A Framework for End-User Programming of Smart Homes Using Mobile Devices

Paul Wisner and Dimitris N. Kalofonos
Nokia Research Center Cambridge
Cambridge, MA 02142, USA
{paul.wisner, dimitris.kalofonos}@nokia.com

***Abstract*— A crucial challenge to enabling multi-device home networked systems is providing a way for non-expert consumers to combine, configure and control the available functions. Unfortunately, the typical consumers find it difficult to perform tasks using even stand-alone programmable CE devices; programming and controlling smart homes involving a number of connected devices and services represents a significantly bigger challenge. We present our architecture for end-user programming of novel application concepts in smart homes. Our architecture has an end-user layer, a scripting layer, a semantic adaptation layer, a distributed middleware layer and a local system layer. We describe the overall end-user's process for creating applications. We discuss a part of our implementation that automatically discovers and composes devices into arrangements that satisfy the user's goal.**

Keywords: end-user programming, service composition, home middleware, mobile devices, pervasive computing.

I. Introduction

The wide availability of affordable networking equipment, developments in the Consumer Electronics (CE), Personal Computer (PC) and mobile industries (e.g. [1], [2]), and the increasing importance of Web Services (WS) [3] will soon enable the proliferation of *smart homes*. Smart homes consist of networked devices and services which expose well-defined programming interfaces that allow the creation of distributed applications, through which end-users interact and control their smart environment. Currently, these applications offer the specific static functionality that the device manufacturers or software vendors intended; there is no way for consumers to "program" their smart spaces to perform customized tasks that match their intent. Sophisticated and knowledgeable end-users could create new applications to perform customized tasks, but it is well beyond the ability of non-expert consumers to modify pre-programmed functionality to meet their own needs.

Our research aims to address this problem by enabling end-users to easily use their mobile devices to instruct their home devices and services to interact with each other and to dynamically react to events happening in their environment. In our view, mobile phones are ideal devices to program and control the smart home. Today's smartphones are mobile always-on networked computers that are with us all the time; they are already part of the digital home ecosystem [1]; they resemble the consumer notion of universal remote controls, but are also personal and much more capable (e.g. processing, storage, multimedia, networking) and with support for a multitude of user interaction modalities (e.g. GUI, voice, gestures, touch); finally, all consumers have them and know how to use them, regardless of their expertise and ownership of computers in general.

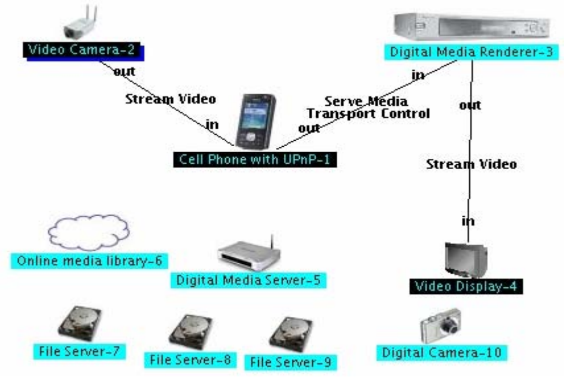


Figure 1 Device composition proposes "wirings" among devices.

In this paper, we propose a framework consisting of middleware and intuitive user-level tools that allow users to create and run on their mobile phones customized programs that implement the intended smart home tasks. In our system the user selects devices based on an intuition that it is possible to use them together. Next, the system uses automatic device composition to propose possible "wirings" among the devices[1] (Figure 1). Our approach contrasts with the approach that suggests using predefined task specifications (e.g. [4]). We believe that these are not required to obtain meaningful automatic compositions; instead, we rely on the user's intuition of which devices can be used together (e.g. [5]). The proposed system finds ways that the devices can collaborate and uses domain knowledge and user input to generate the desired application.

In the proposed framework, the process of programming the desired task begins with the user conceiving some customized task. The framework performs device and service discovery of the local environment and a user-level tool, referred to as *Compose Tool*, displays a list of available devices and allows the user to select the devices to be involved in the task. This tool then runs its composition algorithm to search for possible

[1] In this paper we write 'devices' to refer to services and devices. While it is more precise to write about connecting services, we believe that it is easier for end-users to think in terms of devices.

ways to connect or “wire” these devices together and generates possible composition proposals. One of these proposals is selected, either automatically by the system based on a heuristic evaluation of optimality, or manually by the user. Once a proposal has been selected another user-level tool, referred to as *EUP tool*, builds a list of EVENTS and ACTIONS related to all the devices in the selected composition. The EUP tool next guides the user through an intuitive process to specify the application’s behavior by creating a process design, which may include a *Control User Interface* (Control UI). The system may use domain knowledge about the devices involved in the composition to further assist the user. Once the user completes the definition of the process, the EUP tool creates one or more scripts which concretely implement the desired task. Finally, when the user decides to launch this task at any time in the future, the framework provides a runtime environment that executes the script(s) implementing the task.

The rest of this paper is organized as follows: Section II presents related research; Section III describes a motivating usage scenario for our work; Section IV describes the proposed architecture and its components; Section V presents our current prototype implementation; Section VI gives our directions of future work; and finally, Section VII presents our conclusions.

II. Related Work

The availability of standards allowing the syntactic and semantic description and composition of Web Services, such as the Ontology Web Language for Services (OWL-S) [6] and the Business Process Execution Language for Web Services (BPEL4WS) [7] has spurred significant research efforts to create frameworks and tools that allow the composition of semantically annotated Web Services to achieve user-specified tasks (e.g. [8], [9], [10]). In general we can distinguish these approaches in two categories: (a) *automatic composition* and (b) *semi-automatic* or *interactive* composition. Automatic Web Services composition approaches (e.g. [8], [10]) 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. Although these approaches put less demanding requirements on end-users by allowing them to express their intent in a more vague/natural manner, it is likely that they lead to a composite task which may be (significantly) different from the task the end-user originally intended. On the other hand, semi-automatic or interactive Web Service composition approaches (e.g. [11], [12], [13]) 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.

The Recombinant Computing approach [4], [5], developed in Xerox PARC, 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. In the recombinant approach, users have knowledge about the devices they encounter in their environments and, therefore, must be the final arbitrators of the semantics of the entities they are interacting with. Recombinant computing creates semantically neutral interfaces, called recombinant interfaces, which are programmatic interfaces that specify the ways in which components interact with one another and extend one another's behavior at runtime. Of particular interest in the recombinant computing approach is the use of task-oriented templates [4].

The Task Computing approach [14], [15], [16] developed in collaboration by the Fujitsu Laboratories of America (FLA) and the MIND Lab of the University of Maryland, exposes the functionality found in smart environments (i.e. networked devices, applications, content, and web services) as Semantic Web Services, which in turn the user can discover and arbitrarily compose. Task Computing also 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.

DiamondHelp [17] uses the Task base approach in a mixed initiative user interface. In mixed initiative systems the user interface leads the user through a sequence of steps. The user has the option to do things himself steering the system in another direction. The result is a two-way dialog instead of a user answering questions posed by a computer. The book “Programming by Example” [18] discusses ways for end-users to describe behavior by demonstrating a sequence of steps. Others suggest visual programming in which users manipulate graphical elements on the screen to specify behavior.

III. A Motivating Usage Scenario

Tom has recently installed security cameras outside his house. These security cameras expose an interface to network peers that supports subscriptions to motion-detection events. He would like to be alerted when his security cameras detect motion outside the front or back door. He would like his phone to beep and also display the video on its small screen. He would also like to have a button to show the security camera his large television screen. He decides to try the new software that runs on his smartphone that allows non-expert users to easily program and control their smart homes. The interaction is very intuitive: his phone automatically discovers all the services available and guides him to create the “view visitor” application 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. In this case the security camera's motion detection events trigger a script that displays a control UI on the phone. Tom has created buttons on the control UI labeled "Show on Phone" and "Show on TV". The first button displays the security camera's video on the phone using its built in UPnP Digital Media Renderer (DMR). The second button directs the video to a stand-alone DMR that is connected to the television screen. After trying it out, Tom decides to modify the application to pause his Digital Video Recorder (DVR) before displaying the security camera.

Later, Tom realizes that even when he is not at home his phone will be alerted and display the security camera. Tom starts thinking of other ideas. He could add a button to "Record' video to his Digital Media Server and another button to "Call the Police". He is also fighting the impulse to go out and buy a video screen for his front door so that he can add a feature for a two-way video conversation with the person at the door through his phone's camera.

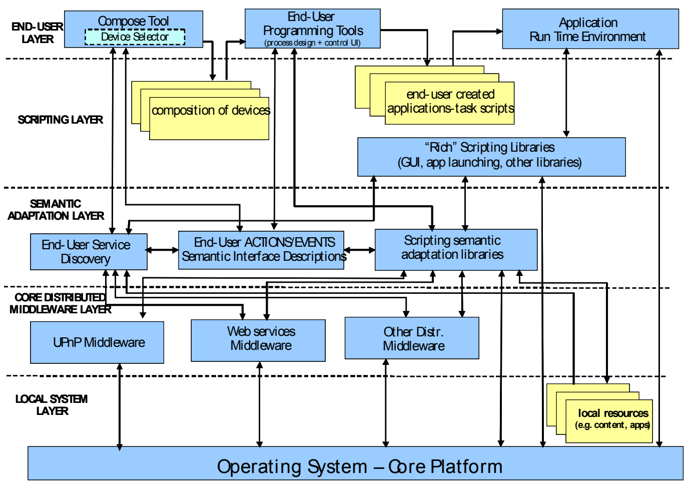

END-USER LAYER
Compose Tool
Device Selector
End-User Programming Tools
(process design + control UI)
Application Run Time Environment
SCRIPTING LAYER
composition of devices
end-user created applications-task scripts
"Rich" Scripting Libraries (GUI, app launching, other libraries)
SEMANTIC ADAPTATION LAYER
End-User Service Discovery
End-User ACTIONS/EVENTS Semantic Interface Descriptions
Scripting semantic adaptation libraries
CORE DISTRIBUTED MIDDLEWARE LAYER
UPnP Middleware
Web services Middleware
Other Distr. Middleware
LOCAL SYSTEM LAYER
local resources (e.g. content, apps)
Operating System – Core Platform

Figure 2: System architecture of the proposed framework.

IV. System Architecture

The general architecture of the proposed framework is shown in Figure 2 and can be distinguished in five layers: (i) the end-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 description of each of these layers.

A. End-User Layer

The end-user layer is responsible for the interaction with the end-user. It serves two roles: (a) to encode the user's intent into an application, and (b) to launch and execute that application. At every level user-interactions must be intuitive. The *Device Selector* uses user-level service discovery as exposed by the semantic adaptation layer to present a list of available devices to the user. The user selects the devices to be used in the task. The *Compose Tool* then generates a set of proposed compositions in the form of wiring diagrams, using information provided by the semantic interface descriptions. Each of these compositions specify how the devices will be connected together for device collaboration. One of these proposals is selected, either automatically by the system or manually by the user.

The *End-User Programming tool (EUP-tool)* then examines the selected composition and pulls information about the devices from the semantic adaptation layer. The semantic adaptation layer provides EVENTS, ACTIONS and domain knowledge corresponding to the chosen devices. Domain knowledge is the key to making EUP easy. The EUP tool may suggest a skeleton process design and the user can manipulate objects in a GUI to modify it. Besides facilitating the process design, the EUP-tool may include an editor which allows the user to create a *Control UI* for the task with buttons, menus and output areas. Buttons and menu items also trigger EVENTS.

The resulting application consists of one or more scripts which instantiate the created process design (from EUP-tool) based on the selected device composition (from Device Selector-Composer) and optionally the Control UI (from EUP-tool). The application executes in the Application Run-Time Environment (RTE).

B. Scripting Layer

The scripts composed by the EUP-tool 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 to explore the possibility of the EUP-tool creating actual native source code or detailed specifications for native code. 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 libraries would also make available to the scripts the APIs with the functionality exposed by the semantic adaptation layer, i.e. *End-user Service Discovery* and the *Scripting Semantic Adaptation Libraries.*

C. Semantic Adaptation Layer

The semantic adaptation layer is responsible for mapping concepts perceived by the end-users to functionality provided by the underlying system. At the heart of this layer are the *End-User Semantic Interface 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 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-tool to guide the user in the composition of tasks and by the Compose Tool to create device composition proposals.

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 proposed EUP-framework.

D. 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.

E. Local System Layer

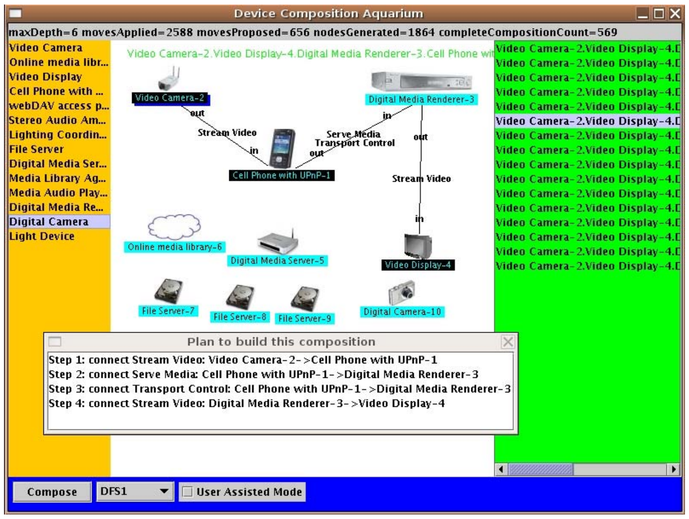
The local system layer includes the core platform and OS facilities, as well as the local user resources (e.g. local applications and content). The local user resources may also be exposed by the semantic adaptation layer as services, for example as described in the “task computing” approach [14].

V. A PROTOTYPE IMPLEMENTATION

We have used our tool called *Device Composition Aquarium* as a prototype of the Compose Tool. Using the Aquarium we simulate automatic device discovery and apply a composer to arrange devices into compositions that satisfy the user’s goal. The result is a composition that is the input to the EUP tool. The user interaction with the Device Composition Aquarium embodies some of our initial thinking about how the end-user will interact with the *Compose Tool*. The final user interface will be redesigned for a smart phone.

A. Device Composition Aquarium

The Device Composition Aquarium provides facilities for simulating a smart space environment, executing a composition algorithm in that environment and displaying a graphical depiction of the results. The Aquarium supports both the fully automatic and interactive (user-assisted) composition modes described in section II. Users of the Aquarium, users can test and evaluate automatic device composition in either mode.



```
maxDepth=6 movesApplied=2588 movesProposed=656 nodesGenerated=1864 completeCompositionCount=569
```

Plan to build this composition

```
Step 1: connect Stream Video: Video Camera-2->Cell Phone with UPnP-1
Step 2: connect Serve Media: Cell Phone with UPnP-1->Digital Media Renderer-3
Step 3: connect Transport Control: Cell Phone with UPnP-1->Digital Media Renderer-3
Step 4: connect Stream Video: Digital Media Renderer-3->Video Display-4
```

Figure 3 Device Composition Aquarium

The Aquarium screen has four main areas (Figure 3). The panel on the left shows a list of all known device types. The center panel represents the devices in the local environment. The user can select device types and create instances in the local environment panel. This simulates the information that will be provided by the End-User Discovery module in our architecture (Figure 2). The user then expresses a composition goal by selecting one or more devices. Next the user presses the “Compose” button to invoke the composition algorithm. The composition algorithm produces a set of composition proposals which are listed in the panel on the right and the first composition is automatically selected. The center panel displays the connections between devices as indicated by the selected composition. At the top of the screen there is some diagnostic information to help evaluate the efficiency of the composer algorithm..

The main Aquarium window displays a composition as a directed graph. A separate dialog labeled “Plan to build this composition” shows the steps to make the composition. This sequence matches how a composition is stored internally. Later, the sequence can be conveniently translated into a script for assembling the real devices into the composition.

The output of a composer is shown in the top box of Figure 4. In the diagram the composition proposal is represented as a plan for connecting the interfaces of devices together. The circles are nodes that correspond to device instances shown in the center box. Each device instance has a pointer to its device type shown in the bottom box. In the Aquarium, the device types are loaded from XML test data. This arrangement is for testing purposes only. In the EUP framework, each device type would come from a device that was discovered by the service discovery module. For example, UPnP devices and Web Services have description documents that give the same details as those specified in our XML test data. The composition algorithm determines interface compatibility using the simple mechanism of matching the interface names shown in the bottom box of Figure 4. In the future we would like to investigate using the Semantic Adaptation layer to determine interface compatibility based on the semantic matching research associated with OWL-S [6].

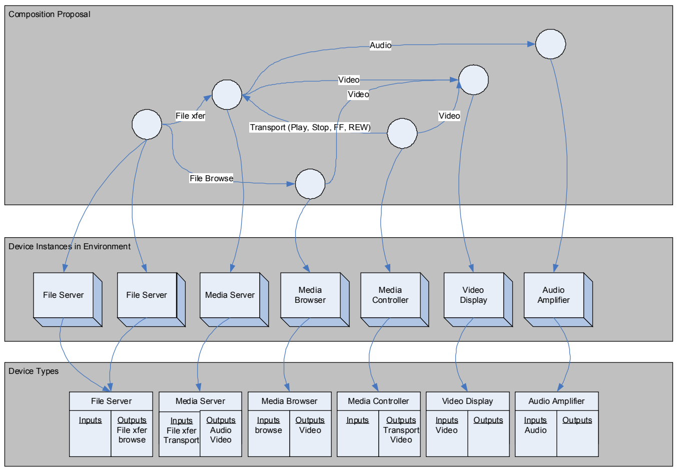


Figure 4 Compositions, Devices and Types.

B. *Composition Algorithms*

We have implemented an initial composition algorithm called Depth First Search 1 (DFS1) [19]. DFS1 implements a standard exhaustive depth first search. DFS1 is a good place to start because it has excellent completeness, meaning it always finds all possible compositions. DFS1 takes as input a set of devices selected by the user that must be included in a composition. The "goal" of the composer is to generate compositions that include all the selected and have no unsatisfied mandatory inputs or outputs. DFS1 starts with a composition containing a single device from the user's selections. It generates a list of next "moves" for the current device (connections to interfaces on other devices). Each "move" is a different direction in the search path.

A more sophisticated composer implementation can potentially produce "better" compositions in less time and with lower peak memory usage. Russell, Norgiv [19] list four ways to evaluate performance of such algorithms:

- **Completeness:** does it always find a solution when there is one?
- **Optimality:** does it find an optimal solution?
- **Time complexity:** how long does it take?
- **Space complexity:** peak memory usage

Depth first search explores each search path completely before moving on to the next move in the search path. Exploring each path completely has lower peak memory requirements than another standard technique called breadth first search – in which all search paths are explored concurrently. Breadth-first finds the solutions with the fewest devices earliest – but a simplest solution may not be the best. DFS1 has no notions of optimality; it does not know which compositions are better. If processor cycles are plentiful, DFS1 has acceptable time space complexity- in the worst cases it completes in a second or two. Better composer algorithms can be developed using informed search techniques such as "best first" or "A*" as described in AI text books such as [19]. Informed search reduces the space complexity and time complexity because it only explores the most promising paths. However, completeness is threatened because the composer may not find the best solution if the search path does not initially seem to be a promising. This is a pitfall of artificial intelligence techniques: they tend to make the same mistakes as a person. The main challenge for informed search is finding a good *heuristic function.* A heuristic function evaluates the current state and gives a score to each potential next step in the search. Informed search strategies use heuristic evaluation to guide its search. For example, you may have a heuristic (a rule of thumb) that states that it is good to connect as many media libraries as possible to a media server. So, given a composition that includes a media server, a higher score would be given to paths that add media libraries. MIT's Hyperglue project [20] uses context awareness as input to a heuristic function: the idea is that where you are and what you are doing are important inputs to evaluating the optimality of a composition.

VI. FUTURE DIRECTIONS

To make the Compose Tool work, we will need components from our Semantic Adaptation Layer. This involves designing a conceptual module such that an end-user will understand how to make use of services, events, actions and tasks.

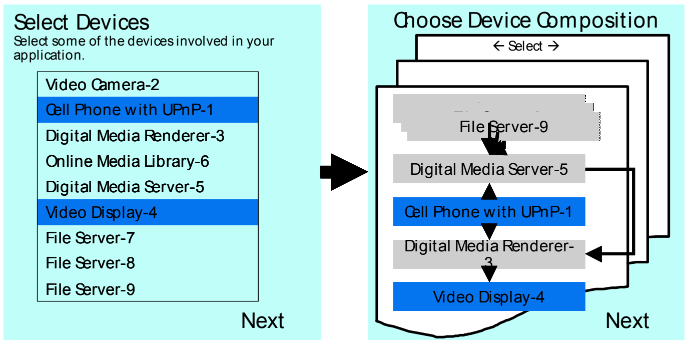


Figure 5 – A Compose Tool user interface for phone.

We plan to transform the Device Composition Aquarium into a *Compose Tool* that runs on a mobile phone. This will require us to rethink our visual metaphors. Figure 5 shows at least one way this might be possible. The informed search described above will provide a way to reduce the number of composition proposals from which the user must select.

Next we will develop the End-User Programming tool. We envision that the user will modify the process design by manipulating a GUI interface. The user will answer questions and make association between objects on the screen. We aim at avoiding script authoring on the part of the end-user. Task Computing [14] avoids scripting by obtaining process models from libraries of predefined task specifications. Task Computing starts with the user selecting a task from a library of choices. We would like to investigate a reverse approach in which the user first narrows the possibilities by first choosing the devices and then the system selects only the task specifications that can be implemented using the devices in the composition. The hope is that in most cases a task specification will already exist and the user can get the intended behavior by selecting a high-level description. However, our main emphasis will be to make it easy for the user to devise a new process that did not previously exist. We are also interested in investigating "Programming by example" as way for a user to demonstrate a sequence of steps [18].

We will develop the Application Run-Time Environment to execute the application. The RTE must support the instantiation of the device composition and execution of scripts. The scripts will include calls to a library of functions in our scripting library. Finally, we will need to assemble existing Distributed Middleware Layer technologies so that the Semantic Adaptation Layer can unify them for the end-user.

VII. CONCLUSIONS

We have described a system that aims at overcoming the difficulties involved in combining the capabilities of devices and their services available in a smart home. We described a specific way to automate the steps needed to compose an application that controls smart spaces. Our proposal supports truly ad hoc application concepts by avoiding the requirement for predefined "task descriptions".

Our architecture unites the capability of various distributed middleware protocols with the capabilities on the mobile phone. Semantic adaptation presents the capabilities in a way that is meaningful to an end-user.

The Device Composition Aquarium has allowed us to inspect the results of the DFS1 composer algorithm. Even though this is a simple, well known algorithm, DFS1 solves any composition request for which a solution exists. Automatic composition reveals possibilities that may not occur to the user.

REFERENCES

[1] Digital Living Network Alliance (DLNA), "Digital Living Network Alliance Home Networked Device Interoperability Guidelines Expanded: March 2006", March 2006.

[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] 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 Proc. of DIS '02, June 2002.

[5] 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.

[6] DAML Services, http://www.daml.org/services/owl-s/

[7] Business Process Execution Language for Web Services (BPEL4WS), http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[8] 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.

[9] Yasmine Charif and Nicolas Sabouret: "An Overview of Semantic Web Services Composition Approaches". Electronic Notes in Theoretical Computer Science 85 No. 6 (2005).

[10] 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).

[11] 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.

[12] 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.

[13] 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.

[14] Fujitsu Laboratories of America: 'Task Computing' web site, http://taskcomputing.org/

[15] 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.

[16] 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.

[17] Rich, C.; Sidner, C.; Lesh, N.; Garland, A.; Booth, S.; Chimani, M., "DiamondHelp: A Collaborative Interface Framework for Networked Home Appliances", *IEEE International Conference on Distributed Computing Systems Workshops*, pp. 514-519, June 2005

[18] Henry Lieberman (editor): "Your Wish is My Command: Programming by Example", Morgan Kaufmann, Academic Press 2001.

[19] Russell, Norvig. "Artificial Intelligence: A Modern Approach", Prentice Hall, 2003.

[20] Stephan Peters, Gary Look, Kevin Quigley, Howard Shrobe and Krzysztof Gajos: "Hyperglue: Designing High-Level Agent Communications for Distributed Applications". M.I.T. Computer Science and Artificial Intelligence Labroratory Technical Report. MIT-CSAIL-TR-2006-017, March 1, 2006.