Task-Oriented Systems for Interaction with Ubiquitous Computing Environments

Chuong C. Vo, Torab Torabi, and Seng W. Loke

Department of Computer Science, La Trobe University, VIC 3086, Australia
{c.vo,t.torabi,s.loke}@latrobe.edu.au

Abstract. We present the design and implementation of a task-oriented system, that dynamically binds user tasks to available devices and services within a ubiquitous environment. The system provides users with a task-based interface for interacting with the environment. This interface is re-oriented around user tasks, rather than particular functions of individual devices or services. The system has two main feature: (1) context-aware task recommendation and (2) task guidance, execution, and management. The fundamental input into the system is task descriptions, XML specifications that describe how tasks are executed. The abstraction of these descriptions allows tasks to be executed in diversely ubiquitous computing environments.

Key words: Task-Based Interface, Smart Space, Ubiquitous Computing

1 Introduction

A ubiquitous computing environment consists of a rich set of casually accessible, and often invisible devices and software services [1]. They are expected to seamlessly integrate and cooperate in support of human tasks. However, it is often difficult for users to use such the environment to accomplish their tasks. According to a recent study [2], half of all reportedly malfunctioning devices returned to stores are in full working order; customers just couldn't figure out how to operate them. In another user study [3], some participants saw technologies as something excessive, sometimes useless and invasive while others (particularly, mature adults with non-technical backgrounds) only used them to perform basic tasks. A study on usability of smart conference rooms [4] also found that the users tended to rely on experts (wizards) for technical operations. This phenomenon is often referred to as the *usability crisis* in digital products including ubiquitous environments. There are at least two factors that lead to this crisis: *overload of functions* and *complexity of user interfaces* [5].

As an environment is increasingly embedded with computers and devices, together with the devices we carry with us, the range of available tasks will explode. This explosion would overwhelm users, hindering them from recognising and performing available tasks. Since devices and services are distributed and sometimes invisibly vanished into the environment's physical infrastructure, many of which users may not even be aware of [6]. Moreover, to accomplish a task, a user must determine how to split it into sub-tasks for each device or service and then find particular functions of each device or service to accomplish the sub-tasks [7]. The explosion of functions makes this work difficult for the user, especially in unfamiliar environments.

The complexity of devices' user interfaces essentially comes from three factors. First, since devices have no knowledge of what task a user intend to accomplish, they often provide the user with all functions which are often in form of buttons and menus. The user needs to switch between the devices' interfaces and to appropriately combine buttons and menus for accomplishing the task. This requires the user to understand the task decomposition and the devices' capabilities. Second, since there is little or no goal-action consistency [8] between the interfaces, the user must learn to operate them separately (i.e., there is no "transfer of training"). Third, it is easy to add features to devices, hence resulting to complexity that has exceeded the capacity of user interfaces [5]. Therefore, controls get overloaded, leading to heavily-moded interfaces, push-and-hold buttons, long and deep menus [9].

Our solution to these problems is based on the concept of *task-driven computing* [10, 11, 12]. We have developed a task-oriented system called TaskOS, acting as a bridge that links tasks and available functions of a ubiquitous environment and that shields users from variations in devices' and services' availability. TaskOS provides users with a task-based user interface, interfaces are organised according to users' tasks. In particular, TaskOS has the following features:

- *Context-aware task recommendation:* recommend users for tasks based on their context and available devices and services in the current environment.
- *Proactive task guidance:* provide users with instructions to complete a task.
- *Task execution management:* manage task execution, multi-tasking, resource conflicts, and adaptation to variations in availability of devices and services.

The remaining of this paper is organised as follows. Section 2 presents an overview of how TaskOS works. Section 3 presents a generic conceptual architecture for TaskOS. Section 4 presents an implementation and a performance evaluation of TaskOS. Section 5 presents related work. The conclusion and future work are given in Section 6.

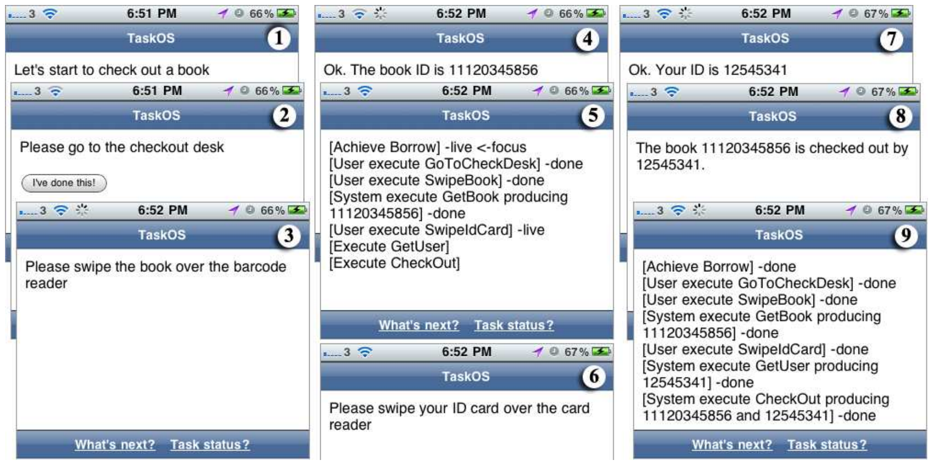
2 Envisioned Operation of TaskOS

Developers create or modify a task in a *task description*, an XML specification describing how the task is executed. They register the description with a *task repository*, where it becomes available as a recommendation to users according to their context. Fig. 1 shows a description of the "*checkout a book*" task.

Automatically or when asked by a user (e.g., let's call Bob), TaskOS recommends him for relevant tasks possible in the current environment based on his context. He can also issue queries to search for an intended task. Once Bob selects a task to perform, TaskOS executes its description and guides him to

```
<taskModel about="models:BookCheckOut"
  xmlns="http://ce.org/cea-2018"
  xmlns:dc="http://purl.org/dc/elements/1.1">

  <task id="Borrow">
    <dc:title>check out a book</dc:title>
    <subtasks id="borrowing">
      <step name="go" task="GoToCheckOutDesk"/>
      <step name="identifybook" task="IdentifyBook"/>
      <step name="identifyuser" task="IdentifyUser"/>
      <step name="checkout" task="CheckOut"/>
      <binding slot="$checkout.book" value="$identifybook.book"/>
      <binding slot="$checkout.user" value="$identifyuser.user"/>
    </subtasks>
  </task>
  <task id="GoToCheckOutDesk">
    <dc:title>go to the checkout desk</dc:title>
  </task>

  <task id="IdentifyBook">
    <dc:title>identify the book</dc:title>
    <output name="book" type="String"/>
    <subtasks id="IdentifyingBook">
      <step name="scanbook" task="SwipeBook"/>
      <step name="getbook" task="GetBook"/>
      <binding slot="$scanbook.external" value="true"/>
      <binding slot="$this.book" value="$getbook.book"/>
    </subtasks>
  </task>

  <task id="IdentifyUser">
    <dc:title>identify the borrower</dc:title>
    <output name="user" type="String"/>
    <subtasks id="IdentifyingUser">
      <step name="scanidcard" task="SwipeIdCard"/>
      <step name="getuser" task="GetUser"/>
      <binding slot="$scanidcard.external" value="true"/>
      <binding slot="$this.user" value="$getuser.user"/>
    </subtasks>
  </task>
  <task id="SwipeBook">
    <dc:title>swipe the book over the barcode reader</dc:title>
    <script>Packages.TaskOS.callService("scanbookid");</script>
  </task>

  <task id="GetBook">
    <output name="book" type="String"/>
    <script>
      $this.book = Packages.Event.getValue();
      Packages.TaskOS.print("The book ID is " + $this.book);
    </script>
  </task>

  <task id="SwipeIdCard">
    <dc:title>swipe your ID card over the card reader
    </dc:title>
    <script>
      Packages.TaskOS.callService("scancardid");
    </script>
  </task>

  <task id="GetUser">
    <output name="user" type="String"/>
    <script>
      $this.user = Packages.Event.getValue();
      Packages.TaskOS.print("Your ID is " + $this.user);
    </script>
  </task>

  <task id="CheckOut">
    <input name="book" type="String"/>
    <input name="user" type="String"/>
    <script>
      Packages.TaskOS.callService("checkoutbook " +
        $this.book + " " + $this.user);
      Packages.TaskOS.print("The book " + $this.book +
        " is checked out by " + $this.user + ".");
    </script>
  </task>
</taskModel>
```

Fig. 1. Description for the "*checkout a book*" task written in CE-TASK [13].



1: Let's start to check out a book

2: Please go to the checkout desk

3: Please swipe the book over the barcode reader

4: Ok. The book ID is 11120345856

5:
[Achieve Borrow] -live <-focus
[User execute GoToCheckDesk] -done
[User execute SwipeBook] -done
[System execute GetBook producing 11120345856] -done
[User execute SwipeIdCard] -live
[Execute GetUser]
[Execute CheckOut]

6: Please swipe your ID card over the card reader

7: Ok. Your ID is 12545341

8: The book 11120345856 is checked out by 12545341.

9:
[Achieve Borrow] -done
[User execute GoToCheckDesk] -done
[User execute SwipeBook] -done
[System execute GetBook producing 11120345856] -done
[User execute SwipeIdCard] -done
[System execute GetUser producing 12545341] -done
[System execute CheckOut producing 11120345856 and 12545341] -done

Fig. 2. Task-based interface for executing the "*checkout a book*" task.

complete it via a task-based user interface. Fig. 2 shows a number of screens TaskOS presents to him for completing the "*checkout a book*" task.

Bob can perform multi-tasks concurrently. When he comes back to an unfinished task, TaskOS presents its current step to him, so he can continue it. He can suspend the task interface on one interaction device and resume it on another device. He can query status of a task to see what steps have or haven't been done. For example, the screens 5 and 9 in Fig. 2 show two statuses of a task. The user can also request TaskOS to cancel, undo, or redo an ongoing task.

3 Generic Conceptual Architecture

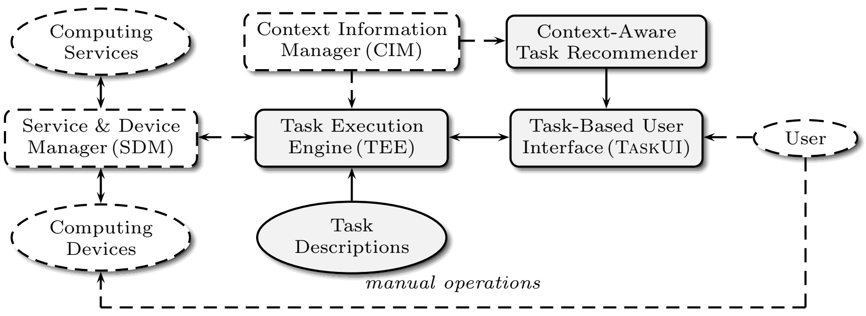

Computing Services
Context Information Manager (CIM)
Context-Aware Task Recommender
Service & Device Manager (SDM)
Task Execution Engine (TEE)
Task-Based User Interface (TaskUI)
User
Computing Devices
Task Descriptions
manual operations

Fig. 3. Generic conceptual architecture for TaskOS. The components with grey-fill and solid border are our focus in this paper.

Fig. 3 illustrates an architecture for TaskOS. The *Service & Device Manager* (SDM) manages devices and services. The *Context Information Manager* (CIM) manages context information and answers context queries to other components. This paper focuses on the *Task Execution Engine* (TEE), *Context-Aware Task Recommender* (TaskRec), *Task-Based User Interface* (TaskUI) components and *task descriptions*.

TaskRec [14] suggests to users possible and relevant tasks based on their context and environment capability. TaskUI is an interface for users to interact with TaskOS. TEE loads, verifies, and executes task descriptions. While executing a task, if TEE needs input from users or wants to present information or instructions them, it sends an interface request to TaskUI. How TaskUI presents this request depends on the current interface modality; it may be visual, spoken, tactile, or multimodal. If the users perform manual operations on devices, TaskUI will simply present them necessary instructions.

A task description specifies steps involved in completing a task. It includes *user steps* performed by humans and *system steps* performed by machines. A *primitive step* is a manual operation on a device or an abstract call to a service or device's function. To execute a call, TEE passes it to SDM for reasoning about an available service or device's function. Hence, developers can abstractly specify calls in a task description; so they can be executed in diverse environments where the availability of devices and services is often unforeseen by the developers.

4 Implementation

This section presents how we describe tasks, how we implement TaskRec that recommends tasks based on user context, and how we implement TEE and TaskUI that executes tasks and provides task guidance.

4.1 Task Description

To describe tasks, we use CE-TASK [13]. CE-TASK allows us to describe high-level, multi-device tasks abstractly from networking technologies and interaction modality. Since it is a mark-up language, CE-TASK descriptions are human-readable and machine-interpretable. Its key expressive features include tasks, input and output parameters, preconditions and postconditions, grounding, task decomposition, temporal order, data flow, and applicability conditions. Fig. 1 shows a CE-TASK description for the task "*checkout a book*".

4.2 Task Recommendation

To recommend tasks for users, we implement *place-based* and *pointing-based* task recommender called TaskRec. We model a place as a hierarchy of sub-places. The lowest level in the place hierarchy is device's zone. Then we associate a task with one or more places. TaskRec keeps inquiring CIM, that monitors users' locations, to get a user's current place for *every second*. Then it recommends tasks which are associated with that place. In our experiment, it took no longer *two seconds* for a user to get task recommendations once s/he changed the place. This latency includes the network latency for sending location information to CIM, the time for updating location at CIM, the one-second interval for each place enquiry at TaskRec, and the time for generating recommendations and sending them to the user device. In the following, we present how to estimate users's current place and pointing directions.

To identify outdoor places, we use the global positioning system. We represent a location of an outdoor place by two attributes: a *geo-coordinate* (latitude and longitude) representing the place's centre and a *radius* representing the place's area. So, if the distance between a user's and a place's geo-coordinates is less than its radius, s/he is seen located in that place, hence s/he is also seen located in its ancestors in the place hierarchy. The user can switch between these places to get different task recommendations.

To estimate outdoor pointing direction, we use compass heading. For each place (called i) in the user's surrounding, CIM computes an angle (β_i) formed by three points: north pole (P_0), user's current geo-coordinate (P_u), and place's geo-coordinate (P_i) using Formula 1. It then compares β_i with the current compass heading (α) to get a difference (δ_i). The smaller δ_i the more likely the user is pointing at the place i. Therefore, if the smallest difference δ_j found is less than an angle threshold, CIM infers that the user points at the place j.

$$\begin{aligned}\beta_i &= \arccos\left(\frac{\overrightarrow{P_uP_0}\cdot\overrightarrow{P_uP_i}}{|\overrightarrow{P_uP_0}|\times|\overrightarrow{P_uP_i}|}\right)\\ &= \arccos\left(\frac{(x_0-x_u)(x_i-x_u)+(y_0-y_u)(y_i-y_u)}{\sqrt{(x_0-x_u)^2+(y_0-y_u)^2}\times\sqrt{(x_i-x_u)^2+(y_i-y_u)^2}}\right), \qquad (1)\end{aligned}$$

where (x_u, y_u), (x_i, y_i), and (x_0, y_0) represent the latitude and longitude of P_u, P_i, and P_0 respectively. In our implementation, we selected P_0 at $(90, y_u)$.

To identify indoor places, we use the Bluetooth and RFID technologies. In our experiment, each user registers with TaskOS for one or more Bluetooth addresses and each place is associated with a Bluetooth scanner. The scanner keeps enquiring for Bluetooth devices. Once it discovers a Bluetooth device, it sends to CIM a three-tuple of ⟨*place'id*, *discovered Bluetooth address*, *time-stamp*⟩ allowing CIM to infer a user's current place. However, if a user is in an overlapped coverage of two or more scanners, CIM changes the user's place as frequently as it receives a report from one of these scanners; causing the task list to be unstable. To address this problem, CIM uses a time threshold for which if a user's current place keeps changing within this threshold, CIM will use their nearest ancestor in the place hierarchy as the current place. To avoid these overlapping cases, we use short-range RFID technologies. Accordingly, a place is associated with a 10cm-range RFID reader. A user swipes a passive RFID tag over the reader to update her/his current place.

To estimate indoor pointing direction, we use the Cricket system [15]. Cricket consists of beacons and listeners. It uses Time-Difference-Of-Arrival between radio frequency and ultrasound to estimate a distance between a beacon and a listener (the accuracy is between 1cm and 3cm). A beacon is worn by a user or fixed to the user's personal device. A device is attached with a listener (devices are places at the lowest level in our place hierarchy). The listener keeps scanning for beacons within its range. Once it discovers a beacon, it sends CIM a four-tuple of ⟨*device'id*, *discovered beacon'id*, *distance*, *time-stamp*⟩. CIM keeps comparing fresh distances (within last *two-seconds* in our experiment) between the same beacon and the listeners, that report these distances to CIM, to infer the device currently nearest to the user. The tasks which are associated with this device are recommended to the user.

4.3 Task Execution & Task-Based User Interface

We implement TEE based on a reference implementation of a task engine in [5]. TEE can listen to CIM for external events and react accordingly. For example, the "*checkout a book*" task has three user sub-tasks: "*go to the checkout desk*", "*scan the book'id*", and "*scan the ID-card*". For the first sub-task, assume that the system cannot recognise the user is already at the checkout desk, it needs her/him to confirm once s/he has done this. So, it asks TaskUI to present her/him a button "I've done this" (see Step 2 in Fig. 2). For the second and third sub-tasks, CIM notifies TEE once the user has done each of them. There is no button "I've done this" on TaskUI as shown in Steps 3 and 6 in Fig. 2.

By introducing SDM in TaskOS, a service call in a task description is abstractly defined as `TaskOS.callService('<service name and arguments>')`. TaskOS will pass the call to SDM for reasoning about a specific service.

We implement TaskUI as a web-page using Ajax technologies [16]. An advantage of using web-based user interface is the portability of interface from one computer to another computer without a need of installing the system on them. Ajax allows us to hide communication between TaskUI and other components of TaskOS from the user, and to update TaskUI autonomously.

5 Related Work

InterPlay [17] allows users to express their tasks via a pseudo-English interface, then the system does the rest to achieve them. But the users must learn how to express their tasks properly. Similarly, a task execution engine [18] can discover and bind the best available resources for executing a task. However, these systems do not recommend users possible tasks or provide the users task guidance.

Huddle [7] automatically generates interfaces for tasks involving multiple connected devices, yet it only supports multimedia tasks that rely on data flows.

A Task Computing Environment (TCE) [11] assists users in completing tasks. It represents a task as a composition of web services. Since like workflows, web services and compositions thereof are autonomous, TCE only supports autonomous, one-step, or batch tasks[1], such as "*exchange e-business card*". In particular, the task of exchanging e-business cards is a batch execution of two services: one returning someone's contact, the other adding this contact into someone else's contact list. TCE does not support multi-step, interactive tasks such as '*borrow a book*'. To complete this kind of tasks, users must interact with several devices and/or with an interactive user interface to provide inputs (e.g., user identity and book call number) and to get instructions for completing tasks, e.g., where the book is located and how to get there.

The Aura system [19] supports the migration of tasks between environments. Since Aura's task description language is designed only for capturing the status of tasks, it cannot describe envisioned tasks, which will be executed in the future.

6 Conclusion

This paper has presented the concept, design, and implementation of a task-oriented system called TaskOS for ubiquitous computing environments. TaskOS provides users with a task-based user interface called TaskUI. The system is able to recommend users with possible tasks based on users' location, surrounding devices, and users' pointing gestures. It can execute task descriptions and guide users through the task executions.

The future work includes developing a graphical tool for authoring task descriptions, implementing other features of TaskOS such as undo, redo, task-recording- replaying, why menus, and adaptation to variations in resource availability, developing algorithms and techniques for efficient and effective storing,

[1] A batch task requires its inputs to be given before being executed.

retrieving task descriptions, extending CE-TASK to support recognising possible tasks in an environment, and mining task instructions on the internet.

References

1. M. Weiser. The computer for the 21^{st} century. *Sci. American*, 3(265):94–104, 1991.
2. E. Den Ouden. *Developments of a Design Analysis Model for Consumer Complaints: revealing a new class of quality failures.* PhD thesis, Technische Universiteit Eindhoven, 2006.
3. M. C. Brugnoli, J. Hamard, and E. Rukzio. User expectations for simple mobile ubiquitous computing environments. In *Proceedings of the Second IEEE International Workshop on Mobile Commerce and Services*, pages 2–10, 2005.
4. G. Golovchinsky, P. Qvarfordt, B. van Melle, S. Carter, and T. Dunnigan. DICE: Designing conference rooms for usability. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 1015–1024, 2009.
5. C. Rich. Building task-based user interfaces with ANSI/CEA-2018. *Computer*, 42(8):20–27, 2009.
6. S. A. N. Shafer, B. Brumitt, and J. J. Cadiz. Interaction issues in context-aware intelligent environments. *Hum.-Comput. Interact.*, 16(2):363–378, 2001.
7. J. Nichols, B. Rothrock, D.H. Chau, and B.A. Myers. Huddle: Automatically generating interfaces for systems of multiple connected appliances. In *Proceedings of the Symposium on User Interface Software and Technology*, pages 279–288, 2006.
8. A Monk. Noddy's guide to consistency. *Interfaces*, 45:4–7, 2000.
9. H. Lieberman and J. Espinosa. A goal-oriented interface to consumer electronics using planning and commonsense reasoning. *Know.-Based Syst.*, 20(6):592–606, 2007.
10. Z. Wang and D. Garlan. Task-driven computing. Technical report, School of Computer Science, Carnegie Mellon University, 2000.
11. R. Masuoka, B. Parsia, and Y. Labrou. Task computing—the semantic web meets pervasive computing. In *Proceedings of the Second International Semantic Web Conference*, pages 866–881, Florida, USA, 2003.
12. S.W. Loke. Building taskable spaces over ubiquitous services. *IEEE Pervasive Computing*, 8(4):72–78, 2009.
13. Consumer Electronics Assoc. Task model description (CE Task 1.0), ANSI/CEA-2018, Mar. 2008.
14. C.C. Vo, T. Torabi, and S.W. Loke. Towards context-aware task recommendation. In *International Conference on Pervasive Computing*, pages 289–292, 2009.
15. N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proceedings of International Conference on Mobile Computing and Networking*, pages 32–43, 2000.
16. J. Garrett. Ajax: A new approach to web applications. Online, Feb. 2005.
17. A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K.H. Yi. InterPlay: A middleware for seamless device integration and task orchestration in a networked home. In *Proceedings of International Conference on Pervasive Computing and Communications*, pages 298–307, 2006.
18. A Ranganathan. *A Task Execution Framework for Autonomic Ubiquitous Computing.* PhD thesis, University of Illinois at Urbana-Champaign, 2005.
19. D. Garlan, D.P. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.