A Survey of Task Representations and Their Applicability to Smart Environments

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

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

c.vo@latrobe.edu.au, t.torabi@latrobe.edu.au, s.loke@latrobe.edu.au

Abstract

A smart environment includes a rich set of computers, digital appliances, sensors, actuators, and Web services. When a smart environment is added with more devices and features, its complexity of use will increase. Task-driven computing, which allows users to interact with smart environments in terms of tasks instead of separate applications on different devices, aims to reduce this complexity. Existing task-driven computing systems often develop their own language or adopt an existing language for representing user tasks which are supported by the systems.

A number of languages for representing user tasks have been developed in different application domains. In this paper, we present our review on the applicability and extensibility of the major languages for representing user tasks in smart environments and in other domains. Our review is based on a set of desirable language properties. We propose these properties based on our analysis of existing task-driven computing systems and of our envisaged generic task-oriented computing system for future smart environments. We propose a categorisation of the languages and then describe representative languages from each family. A comparison among the languages is also given. The survey shows the large extent of common features and key differences among the languages in different domains. It also shows that while the existing languages have been developed for different domains and meet a majority of the desirable properties, there is no language that completely satisfies the proposed set of the desirable properties for representing tasks in smart environments. In this paper, we propose common operators, constructs, and task attributes that a new task representation language for smart environments would provide.

1 Introduction

Mark Weiser [1] described a *smart environment* (or a *smart space* [2]) as "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in everyday objects of our lives, and connected through a continuous network". On one hand, adding smart technologies into a smart environment will obviously enable more new tasks (*i.e.*, more tasks become feasible due to adding smart technologies), provide more ways to achieve the same task, and simplify the accomplishment of existing tasks [3]. On the other hand, when a smart environment is embedded with hundreds of services and digital devices, users often spend more time and effort in understanding and configuring the entire environment including devices [4]. Many people find new technologies (*e.g.*, new digital devices with their added features plus the features enabled by different combinations of these devices) too complicated to

operate [5]. It is even much more complicated for users to combine available devices and services to achieve their high-level goals [6].

Task-driven computing [7, 8] aims to reduce this complexity as it shifts users' focus to what they want to do (*i.e.*, focusing on the tasks at hand) rather than on the means for accomplishing those tasks. Let's consider the following example of how task-driven computing can support users in achieving their goals within a smart environment.

Assume that a user is in a public library and wants to contact a friend to discuss about a new book. To accomplish this task in the current practice (*i.e.*, without the support of task-driven computing), she/he will need to decide whether she/he should make a video call, send an MMS message, send an email, or post a message on the friend's Facebook wall. This process of decision making requires her/him to process a large amount of information about the current context (*e.g.*, the feasibility of each method, the friend's status, the urgency, the privacy, and the data type of communication). Once she/he has decided a method for communication (*e.g.*, send an email), she/he needs to think of devices (*e.g.*, a PC) and applications (*e.g.*, Outlook) to use. In this case, she/he needs to know how to use Outlook for sending an email with attached images/videos. She/he also needs to know how to configure Outlook with her/his email account.

To accomplish the same task above in a task-driven smart environment (we could call it a *taskable space* [9]), she/he 'tells' the smart environment (perhaps via a personal mobile device or a user interface nearby) something like "I want to contact my friend". Next, the smart environment will load a representation of the task (labelled "Contact someone"); and based on the current context, it guides the user to accomplish the task. The input for these steps could be provided by either a context information manager or the user.

We envisage ubiquitous future smart environments (*e.g.*, home, library, shopping mall, factory, and city centre) where the users should not handle directly functionalities provided by individual applications, services, and devices but rather their high-level tasks [9], *e.g.*, 'watch a movie', 'borrow a book', 'contact someone', and 'present a seminar'. One of the first steps towards this vision is a language for representing tasks in smart environments. We call such a language a *task representation language* (TRL). A TRL provides a formal syntax and semantics for specifying how a task can be achieved. A *pragmatic task representation*–a document written in a TRL–includes the activities involved in completing a task, including both activities carried out by humans and those performed by machines.

Given the numerous ways of representing tasks in different domains, the obvious question is whether such existing languages meet the needs for this relatively recent notion of smart environments, and we aim to address this question in this paper. We compare candidate languages for representing tasks in smart environments. We analyse and discuss their similarities and differences based on a set of desirable language properties. Two main aspects of the languages, including their design principles and expressive power, are analysed. The design principles involve application domains and languages' characteristics. The expressive power includes language constructs (*e.g.*, tasks, goals, sub-tasks, events, and data), temporal operators and inter-task operators for representing relationships between (sub-)tasks, and task attributes for representing descriptive aspects of tasks such as task name, task goal, task type, and trigger conditions. Our selection of languages aims to be representative of different kinds of languages and to be comprehensive - clearly, we cannot be exhaustive.

The existing languages have been developed for representing tasks in different domains (*e.g.*, interactive systems design, multi-agent systems, robotics, network protocols, and

business processes), and meet some of the desirable properties. However, there is no language that completely satisfies our proposed set of the desirable properties (*e.g.*, runtime executability and context-awareness). That is, the tasks represented in these languages are too coarse to support the context-aware runtime executions of the tasks. In this paper, we propose common operators, constructs, and task attributes that a new task representation language should provide, and elaborate how the languages we reviewed incorporated some of these attributes. Our criteria for task representation languages for smart environments are based on a general vision of task-oriented computing. Instead of just one language, we envisage a family of languages for representing tasks for the ubiquitous smart environments of the future, and this paper aims to outline what features such languages might need. While our perspective is to find suitable representation of tasks for smart environments, we also make importance observations about how the notion of task has been conceived and represented across application domains.

The paper is organised as follows. In Section 2, we first present notions of tasks, task-driven computing, task-driven computing systems. Then we present reasons why we need a language for representing user tasks in smart environments. Next, in Section 3, we presents our envisioned generic task-oriented computing system for smart environments. According to this system, Section 4 proposes a set of desirable properties of a language for representing tasks in smart environments. Section 5 presents a review of the existing popular languages for representing tasks in general; we classify these languages based on their application domains. A discussion and a comparison between these languages are given in Section 6. In this section, we also summarise common language elements that a new task representation language should provide, together with guidelines for selecting such a language. Finally, Section 7 presents future work and conclusion.

2 Introduction to Task Computing

2.1 Tasks in Smart Environments

The notion of task is informally intuitive but can be hard to formally defined. There have been many definitions for tasks in smart environments (*e.g.*, [7, 9–12]). Generally, our definition for tasks in smart environments as follows: "*A task is a collection of other relevant tasks (we call them sub-tasks). A task that cannot be decomposed further or that is unnecessary to be decomposed further in the current smart environment is called a primitive task or an action.*" A task is related to some goals that may be different for different users in different situations. For example, a task of contacting someone can be a collection of sub-tasks including looking up a number, pressing the call button, holding the conservation, and pressing the end call button. Some of these sub-tasks (*e.g.*, look up a number) can then be further decomposed. The task of contacting someone could be related to different goals, *e.g.*, to tell a story, to inform a situation, or to send a signal (*e.g.*. I've arrived).

We draw a distinction between task and service for the purpose of our discussion. A task is essentially different from a service. Services (*e.g.*, Web services) are means to accomplish tasks. They are the performance of associated tasks [9]. A task is a user-oriented and high-level term, which is associated with what a user wants to accomplish, while a service is a system-oriented and low-level term, which describes what functionalities the system provides [13].

2.2 What is Task Computing?

Task-driven computing [7] (or also called *task computing*[14], *task-oriented computing* [9], *Task-Centric Computing*, or *plan-driven computing* [15]) allows users to focus on the tasks they want to accomplish rather than how to accomplish them. In other words, task computing aims to transfer users' focus from the computer to the task at hand, to help a person "to forget that he/she is using a computer while actually interacting with one" [16]. Task computing is computation to fill the gap between tasks (what the user wants to be done), and services (functionalities that are available to the user) [8, 9]. The fundamental objective of task computing is to present to the user the tasks that are possible/applicable in the user's current context and to guide the user through the execution of those tasks. Tasks may span over multiple applications and multiple computing platforms (not a single device).

One of the approaches to task computing is to re-orient the user interface (called *task-based user interface* for user interactions with the smart environment) around the tasks for the user, rather than the functionalities of individual devices or applications within the environment [17]. This notion has been made popular by the Apple Siri application[1] and also the Windows task-centric user interfaces[2]. In other words, there is a transition from a function-oriented interaction with devices (*i.e.*, menu-based dialogue structures with a fixed interaction vocabulary) to a task-oriented interaction with lower larger interconnected systems (*i.e.*, conversational dialogue structures with an unrestricted interaction vocabulary) [18].

2.3 What is a Task Representation?

A *task representation* (others may call it a *task model*, *task description*, *task specification*, *task formalism*, *task expression*, *situated flow* [19], or *pervasive workflow* [20]) is a plan to achieve the goal of the task. It is a document written in a task representation language, which includes the steps involved in completing the task, the resources needed by each step, the causal, and data flow relations between those steps [10, 12, 15]. A representation of a task usually decomposes the task into sub-tasks. When we reach a sub-task which cannot be further decomposed we have a basic (atomic or primitive) task. In some cases, a basic task requires one single physical action or one service to be performed.

We should make a distinction between a task representation and a task instance. A task representation defines a generic task (called *task class*) with *types* of required data and services. A *task instance* is a parameterised representation of a generic task. It corresponds to an actual or hypothetical occurrence of a task class [12]. 'Contact someone' is an example of a generic task. But 'Bob makes a phone call to Alice' is an instance of this generic task.

2.4 The Need for a Standardised Task Representation Language

Selecting a suitable language for representing tasks in smart environments has been seen as an important challenge when designing task-oriented systems [9]. Separating task representations from the system implementation can provide the following benefits:

- From the perspective of application developers, this separation of concerns allows

[1] http://www.apple.com/iphone/features/siri.html

[2] http://www.winsupersite.com/article/product-review/how-it-works-inductive-user-interfaces

for the independent modification (*e.g.*, extension) of the systems and task representations. If task representations are written in a standard language, they can be reused with different systems in different smart environments or inter-environments.

- Task representations also provide a higher level of abstraction in communicating the "intent" between systems and end users [15]. End users can present the system with the task they want to achieve, instead of the individual devices or applications that they will use. The system could then use the task representation to guide its interaction with the user to accomplish the task. Moreover, by knowing what task the user wants to do, the system could automatically reserve necessary resources and configure devices the user may soon need as identified in the task representation.

- Given an explicit representation of a particular task, the system could guide users through the steps of that task if they are unfamiliar with it. If the system detects a failure of a service, it can use the representation of the task to suggest an alternate course of action that would still achieve the intended goal.

- Task representations enable sharing, composing, extending, and executing abstract task representations in different smart environments by useful abstractions provided by the language (*e.g.*, we can compose a representation of a task "borrow a book" once but it can be used or extended for different libraries).

- With a formal language, it is possible to develop an analytical theory and tools for automatic verification, validation, and conformance testing of task representations.

3 An Envisaged Generic Task-Oriented System for Smart Environments

Before we present a set of desirable properties of a language for representing tasks in smart environments, we clarify what the task-oriented system for smart environments that we will be developing. This section presents a generic conceptual architecture for our envisaged task-oriented system (TaskOS [21]), which is a common software base for future smart environments ranging from kiosks, personal offices, meeting rooms, shopping malls, to city centres. TaskOS operates at the level of tasks which are supposed to be commonly supported within the smart environments, rather than individual resources (*e.g.*, devices, applications, and services) which are available in the environments. This architecture generalises from our own earlier prototyping and other task-oriented systems [5, 11, 12, 14, 22–24].

The main feature of TaskOS is to select a task representation from its task library and generate a parameterised task representation (*i.e.*, task instance) which is likely to best achieve the task in light of users' preferences, context and the current resource availability. Once the task instance is generated, it guides the user and monitors the execution of the task instance. In particular, TaskOS provides two main features: context-aware task recommendation and context-aware guidance of task execution.

3.1 Context-Aware Task Recommendation

What is context-aware task recommendation? TaskOS recommends possible, relevant tasks for users based on their current context and the environment's capabilities (*e.g.*,

device and service availability in the environment) [25]. For example, Bob is attending a conference. When he is approaching a smart conference room, TaskOS for this conference room recommends him several tasks such as 'register as a presenter', 'publish presentation slides', 'print a document', 'view conference agenda', and 'make a slide-show'. This recommendation is generated based on his context such as location and role. When he points his mobile device at an air-conditioner in the room, there are tasks recommended for him such as 'set temperature' and 'set fan'. When he is in the proximity of a coffee machine, tasks such as 'make coffee' and 'make tea' are recommended for him. These two examples of task recommendations are pointing-direction and proximity awareness

Why context-aware task recommendation? Users in smart environments are perhaps surrounded by hundreds of services and devices including sensors, actuators, appliances, and computers. This would result in an overload of information hindering the users from getting to know what tasks are possible and relevant to their current context within a smart environment. When smart technologies vanish into everyday objects, users may not recognise extra features these objects can provide beyond their normal functions. MediaCup [26] is an example of such an object. It is a cup augmented with sensing, processing and communication capabilities to offer some advanced features while preserving its normal appearance, purpose, and use. Moreover, devices and services in a smart environment are frequently added and removed (*e.g.*, due to the movement of users with their mobile devices or the upgrading of existing stationary devices and services). This does not allow a user to make an assumption that a task which was possible yesterday is possible today, or a task was impossible yesterday is impossible today. In other words, the tasks supported by a smart environment can vary over time. As a result, the users may not be aware of what tasks are currently supported by a smart environment, especially for an environment which is unfamiliar to the users. This is why context-aware task recommendation can be useful.

3.2 Guidance and Management of Task Execution

In current practice, to accomplish a task, a user may need to decompose the task into sub-tasks recursively and map the sub-tasks with appropriate features supported by individual services and/or devices within the environment [5]. This requires the users to learn how to operate each device and how the functions of a device can be integrated with the others. This learning task is time consuming, especially for the environments where the users may come for a short time and leave, such as a conference room. TaskOS provides proactive task guidance for the user to accomplish an intended task. The user may not need to know which devices and services to use before s/he starts the task. TaskOS is also able to deal with task interruption, task migration, multi-tasking, resource conflicts, and adaptation to the variation in availability of resources in smart environments.

3.3 A Generic Conceptual Architecture of TaskOS

Figure 1 illustrates an architecture for TaskOS. In this architecture, *Service & Device Manager* (SDM) manages devices and services of the smart environment. *Context Information Manager* (CIM) manages context information and answers context queries to other components of the system. *Context-Aware Task Recommender* (TaskRec) [25] suggests to users possible and relevant tasks based on their context and environment capability. *Task-Based User Interface* (TaskUI) is an interface for users to interact with TaskOS. *Task Execution Engine* (TEE) loads, verifies, and executes task representations. During

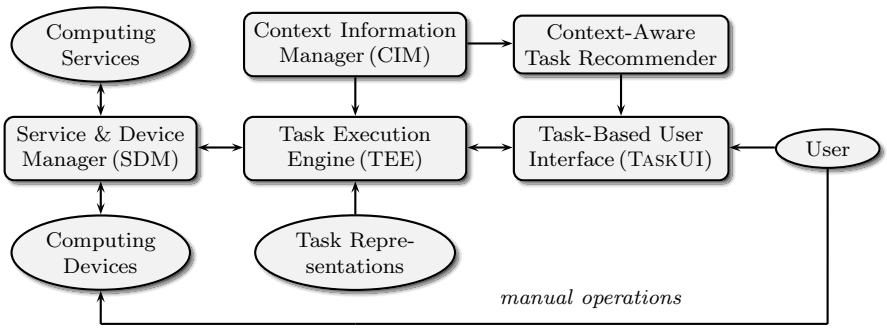

Figure 1: A component architecture for TaskOS.

executing a task, if TEE needs input from users or wants to present them with information or instructions, it sends an interface request to TaskUI. How TaskUI presents this request depends on the current interface modality being used; it may be visual, spoken, tactile, or multimodal. If the users perform manual operations on devices, TaskUI will simply present them necessary instructions.

Because task representations are fundamental inputs to TaskOS, a language is hence required for formally representing tasks. In the next section, we propose a set of desirable properties that such a language may provide in order to realise the features of TaskOS aforementioned.

4 Properties of a Task Representation Language for Smart Environments

Towards a task-oriented system for smart environments aforementioned, we propose properties of a desirable language for representing tasks in smart environments.

- *Abstraction/Genericity*: Task representations can be reusable in different smart environments (*i.e.*, the smart environments differ in devices, services, and underlying technologies). For example, the same representation of a task of "showing photos on a (large) display" can be reused either in a seminar room for showing photos from a digital camera on a projector screen or in a home theatre showing photos from a mobile phone on a TV display. In other words, task representations do not need to be bound to any specific devices or services. They can be platform-independent, device-independent, service-independent, and infrastructure-independent. Consequently, creators of task representations do not need to be aware of specific devices and services which will be involved in execution of these tasks. The abstraction of task representations also enables a task execution engine to adapt to the availability of devices and services at runtime.

- *Hierarchical structure*: With TaskOS, we would like to re-orient interactions between users and smart environments around tasks rather than individual devices (including individual applications on each device as iOS or Android do) and services. Organising a task and its subtasks in a hierarchical structure is a natural solution for generating task-oriented interactions. A hierarchical structure also enables development of mechanisms for navigating, activating, suspending, resuming task steps, and generating instructions during task execution.

- *Support for semantics*: To support for executing abstract task representations at runtime, a task representation language should provide mechanisms for resolving concepts, functions, services, resources, expressions, and devices which are specified abstractly within a task representation. These mechanisms are to semantically map abstract concepts into real objects provided by a smart environment. The semantic richness of a language allows reasoning about tasks so that tasks can be instantiated dynamically, giving the possibility to query and compare task descriptions. One way to achieve this would be to develop an ontology that captures domain concepts and relationships among them.

- *Interpretability*: Task representations must be interpretable/executable. A task execution engine can go through a task representation, automatically interpreting the representation, executing described actions (perhaps in collaboration with the user), and actually carry out the task.

- *Markup language*: Markup languages such as XML-based languages are recently preferred because of their ease of incremental updating, inheritance, and namespace standardisation. They offer a means for structuring representations in a way that tradeoffs between human-readability and machine-processibility.

- *Support for context-aware task recommendation*: Task representations should include descriptive information which is used for evaluating the feasibility and relevancy of tasks based on the capabilities of a smart environment and user's context. For example, adding/removing appliances to/from a smart environment may change the feasibility of tasks. The tasks then can be ranked and recommended for the user according to their feasibility and relevance.

- *Support for proactive task guidance*: TASKOS does not only execute task representations, it also needs to explain and reason about its decisions. Therefore, task representations should include information which can be used at runtime to guide the user through the accomplishment of tasks and to answer the user the following questions [12]:

  - What task to do next?
  - How to do it?
  - What are the inputs/outputs?
  - Did the task succeed?
  - Why did the task failed? How to fix the problem?

  Task guidance also involves mixed initiative dialog, context-sensitive explanation, progress indicator, and a special "undo" function. The system can assist the user by suggesting options (*e.g.*, default actions and parameters) and explaining their implications.

- *Support for task scheduling*: Inter-dependencies among (sub)tasks are often required. For example, a certain task A has to be performed before task B can be started. Hence, a language should provide a set of temporal operators for scheduling task performance. Examples of such operators are sequence and concurrency. The former denotes constraints that tasks must be performed in a specific order whereas the latter defines a possibility for tasks to be performed concurrently.

```
<!-- Play Movie Task -->
<iplay:Task>
  <taskVerb>PLAY</taskVerb>
  <taskSubject>
    <Subject rdf:ID="Movie">
      <requires rdf:resource="#AudioRenderer"/>
      <requires rdf:resource="#VideoRenderer"/>
    </Subject>
  </taskSubject>
</iplay:Task>
```

Figure 2: An InterPlay representation of a *Play Movie* task.

- *Formal definition*: Syntax and semantics of a language should have a complete and formal definition. In particular, the formal definition–on which the semantics of a language is based–should support the development of a theory for automatic verification, validation, and conformance testing of task representations.

5 An Overview of Task Representation Languages

In this section, we review a number of languages that were developed with different objectives in mind, but that have all been used to represent tasks. These languages may be more or less formal, depending on their intended use and audience. There is no easy way of categorising these languages along a single dimension, as they often cut across several dimensions. According to their application domains, we categorise the languages into groups: *smart environments and task computing*, *interactive systems design*, *multi-agent systems and robotics*, and *business processes and workflows*. For each of the categories, we review the languages in chronological order.

5.1 Smart Environments and Task Computing

5.1.1 InterPlay

InterPlay [23] is a system for home environments that allows users to issue multimedia tasks in *pseudo sentences* without having to consider where a particular content is located and how to achieve those tasks. The minimal pseudo sentence representation for a task consists of a *verb*, a *subject* (content-type or, content), and *target device(s)*. An example of such pseudo sentence would be: 'Play' (verb), 'The Matrix' (subject), 'Living Room DTV' (target device), which means "Play the DVD 'The Matrix' onto the DTV in the Living Room". A more elaborated pseudo sentence can include additional information, such as device location and device settings such as "wide screen". A user composes a pseudo sentence by choosing the verb, the subject, and the target device from the three predefined lists. For example, to play a movie on TV, the user can first select the TV, next select the "Play" verb, and finally selects the "Movies" as the content type.

An InterPlay's task representation consists of three types of information: sentence description on what the user can do, what functionalities are needed, and the mapping to how to the task is done. A sample representation for the 'Play Movie' task is shown in Figure 2. InterPlay can only execute single-step tasks where all inputs are pre-determined before task execution. It cannot support complex tasks composing of multiple subtasks and temporal relationships between them (*e.g.*, ordered and parallel). Also, it does not have features for specifying pre-conditions and post-conditions for steps in a task decom-

Operator	*Description*
inOrder	Tasks must occur in specified order.
anyOrder	Tasks can occur in any order, but all must occur.
Optional	Tasks may or may not occur.
oneOrMore	One or more of tasks may occur.
zeroOrMore	Zero or more of tasks may occur.
repeatExactly	Tasks must occur a given number of times.
Choice	Exactly one of the tasks may occur.

Table 1: Operators available in PETDL.

position.

5.1.2 ANSI/CEA-2018 Task Model Description (CE-TASK)

CE-TASK [27] is a standard language specifically designed for representing tasks relevant to consumer electronics devices. CE-TASK represents tasks in terms of *subtasks* and *steps*. Steps are can be grounded to actual device functions via *JavaScript*. Other elements in a CE-TASK task representation are *input* and *output* parameters, *pre-conditions*, *post-conditions*, *grounding scripts*, *user intent concepts*, data flow between subtasks (*data bindings*), *applicability conditions*, and *initialisation scripts*. An applicability condition helps the system choose an appropriate decomposition when there is more than one. An initialisation script is not associated with any tasks and is intended to be executed exactly once when the containing task is loaded. User intent concepts can be specified in an OWL ontology which describes semantic concepts and their relationships support for reasoning about types, values, and functions declared within a task representation. To control steps sequencing, CE-TASK provides temporal operators such as *Ordered*, *Requires*, *Min-Occurs*, and *Max-Occurs*. However, CE-TASK doesn't provide a parallel operator and a mechanism for synchronisation between parallel tasks. Figure 3 shows a CE-TASK representation for the task of borrowing a book from the library.

5.1.3 Context-Adaptive Task Model

The context-adaptive task model[28] (CATM) breaks down a task hierarchically into subtasks. The root of the hierarchy represents the task and is associated with a *context situation* which defines the context conditions whose fulfilment enables the execution of the task. Subtasks inherit the context conditions of their parent task. Figure 4 shows the sample representation of the *WakingUp* task. This model provides some temporal operators: *Exclusive* (*i.e.*, only one subtask will be executed-disabling the others), *Enabling*, *Concurrency*, and *Enabling with condition*.

5.1.4 Pattern-based Event and Task Description Language (PETDL)

PETDL [29] is an XML mark-up task representation language that is based on GOMS[30]. Its main purpose is to facilitate monitoring of task execution by matching events generated from the system with events declared in a task representation. Although the language is not designed for the purpose of representing tasks which can be executed to change the state of the physical world, it does provide some significant temporal operators as summarised in Table 1. Figure 5 shows how a calendaring task in Microsoft Outlook could be specified in PETDL.

```
[fontsize=,samepage=true]
<taskModel about="urn:computer.org:cetask:library"
  xmlns="http://ce.org/cea-2018"
  <task id="Borrow">
    <input name="book" type="Book"/>
    <subtasks id="borrowing">
      <step name="go" task="GoToLibrary"/>
      <step name="choose" task="ChooseBook"/>
      <step name="check" task="CheckOut"/>
      <binding slot="$choose.input"
        value="$this.book"/>
      <binding slot="$check.book"
        value="$choose.output"/>
    </subtasks>
  </task>

  <task id="GoToLibrary"/>

  <task id="ChooseBook">
    <input name="input" type="Book"/>
    <output name="output" type="Book"/>
    <subtasks id="initial">
      <step name="lookup"
        task="LookupInCatalog"/>
      <step name="take" task="TakeFromShelf"/>
      <binding slot="$lookup.book"
        value="$this.input"/>
      <binding slot="$take.book"
        value="$this.input"/>
      <binding slot="$take.location"
        value="$lookup.location"/>
      <binding slot="$this.output"
        value="$this.input"/>
    </subtasks>
    <subtasks id="alternative">
      <step name="search"
        task="UseSearchEngine"/>
      <step name="take" task="TakeFromShelf"/>
      <applicable>
        $this.success == false
      </applicable>
      <binding slot="$take.book"
        value="$search.book"/>
      <binding slot="$take.location"
        value="$search.location"/>
      <binding slot="$this.output"
```

```
[fontsize=,samepage=false]
        value="$search.book"/>
    </subtasks>
  </task>

  <task id="LookupInCatalog">
    <input name="book" type="Book"/>
    <output name="location" type="string"/>
    <postcondition>
      $this.location != undefined
    </postcondition>
    <script>
      $this.location = lookup($this.book);
    </script>
  </task>

  <task id="TakeFromShelf">
    <input name="book" type="Book"/>
    <input name="location" type="string"/>
  </task>

  <task id="UseSearchEngine">
    <input name="query" type="string"/>
    <output name="book" type="Book"/>
    <output name="location" type="string"/>
    <postcondition>
      $this.book != undefined
    </postcondition>
    <script>
      $this.book = search($this.query);
      if ( $this.book != undefined )
        $this.location = lookup($this.book);
    </script>
  </task>

  <task id="CheckOut">
    <input name="book" type="Book"/>
    <script>
      print("["+$this.book+" checked out!]");
    </script>
  </task>

  <script init="true">
    <!- initialisation script ->
  </script>
</taskModel>
```

Figure 3: A CE-TASK representation of a *Borrowing Book* task.

```
<org.pros:TaskModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
  ...
  <Task xsi:type="org.pros:SystemTask" name="turn on bathroom heating" ID="WakingUp_TOBAHC"
    isChildOf="//@Task.0">
    <ContextPrecondition ContextPreconditionString="BathroomTemperature<28">
    <TemporalRelationship TemporalRelationshipType=">>[10 min]>>"
      TemporalRelationshipTo="//@Task.2" TemporalRelationshipFrom="//@Task.1"/>
    <service href="SmartHomeServices.serviceModel#//@Service.3"//@Method.switchOnHeating"/>
  </Task>
    <Task xsi:type="org.pros:SystemTask" name="turn on the radio" ID="WakingUp_TOR"
      isChildOf="//@Task.0">
    <TemporalRelationship TemporalRelationshipType="|||"
      TemporalRelationshipTo="//@Task.3" TemporalRelationshipFrom="//@Task.2"/>
    <service href="SmartHomeServices.serviceModel#//@Service.2"//@Method.turnOnRadio"/>
  </Task>
  ...
</org.pros:TaskModel>
```

Figure 4: A CATM representation of a *WakingUp* task.

5.1.5 Other task-oriented systems

A task computing system called Task Computing Environment [8] (TCE) is designed for smart environments to assist users in completing every tasks such as exchanging e-business

```
<task name="Manage Schedule">
  <task name="Schedule Appointment From Email">
    <inOrder>
      <event name="OpenMailItem"/>
      <optional>
        <event name="SwitchFocus"/>
      </optional>
      <anyOrder>
        <task name="AddAppointment">
          <inOrder>
            <event name="OpenAppointmentItem"/>
            <oneOrMore>
              <event name="ChangeAppointmentItemProp"/>
            </oneOrMore>
            <event name="SaveAppointmentItem"/>
            <event name="CloseAppointmentItem"/>
          </inOrder>
        </task>
        <event name="CloseMailItem"/>
      </anyOrder>
    </inOrder>
  </task>
</task>
```

Figure 5: A PETDL representation of a *Calendaring* task.

cards and showing presentation. The system represents tasks as web services using OWL-S which is discussed later in this paper.

The Aura system [22] supports the migration of user tasks (*i.e.*, task instances) between smart environments. Because the main aim of Aura's task representation language is to capture the status of task instances for the purpose of task migration, Aura lacks features for representing envisaged tasks (*i.e.*, task classes). Importantly, we could not find a completed publication on their language.

The task execution framework [11, 31] focuses on the adaptation of task executions in smart environments to the changing contexts and resource availabilities. In this framework, developers first develop primitive activities that perform actions like starting, moving or stopping components, changing the state of devices, services, or applications. They then develop workflows that compose a number of primitive activities into a task. These primitive activities and workflows are written in the C++ language.

5.2 Interactive Systems Design

Most of these languages provide graphical notations for graphically representing human-machine interactions of interactive systems. The representations are mainly used for communication among user interface designers, system implementers, and consumers. Therefore, they are often not machine-interpretable and executable. However, it is useful to review what aspects of tasks are represented in these languages.

5.2.1 UAN

The User Action Notation[32, 33] (UAN) is a textual notation which is used to represent the dynamic behaviour of a graphical user interface relevant to completion of a task. UAN was initially intended as a communication mechanism between user interface designers and system implementers. In UAN, a task is represented in a quasi-hierarchical structure of user's actions performed on the user interface. Pre-conditions and post-conditions can also be included in a task representation. UAN also provides operators to express temporal

Attribute	*Description*
Name	Symbols represent for the task identification.
Prototype	A prototype task from which the newly defined task inherits information.
Task-type	There are five types of tasks. *User tasks* are tasks that the user performs. *Presentation tasks* are to present information to the user. *Application tasks* are tasks that the application performs without user involvement. *Interaction tasks* represent low-level actions such as mouse clicks. *Undetermined tasks* a way to delay committing to a specific task type.
Goal	Specifying what a task does.
Effects	The actions to be performed when the task is executed. They include method invocation, parameter setting, task status, and data representation.
Parameters	The input/output data that the task consumes/produces.
Precondition	The conditions that must be true before the task can be executed.
Is-optional	Whether the task is optional, and does not need to be performed.
Is-resumable	Whether the task can be resumed after it is interrupted.
Is-interruptible	Whether the task can be interrupted once it is started.
Is-loop	Whether the task can be performed multiple times, provided that the preconditions remain true.
Is-reentrant	Whether separate instances of a task can be spawned at run-time. If the task is not reentrant, only one instance of the task is used.
Sub-tasks	Specify the sub-tasks of a task and in what order they need to be executed.

Table 2: Attributes of a task in MASTERMIND.

relationships among tasks (*e.g.*, *Sequence*, *Iteration*, and *Concurrency*). Again, UAN task representations are not machine-executable.

5.2.2 ADEPT

ADEPT[34] is a design environment for prototyping user interfaces based on task models. ADEPT has adopted a modified version of Task Knowledge Structure[35] (TKS) as the technique for representing tasks both textually and graphically. A TKS task representation describes a sequence of actions to be performed on the user interface to achieve the goal. In ADEPT, there are four types of temporal operators: *Sequence*, *Interleaved*, *Parallel*, and *Choice*.

5.2.3 MASTERMIND

MASTERMIND [36] is a model-based user interface development environment. It was developed based on HUMANOID [37] and UIDE [38], two model-based systems. HUMANOID is based on presentation models whereas UIDE is based on dialogue models. In MASTERMIND, a task representation contains a goal, pre-conditions, effects, information requirements, and sub-tasks. It provides four types of temporal operators: *Sequence*, *Parallel*, *Unrestricted*, and *One-of*. A task representation also contains a set of *flags* that controls subtasks sequencing. Table 2 summarises available attributes for representing a MASTERMIND task.

5.2.4 DIANE+

DIANE+[39] aims to facilitate automatic generation of user interfaces for interactive systems. It provides a graphical notation to represent tasks. The language has two main concepts which are *operation* and *precedence*. An operation is either a process or a set

Operator	*Description*
Enabling	One task enables the other when it terminates.
Disabling	The first task is deactivated once the first action of the second task has been performed.
Concurrent	Tasks can be concurrently executed in any order.
Choice	It is possible to choose from a set of sub-tasks and once the choice has been made, the task chosen can be performed and other sub-tasks are not available at leat until it has been terminated.
Suspend-Resume	It is possible for the second task to interrupt the first task and when the second task is terminated, the first task can be resumed.
Order independence	Both sub-tasks have to be performed but when one is stared it has to be finished before starting the second one.
Order dependence	Both sub-tasks have to be performed in a specified sequence.
Concurrency with information exchange	Two sub-tasks can be executed concurrently but they have to synchronise in order to exchange information.
Enabling with information passing	One task provides some information to the other tasks other than enabling it.
Recursion	Nesting the task itself in its decomposition.

Table 3: Temporal operators in ConcurTaskTrees.

of sub-operations. A precedence is a sequencing link between operations. DIANE+ defines a rich set of attributes of an operation: *Type* (*e.g.*, *automatic*, *interactive*, *manual*), *Mandatory* (*e.g.*, *required*, *optional*, *constrained*–the mandatory depends on a condition), *Trigger* (*e.g.*, *user-triggering*, *system-triggering*), *Pre-condition*, and *Post-condition*. DIANE+ defines two types of entities: *Task* and *Event*. The temporal operators in DIANE+ are *Ordered*, *Unordered*, *Loop*, *Required choice*, *Free choice*, *Parallel*, *Default operations*, and *Optionality*.

5.2.5 ConcurTaskTrees

ConcurTaskTrees [40] (CTT) is a diagrammatic notation for representing tasks. Its main aim is to support user interface design for interactive systems. CTT represents a task as a hierarchical structure. It defines three main types of entities in a task representation: *Task*, *User role*, and *Object*. CTT specifically concentrates on user interface related tasks. It lacks concepts of physical tools including devices and services which are often required for execution of tasks in smart environments. CTT extends LOTOS[41] and provides several temporal operators as described in Table 3. CTT also defines a number of elements and attributes for a task as described in Table 4 and Table 5.

5.2.6 Teallach

Teallach [42] is a task-based user interface development environment for object database systems. Teallach's task representation language provides support for modelling both the structure of tasks and the flow of information between them. A task Teallach is a goal-oriented hierarchy, with its leaf nodes representing interaction tasks. Specifically, in the Teallach language, all tasks have an option of being *cancellable* which specifies that a task can be cancelled while being executed. The language also allows specifying the passing of information into and out of tasks and the *state* associated with any none-primitive tasks.

Element	*Description*
Name	Extended name associated with the task
Type	Task type
Description	Explanation of what the task stands for
Platform	Set (possibly empty) of platforms supporting the task
Precondition	A possible condition that has to be valid in order to perform the task
TemporalOperator	The temporal relationship existing between the concerned task and its sibling right (if the latter exists)
TimePerformance	An estimation of the amount of time (min/max/average) needed for performing the task
Parent	Reference to the parent task (if the latter exists)
SiblingLeft	Reference to the sibling left task (if the latter exists)
SiblingRight	Reference to the sibling right task (if the latter exists)
Object	Set (possibly empty) of objects manipulated by the task
Subtask	It referres to a task which is represented, in turn, by a high level task
Access rights	User access rights to perform the task

Table 4: Elements of a task in ConcurTaskTrees.

Attribute	*Description*
Identifier	The identifier associated with the task
Category	The category associated with the task, which could be one of the following values: abstraction–tasks that have subtasks belonging to different categories, user, interaction, and application
Iteration	A flag indicating whether or not the performance of the task could be repeated until it is terminated by other tasks.
Finite Iteration	Specify how many times the task will be performed.
Optional	It indicates if the performance of the task is mandatory (false) or not (true)
PartOfCooperation	It indicates whether or not the task is involved in a cooperative task
Frequency	The estimated frequency associated with the task (low/medium/high)

Table 5: Attributes of a task in ConcurTaskTrees.

5.2.7 TaskMODL

TaskMODL [43] is a graphical language which supports design of user interfaces by using task models. The language represents a task in terms of sub-tasks, a set of required resources (*e.g.*, actors–who perform the task, tools, and data), pre-conditions, post-condition, output, and task cardinality (*i.e.*, how many times the task must be repeated). The language consists of four operators for controlling the sequence of sub-tasks: *Unconstrained*, *Non-overlapping*, *Sequence*, and *Choice*.

5.2.8 TOMBOLA

TOMBOLA [44] is another graphical language for modelling tasks in support for user interface design. The language provides several temporal operator: *Sequential* (*i.e.*, the sub-tasks must be executed in a fixed sequence), *Serial* (*i.e.*, the sub-tasks can be executed in an arbitrary sequence), *Parallel* (*i.e.*, the sub-tasks can start and end at random), *Simultaneous* (*i.e.*, all sub-tasks must start in an arbitrary sequence before any task can end, hence at least one moment exists where all sub-tasks are running simultaneously), *Alternative* (*i.e.*, exactly one randomly selected sub-task must be executed); *Optional* (*i.e.*, one or no sub-task at all must be executed), and *Loop* (*i.e.*, a single task is repeatedly executed while a given condition is true).

Attribute	*Description*
Priority	A level of priority (very, rather, not very)
Frequency	A task is characterised by a frequency (high, medium, low)
Executant	Can be user, system, interactive, abstract, or unknown. When the executant is a user, the task is characterised by methods (*e.g.*, sensorimotor, cognitive). The user who is associated with the task is characterised by a name, a level of experience (beginner, average, expert) and skills
Objects	A set of objects used by a task (including Event and User)
Effects	The description of the effects observable (or desirable) by the user during the task. It is the system response to the user.
Modality	The nature of the action of the executant "User". Two values are possible: Sensorimotor–it is about a physical action (for example, to fill the printer with paper); Cognitive–it is about a mental action (for example, mental calculation of travel expenses)
Optional	Indicates if the task is mandatory or optional
Interruptible	Indicates if the task is interruptible or not
Trigger event	Indicates that the execution of the task is forced by an event
Actor	This characteristic contains the list of the users authorised to carry out the task
Iteration	Specifies the iterative aspects of a task which be carried out several times and/or until the completion of a condition. This condition relates to the objects handled by the user.
Events	The events that will be generated by the execution of the task
Pre-condition	Specify a condition that must be true before the task can be executed
Post-condition	Specify a condition that must be true once the task has finished. Otherwise, the task is seen unsuccessfully finished

Table 6: Attributes of a task in K-MAD.

5.2.9 useML

useML [45] is a language that can be used to represent human-machine interfaces. The following binary temporal operators are provided in the useML version 2.0: *Choice* (*i.e.*, exactly one of the tasks will be fulfilled), *Order Independence* (*i.e.*, two tasks can be accomplished in an arbitrary order, however when the first task has been started, the second one has to wait for the first to be finalised or aborted), *Concurrency* (*i.e.*, two tasks can be accomplished in an arbitrary order, even parallel at the same time), *Sequence* (*i.e.*, two tasks must be accomplished in the given order. The priorities (*i.e.*, the order of temporal execution) are defined as follows: Choice > Order Independence > Concurrency > Sequence.

5.2.10 K-MAD

K-MAD [46] is a graphical tool for modelling tasks. In K-MAD, a task is defined by a name, a ID number, a goal, a performance duration, feedback information (observable effects by the user), multi-media (associating the task to multi-media information such as video or sound). Besides, a task may have attributes as described in Table 6. K-MAD also provides scheduling operators: *Sequencing*, *Enabling*, *Choice*, *Concurrency*, and *No-order*.

5.3 Multi-Agent Systems and Robotics

In multi-agent systems & robotics, tasks are assigned to autonomous software agents and robots. The agents and robots are able to negotiate about their tasks with each other so that they can maximise the quality of the task accomplishment. Task specification

languages in these systems hence provide a rich set of operators that allow us to specify communication and synchronisation between agents/robots. Most of them are akin to programming languages.

5.3.1 TÆMS

The TÆMS language [47, 48] was designed for representing tasks assigned to software agents. A TÆMS task representation is an annotated task decomposition tree. The tree consists of a sequence of sub-tasks and methods which describe how the task may be performed. Sub-tasks can be further decomposed in the same manner. Methods, on the other hand, are terminal, and represent primitive actions that an agent can perform to produce some output. Sub-tasks may have interrelations which describe how their executions will affect other sub-tasks in the tree. Interrelations may also span different task trees assigned to multiple agents.

5.3.2 APEX Procedure Definition Language (PDL)

PDL [49, 50] aims for representing agents' tasks in multiple-task environments. The language has a central construct called *procedure*. A procedure consists of a goal followed by one or more *steps*. The steps primarily describe activities needed to accomplish the goal. The steps can be concurrently executed. Some constraints can also be specified in PDL such as a *pre-condition* for starting a step.

To resolve the problem of resource conflicts in multiple-task environments, tasks are associated with a *priority*. A priority declaration specifies a numeric value ranging from 1 to 10 for urgency and importance of a task. PDL allows us to specify how these values should be computed and under what circumstances they should be recomputed at runtime. Another interesting feature of PDL is that, to decide which ongoing tasks should be interrupted in cases of failure or resource conflict, each task is assigned with an *interrupt-cost*. The task with the least interrupt-cost will be interrupted first when a resource conflict occurs.

A PDL extension [51] adds three constructs to PDL: *Repeat-until* for representing certain repetitive steps, *Repeat-times* to repeat an action a specified number of times, and *Repeat-At-Interval* to cause an action to repeat after a specified amount of time.

5.3.3 COLLAGEN

COLLAGEN [52, 53] is a collaboration manager for users and software agents. The communication between the user and the agent is constructed based on task models. In COLLAGEN, a specification of a task model is called a *recipe*. A recipe contains a sequence of *steps* to achieve a given goal (the objective of the recipe). Each of the non-primitive steps is further decomposed into sub-steps until primitive *actions* are reached. The COLLAGEN language also supports a number of features, including *optional* and *repeatable* steps, *temporal order* and *constraints*, *parameters*, *pre-conditions*, and *post-conditions*. COLLAGEN task models are defined in an extension of the Java language which is automatically processed to create Java class definitions for recipes. Figure 6 gives an example recipe for the Recording TV Program task using a VCR.

```
public recipe RecordRecipe achieves RecordProgram {
    step DisplaySchedule display;
    step AddProgram add;
    optional step ReportConflict report;
    constraints {
        display precedes add;
        add precedes report;
        add.program == achieves.program;
        report.program == achieves.program;
        report.conflict == add.conflict;
    }
}
```

Figure 6: A example task described in COLLAGEN.

Operator	*Description*
Enable	Specify that a node cannot become enabled until some specified event occurs. The event can be either the passage of time, a specific state transition of another node, or some external event.
Sequential	Specify that a node execution is blocked until the executions of other specified nodes are completed.
Serial	One of the two task components is performed (the user has a choice in the execution of the task).
Parallel	Indicate that some tasks can be executed concurrently.
Wait	This constraint makes the task execution blocking until the specified condition have been satisfied.
Terminate	Specify that a node and all its children are to be terminated when some specified event occurs.

Table 7: Operators available in TDL.

5.3.4 TDL

TDL [54] is an extension of C++ that is used to represent tasks assigned to robots. Such a task once executed at runtime will dynamically generate task decompositions. A task decomposition includes sub-tasks (nodes) as well as synchronisation constraints between sub-tasks and between parent and children tasks. In a TDL task decomposition, each node has an action associated with it, which is essentially a parameterised piece of code. This action can perform computations, or dynamically add child nodes to that node, or perform some physical action in the world. Hence, the task decompositions are generated dynamically. The TDL operators are described in Table 7.

5.3.5 Little-JIL

Little-JIL [55, 56] is a language for representing processes assigned to agents. A process is represented as a hierarchy of steps. Each of these steps can be multiply instantiated at runtime according to the *cardinality* attribute expressing the optionality or repetition of a step. A step may require some resources which are representations of entities. Resources may include agents, tools, and various physical artifacts. A step may include post-conditions and pre-conditions to ensure that all of the conditions needed to begin a step are satisfied and that the step has been executed correctly when it is completed. A step may have a deadline that is a point in time by which a step must have completed. Non-leaf steps consist of one or more sub-steps whose execution sequence is determined by one of the step kinds such as *Sequential*, *Parallel*, *Try*, and *Choice*.

Little-JIL also allows us to specify *exceptions* and *handlers* attached to steps. Exceptions and handlers are used to catch and fix up exceptional conditions or errors during

executing steps. Handling an exception determines whether the step will continue its execution, successfully terminate, restart the execution at the beginning, or throw other exceptions. Finally, parameters passed between steps for exchanging information can also be specified in a Little-JIL task representation.

5.3.6 ECO

The ECO language [57] is means to organise digital devices (each device acts as an agent) working together in an orchestrated way. Consider the following representation of a composite task, expressed in ECO: "make coffee; dim lights; wait for lights; download news; wait for news; show news on TV;", an ECO task representation consists of sub-tasks and *dependency statements*. The sub-tasks are executed concurrently subject to explicit prerequisites which are represented by "wait for" statements. ECO uses *verb-object phrases* to represent sub-tasks. Because of its simplicity, ECO lacks features for representing task attributes and data types. Although ECO allows end-users to easily specify their high-level tasks, the system must have a more sophisticated language processor to understand phrases written in a natural language. Currently, there is no such a language processor for ECO.

5.4 Business Processes and Workflows

Workflows define the sequence of tasks to execute in order to achieve some goals. They are used to automate business processes, in whole or in part, and allow passing documents, information, or tasks from one participant to another for action, according to a set of procedural rules. Languages such as YAWL [58] and BPEL[3] are used to define workflows (in terms of invocations of web services). The first limitation of most workflow systems is that workflow representations are static in nature and cannot adapt dynamically to changing resource availabilities or different contexts [59]. The second limitation is that while the languages often focus on representing automated processes, interactions between humans and devices are not much considered. In this section, we only review some popular business process languages.

5.4.1 The Business Process Execution Language

The Business Process Execution Language[4] (BPEL) is a standard language for representing business processed based on web services. A business process is a collection of related, structured activities that achieve a particular business goal. In BPEL, the activities of a business process are represented by web services. BPEL provides several operators such as *Sequencing*, *Loop*, *Choice*, *Concurrency*, *Pre-condition*, and *Post-condition*. Because BPEL only focuses on automated processes (*i.e.*, invocations of web services), interactions between humans and machines are not considered.

5.4.2 The Ontology Web Language for Services

The Ontology Web Language for Services[5] (OWL-S) is a markup language for representing web services's properties and capabilities. A representation of a web service has three

[3]http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
[4]http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
[5]http://www.w3.org/Submission/OWL-S/

parts: service profile, process model, and grounding. The service profile describes a service's functional attributes such as input parameters, output parameters, pre-conditions and effects. The process model describes how a service works. It contains either an atomic process that can be directly executed or a composite process that is a combination of sub-processes. Sub-processes are either atomic processes or composite processes connected by a set of control constructs including *Sequence*, *Unordered*, *Choice*, *If-Then-Else*, *Iterate*, *Repeat-Until*, *Repeat-While*, *Split*, and *Split+Join*. The grounding contains the details of how to access a service by specifying a communication protocol, parameters to be used in the protocol and the serialisation techniques to be employed for the communication.

6 Comparison and Discussion

In this section, we summarise and compare the expressive power of the languages which we have reviewed. The expressive power of a language defines its capability to express a wide range of aspects of a task. The aspects can be divided into two parts. The task definition part contains informational attributes and characteristics of a task. Some examples of such the attributes are such as task name, unique identifier, description, goal, executor (*i.e.*, who performs the task), significance (*i.e.*, the importance of the task), and location, to name some. The other part specifies temporal operators of a task in relation to other tasks and/or to other sub-tasks. Examples of the operators are *Choice*, *Sequence*, and *Concurrency*.

6.1 Expressive Power of Task Attributes

Table 8 presents a dictionary of common task attributes covered by the reviewed languages. An attribute may be represented by different names in different languages. For example, *Post-condition* is named *Effect* in Little-JIL [55]. A task attribute can be one or both of two types: descriptive attribute (*e.g.*, *Frequency* and *Time-performance*) and operational attribute (*e.g.*, *Applicability* condition), denoted by D and O respectively in Table 8. B denotes that an attribute can be both descriptive and operational (*e.g.*, *Access-right*).

ConcurTaskTrees [40] and CE-TASK [27] support most of the attributes. While CE-TASK mainly covers operational attributes necessary for executing a task at runtime, CTT typically covers descriptive attributes which are often useful prior to executing a task. *Iteration*, *Optionality*, *Pre-condition*, and *Post-condition* are the most common attributes supported by the languages. Some attributes (*e.g.*, *Access-right*, *Applicability-condition*, *Deadline*, and *Platform*) are supported by only one or two language. This can be understood as these languages were designed for different specific-purposes.

6.2 Expressive Power of Operators

Table 9 presents a dictionary of common temporal and inter-task operators. Similar to the task attributes, some different words are used in different languages for representing the same meaning of one operator such as *Concurrency* and *Parallel*.

Table 10 summarises the presence of operators in each language. Some of the languages (*e.g.*, InterPlay [23] and COLLAGEN [53]) are not considered in this comparison because we could not find any publications that describes the operators they support. Some operators allow specifying of information exchange between sibling tasks. However, this

Attribute	*Synonym*	*Common meaning*	*Type*
Access-right	Role	Specify roles or actors who can perform a task.	*B*
Applicability		A condition that helps the system choose an appropriate decomposition of a task when there is more than one.	*O*
Cancellable		Specify whether a task can be cancelled.	*O*
Concept		The semantics of concepts declared within a task representation.	*O*
Data-flow	Binding	Specify the data flow between sub-tasks of a task.	*O*
Deadline	End-time	A point in time by which a task must be completed.	*B*
Exception-handler		Specify a handler to catch and fix up exceptions and errors during executing a task.	*O*
Frequency		The estimated frequency of a task (*e.g.*, low, medium, high).	*D*
Grounding script	Binding script	Specify a script that binds a primitive task to actual devices' functions or service.	*O*
Initialisation-script		A script that is not associated with any sub-tasks within in a task, and is intended to be executed exactly once when the task is loaded.	*O*
Input	Object	Input data (object) manipulated when performing a task.	*O*
Iteration	Loop	Indicates the time a task must be performed repeatedly.	*B*
Interruptible		Whether a task is interruptible.	*B*
Interrupt-cost		A cost of interrupting a task.	*B*
MaxOccurs		A task should occur a maximum of the given MaxOccurs time.	*B*
MinOccurs		A task should occur a minimum of the given MinOccurs time.	*B*
Optionality	Elementary	Indicates if the task is mandatory or optional.	*B*
Output	Object	Output data (object) of a task.	*O*
Platform		Set (possibly empty) of platforms supporting a task.	*B*
Post-condition	Effect	It is a necessary condition for success of a task. A task should not be executed if its post-condition is already true.	*O*
Pre-condition		A task is executable only if its pre-condition associated is satisfied.	*O*
Priority		Represent the urgency and/or importance of a task.	*O*
Reset	Restart	Specify a condition when to restart a task.	*O*
Resources		Specify resources (*e.g.*, executers and tools) needed by a task.	*O*
Start time		A task should be executed after this time.	*B*
State		Execution state associated with a task during its performance.	*O*
Suspend		Specify what to do when a task is interrupted.	*O*
System-feedback		The feedback the system should present to the user at a point of time during task performance.	*O*
Task-Identifier		Unique identifier of a task.	*B*
Task-Type		Specify the type of a task (*e.g.*, automatic, interactive, or manual).	*B*
Termination		Specify a condition or events for terminating a task.	*O*
Time-Performance	Duration	An estimated amount of time needed for performing a task.	*D*
Trigger	Initiator	What will trigger a task to start (*i.e.*, user and system).	*O*

Table 8: Common task attributes covered by the reviewed languages.

Operator	*Synonym*	*Meaning*
AnyOrder	Serial	Tasks can be performed in any order.
Cancel		The completion of task A cancels the execution of task B.
Cause		The completion of task A causes the execution of task B completed.
Choice	Alternative	One of the given tasks must be performed.
Concurrency	Parallel	Task A and task B can be performed concurrently.
Disabling	Exclusive	The execution of task A disables task B.
Enabling	Require	Specify that a task cannot begin until another task has completed.
Facilitate		The completion of a task may increase the quality of other tasks.
Free-Choice	ZeroOrMore	Any (including zero and all) of the given tasks can be performed.
Hinder		The completion of a task may decrease the quality of other tasks.
Oneway-Interleave	Interrupt	The execution of task A interrupts the execution of task B, but not vice versa.
Mutual-Interleave	Non-overlap	The execution of task A interrupts the execution of task B, and vice versa.
Race	Try	Tasks in a race are executed in parallel. When one of them completes, the rest are forced to completion (aborted).
Sequence	inOrder	Tasks must be performed in a specific order.
Waiting	Wait-for	Specify the strategy of delaying a step until specified events occur.

Table 9: Common operators provided by the reviewed languages.

indication is an informative element that does not modify the global executing sequence (*e.g.*, two tasks linked by a concurrent-with-information-exchange operator have the same behaviour as two tasks linked by a concurrent operator). Therefore, we do not consider these types of operators in our comparison.

As can be seen from Table 10, the *Sequence* operator is supported by all the languages

	2006–PETDL	2008–CE-TASK	2010–CATM	1990–UAN	1993–ADEPT	1995–MASTERMIND	1996–DIANE+	1997–CTT	1999–Teallach	2002–TaskMODL	2003–TOMBOLA	2004–useML	2006–K-MAD	1995–TÆMS	1998–PDL	1998–TDL	2000–Little-JIL	2007–ECO	2003–BPEL	2004–OWL-S
AnyOrder	✓	✓		✓			✓	✓	✓	✓	✓	✓	✓			✓			✓	✓
Cancel														✓						
Cause														✓						
Choice	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓				✓		✓	✓
Concurrency			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Disabling			✓					✓						✓						
Enabling		✓	✓					✓					✓	✓		✓				
Facilitate														✓						
Free-Choice	✓					✓	✓		✓										✓	✓
Hinder														✓						
Oneway-Interleave				✓	✓															
Mutual-Interleave				✓						✓										
Race															✓		✓			
Sequence	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Waiting				✓											✓	✓		✓		✓
	Smart Spaces			Interactive Systems Design										Multi-agent Systems					Workflows	

Table 10: Expressive power of operators by each language.

while *Choice*, *Free-choice*, *Concurrency*, *AnyOrder*, *Enabling*, and *Waiting* are the "should supported" operators. Some operators (*e.g.*, *Cancel*, *Cause*, *Facilitate*, and *Hinder*) are supported by small number of the languages (*e.g.*, TÆMS). TÆMS is a language for describing agents' tasks, hence it offers special operators that enable negotiation and collaboration between agents. There is no dominating language that seems to support most of the operators.

6.3 Satisfying Proposed Properties

	PETDL	CE-TASK	CATM	UAN	ADEPT	MASTERMIND	DIANE+	CTT	Teallach	TaskMODL	TOMBOLA	useML	K-MAD	TÆMS	PDL	TDL	Little-JIL	ECO	BPEL	OWL-S
Abstraction	−	✓	−	−	−	−	−	+	−	−	−	−	−	−	−	✓	−	✓	−	−
Hierarchy	✓	✓	✓	−	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	−	✓	✓
Semantics	−	✓	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	✓
Interpretable	✓	✓	✓	−	−	−	−	+	−	−	−	−	−	✓	✓	✓	✓	✓	✓	✓
Markup language	✓	✓	✓	−	−	−	−	✓	−	−	−	−	−	−	−	−	−	−	✓	✓
Context-aware	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Task guidance	−	✓	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Task scheduling	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Formal definition	✓	✓	+	−	−	−	−	✓	−	−	−	−	−	✓	✓	✓	✓	−	✓	✓
	Smart Spaces			Interactive Systems Design										Multi-agent Systems					Workflows	

Table 11: Comparison between the languages in terms of satisfying our proposed properties. −: not meet; +: partially meet; ✓: fully meet.

According to our proposed properties (see Section 4) for a language that is applicable to represent user tasks in smart environments, Table 11 provides a comparison between the languages against these properties. Accordingly, many of the properties are not satisfied by the languages in the user interface design domain except CTT. Even as CTT provides a markup language called TERESAXML, TERESAXML task representations are interpretable but only intended for simulation of the execution of tasks. The lan-

guages for multi-agent systems are like-programming languages, hence interpretable but they are not markup languages and do not support for representing semantics.

As can be seen in Table 11, there is no language that support all operators for task scheduling as discussed in Section 6.2 (hence, they are marked as "partially meet" in the table). There is also no language that provides elements for representing context information which is useful for task recommendation and execution except CATM. Only CE-TASK provides the task guidance.

CE-TASK can be seen as a candidate when it meets most of the proposed properties of a language for smart environments. One of the notable features of CE-TASK is that it allows us to semantically and abstractly specify user-intended concepts within a task representation by the use of ontologies of concepts. This enables reasoning about concepts such as functionalities, input/output data, and any objects involved in executing a task at runtime. However, CE-TASK is lacking of several operators such as *Parallelism*, *Enabling*, and *Try* that can be borrowed from CTT and the other languages. CE-TASK also need elements for including context information in task representations.

7 Conclusion & Future Work

In this paper, we have surveyed representative task languages from different domains and considered their applicability for representing user tasks in smart environments. Specifically, according to advances of pervasive computing technologies and problems faced by users in exploiting these technologies for achieving their routine tasks, we have presented our envisioned task-oriented system (we have called it ToS) for future smart environments in which the user will interact with their environments in terms of their intend goals, not in terms of functions provided by devices or applications within the environments. We have set out requirements for a language that will be used to specify user tasks in smart environments. The survey has focused on two main metrics for comparison between the languages. The metrics are the expressive powers of the languages in terms of task attributes and temporal operators.

Our analysis has shown that while there is no language that satisfies all the proposed set of language properties, a combination of some languages will produce a powerful task description language for our task-oriented system. Because CE-TASK meets most of the requirements, we would like to keep CE-TASK as the core for our future language. Then, we will extend it by: (1) reusing descriptive task attributes from user interface design's languages (especially CTT), (2) reusing operators of multi-agent systems' languages because of the rich set of operators provided by these languages, (3) adding other task attributes and operators on demand during development of real tasks for several common smart places. While there is no dominating language that seems to support most of the operators and attributes, we've also observed the following:

- *Iteration*, *Optionality*, *Pre-condition*, and *Post-condition* are the most common task attributes supported by the languages.
- *Sequence*, *Choice*, *Free-choice*, *Concurrency*, *AnyOrder*, *Enabling*, and *Waiting* are the "should supported" operators.

We have described an vision of advanced smart environments, and based on this vision, we outlined properties of a desirable language to be used for representing tasks in smart environments. We noted that while not all properties is satisfied by the languages we

reviewed, several languages such as CE-TASK do satisfying a majority of these properties. Hence, extensions or combinations of these languages could provide a promising solution for smart environments. A standardised task representation language with the described properties would enable task-based smart environments to be ubiquitous–different smart environments can interoperate and task representations can be re-used across different smart environments.

References

[1] M. Weiser, R. Gold, and J.S. Brown. The origins of ubiquitous computing research at PARC in the late 1980s. *IBM systems journal*, 38(4):693–696, 1999.

[2] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[3] Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, and Trevor F. Smith. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 147–156, New York, NY, USA, 2002. ACM. ISBN 1-58113-515-7. doi: http://doi.acm.org/10.1145/778712.778736.

[4] Maria Cristina Brugnoli, John Hamard, and Enrico Rukzio. User expectations for simple mobile ubiquitous computing environments. In *WMCS '05: Proceedings of the Second IEEE International Workshop on Mobile Commerce and Services*, pages 2–10, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2391-9.

[5] Jeffrey Nichols, Brandon Rothrock, Duen Horng Chau, and Brad A. Myers. Huddle: Automatically generating interfaces for systems of multiple connected appliances. In *Proceedings of the* 19^{th} *annual ACM symposium on User interface software and technology*, pages 279–288, NY, USA, 2006. ACM.

[6] G. Vanderhulst, K. Luyten, and K. Coninx. Pervasive maps: Explore and interact with pervasive environments. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 227–234, 2010.

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

[8] R. Masuoka, B. Parsia, and Y. Labrou. Task computing–The semantic web meets pervasive computing. In *Proceedings of the Second International Semantic Web Conference*, Lecture Notes in Computer Science, pages 866–881, Florida, USA, 2003. Springer Berlin Heidelberg.

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

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

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

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

[13] Gang Pan, Yuqiong Xu, Zhaohui Wu, Laurence Yang, Man Lin, and Shijian Li. TaskShadow: Toward Seamless Task Migration across Smart Environments. *IEEE Intelligent Systems*, 26(3):50–57, May–June 2011. doi: 10.1109/MIS.2010.32. URL `http://doi.ieeecomputersociety.org/10.1109/MIS.2010.32`.

[14] R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin. Ontology-enabled pervasive computing applications. *IEEE Intelligent Systems*, 18(5):68–72, 2003.

[15] Gary Look, Stephen Peters, and Howard Shrobe. Plan-driven ubiquitous computing. In *Artificial Intelligence in Mobile System (AIMS)*, 2003.

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

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

[18] T. Heider and T. Kirste. Supporting goal-based interaction with dynamic intelligent environments. In *Proceedings of European Conference on Artificial Intelligence*, pages 596–600. IOS Press, 2002.

[19] F. Kawsar, G. Kortuem, and B. Altakrouri. Designing pervasive interactions for ambient guidance with situated flows. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 371–375, 2010.

[20] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Implicit interaction design for pervasive workflows. *Personal Ubiquitous Comput.*, 15:399–408, April 2011. ISSN 1617-4909.

[21] C. C. Vo, T. Torabi, and S. W. Loke. Towards a task-oriented framework for smart spaces. In *Proceedings of the 5th International Workshop on Artificial Intelligence Techniques for Ambient Intelligence*, Kuala Lumpur, Malaysia, July 2010.

[22] 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.

[23] Alan Messer, Anugeetha Kunjithapatham, Mithun Sheshagiri, Henry Song, Praveen Kumar, Phuong Nguyen, and Kyoung Hoon Yi. InterPlay: A middleware for seamless device integration and task orchestration in a networked home. In *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 296–307. IEEE Computer Society, 2006.

[24] Henry Lieberman and José Espinosa. A goal-oriented interface to consumer electronics using planning and commonsense reasoning. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 226–233, New York, NY, USA, 2006. ACM. ISBN 1-59593-287-9.

[25] C.C. Vo, T. Torabi, and S.W. Loke. Towards context-aware task recommendation. In *Proceedings of the* 4^{th} *International Conference on Pervasive Computing*, pages 289–292, Taiwan, 2009.

[26] Michael Beigl, Hans-W. Gellersen, and Albrecht Schmidt. Mediacups: experience with design and use of computer-augmented everyday artefacts. *Computer Networks*, 35(4):401–409, 2001.

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

[28] Estefanía Serral, Pedro Valderas, and Vicente Pelechano. Automating routine tasks in ami systems by using models at runtime. In *Proceedings of the First international joint conference on Ambient intelligence*, AmI'10, pages 1–10, Berlin, Heidelberg, 2010. Springer-Verlag. URL `http://dl.acm.org/citation.cfm?id=1926743.1926744`.

[29] B.P. Bailey, P.D. Adamczyk, T.Y. Chang, and N.A. Chilson. A framework for specifying and monitoring user tasks. *Computers in Human Behavior*, 22(4):709–732, 2006.

[30] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[31] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R.H.Campbell, and D. Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *Third IEEE International Conference on Pervasive Computing and Communications*, pages 7–16, Kauai Island, Hawaii, 2005.

[32] A. C. Siochi and H. R. Hartson. Task-oriented representation of asynchronous user interfaces. *SIGCHI Bull.*, 20(SI):183–188, 1989.

[33] H. Rex Hartson, Antonio C. Siochi, and D. Hix. The UAN: a user-oriented representation for direct manipulation interface designs. *ACM Trans. Inf. Syst.*, 8(3): 181–203, 1990.

[34] S. Wilson, P. Johnson, C. Kelly, J. Cunningham, and P. Markopoulos. Beyond hacking: a model based approach to user interface design. In *Proc. of HCI'93*, pages 217–231, 1993.

[35] H. Johnson and P. Johnson. Task knowledge structures: Psychological basis and integration into system design. *Acta Psychologica*, 78:3–26, 1991.

[36] Pedro A. Szekely, Piyawadee Noi Sukaviriya, Pablo Castells, Jeyakumar Muthukumarasamy, and Ewald Salcher. Declarative interface models for user interface construction tools: the MASTERMIND approach. In *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*, pages 120–150, London, UK, UK, 1996. Chapman & Hall, Ltd. ISBN 0-412-72180-5.

[37] P. Szekely, Luo P, and R. Neches. Facilitating the exploration of interface design alternatives: The HUMANOID model of interface design. In *Proceedings SIGCHI'92*, pages 507–515, May 1992.

[38] J. Foley, W. Kim, S. Kovacevic, and K. Murray. *Architectures forIntelligent User Interfaces: Elements and Prototypes*, chapter UIDE - An Intelligent User Interface Design Environment, pages 339–384. Addison-Wesley, Reading MA, 1991.

[39] J. Tarby and M. Barthet. The DIANE+ method. In *Computer-Aided Design of User Interfaces*, pages 95–119, Namur, 1996. Namur University Press.

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

[41] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Comput. Netw. ISDN Syst.*, 14(1):25–59, 1987. ISSN 0169-7552.

[42] Tony Griffiths, Norman W. Paton, Carole A. Goble, Adrian West, Peter J. Barclay, Jessie Kennedy, Michael Smyth, Jo McKirdy, Philip D. Gray, and Richard Cooper. Teallach: A model-based user interface development environment for object databases. In *Proceedings of the User Interfaces to Data Intensive Systems*, Washington, DC, USA, 1999.

[43] Hallvard Trtteberg. *Model-based User Interface Design*. PhD thesis, Norwegian University of Science and Technology, 2002.

[44] H. Uhr. TOMBOLA: simulation and user-specific presentation of executable task models. In *Proceedings of HCI International 2003*, pages 263–267, Crete, Greece, (2003), 2003. Lawrence Erlbaum Associates.

[45] K. Mukasa and A. Reuther. The useware markup language (useml)–development of user-centered interfaces using XML. In *Proc. of the 9th IFAC/IFIPS/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems*, 2004.

[46] Mickaël Baron and Dominique Scapin. K-made user manual. Online, Oct. 2006. URL `http://kmade.sourceforge.net`. v0.1.1.

[47] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, Department of Computer Science, University of Massachusetts, 1995.

[48] B. Horling, V. Lesser, R. Vincent, T. Wagner, A. Raja, S. Zhang, K. Decker, and A. Garvey. The TMS white paper. Multi-Agent Systems Lab, University of Massachusetts, 1999. URL `http://mas.cs.umass.edu/research/taems/white/taemswhite.pdf`.

[49] M. Freed. *Simulating human performance complex, dynamic environments*. PhD thesis, Department of Computer Science, Northwestern University, 1998.

[50] M. Freed. Managing multiple tasks in complex, dynamic environments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 921–927, 1998.

[51] M. Freed and R. Remington. GOMS, GOMS+ and PDL. In *Working Notes of the AAAI Fall Symposium on Simulating Human Agents*, 2000.

[52] Charles Rich and Candace L. Sidner. Collagen: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3-4):315–350, 1998. ISSN 0924-1868.

[53] Jacob Eisenstein and Charles Rich. Agents and GUIs from task models. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 47–54, New York, NY, USA, 2002. ACM. ISBN 1-58113-459-2.

[54] R.G. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of Intelligent Robotics and Systems*, 1998.

[55] Aaron G. Cass, Barbara Staudt Lerner, Stanley M. Sutton, Jr., Eric K. McCall, Alexander Wise, and Leon J. Osterweil. Little-JIL/Juliette: a process definition language and interpreter. In *Proceedings of the* 22^{nd} *international conference on Software engineering*, pages 754–757, 2000.

[56] A. Wise. Little-JIL 1.5 Language Report. Technical report, Department of Computer Science, University of Massachusetts, Amherst, MA, Oct. 2006. URL `http://laser.cs.umass.edu/techreports/06-51.pdf`.

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

[58] WMP Van der Aalst and AHM Ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[59] Anand Ranganathan and Roy H. Campbell. Self-optimization of task execution in pervasive computing environments. In *Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)*, 2005.