LEVELS OF ABSTRACTION IN PROGRAMMING DEVICE ECOLOGY WORKFLOWS

Seng W. Loke, Sea Ling, Gerry Butler and Brett Gillick
School of Computer Science and Software Engineering
Monash University, CaulfieldEast, VIC 3145, Australia

Keywords: device ecology, workflow, abstraction levels, Web services, end-user control.

Abstract: We explore the notion of the workflow for specifying interactions among collections of devices (which we term *device ecologies*). We discuss three levels of abstraction in programming device ecologies: high-level workflow, low-level workflow and device conversations, and how control (in the sense of operations issued by an end-user on such workflows or exceptions) is passed between levels. Such levels of abstraction are important since the system should be as user friendly as possible while permitting programmability not only at high-levels of abstraction but also at low levels of detail. We also present a conceptual architecture for the device ecology workflow engine for executing and managing such workflows.

1 INTRODUCTION

We envision *device ecologies* comprising collections of devices (in the environment and on users) interacting synergistically with one another, with users, and with Internet resources, undergirded by appropriate software and communicating across the living room or across nations. There has been significant work in building the networking and integrative infrastructure for such devices, within the home, the office, and other environments and linking them to the global Internet. For example, UPnP (UPnP Forum, 2000a), SIDRAH (Durand et al., 2003) and Jini (Microsystems, 2001) provide infrastructure for devices to be inter-connected, find each other, and utilize each other's capabilities. Embedded Web Servers (Bentham, 2002) are able to expose the functionality of devices as Web services. Approaches to modelling and programming such devices for the home have been investigated, where devices have been modelled as software components, collections of objects (Association of Home Appliance Manufacturers, 2002), and Web services (Matsuura et al., 2003). Recent work has developed frameworks for aggregating, composing and building connections among networked devices (Omojokun and Dewan, 2003; Kumar et al., 2003; Butler, 2002; Newman et al., 2002; Kohtake et al., 2003; Vildjiounaite et al., 2003; Sousa and Garlan, 2003; Masuoka et al., 2003). However, there has been little work on specifying at a high level of abstraction (and representing this specification explicitly) how such devices would work together at the user-task or application level, and how such work can be managed. Our earlier work in (Loke, 2003) introduced *device ecology workflows* as a metaphor for thinking about how collections of these devices (or devices in a device ecology) can work together to accomplish a purpose. (Rodrigues et al., 2004) investigates mechanisms to permit a robot to recognize valid commands in spoken sentences that may not be entirely grammatically correct. This forms a building block for the input of device commands.

In this paper, we define levels of abstraction for programming device ecology workflows and describe how these levels interact. We also show how workflow operations issued by a user at one level of abstraction can map down to operations at another level of abstraction and how exceptions in the lower level can be reflected to the upper level. We aim to provide a framework for programming device ecologies where new levels of abstraction can be built from the lower levels, as needed. We contend that such a formally grounded construction is important for task-based programming in general (e.g., (Masuoka et al., 2003)). Our contribution is to focus on the workflows designated by multiple commands to different devices, and the mapping of these workflows between languages at different levels of abstraction. Since the upper level is intended to form the user interface, we represent workflows in an English-like language. The lower level could be represented by languages such as BPEL4WS (Microsoft et al., 2003) or

DysCo (Piccinelli et al., 2003). In our work we have used BPEL4WS.

The rest of this paper is organized as follows. Section 2 describes the notion of device ecology workflows via an example and also introduces two device ecology workflow languages at different levels of abstraction and how they formally relate to each other. We also consider conversations with devices at the lowest level of abstraction. The user should have control over the workflow execution and tasks within the workflows. In Section 3, we consider how user specified operations on workflows are related across the different levels of abstraction and how exceptions are reflected up to higher levels. We describe the conceptual architecture of a system for executing the multi-layered workflows in Section 4 and briefly outline our current prototype. We conclude with future work in Section 5.

2 PROGRAMMING DEVICE ECOLOGIES

2.1 Device Ecology Workflows

Devices can work together with other devices, the user (e.g., seeking approval for critical tasks) or Web resources in accomplishing its goals, either as initiated by users or by proactive smart devices. As illustration, consider a device ecology workflow involving a television, a coffee-boiler, bedroom lights, bathroom lights, and a news Web service accessed over the Internet.

Figure 1 graphically depicts this workflow. The dashed arrows represent sequencing, the boxes are tasks, the solid arrow represents a control link for synchronization across concurrent activities, and free grouping of sequences (i.e., the boxes grouped into the large box) represents concurrent sequences. This workflow is initiated by a wake-up notice from Jane's alarm clock which we assume here is issued to the Device Ecology Workflow Engine when the alarm clock rings. This workflow can be described using BPEL4WS or other Web service workflow languages such as DysCo, and executed using a specialized workflow engine as outlined in (Loke, 2003).

2.2 User Command Language

One could create device ecology workflows as mentioned above using some tools. However, abbreviated commands would make end-user programming simpler. Such commands can then map down to lower level workflows. We consider a small example of such

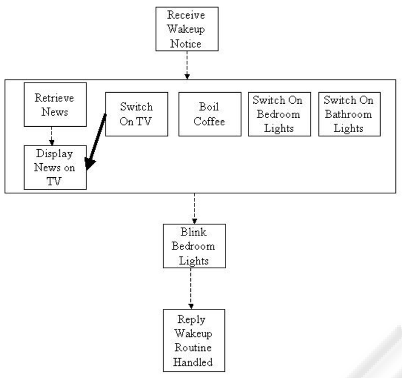

Figure 1: An Example Device Ecology Workflow

a two-level model. The top level is a small user command language, which we call *eco*, comprising commands that can be combined for sequential or parallel execution. For simplicity, we consider a device ecology with only a few devices. Inspired by (Omojokun and Dewan, 2003), we consider two kinds of commands, those which affect a single device and those which affect multiple devices. Commands affecting multiple devices are directed to a pseudo device that issues single-device commands to the appropriate devices. In fact, the Device Ecology Workflow Engine performs this function. It need only be provided with a predefined command set, expressed in *eco*, describing how to break down multi-device commands. This procedure can be nested to any depth, where each level of nesting corresponds either to a recursive call to a single instance of the Engine, or to the invocation of a separate instance.

The *eco* command language is defined as follows in EBNF:

```
Sentence ::= Clause (";" Clause)* "."
Clause ::= ActionClause | WaitClause
ActionClause ::= Prep* Opn Parm*
    Prep* Device
WaitClause ::= "wait for" Device
```

The symbols *Device*, *Opn*, *Parm* and *Prep* are terminals. *Device*, which is always a noun in the English-like language, designates the appliance that an operation (a verb) is directed to. *Parm* designates optional parameters. Prepositions (*Prep*) allow for the insertion of words such as *the* or *a* to improve readability. *wait for* specifies a synchronization point.

An example of an expression in the above language is as follows:

```
turn on room lights; close drapes;
  show news on television.
```

Each of the above expressions in the language can then be translated into a workflow expressed in a lower level language. Since we assume that each device can be operated on via invocation of Web services, this lower level language is BPEL4WS. Note that some of the commands, although different in the command language, might invoke the same Web services with different parameters. The commands in the task language are a combination of a verb and a noun, some commands with parameters such as ‘news’. In practice, the vocabulary of verbs and nouns can be based on a task ontology such as CLEPE (Ikeda et al., 1998). Alternatively, the nouns and verbs can be extracted dynamically by a service discovery process. This is the approach we have taken here, which involved dynamically creating the parser’s lexical analyzer from terminal symbols discovered at runtime.

The reason for the additional level of commands above the BPEL4WS level is abstraction, so that the user does not think of device ecology workflows in terms of Web services but rather in terms of what he/she observes or would expect of the devices in everyday terms. For example, a user who does not know anything about Web services can still command the device ecology based on the high-level commands.

Moreover, many of these high-level commands are applicable in different settings. For example, any room that has lights can be commanded with “turn on all lights” but such a command will translate into a different low-level workflow, depending on what lights are available in the room itself. Such translations from high-level commands to low-level workflows can be pre-defined (e.g., by an administrator who is either a vendor or a savvy user) for each room using methods such as that presented above. Hence, the high-level command to turn on all lights has a different meaning or interpretation which is predicated on the actual room (i.e., the actual device ecology) the command is issued against. We term such commands *polydeco commands*, referring to commands whose meaning is device ecology dependent.

There would also be commands that are applicable for different devices, and depending on the device, such commands will take on different meanings (and interpretation). For example, the command “switch on” can be applied to a light or to the television and the command “open” can be applied to drapes or to doors. We term such commands *polydevice commands*. Similar to polydeco commands, the actual meaning of the polydevice commands can be pre-defined, i.e., mappings from each command on a device can be mapped to a conversation with the device.

Ideally, a user, through experience of and general knowledge about the world, knows intuitively how to command devices and device ecologies, and so, does not need to learn about Web services or learn a new command set for every room visited or for every device encountered. There will be devices that an individual would not know about (e.g., new innovations) or would not know the full features of - the user will then need to learn new commands, perhaps adding to those already available by general knowledge.

2.3 Device Ecology Workflows in BPEL4WS

BPEL4WS represents concurrent execution of processes by service invocations within a *flow* construct, and sequential execution by a *sequence* construct. Interfaces between processes and with the environment are represented by messages whose type and structure can be defined from primitive types. Messages are received into variables, which form the basis of communication between processes.

2.4 Translation

Each command in the top level language is translated into a device ecology workflow in BPEL4WS. We assume that there is a device ecology workflow engine for executing BPEL4WS specifications - we return to this point later. A command might be translated into a set of alternative workflows in the lower level language, where if one alternative fails during execution, another can be tried. *Eco* expressions are translated into BPEL4WS *flow* constructs, except where the *wait* operator signifies sequential execution. For example, the following expression:

```
turn on room lights; wait for lights;
show news on television.
```

is translated into the following core commands in BPEL4WS. (*assign* commands have been omitted in the interests of brevity.):

```
<sequence>
 <invoke partnerLink="lights"
   portType="lights:lightsSoap"
   operation="turn"
   inputVariable="turnIn">
 </invoke>
 <invoke partnerLink="television"
   portType="tv:tvSoap"
   operation="show"
   inputVariable="showIn">
 </invoke>
</sequence>
```

where [`turn on lights`] maps to a lower level workflow where a number of lights are turned on (corresponding to a number of Web service calls) and [`close drapes`] is a Web service call.

The mappings from high-level commands to lower level workflows need to be kept in order that faults at the lower level might be reflected up to the corresponding commands, since the user perceives the commands as the units of activity.

2.5 Conversations with Devices

In the lower level workflow, we have assumed that each task refers to a Web service call. In a simple extension to this model, each such task on a device might require a series of invocations on one or more Web service calls on the device. For example, to turn on a light, the system might first make a Web service call to query the status of the light, and if the light is off, then make a call to another Web service (also to the same device) to switch the light on. In short, in general, a conversation with a device (comprising several Web service exchanges of the kind modelled by the Web Services Conversation Language (World Wide Web Consortium, 2002)) might be required for a workflow task on the device.

Finite state machines can be used to model such conversations as in (Benatallah et al., 2003). For example, to switch the light on would involve first invoking the get status service and then based on the result returned, possibly invoking another Web service to turn the light on. The device itself might make calls back to the device ecology workflow engine that is executing the workflow - for example, to notify a subscriber who registered to be notified of an event.

UPnP devices will tend to require such conversations. For example, the UPnP specification for a printer device (UPnP Forum, 2000b) has *actions* (which can be viewed roughly as method calls) such as *GetPrinterAttributes*, *GetJobAttributes*, *CreateJob* and *CancelJob*. Hence, cancelling a job with a printer might involve first inquiring about a job before issuing a cancel, or a task to print a document might involve sending the job to the printer and then waiting for it to finish (either checking the job status, or if supported, registering to be notified of a job completion event).

3 END-USER CONTROL OF DEVICE ECOLOGY WORKFLOW EXECUTION: INTER-LEVEL WORKFLOW MANAGEMENT

Execution of a high-level device ecology workflow will result in execution of low-level device ecology workflows, which, in turn, will result in conversations with devices. We allow the user to control workflow execution by issuing workflow operations to the workflow engine.

The mapping between levels of workflows implies that interruptions to workflow execution due to faults will need to be reflected upwards to the higher level workflow. For example, the command to turn on room lights will fail if one of the lights cannot be turned on. At this point, either a rollback occurs in which lights turned on are switched off or in cases where a rollback is inappropriate or impossible, some compensatory action needs to be performed.

In addition, control at the higher level workflow needs to be propagated to the lower levels. For example, if the command to turn on room lights has been issued, but the user decides to revoke this command (or simply to terminate the higher-level workflow in the midst of execution) and the resulting lower-level workflow has started but not yet completed (e.g., only one of three lights have been switched on), then the lower-level workflow must also be stopped (and rollback or compensatory actions performed). What we have is a situation akin to nested transactions in databases. However, we have greater complexity in the case where several different operations are possible on device ecology workflows, including start, stop, but also operations such as suspend and resume, undo (similar to rolling back), which when issued for a high-level workflow must be reflected down to lower-levels, and from the lower-levels down to the individual workflow task level (i.e. to the device conversations). For example, a cancel operation on a task of a high-level workflow might translate down to cancelling the corresponding low-level workflow which in turn translates down to cancelling a device conversation. But different possible translations are possible. Suppose that a low-level workflow instance is executing and in the middle of a conversation being carried out with the device, cancelling the corresponding low-level workflow translates down to certain other service invocations to reverse the state of the device, instead of simply stopping the conversation. Also, cancelling a high-level workflow task might translate down not to cancelling the low-level workflow but triggering certain compensatory tasks. For greater flexibility, what is required is a means to specify the translation semantics of operations on the high-level workflow to operations on the low-level workflow, and of operations on the low-level workflow to operations on device conversations.

In the following subsections, we introduce operations on tasks and operations on workflows and illustrate their correspondences.

3.1 Operations on a Task

Given a task in a workflow, the state diagram in Figure 2 shows the operations on the task and what states the task would move to as a result. A task can be in any of five states: not started, executing, suspended, completed, and abnormally terminated, and the allowable operations are defined to be start, cancel, done, undo, pause, and resume. Not all the operations make sense in all the states, and so the diagram only shows the operations which makes sense in each

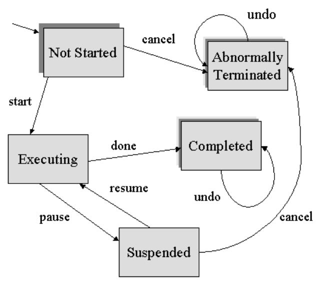

Figure 2: State diagram for operations on a task.

of the states. Moreover, only changes on the task state due to the operations are shown, and not the effects of the operations on the devices (or the devices' states). The task state is stored within the device ecology workflow engine.

A task either maps to a (1) conversation in the case of the task being within a low-level workflow, or to a (2) low-level workflow in the case of the task being within a high-level workflow (e.g., the task is a high-level single device or multidevice command as in Section 2).

1. If a task involves a conversation with the device, the task state of "executing" would mean that the conversation is happening and service calls are being made with the device, "suspended" would mean that no service calls are currently being made even if some calls have already been made (e.g., in the middle of a conversation) and so, resuming would mean the conversation with the device is continued, "completed" would mean that the conversation completed normally as pre-specified, and "abnormally terminated" would mean that the task did not involve a conversation that completed based on what is prescribed as normal (the conversation was abruptly cancelled). Typically, we do not expect it be possible to cancel or pause in the middle of a service call, and so, a cancel or pause issued to the task while a service call is made will take effect only after the call has returned. However, it is sometimes possible to terminate an on-going operation of a device - e.g., cancelling a print job.

   Starting a task would mean beginning a conversation between the device ecology workflow engine and the device. Operations start, cancel, pause, and resume have the same semantics for different devices, whereas the operation undo would depend on the device. The device can either provide an undo service call which reverses effects since the start of the conversation (or a compensatory service call (e.g., as defined in (Benatallah et al., 2003) for Web services in general) for non-reversible effects to compensate for effects since the start of the conversation), or an undo (a compensatory) service call for identified calls in the conversation. For example, a light might support service call(s) to switch it on and off, which have complementary effects. If the device provide no such (effect reversal or compensatory) calls, then the effect of a user's request to undo a task cannot be effectively performed by the device ecology workflow engine, i.e. undo might be supported on some devices but not for others. Sometimes, it is not that a device doesn't support compensatory service calls but that it is physically impossible to undo an action. For example, cancelling a submitted and queued (but not yet executed) print job is possible, whereas reversing a print job where the document has already been printed is not.

2. If a (high-level) task maps to a low-level workflow, the operations on the high-level task then maps to operations on the corresponding workflow, as explained in the next two subsections.

3.2 Operations on a Workflow

We can now define operations on a (high-level or low-level) workflow in terms of corresponding operations on tasks within the workflow.

- To start a workflow would mean to start the first task of the workflow.
- To cancel a workflow would mean to cancel all tasks of the workflow, whether the tasks have not yet been started, or suspended (a completed task cannot be cancelled but might be undone). Completed tasks are not affected. A cancellation should be issued after a pause.
- To undo a workflow is to undo all completed or terminated tasks, and can only be carried out if all the tasks are either completed or terminated (e.g., if the workflow has been cancelled). An undo should be issued after completion or cancellation of the workflow.
- To pause a workflow is to suspend all currently executing tasks.
- To resume a workflow is to resume all currently suspended tasks.

Workflows for businesses have explored operations such as cancel (Aalst et al., 2003) but the operations we attempt to support are richer here since substantial control needs to be provided for users of device ecologies.

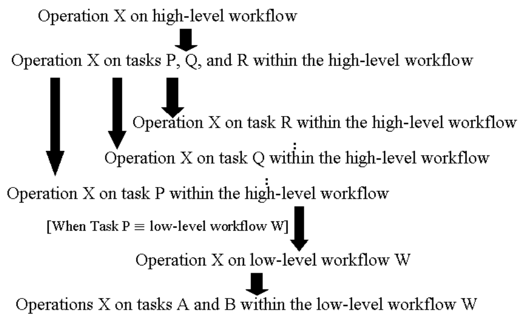
Operation X on high-level workflow
Operation X on tasks P, Q, and R within the high-level workflow
Operation X on task R within the high-level workflow
Operation X on task Q within the high-level workflow
Operation X on task P within the high-level workflow
[When Task P ≡ low-level workflow W]
Operation X on low-level workflow W
Operations X on tasks A and B within the low-level workflow W

Figure 3: An example mapping from operation X on a high-level workflow operation to corresponding operations on tasks in a low-level workflow. The down arrows denotes 'translates down.'

3.3 Correspondence of Operations in High-Level Workflows with Operations in Low-Level Workflows

Operations on high-level workflows are mapped to operations on tasks within the high-level workflow in the manner described earlier. But an operation on a task of a high-level workflow, where the task is mapped not to a conversation with a device but to a low-level workflow, will be mapped to an operation on the low-level workflow. The operation on a low-level workflow then maps to operations on tasks within the low-level workflow in the manner desribed earlier.

Figure 3 shows an example route where an operation on a high-level workflow is mapped to operations on tasks within a low-level workflow. n operation X on a high-level workflow (issued by the user say) is mapped to operation X on three tasks P, Q, and R of the workflow. Each of these operations are subsequently mapped down to device conversations. The figure shows the mapping for the operation X on task P. Because task P corresponds to a low-level workflow W, the operation X on P is effectively an operation X on W. The operation on the workflow W is then mapped to operation X on task A and B in W. The operation X on task A (involving device A, say) and the operation X on task B (involving device B, say) will result in conversations via calls to Web services for device A and for device B.

Using our example on turning on lights, once a workflow for

```
turn on all lights; wait for lights;
open drapes.
```

has been issued, it will start to execute. After some time, before completion, if the user now issues a pause command, the command will be translated down to the task level, and the workflow will be suspended according to the semantics given above. The aim of the operations on tasks and operations on the workflows is to allow user control (at user's will) over them, even during their execution.

3.4 Handling Faults

So far, we have not considered in detail the mapping upwards of faults occurring in service calls during conversations to faults in low-level workflows and then to faults in high-level workflows.

Mapping rules are required in order to define how an exception in a service call within a conversation will be manifested at the high-level workflow, and ultimately to users. Depending on such definitions, the response to an exception might be to cancel and undo operations (including compensatory actions). Moreover, some faults might not be reflected up to higher levels and need only be dealt with at a lower level. If a high level task corresponds to two alternative low-level workflows, a fault in one low-level workflow can be handled by first undoing the effect of the parts of this workflow that has executed and then starting the alternative workflow. Alternatively, some effects of operations might not be rolled back or compensated but can be safely ignored - depending on the application scenario.

As example, consider the high-level task to turn on all lights in the room. Such a high-level task translates down to a low-level workflow and on execution of the low-level workflow, perhaps only two of the three lights are successfully turned on (say light 2 is faulty). One way to handle the problem is ignore the fault and continue with the other tasks in the workflow, which makes sense in this example. Another way to handle this fault is to undo the entire workflow (undoing any successful operations before encountering the fault - switching light 1 off) and stop. A third way to handle the problem is to suspend the workflow when the fault is encountered (say detected when the service call to light 2 fails) and query the user about what to do (e.g., to undo the workflow or ignore the fault). The first solution is less obtrusive, to silently (with respect to the user) ignore faulty operations and complete the workflow, but logging faults detected for future analysis and reporting. All three solutions can be supported by the device ecology workflow engine but the actual behaviour in a particular workflow has to be pre-defined (again by an administrator or a savvy user).

Solutions to deal with dynamic changes in workflow have been discussed at a higher-level of abstraction in terms of business process (Aalst, 2001). Some of the techniques can be applied to our work, but the focus will be on dealing with device failure and service failure.

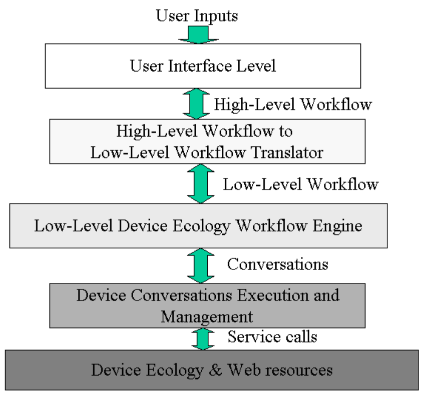
User Inputs
User Interface Level
High-Level Workflow
High-Level Workflow to Low-Level Workflow Translator
Low-Level Workflow
Low-Level Device Ecology Workflow Engine
Conversations
Device Conversations Execution and Management
Service calls
Device Ecology & Web resources

Figure 4: Multilayered Conceptual Architecture for Device Ecology Workflow Engine

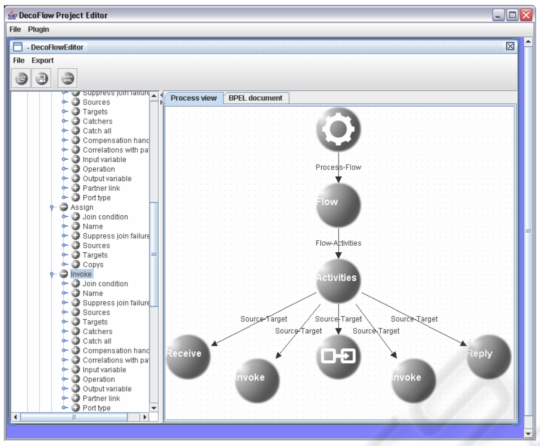

Figure 5: DecoFlow application with a sample BPEL4WS document

4 PROTOTYPE ARCHITECTURE AND CURRENT IMPLEMENTATION

The conceptual architecture for the engine that executes and manages device ecology workflows is shown in Figure 4. Roughly, each layer of the architecture shows the components required to manage a level of abstraction. Note that user operations on an executing workflow such as cancel, pause, etc, can be issued during workflow execution - not shown in the diagram. The system keeps track of the correspondences between device conversations and the associated tasks in the low-level workflow, and between tasks in the high-level workflow and the associated low-level workflows, in order that exceptions are correctly reflected up (if required) through the abstraction levels. If there is an exception in a call to device, it can be traced to its corresponding low-level workflow task and then to its corresponding high-level workflow task. For example, if there is an exception in the call to switch on a light, it can be traced to the high-level task of turning on all lights, and the system might report to the user that there is an error in carrying out that high-level task but allow the user to drill down to specific faults in the lower levels.

We are currently developing a prototype of our system with a subset of BPEL4WS as the low-level workflow language. Figure 5 shows the DecoFlow application with a sample BPEL4WS document loaded for visualisation. On the left side of the editor window is a tree displaying all of the information obtained from the BPEL4WS document. On the right hand side is the visualisation of the BPEL4WS document which is represented by a series of connected nodes which represent the process and activities from the BPEL4WS document.

5 CONCLUSION AND FUTURE WORK

We have discussed three levels of abstraction: high-level workflow, low-level workflow and device conversations, involved in programming device ecology workflows, and how inter-level control is passed between levels. Such levels of abstraction are important since the system should be as user friendly as possible while permitting programmability not only at high-levels of abstraction but also at low levels of detail.

We can consider more than the three levels of abstraction. One can iteratively build higher level command languages over the top level command language, allowing workflows to be specified at an even higher level of abstraction, but all linked within a uniform formal model. Future work also involves continuing our prototype implementation of our model in a device ecology workflow engine which supports the translation semantics and fault handling schemes described earlier, and to extend our BPEL4WS low-level workflow language with tasks that link to conversation specifications. Moreover, other work on dynamically selecting devices for a particular task such as (Kumar et al., 2003; Butler, 2002) are complementary to our work - techniques can be considered in using semantics to select devices for a workflow at run-time. We are also working on methods to analyze device ecology workflows before execution (Loke and Ling, 2004). We are also working on an ontology of polydeco and polydevice commands.

ACKNOWLEDGEMENTS

We would like to thank the Australian Research Council for partial funding of this work.

Trademarks. *UPnP* is a trademark of UPnP Forum. *Jini* is a trademark of Sun Microsystems.

REFERENCES

Aalst, W. (2001). Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317.

Aalst, W., Hoftede, A., Kiepuszerski, B., and Barros, A. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51.

Association of Home Appliance Manufacturers (2002). *Connected Home Appliances Object Modelling, CHA-1-2002*. Available at http://www.aham.org/.

Benatallah, B., Casati, F., Toumani, F., and Hamadi, R. (2003). Conceptual Modelling of Web Service Conversations. Technical Report HPL-2003-60, HP Labs.

Bentham, J. (2002). *TCP/IP Lean: Web Servers for Embedded Systems (2nd Edition)*. CMP Books.

Butler, M. (2002). Using Capability Profiles for Appliance Aggregation. Technical Report HPL-2002-173, HP Labs.

Durand, Y., Vincent, S., Marchand, C., Ottogalli, F., Olive, V., Martin, S., Dumant, B., and Chambon, S. (2003). SIDRAH: A Software Infrastructure for a Resilient Community of Wireless Devices. In *Proceedings of the Smart Objects Conference (SOC'03)*, Grenoble.

Ikeda, M., Seta, K., Kakusho, O., and Mizoguchi, R. (1998). An Ontology for Building a Conceptual Problem Solving Model. In *ECAI98 Workshop on Applications of ontologies and problem-solving model*, pages 126–133, Brighton, England.

Kohtake, N., Matsumiya, K., Takashio, K., and Tokuda, H. (2003). Smart Device Collaboration for Ubiquitous Computing Environment. In *Proceedings of the Workshop on Multi-Device Interface for Ubiquitous Peripheral Interaction at the 5th International Conference on Ubiquitous Computing (UbiComp'03)*.

Kumar, R., Poladian, V., Greenberg, I., Messer, A., and Milojicic, D. (2003). Selecting Devices for Aggregation. In *Proceedings of the WMCSA 2003 (to appear)*.

Loke, S. (2003). Service-Oriented Device Ecology Workflows. In Orlowska, M., Weerawarana, S., Papazoglou, M., and Yang, J., editors, *Proceedings of the International Conference on Service-Oriented Computing, Lecture Notes in Computer Science 2910*, pages 559–574, Trento, Italy. Springer-Verlag.

Loke, S. and Ling, S. (2004). Analyzing Observable Behaviours of Device Ecology Workflows. In *Proceedings of the 6th International Conference on Enterprise Information Systems*, pages 78–83, Portugal.

Masuoka, R., Parsia, B., and Labrou, Y. (2003). Task Computing - the Semantic Web meets Pervasive Computing. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, Florida, USA.

Matsuura, K., Haraa, T., Watanabe, A., and Nakajima, T. (2003). A New Architecture for Home Computing. In *Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Systems (WSTFES03)*, pages 71–74.

Microsoft, IBM, Siebel, BEA, and SAP (2003). *Business Process Execution Language for Web Services Version 1.1*. Available at http://www-106.ibm.com/developerworks/library/ws-bpel/.

Microsystems, S. (2001). *Jini Network Technology*. Available at http://wwws.sun.com/software/jini/.

Newman, M., Sedivy, J., Edwards, W., Smith, T., Marcelo, K., Neuwirth, C., Hong, J., and Izadi, S. (2002). Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments. In *Proceedings of the Conference on Designing Interactive Systems (DIS2002)*. Available at http://www.cs.berkeley.edu/˜jasonh/publications/dis2002-speakeasy-browser.pdf.

Omojokun, O. and Dewan, P. (2003). A High-Level and Flexible Framework for Dynamically Composing Networked Devices. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2003)*.

Piccinelli, G., Finkelstein, A., and Williams, S. (2003). Service-Oriented Workflows: the DySCo Framework. In *Proceedings of the Euromicro Conference*, Antalya, Turkey. Available at http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/euromicro2003.pdf.

Rodrigues, M., Teixeira, A., and Lopes, L. S. (2004). An Hybrid Approach for Spoken Natural Language Understanding Applied to a Mobile Intelligent Robot. In Sharp, B., editor, *Proceedings of the 1st International Workshop on Natural Language Understanding and Cognitive Science*, Portugal.

Sousa, J. and Garlan, D. (2003). From Computers Everywhere to Tasks Anywhere: The Aura Approach. In *Submitted*. Available at http://www-2.cs.cmu.edu/˜aura/docdir/sg01.pdf.

UPnP Forum (2000a). *UPnP Device Architecture*. Available at http://www.upnp.org/.

UPnP Forum (2000b). *UPnP DeviceType: Printer Device Template Version 1.01*. Available at http://www.upnp.org/.

Vildjiounaite, E., Malm, E., Kaartinen, J., and Alahuhta, P. (2003). Networking of Smart Things in a Smart Home. In *Proceedings of the Workshop on the Interaction of HCI and Systems Issues in UbiComp (UBIHCISYS 2003) at the 5th International Conference on Ubiquitous Computing (UbiComp'03)*. Available at http://ubihcisys.stanford.edu/online-proceedings/Ubi03w7-Vildjiounaite-final.pdf.

World Wide Web Consortium (2002). *Web Services Conversation Language (WSCL) 1.0*. Available at http://www.w3.org/TR/2002/NOTE-wscl10-20020314/.