



Research Center

NRC-TR-2006-001

Task-Driven End-User Programming of Smart Spaces Using Mobile Devices

Dimitris N. Kalofonos and Franklin D. Reynolds

Nokia Research Center Cambridge

<http://research.nokia.com>

January 10, 2006

Abstract:

Developments in a number of technology industries and the increasing importance of Web Services will soon enable the pervasive deployment of networked devices and services in environments such as homes, public spaces and offices. These “smart spaces” consist of networked devices and services which expose well-defined programming interfaces that allow the creation of distributed applications, through which end-users interact with their smart spaces. Currently, these applications offer the specific functionality that the device manufacturers or software vendors intended to make available. Even though sophisticated software developers or system administrators can create new distributed applications to perform new customized tasks, it is well beyond the ability of non-expert end-users to modify pre-programmed functionality to meet *their own needs*. In this report we propose a research agenda to enable non-expert end-users to “program” their smart spaces. Our goal is to create a framework that will allow end-users to use their mobile devices to instruct their connected devices and services to perform tasks in a way that *intuitively seems possible* and to dynamically *react to events* happening in their environment. As a first step in that direction, we present an overview of the different approaches to end-user programming and to composition of services in pervasive computing environments, as well as our initial position about a framework that would enable intuitive end-user programming of smart spaces using mobile devices.

Index Terms:

end-user programming
smart spaces
user interaction
service composition

TABLE OF CONTENTS

| | | |
|-------|--|----|
| 1. | Introduction | 3 |
| 1.1 | Problem Statement | 3 |
| 1.2 | A Motivating Scenario | 3 |
| 1.3 | Report Organization | 4 |
| 2. | Overview of End-user Programming Approaches | 5 |
| 2.1 | Current Status of End-User Programming Activities..... | 5 |
| 2.2 | New Programming Languages | 6 |
| 2.3 | Natural Language Processing..... | 7 |
| 2.4 | Visual Programming | 8 |
| 2.5 | Direct Manipulation | 9 |
| 2.6 | Programming by Example (PBE) | 9 |
| 3. | Overview of End-user Service Composition Approaches..... | 11 |
| 3.1 | Background on Semantic Web Services | 11 |
| 3.2 | Approaches to Composition of Semantic Web Services..... | 12 |
| 3.3 | Recombinant Computing | 12 |
| 3.4 | Task Computing..... | 14 |
| 4. | Initial Position for Task-Oriented End-User Programming Framework..... | 17 |
| 4.1 | Overview of Proposed Research Agenda..... | 17 |
| 4.2 | Proposed General Architecture | 17 |
| 4.2.1 | User Layer..... | 18 |
| 4.2.2 | Scripting Layer..... | 19 |
| 4.2.3 | Semantic Adaptation Layer..... | 20 |
| 4.2.4 | Core Distributed Middleware Layer | 20 |
| 4.2.5 | Local System Layer | 21 |
| 4.3 | End-User Programming Tools (EUP-tools)..... | 21 |
| 4.3.1 | General Approach | 21 |
| 4.3.2 | “Guides”: Programming Tools for Naïve Users | 21 |
| 4.3.3 | Control and Program Structure | 22 |
| 4.3.4 | Composition and Debugging of Distributed Services..... | 23 |
| 4.3.5 | Platform Requirements of End-User Programming Tools..... | 24 |
| | References..... | 25 |

1. INTRODUCTION

This report aims at presenting a survey of research and technologies that could be used to enable non-expert end-users to program and control their smart spaces. It also aims at presenting some initial thoughts about a framework that would enable end-users to compose their own tasks and program their smart spaces using their mobile devices.

1.1 Problem Statement

Pervasive Computing environments, often referred to as “smart spaces”, are increasingly becoming a reality. The specifications of the Digital Living Network Alliance (DLNA) [1] and the UPnP Forum [2] are some notable examples of technologies that will likely appear in the near future in mass-market consumer electronics products targeting the “digital home”. Furthermore, Web Services [3] are increasing in popularity as related standards mature and are becoming an important part of the pervasive computing landscape. These developments will soon enable the wide availability of networked devices and services in people’s everyday lives in environments such as homes, public spaces and offices.

Networked devices and services expose well-defined interfaces that allow application developers to create distributed applications through which users can interact with their smart spaces. These applications offer the specific static functionality that the device manufacturers or software vendors intended to make available to the end-users of their products. Even though sophisticated software developers or system administrators can create new applications or scripts that would allow them to perform new customized tasks in their smart spaces, it is beyond the ability of non-expert end-users to modify pre-programmed functionality to meet their own needs.

In this report we will focus on research aiming at enabling non-expert end-users to “program” their smart spaces to perform customized tasks that match their intent, as opposed to only the static tasks the device and software vendors have provided. This is a particularly challenging proposition. Non-expert users/consumers today find it difficult to perform tasks using even relative simple programmable devices, e.g. the task of “programming” a VCR. Programming and controlling smart spaces involving a number of connected devices and services represents a significantly bigger challenge for non-expert users. The long-term goal of this research is to address this problem by creating middleware and user-tools, so that people can use their mobile devices to easily instruct their connected devices and services to interact in a way that intuitively seems possible and to dynamically react to events happening in their environment.

1.2 A Motivating Scenario

Bob is a software engineer and tech-enthusiast who has purchased the latest DLNA-certified devices and added them in his home network. He has purchased a Digital Media Server (DMS), a Digital Media Renderer (DMR) and a Wi-Fi-enabled motion-activated security camera. The broadband cable ISP also offers a service to subscribers to access up to five family-member’s location online, and Bob with his wife Helen have set up their mobile phones to synchronize their location with this service. Furthermore, their mobile operator offers a service that allows them to send SMS to their mobile phones from any home computer. All these devices and services came with vendor applications which allow them to watch

content from their DMS on their DMR, to view live video from the security camera to any home computer, to view each other's location from any home computer, and to send SMS messages from any computer in their home network to their mobile phones. Even though they have paid thousands of dollars to purchase their devices and subscribe to services, Bob and Helen feel frustrated when they realize that they cannot use their smart home to perform a security task that intuitively seems simple: they would like their Wi-Fi camera, upon detecting motion in their home when they are away, to start recording video in their DMS and send an SMS message to their mobile phones to alert them. Bob starts looking at the technical specs and finds out that the DLNA devices use UPnP, while the location and SMS services have a Web services interface. After reading about each of these interfaces and downloading a UPnP and Web Services development toolkit, he decides to start developing in his free time a 'smart-home security' application that will run on his PC and perform the following:

- probes the location of their mobile phones every 5 minutes (Web Service),
- if both are away, activates the security camera and subscribes to motion-detection events (UPnP),
- upon receiving a "motion detected" event, it directs video from the camera to the DMS (UPnP),
- finally, it sends an SMS message to both their phones with a time stamp of the event (Web Service).

Weeks into the project Bob finds out about the new software that runs on Nokia smartphones and allows non-expert users to easily program and control their smart homes. He decides to try it out and installs it on his smartphone. He is surprised to discover that he can now easily create his 'smart-home security' application after interacting with his mobile phone for only half hour. The interaction is very intuitive: his phone automatically discovers all the services available and guides him on creating the 'smart-home security' task by responding to questions, choosing from pull-down menus, pointing and touching with his phone some of the home devices, and entering conditions and parameters to customize the program's behavior to match his intent. At the end of this process he has created an application on his phone, which he can launch to activate the 'smart-home security' task. He can also view a visual representation of the task, either on his phone screen or, by using the Nokia PC Suite, on his 32" hi-resolution LCD monitor. He can also edit the phone application that he created and change its behavior, from minor changes (e.g. modifications of some timing parameters) to more extended changes in the definition of the task (e.g. replace the SMS notification part with a new service that allows him to send video clips from sources such as the security camera via MMS).

Bob asks Helen to try to create her own 'smart-home security' application on her mobile phone. They are both pleased to realize that even though Helen has no idea nor interest in smart home technology, she can follow the same intuitive process on her mobile phone and she can easily create her own 'smart-home security' application. She feels so comfortable with the process that she decides to buy a second Wi-Fi camera and create a new 'baby alarm' application to monitor their 6-month-old baby at night.

1.3 Report Organization

The rest of this report is organized as follows: in Section 2 we present an overview of the different approaches to end-user programming; in Section 3 we present an overview of research that enables end-users to compose and execute complex tasks in pervasive computing environments; finally, in Section 4 we present our initial position about an end-user framework to intuitively "program" smart spaces using mobile devices.

2. OVERVIEW OF END-USER PROGRAMMING APPROACHES

“End-user” programming covers a broad range of technologies and research topics. It is not, strictly speaking, limited to traditional notions of “programming”. Over the years, considerable research into the various related topics has been conducted by a large number of researchers. End-user programming has had some significant successes in application specific domains, but despite these efforts, many significant challenges must be dealt with before general purpose end-user programming will become popular.

The various attempts to create solutions to this problem are collectively referred to as end-user programming even though little or no actual programming may be involved. A wide range of approaches have been experimented with over the years. Most of the research can be grouped into one of the following categories:

- New Programming Languages
- Natural Language Processing
- Visual Programming
- Direct Manipulation
- Programming By Example (PBE) or Programming by Demonstration (PBD).

In the rest of this section, we first present the current status of End-User Programming activities and we provide a brief survey of different approaches proposed in each of the above categories.

2.1 Current Status of End-User Programming Activities

“Every week or so, I record my latest credit card charges in Dollars and Sense, a home accounting program. Before I can enter any charges, I have to do the following: I select the "Edit Transactions" command from a menu. A dialog asks me for the Funding Account and the date range. I scroll through a list of about 20 accounts to find "MasterCard". Then I type in the first day of the current month as the "From" date. Had this program been written specifically for me, it would have had a button labeled "Add MasterCard Charges" that would perform all of these steps. But since this is a generic program, intended for thousands of people with varying accounting needs, an "Add MasterCard Charges" button would not make sense. As a result, I am stuck performing a whole sequence of actions instead of just one, because I am using a generic program to perform a specific task.

This leaves personal computer users in an ironic situation. It is a truism that computers are good at performing repetitive activities. So why is it that we are the ones performing all of the repetition, instead of the computer?”

Allan Cypher from the introduction to “Watch What I Do”

”Today, we have windowed interfaces everywhere, and even a number of iconic object-construction kits. We have macro capture systems of every kind, and scripting languages. But we don't have "end-user programming".”

Alan Kay, Foreword to the book, “Watch What I Do”

“Just read Alan Kay's [Smalltalk history](#). An odd sensation to be reminded of the mood of 1970's computing, and of why I went into this field in the first place, and of

how the vision of computing as "amplifier of human reach" has wilted. I still feel that the "end users" should be customizing programs and writing their own in order to amplify whatever they variously care about and make the computer their servant; why this doesn't happen (basically the "why is programming so hard" question) is still a mystery to me."

From Johnathan Rees' blog, 2004-07-17

Today computing has thoroughly permeated modern industrial society. Even our telephones are customizable computing platforms. The goal of end-user programming research is to make it possible for non-programmers to create their own, custom commands. Yet, after more than 20 years of research, satisfactory general purpose solutions which enable end users to create their own custom commands do not exist. This is not for the lack of effort by the research community. In 2002 and 2003 the EUD-Network [4] hosted a series of workshops on "End-User Development" and in 2003 SIGCHI hosted another workshop on End-user Development [5] attended by many of the leading researchers in the field. The next End-User programming workshop, WEUSE II, The Next Step: From End-User Programming to End-User Software Engineering [6], will be held at CHI 2006 in Montréal, Quebec, Canada, April 23, 2006.

The CHI workshops are associated with the EUSES Consortium [7]. This organization is a collaboration by researchers at [Oregon State University](#), [Carnegie Mellon University](#), [Drexel University](#), [Penn State University](#), [University of Nebraska](#), and [Cambridge University](#) whose goal is to develop and investigate end-user software engineering technologies for enabling End Users to Shape Effective Software. Clearly, end-user programming is an active area of research.

2.2 New Programming Languages

There is a long history of research into programming languages that are suitable for end-user programming. Different approaches have been explored with varying amounts of success including very simple languages, very high level languages, special purpose languages and attempts to make computer languages more like human languages.

Some researchers have explored the use of logic, rules and constraint-based languages. While this work has generated interesting results ([8] provides a useful survey of constraint based programming), ultimately these techniques have limited appeal to end-users. Part of the problem seems to be a lack of computational efficiency (Prolog and similar languages are not known for their speed of execution) and part of the problem seems to be expressiveness – if the language is expressive enough to be generally useful then it is too complex for end-users. It is worth noting the successful use of simple rules for email SPAM filters. However, this is hardly an example of general purpose programming.

Very high level programming languages, such as SETL and Python significantly simplify programming. Special purpose languages like 4G database programming languages and SAS or SPSS (special languages for statistics) have proven to be very usable by knowledge workers who were not professional programmers. Mathematica, MathCad, Maple and other systems have made mathematical programming accessible to large numbers of non-programmers. In most cases, these languages have been very successful at making knowledge workers more productive. They have had much less impact outside of professional or educational situations.

Simple programming languages, especially languages intended for children have long been studied. One of the earliest and most notable of these programming languages for children is Logo. Logo was originally developed by Daniel Bobrow and Wallace Feurzeig at Bolt, Beranek and Newman, Inc., and Seymour Papert, at the Massachusetts Institute of Technology in the 1960's.

When Logo was first built, a critical aspect was taking the computational constructs of the Lisp programming language and designing a child friendly syntax for them. Lisp's "CAR" was replaced by "FIRST", "DEFUN" by "TO", parentheses were eliminated, and so on. Over the years, Logo has continued to enjoy reasonable success. Educational products based on Logo can still be purchased. Much more sophisticated variants of Logo have been developed including variants suitable for professional programmers. Logo continues to inspire research into programming languages and tools for kids, such as Lego MindStorms, a moderately successful robot construction and programming toy kit. Squeak is a version of Smalltalk for kids that can be used to control small mobile robots. The legacy of Logo and MindStorms in Squeak is clear. Scratch is a new system under development at MIT based on Squeak. Scratch emphasizes ease on development and adds visual programming aids to Squeak. Alice and ToonTalk [9] are graphical programming environments intended for use by children.

Researchers have clearly demonstrated that with tools such as Logo, even young children (7-8 years old) can learn to write computer programs. Unfortunately, evidence also suggests that even though most people can learn to program, most people prefer not to program – at least using the tools currently available.

2.3 Natural Language Processing

“The degree to which human, or "natural" language is unlike computer code cannot be overemphasized.” *Jaron Lanier, Cato Institute*

Natural Language programming research is an attempt to teach computers to understand human language. This is not to be confused with attempts to make computer languages understandable by humans. Cobol, HyperTalk and other computer languages have been developed with a syntax that was somewhat closer to a human language than C or Fortran. These languages are all essentially computer languages with only a somewhat human friendly syntax.

Lillian Lee from Cornell University [10] describes Natural Language Processing as the field of computer science devoted to enabling computers to use human languages both as input and as output. She points out that the area is quite broad, encompassing problems ranging from simultaneous multi-language translation to advanced search engine development to the design of computer interfaces capable of combining speech, diagrams, and other modalities simultaneously.

Research in this field can be easily traced back to the 1960's – or even earlier. The eventual goal is to be able to communicate with a computer in the same way we communicate with each other. Despite the predictions of various futurists (for example Stanley Kubrick and his famous film 2001 with HAL the mutinous AI), the current state of the art is somewhat limited. In her survey of the field paper titled, “I'm sorry Dave, I'm afraid I can't do that.: Linguistics, Statistics, and Natural Language Processing circa

2001” [10], she suggests that the core cause of the limited progress in NLP lies with the fundamental challenge of resolving ambiguity.

It is worth noting that most researchers clearly differentiate between natural language processing and speech recognition. Speech recognition usually focuses on recognizing individual spoken words or phrases. Galaxy [11] is an example of such an application. NLP focuses on understanding the semantics of human language. To simplify and focus the research efforts, NLP research often uses written language rather than spoken language. Start [12] is an example of such a system.

The Start Server (<http://start.csail.mit.edu/>) provides users with access to multimedia information in response to questions formulated in English. It has been in more or less continuous operation since 1993. Start seems to do a pretty good job of understanding simple questions expressed in English. It often cannot answer the question because of a lack of domain knowledge. For example, even though Astronomy is one of the suggested topics (and Start has been around for more than 10 years), Start does not appear to know what a Kuiper object is.

Even with the built in simplifications possible by limiting the problem to understanding simple questions, this example illustrates the challenge of building a natural language processing system with enough domain knowledge to be useful.

2.4 Visual Programming

Visual Programming is:

- A programming language that uses a visual representation (such as graphics, drawings, animation or icons, partially or completely)
- A visual language manipulates visual information or supports visual interaction, or allows programming with visual expressions
- Any system where the user writes a program using two or more dimensions
- A visual language is a set of spatial arrangements of text-graphic symbols with a semantic interpretation that is used in carrying out communication actions in the world.

Despite (or perhaps because) of extensive research into the use of visual programming aids, the value of Visual programming is surprisingly contentious. In a well-known article Fred Brooks says this:

“A favorite subject for PhD dissertations in software engineering is graphical, or visual, programming - the application of computer graphics to software design.... Nothing even convincing, much less exciting, has yet emerged from such efforts. I am persuaded that nothing will.”

Then there is the so-called “Deutsch Limit” attributed to Peter Deutsch:

“The problem with visual programming is that you can't have more than 50 visual primitives on the screen at the same time. “

One area where visual programming systems have enjoyed success is in tools to design and develop user interfaces. Several different professional application development toolkits provide GUI builders.

Another active area of research is the use of VPL systems for children. Relevant research projects include ToonTalk, Alice and Scratch. ToonTalk, developed by Ken Kahn is a general-purpose programming system for kids. It attempts to make programming a playful activity. Source code is animated and the programming environment is a video game. Every abstract computational aspect is mapped into a concrete metaphor. For example, a computation is a city, a concurrent object is a house, birds carry messages between houses, a method or clause is a robot trained by the user and so on. The programmer controls a "programmer persona" in this video world to construct, run, debug and modify programs.

Just this year, Apple released with its new operating system, MacOS 10.4, a graphical scripting tool called Automator [13]. Automator has generated a lot of user interest including an active online community (Automator World - <http://www.automatorworld.com/>) and a couple of books (Mac OS X Technology Guide to Automator by Ben Waldie and Automator for Mac OS X Tiger by Ethan Wilde). Automator is much too limited to be considered a general purpose visual programming tool – it is interesting to consider whether it is quickly becoming popular despite its limitations or because of its limitations.

2.5 Direct Manipulation

Excel, Word, SIMCITY, Opening Night, and other WYSIWYG applications are examples of Direct Manipulation. Direct Manipulation has been very successful but it requires explicit and ongoing user interaction. Alone, direct manipulation does not provide sufficient programmability to meet our needs. However, coupled with Programming By Example, sometimes called programming by demonstration, it is a powerful tool.

2.6 Programming by Example (PBE)

“The motivation behind Programming by Demonstration is simple and compelling: if a user knows how to perform a task on the computer, that should be sufficient to create a program to perform the task. It should not be necessary to learn a programming language like C or BASIC. Instead, the user should be able to instruct the computer to "Watch what I do", and the computer should create the program that corresponds to the user's actions. “
Alan Cypher from the Introduction to the book, “Watch What I Do”

Programming by Example (PBE) encompasses a number of approaches to creating programs by giving examples of their behavior or effect. All approaches emphasize working on concrete examples rather than describing a procedure in the abstract.

Macro recording and replay is a very simplified variation of PBE. Macro recorders have been implemented in Emacs (Emacs keyboard macros), Microsoft Office and other applications, such as Statistica. Macro recorders do not usually provide methods to generalize recorded actions. Though not required, it is interesting to note that Emacs and Word macro recording is integrated with a scripting language. Emacs uses Lisp and Office and Statistica use Visual Basic.

The earliest approach, historically called Automatic Programming but now often referred to as Programming by Input/Output Examples, relies on the system's ability to generalize from examples. The

programmer presents examples of data before and after processing; the computer must find a method of transforming the input to the output.

Programming by example (PBE) or sometimes programming by demonstration (the user demonstrates examples to the computer) is radically different from the traditional process of programming. In this approach, a software agent records the interactions between the user and a conventional “direct-manipulation” interface and writes a program that corresponds to the user’s actions. The agent can then generalize the program so that it can work in other situations similar to, but not exactly the same as, the examples on which it was taught. Macro recorders, such as Emacs keyboard macros, can be thought of as a sort of PBE-lite. It is the generalization capability of PBE systems that marks the most important difference between a PBE system and simple macro recorder.

“Your Wish is My Command”, edited by Henry Lieberman [14] is an introduction to PBE technology and the current state of PBE research. “Watch What I Do”, edited by Allen Cypher is another book on the same topic, with many of the same authors, published about 10 years earlier. Both books are valuable collections of papers describing many PBE research projects.

Another recent approach is Programming by Teaching (sometimes called Instructible Systems or Agents), which extends the demonstrational approach by enabling users to give verbal or gestural hints. These focus the system's attention on relevant generalizations.

It would appear the PBE technology has been widely deployed and adopted by the computing industry. However, with the exception of Emacs users and Emacs keyboard macros, it is hard to find people who have used any of these technologies. It is curious that despite the widespread deployment and availability of macro recording in Microsoft Office, most users seem to be unaware of the feature.

It may be that the first successful commercial market for PBE systems will be tools for children. Most children are not committed to traditional programming tools - interactivity and usability are of greater importance. Stagecast Creator by Smith, Allen, and Tesler [15] and ToonTalk by Ken Kahn [9] are examples of PBE tools for kids.

3. OVERVIEW OF END-USER SERVICE COMPOSITION APPROACHES

There has been an intense research interest recently in the area of end-user composition of services in pervasive computing environments to achieve user-perceived tasks (e.g. [16], [17]). Most research focuses on the automatic or semi-automatic (i.e. user-assisted) composition of semantically annotated Web Services (e.g. [18], [19]). Two other notable approaches in this area include the “Recombinant Computing” approach developed in the Xerox PARC [20] and the “Task Computing” approach developed jointly in the Fujitsu Laboratories of America and the MIND Lab of the University of Maryland [21]. In the rest of this section we present more information about these technologies and a survey of related literature.

3.1 Background on Semantic Web Services

Web services [3] are software components developed using different programming languages and deployed on any platform, that are accessible using standard Web technologies. Each web service exposes an XML interface that describes its public operations and its access details specified using the Web Services Description Language (WSDL) [22]. A Web service communicates using the Simple Object Access Protocol (SOAP) [23] on top of Internet protocols. Publishing and Service Discovery of Web Services is achieved through the usage of the Universal Description, Discovery and Integration (UDDI) protocol [24]. UDDI creates a standard interoperable platform that enables applications to dynamically find and use Web services over the Internet.

Although initially the standards developed for web services targeted mainly their syntactic description, the DAML program soon developed the Ontology Web Language for Services (OWL-S) [25]. OWL-S supplies Web service providers with a markup language for describing the properties and capabilities of their Web services in a computer-interpretable form, therefore enabling the automation of tasks, such as automated Web service discovery, execution, and composition [25]. OWL-S service descriptions are composed of three parts: the *service profile*, the *process model* and the *service grounding* (quoting from [17]):

- **OWL-S service profile:** The service profile gives a high level description of a service and its provider. It is generally used for service publication and discovery. The service profile is composed of three parts: (i) an informal description of the service oriented to a human user; (ii) a description of the service’s capabilities, in terms of Inputs, Outputs, Preconditions, and Effects (IOPE); and (iii) a set of attributes describing complementary information about the service.
- **OWL-S process model:** The process model is a representation of the external behavior of the service as a process. This description contains a specification of a set of sub-processes that are coordinated by a set of control constructs (e.g. Sequence or Parallel constructs); these sub-processes are atomic or composite. The atomic processes correspond to WSDL operations. The composite processes are decomposable into other atomic or composite processes by using a control construct.
- **OWL-S service grounding:** The service grounding specifies the information that is necessary for the service invocation, such as the protocol, message formats, serialization, transport, and addressing information. It is a mapping between the abstract description of the service and the concrete information necessary to interact with the service. The OWL-S service grounding is based on WSDL.

Finally, the development of the Business Process Execution Language for Web Services (BPEL4WS) [26] further facilitated the automatic Web service integration, by providing a language for the formal specification of business processes and business interaction protocols.

3.2 Approaches to Composition of Semantic Web Services

The availability of the above standardized technologies allowing the syntactic and semantic description and composition of Web Services has spurred significant research efforts to create frameworks and tools that allow the composition of semantically annotated Web services to achieve user-specified tasks. In general we can distinguish these approaches in two categories: automatic composition and semi-automatic or interactive composition.

- **Automatic** web services composition approaches [17], [19], [27], [28], [29], [30] in general propose user interaction methods and tools that capture the user's intent and translate it into some abstract semantic service description (user-task). Then after discovering the available web services, they try to compose a composite service which semantically matches as much as possible the user-task according to certain criteria and matching algorithms. This composite service is then executed to perform the user task. Of course, given the complexity and heterogeneity of the various pervasive computing environments, it is likely that such approaches will lead to a composite task which may be (significantly) different from the task the end-user originally intended. On the other hand, the benefit of these approaches is that they require in general less user interaction, thus putting less demanding requirements on non-expert end-users and allowing them to express their intent in a more vague/natural manner. An interesting enhancement of these approaches which uses context information was proposed in [31]. Termed 'Context-Aware Service Composition', this work proposes using context information in the composition phase to yield potentially different composite tasks, given the user's current context.
- **Semi-automatic or interactive** Web service composition approaches [32], [33], [34] in general propose user interaction methods and tools that capture the intent of the user interactively and continuously during the composition process of the composite task. The semantics of the available services are used by these systems to guide the user and limit the available choices.

Finally, interesting composition and execution tools for end-user composition of services can be found in [30], [35], [36], [37].

3.3 Recombinant Computing

The Recombinant Computing approach [20], [38], [39], [40], [41], [42] was developed in Xerox PARC as part of projects 'SpeakEasy' and later 'Objé'. According to [20], it uses a model in which each computational entity on the network is treated as a separate component. The presence of new components is detected through the use of dynamic discovery protocols, and it relies upon a **mobile code** framework to deliver the implementations of components needed at runtime (Figure 1). According to [42], central to the recombinant computing approach is the notion that we cannot expect applications to be expressly written to have prior knowledge about all devices they may encounter. But we can expect users to have knowledge about the devices they encounter in their environments. In the recombinant approach, users themselves must be the final arbitrator of the semantics of the entities they are interacting with.

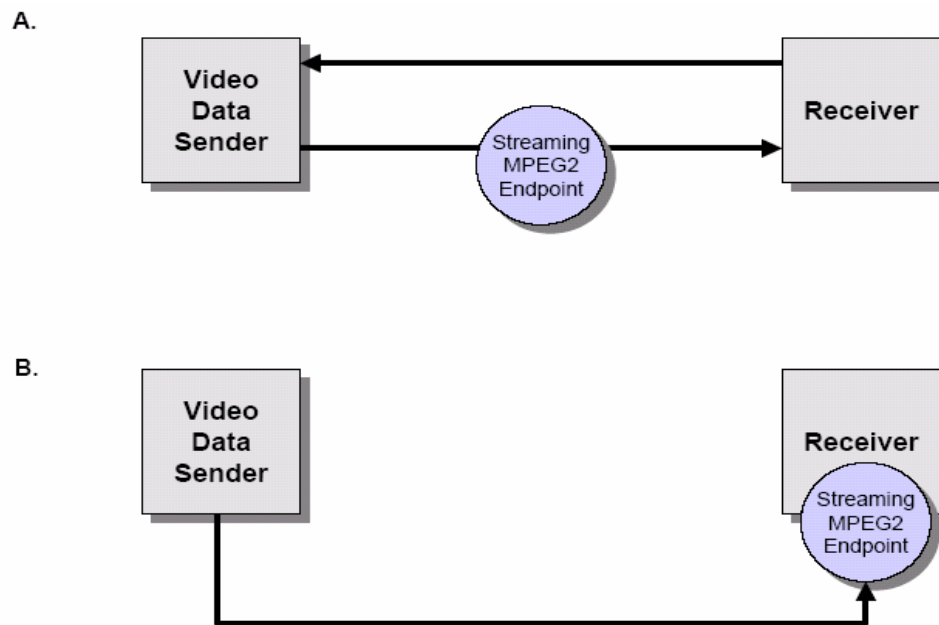


Figure 1: Example of a data transfer service in recombinant computing [20]. The receiver, after discovering the recombinant interface of the data source, retrieves mobile code (A) and begins using the service (B).

The recombinant computing approach creates **semantically neutral interfaces** (called recombinant interfaces) that allow entities to interact in at least a very basic way without having to rewrite code [20]. Recombinant interfaces are programmatic interfaces that specify the ways in which components interact with one another. These interfaces describe the ways in which components can extend one another's behavior at runtime, rather than describing the domain-specific functionality of the components (e.g. printing or file storage).

Of particular interest in the recombinant computing approach is the use of **task-oriented templates** [42]. According to [42], these are “prototypes” of common tasks that can be created and shared by users. Templates contain a partially specified set of connections and components to be fully specified and “instantiated” by users at the time of use. For example, a template named “Give a presentation” might have slots for a file to be displayed and a projector to display it on, as well as for controlling the room lighting, the projector, speakers, and a file viewer (Figure 2). Slots in a template can refer to fully specified components or can be defined generally, using constraints. Templates must be instantiated by users by filling in their slots to make them concrete. If a constrained slot is not fully specified, a selection from components that match the constraints must be made. The framework supports the creation of new templates by example. Users can modify existing tasks by adding and removing connections and components, as well as by changing the components with which the slots are instantiated. At any point, tasks (as well as connections made directly) can be saved as templates. When saving, users are presented with a dialog that allows them to (optionally) generalize each slot according to constraints such as “accept components whose type is ‘Projector’” or “accept components whose owner is the current user.”

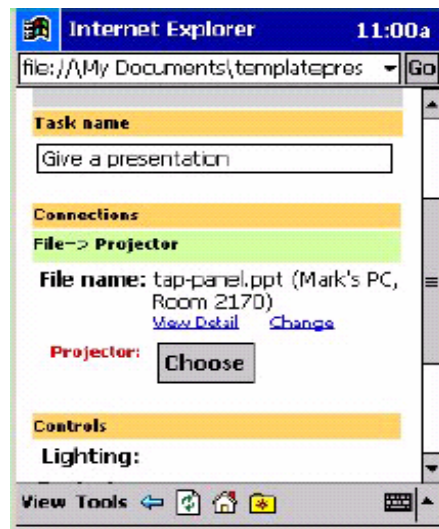


Figure 2: Example of a 'Give a Presentation' task template [42].

3.4 Task Computing

The Task Computing approach [21], [43], [44], [45], [46], [47] was developed in collaboration by the Fujitsu Laboratories of America (FLA) and the MIND Lab of the University of Maryland. According to [45], Task Computing aims to empower non-expert users with the ability to perform complex tasks in information-rich, device-rich, and service-rich environments using mobile devices. Its approach is to expose the functionality in such environments as Semantic Web services, which in turn the user can discover and arbitrarily compose. In Task Computing the functionality exposed as Semantic Web services generally emanates from four different types of sources: networked devices (e.g. UPnP devices), applications (e.g. media player), content (e.g. music file, PIM contact), and web services.

The general architecture of a Task Computing Environment (TCE) is shown in Figure 3. A brief description of the four Task Computing layers follows (quoting from [44]):

- **Realization Layer:** The bottom most layer encompasses the universe of devices, applications, e-services and content, where all functionality available to the user originates.
- **Service Layer:** These various sources of functionality are made computationally available as services, in the sense that service interfaces are employed to access (execute) this functionality. Each service is associated with at least one semantic description, which sometimes may be created on-the-fly as services might be created dynamically. Services are the abstraction of functionality in the Task Computing universe, and semantic descriptions of these services are meant to shield the user from the complexity of the underlying sources of functionality and make it easy (hopefully trivial) for the user to employ these sources in accomplishing interesting and complex tasks.
- **Middleware Layer:** These goals are enabled by the middleware layer components that are in charge of discovering services, deciding how services can be composed, executing services and monitoring service execution, and finally enabling and facilitating a variety of management tasks, including the creation and publishing of semantically described services.

- Presentation Layer:** The most important aspect of Task Computing is the presentation layer, which uses the capabilities of the layers below in order to provide the user with a “Task” abstraction of the complexity of whatever lays underneath. Task Computing researchers have developed a variety of clients for that purpose, such as voice, textual and graphical interfaces, (referred to as Task Computing Clients (TCC)), and a web-based interface (utilizing a web browser). The presentation layer presents to the user an environment where functionality that can be transient and dynamically created can be assembled (in real time) to perform users’ tasks. Since the middleware layer components expose well-defined service API’s, it is possible to create custom applications in the presentation layer in any development environments that can invoke Web Services.

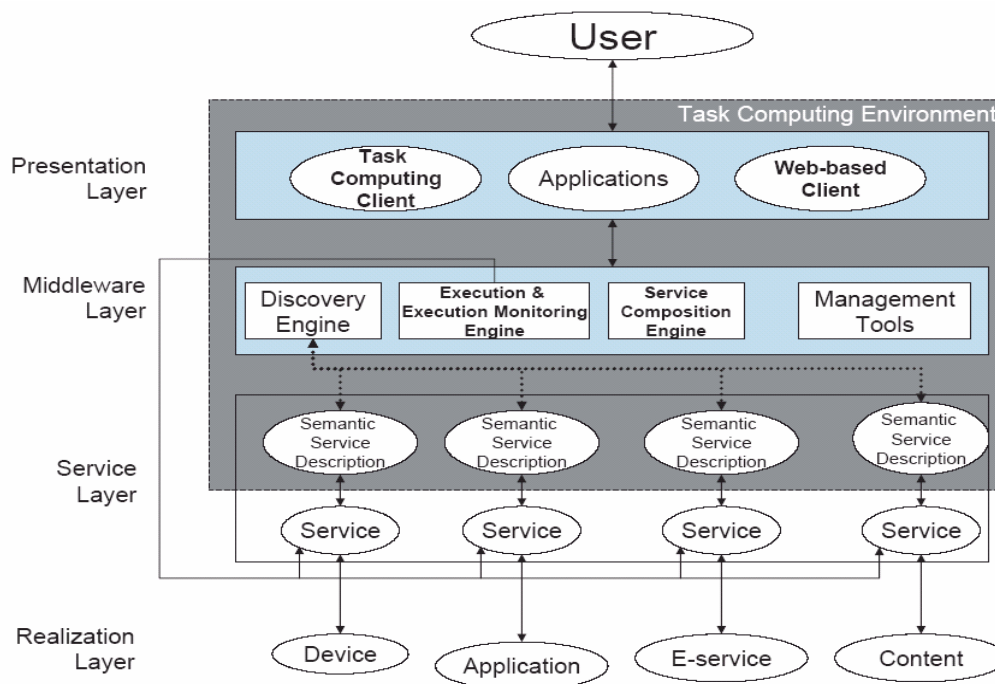


Figure 3: The general architecture of a task computing environment [44].

Task Computing assumes that the vendors of networked devices (e.g. UPnP-enabled devices) and Web services will provide a globally agreed upon semantic description of their services and their interconnection. For services from the other two source types (i.e. local content and applications), the authors have created a user level tool called ‘White Hole’ [44], [45], [46] which “semantizes” them, i.e. creates semantic descriptions based on their respective types. According to [44], Task Computing defines a Semantic Service Discovery Mechanism (SSDM) which uses the underlying Service Discovery mechanisms offered by the Middleware Layer (e.g. UPnP SSDP) to discover Semantically Described Services. For example, UPnP’s discovery mechanism is used to find the UPnP devices on the subnet (not all of which are Task Computing-enabled services) and for each UPnP device, the Task Computing Client invokes one specific UPnP action (*getDescriptionURL*) to determine if the UPnP device represents a Task

Computing-enabled service. If so, the TCC proceeds to download the Semantic Service Description (SSD) from the UPnP device.

Finally, the researchers of the Task Computing approach have developed an embodiment of the various modules of a Task Computing Environment (TCE): a Task Computing Client (TCC) called 'STEER' (Semantic Task Execution EditoR), a tool to "semantize" of content/application objects called 'White Hole', and a tool to "servicize" the semantic objects and publish them called 'PIPE'. Software to experiment with a TCE can be made available to researchers from academic and non-profit research institutes on a non-commercial basis from Fujitsu [21].

4. INITIAL POSITION FOR TASK-ORIENTED END-USER PROGRAMMING FRAMEWORK

4.1 Overview of Proposed Research Agenda

We propose a research agenda with the long range goal of developing technology to enable end-users to program their phone and their personal networks, i.e., smart homes, smart offices, etc. It should not be necessary to use a PC or other computing platform to program the phone or the smart space – users should be able to create their “programs” using only a mobile phone.

It is clear from the survey of the literature and current best practices in end-user programming that these are very ambitious goals. In 1990, “The Programmer’s Apprentice” [48], was published and today professional programmers are still using tools very similar to the tools they were using 20 years ago. Though there have been some advances in the 10 years since Pane and Myers [49] described the state of the art of end-user programming in their report, “Usability Issues in the Design of Novice Programming Systems”, the tools available for end-user programming have not dramatically improved. There have been new versions of different approaches but no breakthroughs. Squeak follows in the footsteps of Logo, Scratch is built on top of Squeak and adds some elements of Visual Programming, Animator is a recent commercial tool from Apple for MacOS 10.4 to simplify scripting that is based on some Visual Programming ideas, the Start and Galaxy natural language processing systems from MIT illustrate how much progress has been made and how far we have to go.

We suggest an incremental strategy with milestones that lead to the final goals rather than an all or nothing approach. For example: programming tools intended to be used by hobbyists or children, such as HyperCard or Logo have enjoyed considerable success over the years and may appeal to a broader audience than programming languages intended for professionals such as Java or Python. Targeting hobbyists might be a useful intermediate step on the path to eventually providing programming tools for all end-users.

We expect that the limitations of the user interface of a mobile device, such as a phone or a PDA, will ultimately drive the research in new directions. The limited keyboard will hamper the sort of text input usually associated with traditional programming and the small displays will limit the amount of text or graphics that can be displayed. This will obviously limit the usefulness of Visual Programming techniques. A new approach that can exploit the limited keyboard and graphics, mobility and perhaps new, special features of a mobile device is called for.

A research agenda that targets end-user programmable smart spaces has important advantages. Smart Spaces emphasize mobility – users move from one space to another – and the importance of mobile users make Smart Spaces a very interesting application domain. And despite the last several years of hype, end-user customization of smart spaces is a relatively new field. There are no significant commercial products to compete with and only limited prior research that directly deals with the special problems of distributed programming for end-users.

4.2 Proposed General Architecture

This section describes our initial thoughts on the general architecture of an End-User Programming (EUP) framework. The proposed general architecture is influenced by ideas found in research presented in

Sections 2 and 3, especially “Task Computing” [21], “Recombinant Computing” (task-oriented templates) [20], [42], and Programming by Example (PBE) [14].

The general architecture of the proposed EUP-framework is shown in Figure 4 and can be distinguished in five layers: (i) the user layer, (ii) the scripting layer, (iii) the semantic adaptation layer, (iv) the core distributed middleware layer, and (v) the local system layer. In the rest of this section we give a brief description of each of these layers. Even though this architecture could be implemented in any host, our focus is on mobile devices running either Symbian (e.g. Nokia Series60 phones) or Linux (e.g. Nokia 770 Tablets).

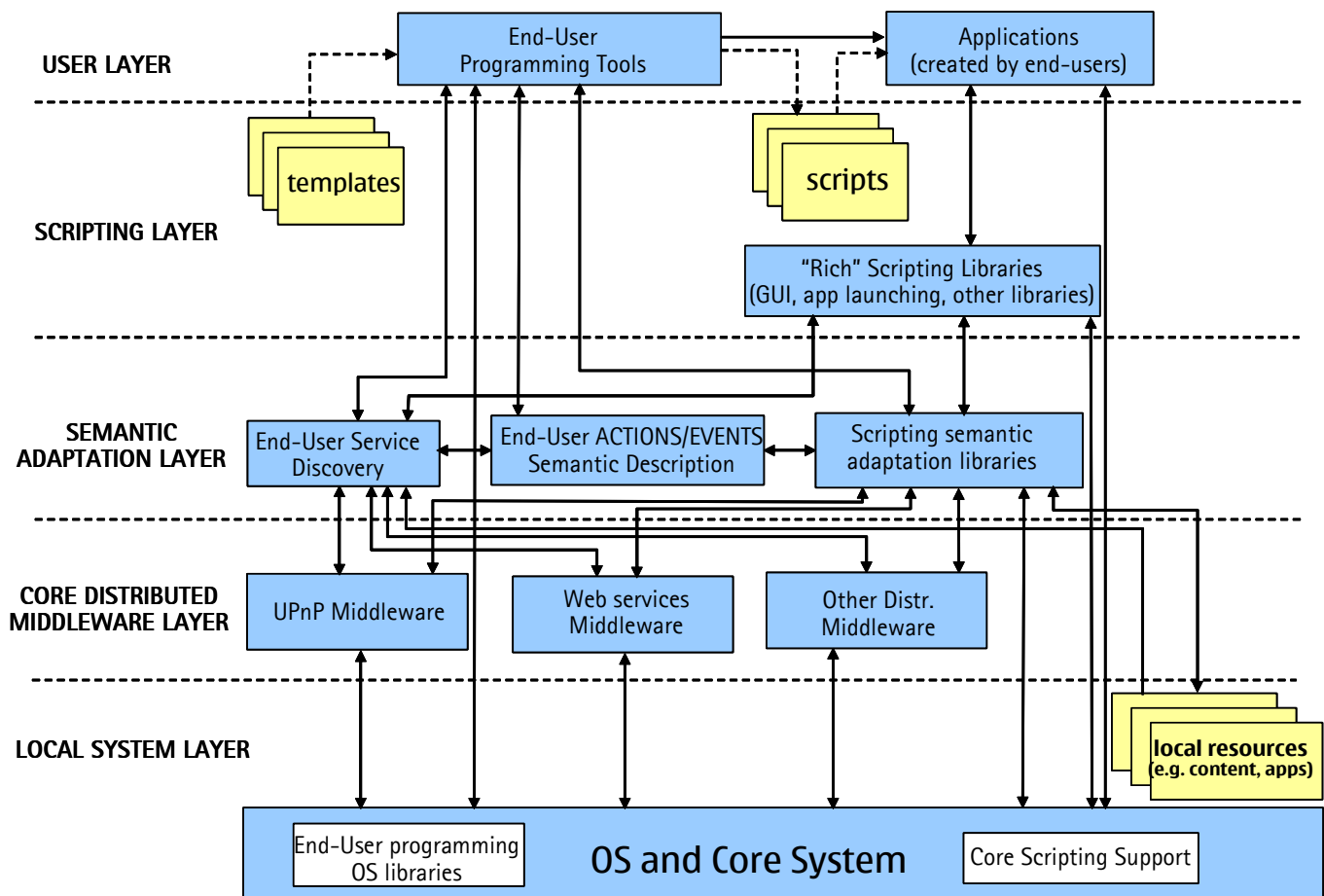


Figure 4: A high-level architecture of the proposed end-user programming framework.

4.2.1 User Layer

The user layer is responsible for the interaction with the end-user. Its role is to capture the user’s intent and communicate the results of the user-actions in a manner that is intuitive and easily understood. Users interact with the *End-User Programming tools* (EUP-tools) to create new customized applications and

scripts that are used to perform their intended tasks. The applications are native applications that provide an appropriate front-end that invokes the scripts composed by the users. For example, as a result of the user interaction with the EUP-tools, an application front-end is chosen among a pool of possible front-ends and installed by the phone's installer. When this application is later launched, it invokes the scripts that were also created by the EUP-tools and which contain the specific functionality composed by the user. The EUP-tools may also use task-oriented script templates to facilitate more meaningful final task compositions. In the process of guiding the user to compose the scripts, the EUP-tools use user-level service discovery and "action/event" descriptions as exposed by the semantic adaptation layer. The EUP-tools may also use the semantic adaptation scripting libraries to gain access to other core distributed middleware functionality.

Ideas about how the EUP-tools could interact with the users to compose the scripts and final applications are examined in Section 4.3.

4.2.2 Scripting Layer

The end-user programs are essentially scripts, which may have a native front-end as described in 4.2.1. The scripting layer may contain *scripting templates* which prescribe certain compositions of available services similarly to the task-oriented templates in [42]. These templates may be created by the provider of the EUP-framework (e.g. Nokia through "Forum Nokia"), can be downloaded by the end-users from 3rd party developers or hobbyist communities, or even created by the end-users themselves. For example, a user may search with Google for a "home-automation" template available for the Nokia EUP-framework and download it to her phone. The EUP-tools will load this template and use the results of the framework's service discovery to customize it and create the final "home-automation" program for her home.

The scripts composed by the EUP-tools (possibly using the scripting templates mentioned above) prescribe the actual functionality to perform the user-tasks. We propose using scripting to create end-user programs in order to avoid on-the-fly compilation of native applications. However, it would be interesting in the future to explore the possibility of the EUP-tools creating actual native source code or detailed specifications for native code, whose compilation would be "outsourced" elsewhere in the user's network (e.g. a home PC) or some remote network service (e.g. an "application building" service provided by "Forum Nokia"). In order for the scripts to provide adequate functionality, we propose that enough native functionality be made available to them through "rich scripting libraries" that would allow scripts to access extensive platform facilities, such as GUI, application launching, inter-process communication, networking etc. These "rich scripting libraries" would also make available to scripts the functionality exposed by the semantic adaptation layer, i.e. end-user service discovery and the scripting semantic adaptation libraries (see Section 4.2.3).

For scripting support we initially consider using Python, but it would be interesting to consider other languages such as Logo or HyperCard, that have proven to be especially easy for novice programmers. Eventually, we would like to investigate the merits of a new scripting language that is optimized for: novice users, authoring scripts on a mobile, handheld device, and safety. The last point is very important. Scripting languages have proven to be very useful for applications that need mobile code. However, Python and most other scripting languages are not intrinsically safe. They cannot be used to build safe

mobile applications without special operating system support. Clearly, even if end-users use their mobile phones to create their new commands, they will not want these new commands to be limited to running on their mobile phones. Some method for pushing portions of the newly created programs to other parts of the smart home will be needed.

4.2.3 Semantic Adaptation Layer

The semantic adaptation layer is responsible for making semantic translations between the concepts perceived by the end-users and the functionality provided by the underlying system. At the heart of this layer are the *end-user semantic descriptions of actions and events*, whose role is two-fold:

- they describe how low-level actions and events (e.g. distributed based on SOAP and GENA, or local based on local file contents or local application functionality) are mapped to the user-level concepts that are exposed to end-users. More specifically, they describe how the *end-user service discovery* and *scripting semantic adaptation libraries* should translate core distributed middleware functionality or local resources to user-perceived concepts. For example a user-level action may be “upload video to media server”. If the core middleware is based on UPnP, these semantic descriptions prescribe how this user-level action is translated to a series of UPnP SOAP actions that would implement it. Furthermore, the scripting semantic adaptation libraries would expose an API call of the form *upload_video(file_name, target_destination)*. The user composed scripts would be able to use this API call.
- they describe how user-level actions/events can interact with each other to compose higher-level tasks. This functionality is used by the EUP-tools to guide the user in the composition of tasks, by limiting his/her available choices or by suggesting alternatives. We note here that a fully-automatic end-user task composition would require a very complex and complete definition of the interactions of all available services, which may be infeasible. However, we envision a more user-interactive composition approach, where the user will be able to infer many of the possible service interactions based on the intuitive concepts exposed by the EUP-tools. This is consistent with the “recombinant computing” approach [20], [42].

Finally, it is important to note that we propose a local-scope approach to the definition of service semantics. Even though it would be desirable to achieve global agreement through standardization about how to expose services to users and how these services interact with each other, it is not required in our approach. Assuming a standardized interface to the core distributed middleware layer (e.g. UPnP, DLNA, web services, etc.), we think it is possible to achieve meaningful end-user programming functionality by using only semantics provided by the EUP-framework provider (e.g. Nokia). Of course, in order to expand the programming capabilities of this framework, an “update” mechanism would be necessary to include new definitions and interactions. This “update” mechanism would be required mainly for the semantic adaptation layer entities.

4.2.4 Core Distributed Middleware Layer

This layer includes all the low-level distributed computing middleware necessary to access devices and services which constitute the user’s smart-space. The EUP-framework is built on top of this lower-layer middleware. Examples include UPnP, web services, Jini, Salutation, JXTA, Bluetooth Services, etc. The interface to the various components of this layer is known and usually standardized. Even though the

various components of this layer may use different methods to expose similar services, they are translated by the semantic adaptation layer to the same user-level concepts. For example, whether a networked printer is using UPnP, Salutation or Bluetooth printing profiles, the user-level concepts of “print”, “select double-sided”, “color”, “high-resolution”, “out-of-paper”, “low-ink” are the same.

4.2.5 Local System Layer

The local system layer includes the core platform and OS facilities, as well as the local user resources (e.g. local applications and content). The system layer provides the native interface and core scripting support and perhaps some native OS EUP libraries. The local user resources may also be exposed by the semantic adaptation layer as services, for example as described in the “task computing” approach [21]. Finally, some platform requirements to support the EUP-tools of the framework can be found in Section 4.3.5.

4.3 End-User Programming Tools (EUP-tools)

4.3.1 General Approach

Build a system to greatly simplify the construction of distributed applications. We will draw inspiration from Automator and PBE systems. As with typical PBE systems, our architecture will provide a framework that makes it possible for a software agent to eavesdrop on a user’s interaction with an application and record the actions for future use. The recording mechanism will be fully integrated with a scripting system that leverages lessons learned from Automator. This system will include the support for describing Stimulus/Response (S/R) behaviors, creating compound behavior from multiple applications and end-user debugging.

4.3.2 “Guides”: Programming Tools for Naïve Users

As we have already mentioned, our long-term goal is to make it easy for a naïve end-user make his/her desires known to the system. Based on the limited appeal of even the easiest programming languages, we propose to investigate the use of tools that do not require end-user to learn a programming language.

In theory, PBE systems seem to offer the promise of enabling end-user to build new commands using current applications with their current UIs. In practice, the results seem somewhat mixed. Part of the problem seems to be the difficulty of inferring from current UIs what can be reasonably generalized. The second difficulty is a bit more subtle. In modern applications, most simple tasks, even if they are repetitive, are easier to perform using the current direct manipulation interface than to use a macro recorder or PBE. One lesson learned from Automator is the benefit to users of being to build new commands out of simple combinations of old commands. This leads us to believe that our system should be able to compose new, simple commands from interactions with multiple, separate programs.

There is anecdotal evidence that suggests using the direct manipulation interface of today’s applications to perform complex operations is difficult. Using these types of interfaces in a PBE system to construct complex new commands would be even more difficult. We have observed that users are comfortable with computerized help and question-and-answer systems. Systems based on interactive, forms based dialogs

or “**guides**”, such as installation wizards are commonly used to help novice users and expert users accomplish unfamiliar tasks.

We propose to explore ways to encourage the creation of form-based dialogues for devices and services that go beyond installation. These dialogues can be thought of as “**guides**” for end-users. These guides can direct users through multi-step operations. Some “help” systems, such as the Windows “Help and Support” system already provide form-based dialogues to assist users with complex tasks. We think these “guides” can serve as the basis for simple to use, general purpose PBE system that can be utilized by end-users.

As noted earlier in this report, various programming by example projects associated with specific applications, such as Emacs keyboard macros, have enjoyed some success. With the exception of the recent Automator from Apple, most programming by example tools have been associated with specific applications. We think that combining programming by example ideas with guides or user dialogs would be a powerful, general purpose and easy to use tool for capturing the intentions of a user.

Various authors in, “Your Wish is My Command” [14] cite different problems or challenges associated with PBE systems. Inferring the intention of the user is a commonly cited problem. A related problem is the generalizability of actions and naming of objects. Consider for example, the case of recording the commands to discover the lights in a room and to turn them on. When the recording is replayed in another room, should the lights in the original room be turned on or should the lights in the current room be turned on?

Usually, this and similar problems are dealt with using simple inference techniques. These techniques can be used to try to guess the correct rule and subsequent interaction with the user is used to verify or refine the guesses. Some researchers have argued against intelligent interfaces, claiming that these interfaces will mislead the user and remove the necessary control. However, according to Myers in chapter 3 of “Your Wish is My Command”, the experience of most PBE researchers is that users expect the system to make increasingly sophisticated generalizations for examples. We will explore both the use of inference techniques and the use of prompts built into guides that will encourage users to express their intentions in a generalized fashion.

4.3.3 Control and Program Structure

Chapter 16 of “Your Wish is My Command” [14] introduces the important idea of Stimulus/Response behaviors to PBE systems. Support for S/R behavior makes it possible to create “programs” that respond to an event (i.e., a stimulus). An event can be a condition not naturally encountered during a recording session, such as a calendar alarm. We will study the use of dialogues to allow users to describe such events or conditions that should act as a stimulus.

Somewhat more problematic issues are actions or responses with consequences and control structure. PBE systems are based on the idea that the systems is watching and recording a sequence of actions performed by a user. This can be thought of as a very rudimentary type of machine learning. However, some actions within a Smart Home, such as sounding the fire alarm or deleting an important file, may be

inconvenient to perform solely for the purpose of training. This problem is not unique to S/R, it is common to most PBE systems.

Capturing program control structure is another problem common to most PBE systems. PBE systems are designed to record a simple sequence of steps taken by a user. The process of translating the recorded sequence of steps is fairly straightforward. However, most programs are not simple sequences of steps – most computer programs have control structure. Iteration and branching are usually expressed differently. Automator is an increasingly popular tool to simplify the construction of scripts. Its success is in large part directly attributable to its simplicity. However, while Automator has methods for capturing a user's intention to iterate, it can only compose a simple sequence of scripts. We will use the lessons learned from previous PBE systems to deal with iteration. However, it remains unclear how users can best express conditional (if this condition then perform this action else do some other action) behavior. We will investigate ways to enhance forms based dialogues to include support for expressing conditions and recording multiple sequences.

An important part of this work will be investigating ways to cope with the UI limitations of mobile phones and ways to exploit their special advantages in a PBE environment. For example, instead of using a browser to search for a printer or other desired device, we may be able to simply touch the printer or desired device using an RFID tag reader. Instead of relying entirely on forms-based dialogues, we will explore the careful use of speech.

4.3.4 Composition and Debugging of Distributed Services

Another challenge to programming smart spaces, especially smart homes, will be integrating applications and devices developed by different vendors and deployed incrementally. Unlike a mid-size enterprise, a typical end-user will not be able to rely on IBM's Consulting Services to integrate his/her home network, devices and applications. If the end-user wants the new alarm clock to control the home heating system, then the end-user will have to program the clock and the heating system and make sure they communicate with each other. This raises two additional topics of research, end-user composition of services and end-user debugging of distributed systems.

Composition of Services: The problem of integrating products and services from multiple providers will be one of the biggest challenges to widespread creation of smart homes. Our end-user programming system will need to have the means for end-users to compose new behaviors by combining these independently created devices and services. We will explore the possibility of automatically combining "guides" to solve complex problems and using PBE inspired recording service to compose the new, compound behaviors. We will leverage research that was reviewed in Section 3.

Debugging Distributed Systems: It follows that if end-users will deploy their smart homes, manage their smart homes and develop distributed applications for their smart homes, they will need tools to debug the distributed applications operating within their smart homes. One recurring problem is the limited support for UNDO. Many smart home applications will not support an UNDO because the effects take place in the real world. It is not possible to UNDO a printed page or microwaved popcorn. In the real world, compensating actions are used when a mistake occurs. Nevertheless, some means for removing an undesirable step or action from a recording will eventually be needed.

Even if the applications worked the very first time – users would still have to deal with problems due to subsequent component failure. Consumer products are often surprisingly reliable, but all products occasionally fail. Some related work has been done with dialogue based help systems and some interesting work has been done with intelligent introspective systems. We will explore both of these approaches.

4.3.5 Platform Requirements of End-User Programming Tools

Chapter 15, section 3 of “Your Wish is My Command” [14] describes a list of functionality a platform needs to provide in order to successfully host a PBE toolkit. These include event and action recording, support for controllability (i.e., event replay) and observability of relevant execution state.

Recording and replay of events and actions are the basis for PBE systems. It is vital that the PBE framework can record the UI events in order and in a timely fashion. It must be possible for the PBE framework to determine the actions triggered by the events so that the same actions can be triggered when the events are replayed.

User interfaces Symbian and POSIX do not provide all these functions and services (nor does any other native commercial operating system).

Our approach will be to create a PBE layer on top of a target operating system as described in Section 4.2. Applications based on this framework will clearly separate the UI from the rest of the application (Symbian applications are already constructed this way). The UI portion of an application will communicate with other components using a new communication channel. This new channel will allow the PBE framework to eavesdrop and record actions. It will also make it possible to replay the recordings at a future time. It is essential that semantics of replayed events are the same as the recorded events.

The unfortunate drawback of this approach is that applications will have to be rewritten to take advantage of the new layer and its services.

REFERENCES

- [1] Digital Living Network Alliance (DLNA), “Home Networked Device Interoperability Guidelines v1.0”, June 2004. <http://www.dlna.org>
- [2] UPnP Forum, “UPnP Device Architecture 1.0.1”, December 2003. <http://www.upnp.org>
- [3] World Wide Web Consortium (W3C), Web Services Activity, <http://www.w3.org/2002/ws/>
- [4] EUD-NET Network of Excellence, <http://giove.cnuce.cnr.it/eud-net.htm>
- [5] End User Workshop, <http://giove.cnuce.cnr.it/chi-eud.html>
- [6] WEUSE II, The Next Step: From End-User Programming to End-User Software Engineering, will be held at CHI 2006 in Montréal, Quebec, Canada, April 23, 2006, <http://eusesconsortium.org/weuse/>
- [7] EUSES (End Users Shaping Effective Software) Consortium, <http://eusesconsortium.org/>
- [8] Roman Barták: “Constraint Programming: In Pursuit of the Holy Grail”. Charles University, Faculty of Mathematics and Physics, Department of Theoretical Computer Science, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic.
- [9] Chapter 2, Ken Kahn, “Your Wish is My Command”, edited by Henry Lieberman, Morgan Kaufmann, Academic Press 2001.
- [10] Lillian Lee “I'm sorry Dave, I'm afraid I can't do that.: Linguistics, Statistics, and Natural Language Processing circa 2001”. Appears in *Computer Science: Re_ections on the Field, Re_ections from the Field* (Report of the National Academies' Study on the Fundamentals of Computer Science), pp. 111.118, 2004.
- [11] Spoken Language Systems Group, Computer Science and Artificial Intelligence (CSAIL) Lab, MIT: GALAXY, <http://groups.csail.mit.edu/sls/technologies/galaxy.shtml>
- [12] Boris Katz, “Annotating the World Wide Web Using Natural Language”, *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, 1997
- [13] White paper describing Apple's Automator scripting tool, <http://developer.apple.com/macosx/automator.html>
- [14] Henry Lieberman (editor): “Your Wish is My Command: Programming by Example”, Morgan Kaufmann, Academic Press 2001.
- [15] Chapter 1, David Canfield Smith, Allen Cypher, Larry Tesler, “Your Wish is My Command”, edited by Henry Lieberman, Morgan Kaufmann, Academic Press 2001
- [16] Alan Davy: “Task-Driven Service Composition for Pervasive Computing Environments”. M-Zones White Paper June 04, white paper 06/04, Ireland, June, 2004.
- [17] Sonia Ben Mokhtar, Nikolaos Georgantas, Valérie Issarny: “Ad hoc Composition of User Tasks in Pervasive Computing Environments”. In *Proceedings of 4th Workshop on Software Composition (ETAPS'05)*. April 2005, Edinburgh. Scotland.
- [18] Yasmine Charif and Nicolas Sabouret: “An Overview of Semantic Web Services Composition Approaches”. *Electronic Notes in Theoretical Computer Science* 85 No. 6 (2005).

- [19] Shalil Majithia, David W. Walker, W.A. Gray: "Automated Web Service Composition using Semantic Web Technologies". Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'04).
- [20] W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy: "The Case for Recombinant Computing". Xerox Palo Alto Research Center Technical Report CSL-01-1. April 20, 2001.
- [21] Fujitsu Laboratories of America: 'Task Computing' web site, <http://taskcomputing.org/>
- [22] World Wide Web Consortium (W3C), Web Services Description Language (WSDL), <http://www.w3.org/TR/wsdl>
- [23] World Wide Web Consortium (W3C), Simple Object Access Protocol (SOAP), <http://www.w3.org/TR/soap/>
- [24] UDDI project, <http://www.uddi.org/>
- [25] DAML Services, <http://www.daml.org/services/owl-s/>
- [26] Business Process Execution Language for Web Services (BPEL4WS), <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>
- [27] Shalil Majithia, David W. Walker, and W. A. Gray: "A Framework for Automated Service Composition in Service-Oriented Architecture". In 1st European Semantic Web Symposium, 2004.
- [28] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan: "Automated Discovery, Interaction and Composition of Semantic Web services". Web Semantics: Science, Services and Agents on the World Wide Web, 1(1):27-46, 2003.
- [29] Project ASTRO: Supporting Automatic Composition of Distributed Business Processes, <http://sra.itc.it/projects/astro/>
- [30] Dan Wu, Evren Sirin, James Hendler, Dana Nau, and Bijan Parsia: Automatic Web Services Composition Using SHOP2. In 13th International Conference on Automated Planning & Scheduling. Workshop on Planning for WebServices., Trento, Italy, June 2003.
- [31] Maja Vukovic, Peter Robinson: "Adaptive, Planning-based, Web service Composition for Context Awareness". International Conference on Pervasive Computing, Vienna, April 2004.
- [32] Evren Sirin, James Hendler, and Bijan Parsia: "Semi-automatic composition of web services using semantic descriptions". In Web Services: Modeling, Architecture and Infrastructure Workshop in ICEIS 2003, Angers, France, April 2003.
- [33] Evren Sirin, James Hendler and Bijan Parsia. Interactive Composition of Semantic Web Services. Poster, In The Twelfth International World Wide Web Conference (WWW 2003), Budapest, Hungary, May 2003.
- [34] Steffen Higel, Tony O'Donnell, Vincent Wade: "Towards a Natural Interface to Adaptive Service Composition". Proc. of the 1st ACM International Symposium on Information and Communication technologies, pp. 169-174, Dublin, Ireland, 2003.
- [35] M. Pistore, P. Bertoli, E. Cusenza, A. Marconi, and P. Traverso: "WS-GEN: A Tool for the Automated Composition of Semantic Web Services". Demo Paper in 3rd International Semantic Web Conference (ISWC2004)

- [36] WS-GEN tool download:
http://sra.itc.it/projects/astro/index.php?option=com_content&task=section&id=30&Itemid=128
- [37] S. Majithia, M. Shields, I. Taylor, and I. Wang: "Triana: A graphical web service composition and execution toolkit". In IEEE International Conference on Web Services (ICWS'04), San Diego, CA, June 06 - 09 2004.
- [38] Using Speakeasy for Ad Hoc Peer to Peer Collaboration. W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, Trevor F Smith, Dirk Balfanz, D. K. Smetters, H. Chi Wong, Shahram Izadi. In Proceedings of CSCW '02. (November 2002) [pdf]
- [39] XEROX PARC: The Objé™ Software Architecture, <http://www.parc.com/research/projects/obje/>
- [40] Mark W. Newman, Shahram Izadi, W. Keith Edwards, Jana Z. Sedivy, Trevor F Smith: "User Interfaces When and Where They are Needed: An Infrastructure for Recombinant Computing". In Proceedings of UIST '02, October 2002.
- [41] W. Keith Edwards, Mark W. Newman, Jana Sedivy, Trevor Smith, Shahram Izadi: "Challenge: Recombinant Computing and the Speakeasy Approach". In Proceedings of Mobicom '02, September 2002.
- [42] Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, Trevor F Smith. "Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environment". In Proceedings of DIS '02, June 2002.
- [43] Ryusuke Masuoka, Yannis Labrou, and Zhexuan Song: "Semantic Web and Ubiquitous Computing - Task Computing as an Example". AIS SIGSEMIS Bulletin, Vol. 1 No. 3, October 2004, pp. 21 - 24 (PDF).
- [44] Zhexuan Song, Yannis Labrou and Ryusuke Masuoka: "Dynamic Service Discovery and Management in Task Computing," pp. 310 - 318, MobiQuitous 2004, August 22-26, Boston, MA, 2004.
- [45] Ryusuke Masuoka, Bijan Parsia, Yannis Labrou and Evren Sirin: "Ontology-Enabled Pervasive Computing Applications". IEEE Intelligent Systems, vol. 18, no. 5, pp. 68-72, Sep./Oct. 2003.
- [46] Ryusuke Masuoka, Bijan Parsia and Yannis Labrou: "Task Computing - the Semantic Web meets Pervasive Computing," 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, October 2003.
- [47] Ryusuke Masuoka and Yannis Labrou: "Task Computing - Semantic-web enabled, user-driven, interactive environments". WWW Based Communities For Knowledge Presentation, Sharing, Mining and Protection (The PSMP workshop) within CIC 2003, Las Vegas, NV, June 2003.
- [48] Charles Rich and Richard C. Waters: "The Programmer's Apprentice", ACM Press Frontier Series, SBN:0-201-52425-2, ACM Press, New York, NY, USA, 1990.
- [49] John F. Pane, Brad A. Myers: "Usability Issues in the Design of Novice Programming Systems", CMU-CS-96-132, August 1996.
- [50] Allen Cypher (editor): "Watch What I Do: Programming By Demonstration", The MIT Press, Cambridge, Massachusetts, London, England, 1993.
- [51] Brad A. Myers, John F. Pane and Andy Ko: "Natural Programming Languages And Environments".

- [52] John F. Pane, Brad A. Myers, and Leah B. Miller: “Using HCI Techniques to Design a More Usable Programming System”. Computer Science Department and Human Computer Interaction Institute, Carnegie Mellon University.
- [53] Link to Programming By Example bibliography
<http://web.media.mit.edu/~lieber/PBE/PBE-Bibliography.html>
- [54] Link to extensive Visual Programming research bibliography
<http://web.engr.oregonstate.edu/~burnett/vpl.html>
- [55] Link to an End-User Programming bibliography
<http://www.hcibib.org/gs.cgi?terms=end+user+programming&highlight=checked>
- [56] Link to another End-User Programming bibliography
<http://www.cs.uml.edu/~hgoodell/EndUser/Biblio/Bibliog.htm>