/ccvo/task" id="make_coffee">
  <title>Make Coffee</title>
  <ui><textview text="Let's make coffee"/></ui>
  <task id="select_coffee_type">
    <title>Select coffee to make</title>
    <ui>
      <textview text="What coffee to make:"/>
      <select name="coffee_type">
        <option value="0" text="Cappuccino"/>
        <option value="1" text="Espresso"/>
      </select>
    </ui>
  </task>
  <if condition="$select_coffee_type.coffee_type == 0">
    <then><include taskid="make_cappuccino"/></then>
    <otherwise><include taskid="make_espresso"/></otherwise>
  </if>
</task>

```

Figure 6: A specification for a “make coffee” task which shows how to specify a decomposition choice and references to external tasks.

### 3.2.2. Task Refinement

Task refinement allows the user to gradually refine a general task to a more specific task at runtime. For example, the user can start with a “control device” task and finally refine this to a “set TV channel” task. TASKCOM supports this feature by the use of service calls in task specifications. In particular, the invocation of a service can return another task specification which immediately replaces it for that service in the current task tree. We also call this feature dynamic task composition. Formally, a task  $t'$  is a refinement of a task  $t$  if the execution of  $t$  returns  $t'$ , we denote this relation by  $\mathcal{R}(t, t')$ . Task refinement is *irreflexive*, *anti-symmetric*, and *transitive*. The irreflexivity means that no task is a refinement of itself, in other words, there is no  $\mathcal{R}(t, t)$ . The anti-symmetry means that for all tasks  $t$  and  $t'$  with  $t \neq t'$ , if  $\mathcal{R}(t, t')$  then  $\mathcal{R}(t', t)$  must not hold. The transitivity means that if  $\mathcal{R}(t, t')$  and  $\mathcal{R}(t', t'')$  then  $\mathcal{R}(t, t'')$ .

Figure 7 illustrates a scenario where the task specifications contain a service call that returns another task specification which is more specific than the previous task. In this scenario, a user wants to control a device remotely. He starts the “control device” task which is very simple and general; its specifications has only one service with no user interface. The execution of the “control device” task directly leads to invocation of the “get locations” service, which in turn returns the specification of a task called “select location”. The “select location” task asks the user to select a location from a location list. After the user selects a location, the runtime engine will invoke the “get devices” service which is specified in the “select location” task. The invocation of the “get devices” service returns a specification of another task called “select device” that presents a list of devices in the previously selected location. Once the user selects a device from the device list, the runtime engine will invoke the “get commands” service which is specified in the “select device” task. This service returns a “select command” task which allows the user to select a command for

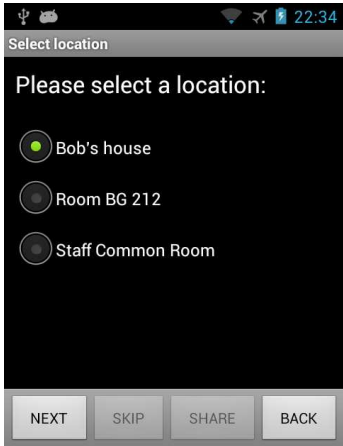
<pre> &lt;service xmlns="http://homepage.cs.latrobe.edu.au/ccvo/task"   id="control_devices"   url="http://&lt;ipaddress&gt;/taskos/get_locations"/&gt; </pre>	<p>[no user interface]</p>
<pre> &lt;task id="select_location"&gt;   &lt;title&gt;Select location&lt;/title&gt;   &lt;ui&gt;     &lt;textview text="Please select a location:"/&gt;     &lt;select name="location"&gt;       &lt;option value="0" text="Bob's house"/&gt;       &lt;option value="1" text="Room BG 212"/&gt;       &lt;option value="2" text="Staff Common Room"/&gt;     &lt;/select&gt;   &lt;/ui&gt;   &lt;service url="http://&lt;ipaddress&gt;/taskos/get_devices"&gt;     &lt;args&gt;       &lt;arg name="locationid" value="&lt;location"/&gt;     &lt;/args&gt;   &lt;/service&gt; &lt;/task&gt; </pre>	
<pre> &lt;task id="select_device"&gt;   &lt;title&gt;Select device&lt;/title&gt;   &lt;ui&gt;     &lt;textview text="Please select a device:"/&gt;     &lt;select name="device"&gt;       &lt;option value="0" text="TV"/&gt;       &lt;option value="1" text="Lights"/&gt;       &lt;option value="2" text="Coffee Machine"/&gt;     &lt;/select&gt;   &lt;/ui&gt;   &lt;service url="http://localhost:8084/taskos/get_commands"&gt;     &lt;args&gt;       &lt;arg name="deviceid" value="&lt;device"/&gt;     &lt;/args&gt;   &lt;/service&gt; &lt;/task&gt; </pre>	
<pre> &lt;task id="select_command"&gt;   &lt;title&gt;Select command&lt;/title&gt;   &lt;ui&gt;     &lt;textview text="What would you like to do?"/&gt;     &lt;select name="command"&gt;       &lt;option value="0" text="Set TV channel"/&gt;       &lt;option value="1" text="Set TV volume"/&gt;     &lt;/select&gt;   &lt;/ui&gt;   &lt;service url="http://&lt;ipaddress&gt;/taskos/controlTV"&gt;     &lt;args&gt;       &lt;arg name="cmd" value="&lt;command"/&gt;     &lt;/args&gt;   &lt;/service&gt; &lt;/task&gt; </pre>	

Figure 7: Some task specifications and their corresponding user interfaces. The user starts with the “control device” task and finishes with the “set TV channel” task.

that device (e.g., TV). Once the user selects a command (e.g., “Set TV channel” in this example), the runtime engine will invoke the service (e.g., the “controlTV” service in this example) which is specified in the “select command” task. The “controlTV” service returns a “Set TV channel” task which presents a list of channels for the user to select.

Note that, as shown in our schema, a service, a task reference, and a decomposition choice are one of many possible instantiations of a task, i.e., we have a notion of *polymorphic tasks*. Hence a service call can also return a task reference (i.e., an `include` element), a service, or a decomposition choice (i.e., an `If-Then-Otherwise` element).

### 3.3. Task Recommendation

There can be hundreds of possible tasks for a user at a time while within a smart environment. Task recommendation [18] is a mechanism to help users cope with the massive number of tasks available to them. Our system recommends relevant tasks to users based on their context. The recommendation process can be triggered by a change in the context (e.g, a change of location) or an event (e.g., the user points his/her mobile device at an object). In our current implementation, we use location information (e.g., using GPS and Bluetooth technologies) and pointing gesture (e.g., using Cricket[14] and magnetic compass technologies) to trigger the task recommendation. Other object recognition and location tracking technologies such as vision-based object recognition [23], bar/QRcode reader [12], RFID [20], and NFC<sup>15</sup> can also be easily integrated into our system to trigger the task recommendation.

In TASKCOM, each task is pre-associated with one or more objects (we call them “*taskable spaces*” or “*space*” for short). A space can be a university, a library, a room, a car, or a particular appliance. A space may include other sub-spaces creating a *space tree*. A space may have a delegated task server which handles all tasks relevant to that space. Figure 8 represents a space tree called “The University” and the associated tasks.

Formally, a space tree is represented as a pair  $(V, E)$ , where  $V$  is the set of nodes denoting spaces and  $E$  is the set of edges denoting hierarchical relations between the spaces. Denote the set of tasks from space  $v$  by  $T_v$ , if the current space of the user is  $v$  in  $V$  and  $v_1, v_2, \dots, v_n (n > 0)$  are the spaces (including  $v$  and the root space) along the path from the current space to the root, then the set of recommended tasks ( $T^r$ ) is the union of the sets of tasks which are associated with the spaces along that path:

$$T^r = \bigcup_1^n T_{v_i}.$$

The task engine continuously listens for changes of the user’s context and triggers the task recommendation algorithm (see Algorithm 1). Based on the context of the user (e.g., the location where the user is currently located or the object the user is pointing to), the system determines the

---

<sup>15</sup>[http://www.nfc-forum.org/resources/white\\_papers/nfc\\_forum\\_marketing\\_white\\_paper.pdf](http://www.nfc-forum.org/resources/white_papers/nfc_forum_marketing_white_paper.pdf)



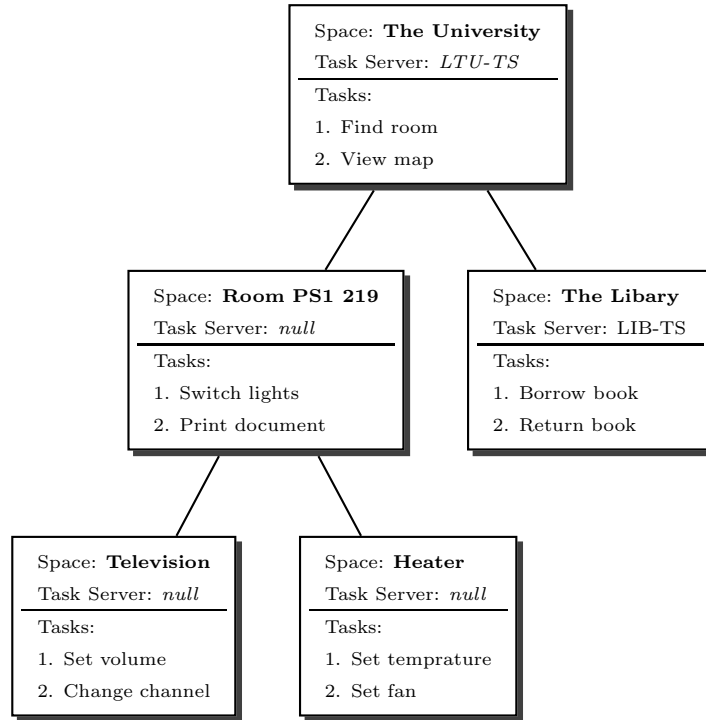


Figure 8: An example of a tree representing the “University” space and associated tasks.

---

**Procedure 1** Task recommendation: It is triggered by changes of the user’s context.

---

**Input:** The user’s context.

**Output:** Recommended tasks.

```

1:  $v \leftarrow \text{context.getCurrentSpace}();$ 
2:  $\mathcal{S} \leftarrow \text{getContainedSpaces}(v);$ 
3:  $T^r \leftarrow \phi;$ 
4: for all  $s \in \mathcal{S}$  do
5:    $T_s \leftarrow \text{getAssociatedTasks}(s);$ 
6:    $T^r \leftarrow T^r \cup T_s;$ 
7: end for
8:  $T^r \leftarrow \text{sort}(T^r);$ 
9: return  $T^r;$ 
  
```

---

<i>Context</i>	<i>Sorted Recommendations of Tasks</i>
Bob enters the University	<ol style="list-style-type: none"> <li>1. Find room</li> <li>2. View map</li> </ol>
Bob enters the Library	<ol style="list-style-type: none"> <li>1. Borrow book</li> <li>2. Return book</li> <li>3. Find room</li> <li>4. View map</li> </ol>
Bob enters the room PS1 219	<ol style="list-style-type: none"> <li>1. Switch lights</li> <li>2. Print document</li> <li>3. Find room</li> <li>4. View map</li> </ol>
Bob points to the television	<ol style="list-style-type: none"> <li>1. Set volume</li> <li>2. Change channel</li> <li>3. Switch lights</li> <li>4. Print document</li> <li>5. Find room</li> <li>6. View map</li> </ol>

Table 1: Different recommendations of tasks in different contexts.

current taskable space and its contained spaces and then queries the task database for the tasks which are associated with those spaces. These tasks are then ordered based on the granularity levels of their associating spaces within the space tree. Tasks associated with a space with finer granularity (i.e., the smaller space enclosing the user) are likely to be placed on the top of the recommendations. For example, let’s assume that Bob’s TASKUI is currently connected to two task servers: “LTU-TS” for the University and “LIB-TS” for the Libary spaces respectively. The space tree is set up as shown in Figure 8. When Bob is moving to a new space or pointing to an object, his current taskable space also changes. As result, the recommended tasks for him are also changed. Table 1 shows different recommended tasks for Bob when his context changes.

Note that, the system also allows the user to turn off the automatic task recommendation, so the user can switch between task spaces manually. This is useful, e.g., when the user wants to control a particular task space remotely. For example, Bob is currently at the University and he wants to turn on the living room’s heater and set it to his preferred temperature before he gets home. Figure 9 shows the setting for the automatic task recommendation and the dialog for switching between task spaces.

### 3.4. Capturing Users’ Intended Tasks

One of the desired features of a task-oriented system is its capability of capturing users’ intended tasks. For this purpose, there have been several approaches such as command phrases [9] and end-user demonstrations [3]. Command phrases allows users to express their intended tasks in

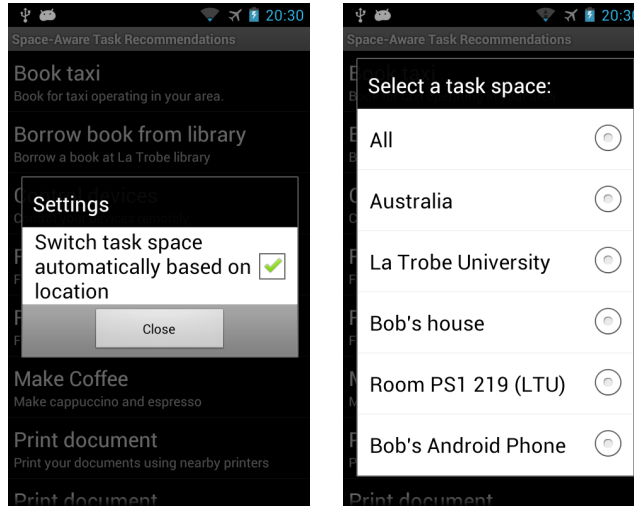


Figure 9: Setting for automatic task recommendation and switching task spaces.

natural language. A command phrase normally consists of a verb, object(s), and adverb(s) (e.g., time and location). For example, “turn off the light in the living room” is a command phrase in which “turn off” is the verb, “light” is the object, and “living room” is the location adverb. The command phrase-based method relies heavily on natural language parsers (e.g., the Berkeley Parser<sup>16</sup>). With the approach using end-user’s demonstrations, the user first trains the system a task they wish to repeat in the future by doing demonstrations of how that task is accomplished. After that, the system observes the user’s actions to recognise the tasks which it has learned previously. This method requires a special environment with sensors (e.g., cameras) so that the user’s actions can be captured. It also heavily relies on pattern recognition techniques.

Our current implementation provides a global instant keyword search method for users to express their intended tasks. The global keyword search allows the users to quickly navigate to a task by providing several keywords of that task. Each task in our system has the data of predefined set of keywords, a title, and a description. The system matches the user’s keywords with the task data and computes the number of matched keywords for each task. The number of matches are used to rank tasks (i.e., the greater number of matches, the higher the rank in the results).

The user searches for a task by either typing, gesturing, or telling keywords. The gesture search lets a user draw the keywords on a touchable screen. Figure 10 shows search screens which are implemented for the Android platform. We are currently using built-in speech recognition APIs provided by the mobile platform. For handwriting recognition, we integrated a tool called Gesture Search [5] into our system.

<sup>16</sup><http://code.google.com/p/berkeleyparser/>

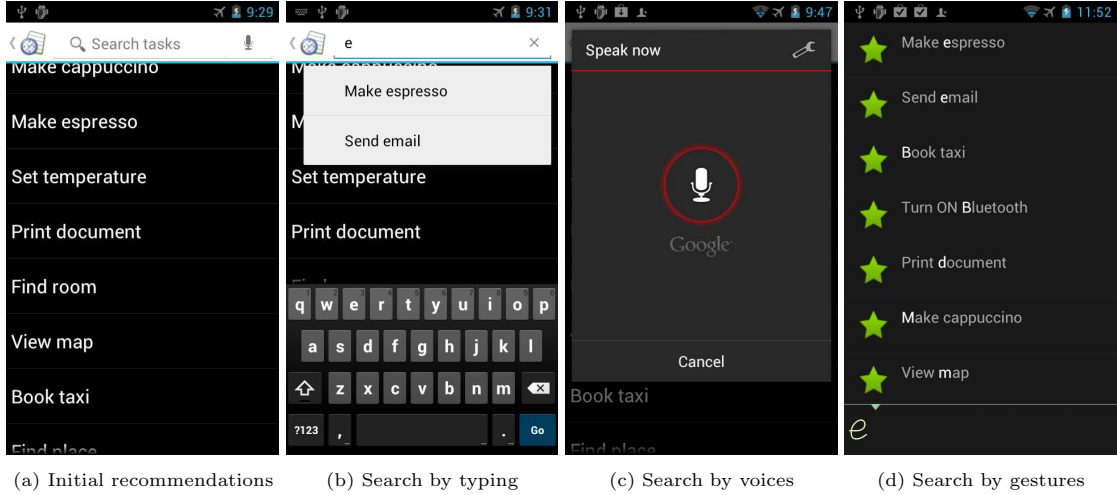


Figure 10: Examples of search screens implemented for the Android platform.

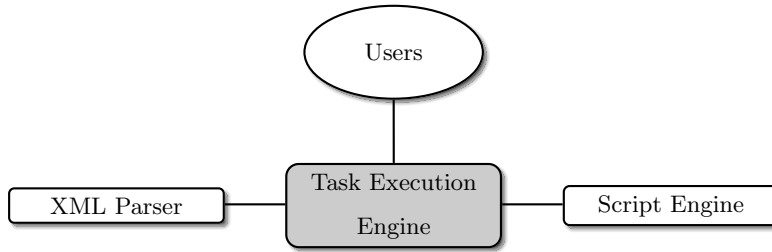


Figure 11: Main components and data model of the task execution engine.

### 3.5. Task Execution

The user accomplishes a task by interacting with the Task Execution Engine (TEE) via an instance of TASKUI which runs on the user’s mobile device. TEE is hosted on a task server which has a repository of task specifications. Figure 11 shows the main components and the data model for TEE. TEE has a database of its users. The XML parser is used to validate and convert XML-based task specifications into task instances for execution. The script engine is used to evaluate expressions (e.g., boolean expressions) at runtime. We currently use the ECMAScript engine.

Figure 12 shows the simplified user model in TEE. Each user first registers with TEE using a unique user name (ID). Every user has a set of recommended tasks, a set of live task instances, a set of shared task instances, and the current active task. The set of recommended tasks is updated on change of the user’s context.

Figure 13 shows the simplified data model of a task instance in TEE. TEE creates an instance of the task once the user starts to execute that task. A task tree represents the decomposition of the task at runtime. The back stack is used to enable the feature that allows the user to go back to a previous sub-task. The variable table is a hash table which stores pairs of  $\langle \text{key}, \text{value} \rangle$  which are returned from invocations of services or the user’s input. These values are passed into expressions or between sub-tasks. The current sub-task of a task instance is a pointer that points to a sub-task (i.e., a node) within the task tree where the user is currently executing.

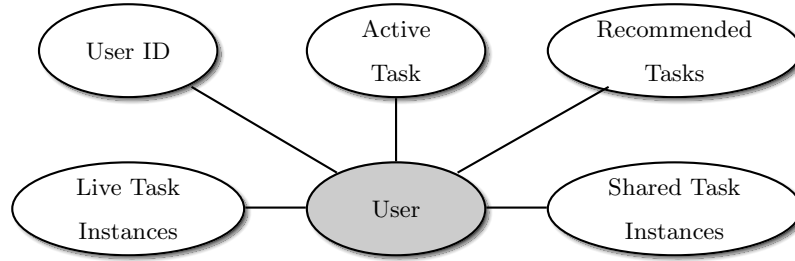


Figure 12: The simplified user model in TEE.

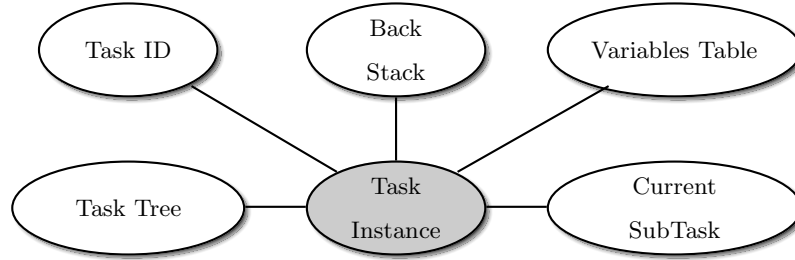


Figure 13: The simplified data model of a task instance in TEE.

As the user switches between tasks or navigates through the sub-tasks of a task instance, the task instances transition between different states in their lifecycle. For example, when the user starts to execute a task for the first time, the user interface of the task comes to the foreground and receives user focus. During this process, TEE calls several procedures to create the task instance and to present its first sub-task for the user to interact with. If the user switches to another task, TEE suspends the current task and moves it into the background (where the task is no longer visible, but the task instance and its state remains intact). Figure 14 illustrates the lifecycle of task instances in TEE.

When the user selects a task to execute or to continue (from either a task recommendation or a task search result), TEE calls the `execute()` method as shown in Procedure 2. If the task has not been started, TEE loads the corresponding task specification, creates an instance for that task (i.e., instantiates the corresponding task tree), and assigns the current sub-task variable to the root task. Finally, TEE resumes the task instance in order to move the current sub-task to the foreground for the user to interact with.

The user interacts with TEE via TASKUI which runs on a mobile device. The user interface for a task instance has several commands such as “Next”, “Back”, “Skip”, and “Share” which allow the user to interact with the task. The user’s input (including the command he/she’s selected) is sent to TEE for processing. Procedures 3, 4, 5, and 6 present the pseudocode, which shows how TEE handles the user’s commands with a task instance. In particular, the `next()` method handles the Next command. It first executes the current sub-task with the arguments which are supplied by the user, then the pushes the current sub-task into the back stack as a mechanism for the Back command. Next if there exist uncompleted sub-tasks, the next uncompleted sub-task is executed to generate the user interface for it, otherwise the root task is seen to be completed and will be

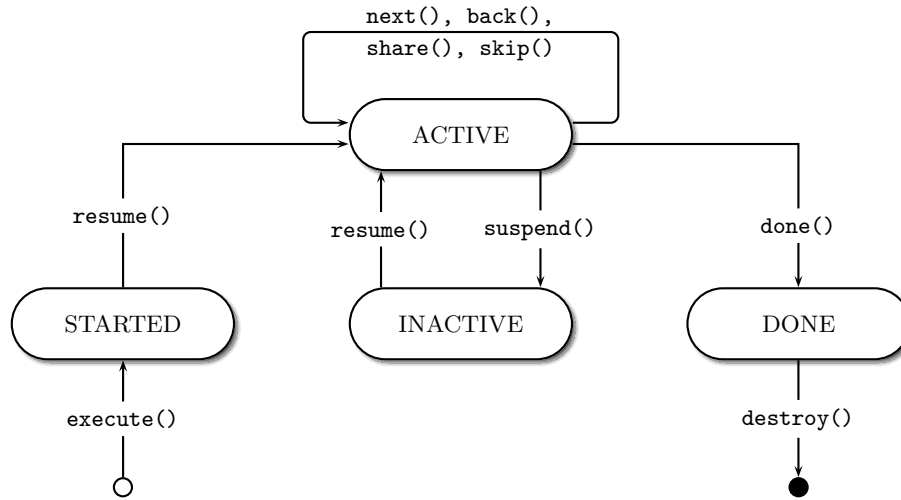


Figure 14: An illustration of the task instance's lifecycle, expressed as a state transition diagram.

---

**Procedure 2** `execute()`: The user starts executing a task given a `taskID`.

---

**Input:** `taskID`.

```

1: if (liveTaskInstances.contains(taskID)) then
2:   if (activeTask.taskID ≠ taskID) then
3:     activeTask.suspend();
4:     activeTask ← liveTaskInstances.get(taskID);
5:   end if
6: else
7:   if (activeTask ≠ NULL) then
8:     activeTask.suspend();
9:   end if
10:  taskTree ← loadTaskSpecification(taskID);
11:  taskInstance ← new TaskInstance(taskTree);
12:  liveTaskInstances ← (liveTaskInstances ∪ {taskInstance});
13:  activeTask ← taskInstance;
14:  taskInstance.currentSubTask ← taskInstance.getRoot();
15:  taskInstance.backStack ← ϕ;
16:  taskInstance.variablesTable ← ϕ;
17:  taskInstance.state ← STARTED;
18: end if
19: activeTask.resume();

```

---

destroyed. The `back()` method handles the Back command. It first checks if the back stack is empty (i.e., there is no previously completed sub-tasks of the current task), it is seen that the user wishes to cancel the current task and will be destroyed. Otherwise, the previous sub-task (which is popped from the back stack) is undone (e.g., simply set its status to “UNDONE”), and re-executed to generate the user interface for it. The `skip()` method handles the Skip command which is available only if the current sub-task is specified “optional”. This method simply sets the status of the current sub-task (and its sub-tasks if they exist) to “DONE”. The `share()` method handles the Share command which is available only if the current sub-task is specified “sharable”. The method gathers the information of the sharee then sends a push notification to the sharee’s mobile device. The notification includes the shared task’s data which allow the task to be resumed on the sharee’s mobile device.

---

**Procedure 3** `next(args)`: The user executes “next” of a given `activeTask`.

---

**Input:** `args`: Values provided by the user.

```

1: currentSubTask.done(args);
2: backStack.push(currentSubTask);
3: currentSubTask ← getNextUnDoneSubTask();
4: if (currentSubTask ≠ NULL) then
5:     currentSubTask.run();           ▷ Generate a message and send it to TASKUI.
6: else
7:     destroy();
8: end if

```

---



---

**Procedure 4** `back()`: The user executes “back” of a given `activeTask`.

---

```

1: if (taskStack.isEmpty()) then
2:     liveTaskInstances ← (liveTaskInstances \ {activeTask});
3:     destroy();
4: else
5:     currentSubTask ← backStack.pop();
6:     currentSubTask.undo();
7:     currentSubTask.run();           ▷ Generate a UI message and send it to TASKUI.
8: end if

```

---

TEE sends TASKUI messages for generating user interfaces of (sub-)tasks. TASKUI uses native user interface elements provided by the mobile platform and custom user interface elements which are provided by third-party applications. For example, to generate a user interface where the user can scan the barcode of his/her ID card using the camera of a mobile device, TEE sends TASKUI a message as shown in Figure 15. In this message, the `UIToken` element is used to keep track of

---

**Procedure 5** skip(): The user executes “skip” of a given activeTask.

---

```
1: currentSubTask.state ← DONE;
2: for all  $t \in$  currentSubTask.getSubtasks() do
3:    $t$ .state ← DONE;
4: end for
5: next(NULL);
```

---

---

**Procedure 6** share(shareeID): The user executes “share” of a given activeTask.

---

**Input:** shareeID: The ID of the user who will get involved in the task.

---

```
1: sharee ← getUser(shareeID);
2: sharee.addSharedTaskInstances(activeTask, sharer);
3: pushNotification(sharee, activeTask, sharer);
```

---

the current user interface on TASKUI. The ui element is copied from the the task specification. Notably, the ui element includes a **barcode** input which is represented on the screen as a text view and a “Tap to scan” button. Tapping on the button will invoke a third-party barcode scanner component which finally returns the scanned barcode back to TASKUI. This example shows how TASKUI can seamlessly integrate third-party services/components for best user experience during task execution.

The **navigationbar** element in the message above is generated dynamically by TEE based on the status of the task. For example, the user can only execute the next sub-task (i.e., the Next button is enabled) if there exists a next sub-task; the user can only skip, share the current sub-task if it is optional, sharable respectively; and the user can only go back to the previous sub-task if there exists a sub-task in a special stack which we call the *back stack* in TASKCOM. The back stack of a task is a mechanism for tracking the sub-tasks which have been executed previously. In our current implementation, the back mechanism is a way for the user to go back and re-execute a

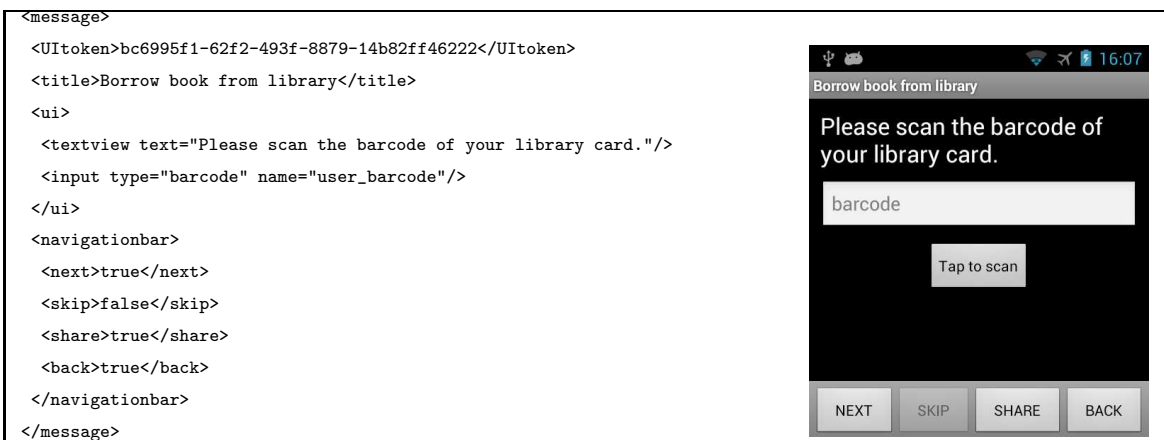


Figure 15: A typical message that TEE sends to TASKUI for generating a user interface for the current sub-task.



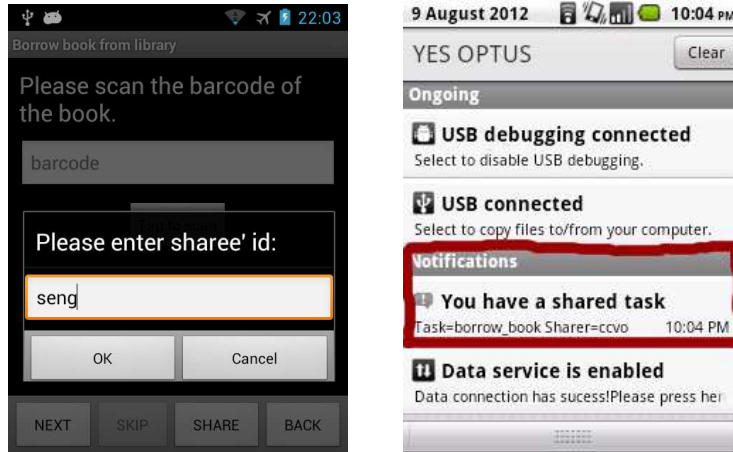


Figure 16: An example of collaboration in borrowing a book remotely between two users; their usernames are “ccvo” and “seng”. The left screen on ccvo’s mobile device shows a dialog asking for a username of the user to collaborate. The right screen on seng’s mobile device shows a notification of the shared task.

previous sub-task. However, it does not undo the impact which has been caused by the execution of that sub-task. To support the undo feature to some extent, the user would need to define the “undo task” which normally reverses the impact or compensates for the impact caused by the execution of the corresponding task.

### 3.6. Resuming Tasks on Different Mobile Devices

Because task instances are completely stored on the task servers and TASKUI only acts as the user interface of the tasks, the user can resume a task instance on any mobile device provided that the mobile device has TASKUI installed. This works like the concept of remote sharing one desktop screen on multiple monitors on the Windows operating system. For example, a user starts a task on a tablet, then for some reason, he/she would like to continue the task on a smartphone.

### 3.7. Collaboration

Our current implementation supports the coordinated mode of collaboration. This mode is used when the sharer wants all participants to share the same view of a (sub-)task and they all can manipulate the task at the same time (e.g., provide inputs, go back, and go next). A (sub-)task is sharable if its “sharable” attribute is specified “true” in the XML task specification.

Figure 16 illustrates a scenario of coordinated collaboration where Bob wants to borrow a book remotely using TASKUI. When he is asked to scan the barcode of the book, he delegates this sub-task to his friend who is currently in the library, so his friend can help him to scan the barcode of the book using a mobile device. To achieve this, Bob clicks on the “Share” button and provides his friend’s identity (e.g., username). His friend will be notified of this delegated sub-task and can help to accomplish it.

#### 4. Discussion

Smart environments are increasingly integrated with complex appliances, providing a huge set of capabilities, so that the end-users wonder which capabilities are available at a point of time, which capabilities they should use, and how to use them for accomplishing their intended tasks. With the task-oriented computing paradigm, like TASKCOM, the system can automatically map high-level user tasks to low-level actions and capabilities (e.g., normally exposed as services) of appliances. That is, the users do not need to be aware of low-level capabilities. Specifically, our goals behind TASKCOM are as follows:

- It helps users quickly discover possible tasks in a smart environment. For example, if the user is currently located in a university campus, with TASKUI on his/her mobile device, the user would quickly discover that there are possible tasks which can be done via the smartphone such as find a route to a particular staff's room, borrow books from the library, and print documents via common printers. If the user points his/her mobile device to a television in a seminar room, he/she would discover that he/she can change the volume and channel remotely using his/her mobile device.
- It allows the users to focus on high-level tasks rather than low-level details of applications and appliances. Indeed, they can accomplish their intended tasks without the knowledge of underlying applications, services, and appliances.
- Resume tasks on different mobile devices and collaborate on shared tasks.

To evaluate the benefits of TASKCOM, let's compare user activities (including mental activities) in accomplishing several tasks using TASKCOM with doing the same tasks without TASKCOM (i.e., via the current application-centric paradigm). For this comparison, we consider the following initial scenario: Bob's smart personal office at his University has computer-controllable lights. Let's assume that the University has a task server which currently provides a set of possible task specifications, one of which allows Bob to control the lights in his office. The University may at anytime update these task specifications (i.e., modify, add, or remove task specifications). We also assume that the University currently provides a mobile application called Smart Office Controller for controlling the lights. Bob has installed this application on his mobile device. Table 2 illustrates this comparison from which we can see that when accomplishing the same tasks, TASKCOM requires less activities than without TASKCOM. Especially, TASKCOM can eliminate many mental activities which are normally require in the current application-centric paradigm.

Our key idea behind the design and implementation of TASKCOM is its use of task specifications to map users's high-level tasks to low-level services and appliances' capabilities. Our specification schema provides the following features:

**Reuse of task specifications.** A new task specification can be created by composing existing

<i>Situation</i>	<i>With TASKCOM</i>	<i>Without TASKCOM</i>
Bob wants to brighten the room. Note that, only the lights are remotely controllable at this point. The window drapes cannot be controlled remotely yet.	<ol style="list-style-type: none"> <li>1. Search for the task (e.g., tell the phrase “brighten room”).</li> <li>2. Select the “Adjust room lighting” task from the search result to start the task. Note that Bob easily recognises this task for his goal because the description of this task includes “brighten the room”.</li> <li>3. Bob is presented a user interface with a slider (seeker) where he can adjust the lights.</li> </ol>	<ol style="list-style-type: none"> <li>1. Think of what appliances should be used for this task: the lights.</li> <li>2. Think of what applications are used to control the lights: Smart Office Controller.</li> <li>3. Think of how the icon of this application looks like in order to map the application to an icon.</li> <li>4. Locate the icon from the application icon list.</li> <li>5. Select the icon to start the application.</li> <li>6. Bob is presented a user interface with a slider where he can adjust the lights.</li> </ol>
Bob’s university has updated the “adjust room lighting” task model which allows him to control the window drapes and the lights at the same time for adjusting the room lighting. The university’s also released a new version of Smart Office Controller for the same purpose. Bob’s task now is to darken the room.	<ol style="list-style-type: none"> <li>1. Search for the task (e.g., tell the phrase “darken room”).</li> <li>2. Select the “Adjust room lighting” task from the search result to start the task.</li> <li>3. Bob is presented a user interface with two sliders where he can adjust the lights and the window drapes.</li> </ol>	<ol style="list-style-type: none"> <li>1. Update Smart Light Controller.</li> <li>2. Learn what’s new in the new version: Bob knows that the application now allows him to control the window drapes as well. These activities are usually needed for every update.</li> </ol> <hr/> <ol style="list-style-type: none"> <li>1. Think of what appliances should be used to achieve the goal of this task: the lights and window drapes.</li> <li>2. Think of what applications are used to control the lights and window drapes: Smart Office Controller.</li> <li>3. Think of how this application’s icon looks like to map the application to an icon.</li> <li>4. Locate the application icon from the application icon list.</li> <li>5. Select the icon to start it.</li> <li>6. Bob is presented a user interface with a slider where he can adjust the lights and window drapes.</li> </ol>
The university’s added a “print document” task model for printing documents via printers across the university. It’s also released a Mobile Printer application for the same purpose. Bob wishes to print a document.	<ol style="list-style-type: none"> <li>1. Search for the task (e.g., tell the phrase “print document”).</li> <li>2. Select the “Print document” task from the search result to start the task.</li> <li>3. Go through several steps to accomplish the printing.</li> </ol>	Simply just do not know that there is an application for this task until someone tells him about it.

Table 2: Comparison user activities in accomplishing several tasks with TASKCOM and without TASKCOM.

task specifications. This is achieved by the use of services and/or task references within task specifications.

**Dynamic task decomposition.** The decomposition of a task can be dynamically added or removed at run-time by the use of services and the back stack. Specifically, an invocation of a service can return a new task specification depending on the provided arguments. The returned task specification will be added to the task decomposition tree as a sub-task by replacing that service. The back stack in TASKCOM allows the user to go back to the previous step which could result in removing the sub-task from the task decomposition tree if that sub-task was added previously as a result of a service invocation.

**Automatic generation of user interfaces for tasks.** Based on task model specifications, TASKUI can generate user interfaces which present instructions for the user to execute the tasks. TASKUI uses native user interface elements and third-party components for representing information to the user and requesting user inputs. By the use of native user interface elements, TASKUI ensures that it does not standardise the look and feel provided by different mobile platforms. And by the use of third-party components, TASKUI hides the concept of applications and invokes the required components (which are provided by the third-party applications) on demand. For example, a map application may have many components, one of which is to compute and show a route on the map. TASKUI will call this component whenever it needs to compute and show a route on the map. Similarly, TASKUI can start a barcode reader application whenever a task requests the user to scan a barcode. This feature relies on our assumption that these components are discoverable and they are allowed to be started from outside their main applications. On the Android platform, this assumption is satisfied.

We have also discussed the application domains where we can apply TASKCOM. We realised that TASKCOM can also be applied in learning activities where teachers ask their students to accomplish some tasks, observe the progress of the students' accomplishment, and provide help remotely. For this purpose, our task specification schema may be customised to allow specifying other aspects of learning tasks such as time and location constraints. For example, a learning task in a biology lab requires the students to accomplish a sequence of sub-tasks which involve the use of available facilities and appliances in the lab. Some of the sub-tasks may be accomplished by directly manipulating the appliances and reporting the results using TASKUI on a mobile device, while other sub-tasks can be accomplished via TASKUI.

The performance of the system depends up several factors; e.g. the network connectivity and the complexity of the task (e.g., the number of sub-tasks and whether the sub-tasks are included directly or by references in a task specification).

## 5. Related Work

There are several limited forms of task-oriented systems in the software industry. On the Windows operating system, when a user inserts a music CD to the CD tray, a dialog is popped up and suggests the user tasks such as “Play CD” or ”Copy Music”. Apple iPhones have an application called Siri<sup>17</sup> which lets users use their voices to send messages, schedule meetings, place phone calls, and more. The most important advantage of Siri is its ability to understand what the user says. In some cases, it can also infer the context that the user is referring to. The fundamental limitations of these task-oriented applications are that they only support a limited number of those predefined tasks which are pre-built into the applications, and that any changes of the tasks require updates of the applications. Also the users are not allowed to personalise the tasks, to define their own tasks, or to add more tasks from different resources.

There are also several systems which are developed by the research community. The Aura system [4] is claimed to support the migration of task instances across different smart environments. In Aura, a task is a composition of virtual services which will be resolved to actual services when the task is instantiated or resumed in a particular environment. To enable task migration, Aura maintains the execution states of tasks globally which is similar to our approach (i.e., TASKCOM also maintains task instances on task servers).

Olympus [15] is a high-level programming model for smart spaces. The main feature of this model is that developers can program applications in terms of abstract entities (e.g., services, applications, devices, physical objects, locations and users) and common operations (e.g., start and stop a component). At runtime, the Olympus framework will resolve these abstract entities into actual entities based on constraints, ontological descriptions of entities, the available resources, and space-level policies. The key elements of the framework are ontologies for hierarchically specifying entities and an algorithm for semantically discovering resources. The authors also claim that their framework can also recover from failures of actions by using alternative resources. While this framework mainly aims at supporting development context-aware applications for smart spaces, it does not address usability concerns of the end-users in smart spaces.

Huddle [10] is one of the approaches which uses task-based user interfaces to address the complexity of consumer electronic devices. Huddle automatically generates task-based user interfaces for multiple appliances based on descriptions of data flows between appliances involved in tasks. The data flows, wiring diagrams, and descriptions of appliances’ functions show how the appliances are connected. However, this system mainly targets multimedia-related tasks which normally have clear descriptions of data flows between media devices. Our system uses data flows between sub-tasks rather than between appliances.

InterPlay [9] is designed to present to users available tasks based on devices’ capabilities, user

---

<sup>17</sup><http://www.apple.com/iphone/features/siri.html>

locations, and preferences. It also allows users to express their tasks in pseudo sentences which consist of a verb, a subject, and target device(s), e.g., “play the Matrix movie on the living room’s theatre”. The system operates based on device descriptions and generic task specifications. Like Huddle, this system does not support composite tasks which involve several sub-tasks. It does not target the manipulation of task execution (e.g., suspension, resumption, and collaboration) but rather only concentrates on ease of use.

Roadie [6] aims to provide users with context-sensitive advice and mixed-initiative assistance for executing tasks on consumer devices. Roadie uses a commonsense knowledge base to automatically generate task specifications and to provide assistance with executing multi-step procedures and debugging help when things go wrong. Because of its dependency on the commonsense knowledge base, Roadie may not be able to support uncommon tasks, such as those related to a private environment or to a new class of devices. Roadie only supports single device tasks while TASKCOM supports multiple device tasks.

ANSI/CEA-2018 [16] is a standard language for describing tasks supported directly on consumer electronics. ANSI/CEA-2018 task specifications are interpretable. The bindings between the task specification and the controlled device(s) are implemented using JavaScript. We believe that the applicability of our framework is broader than this standard because ANSI/CEA-2018 targets direct controlling of consumer electronics individually while our framework can be applied for a network of devices (i.e., entire environments), composite tasks, collaborated tasks, and remote controlling tasks via mobile devices. However, ANSI/CEA-2018 has served as an inspiration for our work.

TCE [8] aims to assist users in completing tasks in smart environments. In TCE, services are semantically described which enables semantic service discovery. The system presents available basic services to users and allows them to either execute a basic service or manually compose tasks using multiple basic services. However, to compose a desired task, the user has to understand the services.

## 6. Conclusion

In this paper, we have described the shortcomings of the current application-centric paradigm when applied to smart environments. We have presented our motivation, the design and implementation of TASKCOM, our task-oriented computing framework for smart environments. TASKCOM allows users to operate appliances in a smart environment at task level, rather than at the level of applications and appliances. We have shown that when accomplishing the same tasks in smart environments, TASKCOM can eliminate many mental activities which are normally required by the application-centric paradigm. As smart environments become more complex, we believe there is a need for a fundamental paradigm shift from “how to do” a task to “what to do”. The framework described in this paper is our attempt in supporting such a shift, at least from the

end-user’s perspective. While we do not think the application-centric notion will be replaced, the task-oriented paradigm can be built over existing applications. Our prototype implementation has demonstrated the feasibility and advantages of the task-oriented computing paradigm. We also contend that while the task computing notion, as reviewed earlier, has been considered in other work, our work is novel in terms of *(i)* comprehensiveness: TASKCOM supports the full range of the task-oriented computing ideal (e.g., task modelling and specification, recommendation, execution, collaboration, and organisation into hierarchical spaces), *(ii)* broad applicability: we argue for the task-oriented computing paradigm for smart environments, including rooms, individual appliances, and personal virtual spaces and *(iii)* extensible open source framework which includes Java-based task server, TASKUI for Android, and an RNC-based task specification language.

There are at least four aspects that we can improve TASKCOM:

- Contextualising the execution of a task: When a task is executed or resumed in a new environment, all service bindings should be resolved based on available appliances and services in the environment. However, there are some cases which the user may not want to contextualise their task. For example, the user wants to remotely turn off the heater at home which he/she forgot to turn off when he/she left home. If this task is contextualised, it will turn off the heater in the user’s office instead of the one at home. But in another scenario where the user suspends the task of viewing television at home and resumes it in his/her office, the task contextualisation will ensure that the office’s television (not the home’s one) is turned on and set to the same settings.
- Task specification creation by end-user demonstration or online “ehow” data (e.g., ehow.com or 43Things.com): The user should be able to generate a new task specification by demonstrating a sequence of actions. The system can monitor a user’s interactions with appliances in an environment, asks the user to label it as a task for future use.
- Speech-based user interface for TASKUI: Our current implementation of TASKCOM supports speech input only for task search while the user interaction for task execution is graphical and text-based. To improve user experience with TASKCOM, we’d like to add speech-based interaction for task execution. For example, instead of showing a graphical user interface where the user can move a slider to set the volume for a television, TASKUI should accept the user’s speech telling it like “seventy”, “up up”, “down down”, or “mute”.
- Trouble-shooting and explanations: The system should provide trouble-shooting and explanations when problems occur while executing a task. This feature could be added to TASKCOM using, e.g., Roadie [6]. Roadie has an AI partial-order planner based on a commonsense knowledge base to provide mixed-initiative assistance and debugging help when things go wrong.

In summary, we believe that TASKCOM is a further step towards coping with the increasing complexity of smart environments. Our reference implementation of TASKCOM and examples of task model specifications are available by contacting the authors or accessible at <https://github.com/ccvo/taskcom/>.

## References

- [1] J.E. Bardram, Activity-based computing: Support for mobility and collaboration in ubiquitous computing, *Personal and Ubiquitous Computing* 9 (2005) 312–322.
- [2] O. Beaudoux, M. Beaudouin-Lafon, DPI: A conceptual model based on documents and interaction instruments, in: *People and Computer XV - Interaction without frontier (Joint proceedings of HCI 2001 and IHM 2001)*, Springer Verlag, 2001, pp. 247–263.
- [3] A.K. Dey, R. Hamid, C. Beckmann, I. Li, D. Hsu, a CAPpella: programming by demonstration of context-aware applications, in: *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, 2004, pp. 33–40.
- [4] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, Project Aura: toward distraction-free pervasive computing, *Pervasive Computing, IEEE* 1 (2002) 22–31.
- [5] Y. Li, Gesture search: Random access to smartphone content, *Pervasive Computing, IEEE* 11 (2012) 10–13.
- [6] H. Lieberman, J. Espinosa, A goal-oriented interface to consumer electronics using planning and commonsense reasoning, *Know.-Based Syst.* 20 (2007) 592–606.
- [7] S. Loke, Building taskable spaces over ubiquitous services, *IEEE Pervasive Computing* 8 (2009) 72–78.
- [8] R. Masuoka, B. Parsia, Y. Labrou, Task computing—the semantic web meets pervasive computing, in: *Proceedings of the Second International Semantic Web Conference, Florida, USA*, pp. 866–881.
- [9] A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, K. Yi, InterPlay: A middleware for seamless device integration and task orchestration in a networked home, in: *PerCom*, pp. 298–307.
- [10] J. Nichols, B. Rothrock, D.H. Chau, B.A. Myers, Huddle: automatically generating interfaces for systems of multiple connected appliances, in: *UIST '06, ACM, NY, USA, 2006*, pp. 279–288.
- [11] D. Norman, *The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution*, MIT Press, 1998.



- [12] E. Ohbuchi, H. Hanaizumi, L. Hock, Barcode readers using the camera device in mobile phones, in: International Conference on Cyberworlds, pp. 260–265.
- [13] F. Paterno, Model-based design of interactive applications, *Intelligence* 11 (2000) 26–38.
- [14] N.B. Priyantha, A. Chakraborty, H. Balakrishnan, The cricket location-support system, in: *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, ACM, New York, NY, USA, 2000, pp. 32–43.
- [15] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R.H.Campbell, D. Mickunas, Olympus: A high-level programming model for pervasive computing environments, in: *PerCom, Kauai Island, Hawaii*, pp. 7–16.
- [16] C. Rich, Building task-based user interfaces with ANSI/CEA-2018, *Computer* 42 (2009) 20–27.
- [17] J. Spohrer, M. Stein, User experience in the pervasive computing age, *MultiMedia, IEEE* 7 (2000) 12–17.
- [18] C.C. Vo, S.W. Loke, T. Torabi, T. Nguyen, TASKREC: a task-based user interface for smart spaces, in: *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, MoMM '11*, ACM, New York, NY, USA, 2011, pp. 223–226.
- [19] Z. Wang, D. Garlan, Task-Driven Computing, Technical Report, School of Computer Science, Carnegie Mellon University, 2000.
- [20] R. Want, An introduction to rfid technology, *IEEE Pervasive Computing* 5 (2006) 25–33.
- [21] M. Weiser, The computer for the 21<sup>st</sup> century, *Scientific American* 3 (1991) 94–104.
- [22] M. Weiser, R. Gold, J. Brown, The origins of ubiquitous computing research at parc in the late 1980s, *IBM systems journal* 38 (1999) 693–696.
- [23] T. Yeh, K. Grauman, K. Tollmar, T. Darrell, A picture is worth a thousand keywords: image-based object search on a mobile platform, in: *CHI '05 extended abstracts on Human factors in computing systems*, ACM, New York, NY, USA, 2005, pp. 2025–2028.