Service-Oriented Device Ecology Workflows

Seng Wai Loke

School of Computer Science and Software Engineering
Monash University, Caulfield East, Victoria 3145, Australia
swloke@csse.monash.edu.au

Abstract. We address the need for a high level of abstraction to describe how devices should work together and to manage their interaction. Our perspective is from workflow, where business processes are managed by a workflow system that assigns tasks, passes them on, and tracks the progress. One can envision device ecologies for different purposes and situations but this paper focuses on a device ecology example within the home environment. We illustrate how a workflow model can be applied to describe and manage device ecologies – in particular, we treat devices as Web services and utilize the Business Process Execution Language for Web Services (BPEL4WS) for describing workflows in device ecologies. We also show how the DySCo workflow algebra can be employed to model device ecology workflows and discuss how to model the impact of these workflows on devices' observable states. The result of this work is a starting point for a workflow based programming model for device ecologies.

1 The Rise of Device Ecologies

We are surrounded by appliances, and increasingly so. The American Heritage Dictionary defines an appliance as "a device or instrument designed to perform a specific function, especially an electrical device, such as a toaster, for household use." Usability experts have advocated special-purpose devices (in contrast to general-purpose PCs) which, with their supporting computational and communication infrastructure, fit the person and tasks so well, are sufficiently unobtrusive and inter-connectivity seamless, that the technological details become virtually invisible compared to the task. These devices are often called *information appliances*, or *Internet appliances* if they have Internet-connectivity [5,16]. Part of "smart" behaviour is this interaction among devices and resources. The current Internet and networking technologies, and developments in wireless networking enable the "smart" devices to communicate with one another and with Internet or Web resources. The devices might need to interact effectively in order to accomplish the goals of its user(s), and such interaction ability, whether occurring over very short ranges or across continents, can open up tremendous possibilities for innovative applications.

The American Heritage Dictionary defines the word "ecology" as "the relationship between organisms and their environment." We perceive the above mentioned developments as yielding a computing platform of the 21st century that takes the form of *device ecologies* consisting of collections of devices (in the environment and on

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 559–574, 2003.

users) interacting synergistically with one another, with users, and with Internet resources, undergirded by appropriate software and communication infrastructures that range from Internet-scale to very short range wireless networks. These devices will perform tasks and work together perhaps autonomously but will need to interact with the user from time to time.

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, AutoHan [19], UPnP [12], OSGI [13], Jini [21], and SIDRAH (with short range networking and failure detection) [8] define infrastructure and mechanisms at different levels (from networking to services) for devices to be inter-connected, find each other, and utilize each other's capabilities. Embedded Web Servers [3] are able to expose the functionality of devices as Web services. Embedding micro-servers into physical objects is considered in [15]. Approaches to modelling and programming such devices for the home have been investigated, where devices have been modelled as software components [9], as collections of objects [1], as Web services [14], and as agents [18,6]. 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.

In this paper, we address the need for a high level of abstraction to describe how devices should work together and to manage their interaction. Our perspective is from workflow, where business processes are managed by a workflow system that assigns tasks, passes them on, and tracks the progress. We can have device ecologies for different situations and purposes. Here, we focus on device ecologies within the home environment. We first provide some background in Section 2. Then, in Section 3, we illustrate how a workflow model can be applied to describe and manage device ecologies – in particular, we treat devices as Web services and utilize the Business Process Execution Language for Web Services (BPEL4WS) [2] for describing workflows in device ecologies. For short, we call such device ecology workflows *decoflows*. The idea of a DecoFlow Engine is outlined in Section 4. We also show, in Section 5, how the DySCo [17] workflow algebra can be employed to model decoflows and discuss how to model the impact of decoflows on devices' observable states. We conclude with future work in Section 6.

2 Preliminaries

2.1 Modelling Devices

We model the observable and controllable aspects of devices as Web services as done in [14]. Such device modeling is not inconsistent with emerging standard models for appliances such as the AHAM Appliance Models [1], where each appliance (such as clothes washer, refrigerator, over, room air conditioner, etc) is modelled as a collection of objects categorized according to subsystems. In this paper, we assume that

aspects of devices can be directly observed and controlled by means of a collection of Web services, described using the Web Services Description Language (WSDL) [7]. We note that there will be aspects of the device which are not exposed as Web services.

2.2 BPEL4WS

BPEL4WS is an XML language for specifying business process behaviour based on Web services. Quoting from [22]:

"It (BPEL4WS) allows you to create complex processes by creating and wiring together different activities that can, for example, perform Web services invocations, manipulate data, throw faults, or terminate a process. These activities may be nested within structured activities that define how they may be run, such as in sequence, or in parallel, or depending on certain conditions."

BPEL4WS has language constructs for manipulating data, handling faults, compensating for irreversible actions, and various structured activities. BPEL4WS assumes that there is a central engine which is executing the workflow. A BPEL4WS process or workflow is viewed as the central entity invoking Web services associated with its (business) partners and having its own services invoked by partners (e.g., when the partner initiates the business process or receives results for a previously sent request). We describe the syntax and semantics of the BPEL4WS constructs together with our example below.

3 Modelling Device Ecology Workflows

3.1 An Example Decoflow

We consider a decoflow for someone we call Jane involving a television, a coffee-boiler, bedroom lights, bathroom lights, and a news Web service accessed over the Internet. Figure 1 describes this decoflow. 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 decoflow is initiated by a wake-up notice from Jane's alarm clock which we assume here is issued to the Device Ecology Workflow Engine (which we call the *Decoflow Engine*) when the alarm clock rings. This notice initiates the entire workflow. Subsequent to receiving this notice, five activities are concurrently started: retrieve news from the Internet and display is on the television, switch on the television, boil coffee, switch on the bedroom lights, and switch on the bathroom lights. Note the synchronization arrow from "Switch On TV" to "Display News on TV", which ensures that the television must be switched on before the news can be displayed on it. After all the concurrent activities have completed, the final task is to blink the bedroom lights, in order to indicate to Jane that the decoflow tasks have completed. This scenario was inspired by [4].

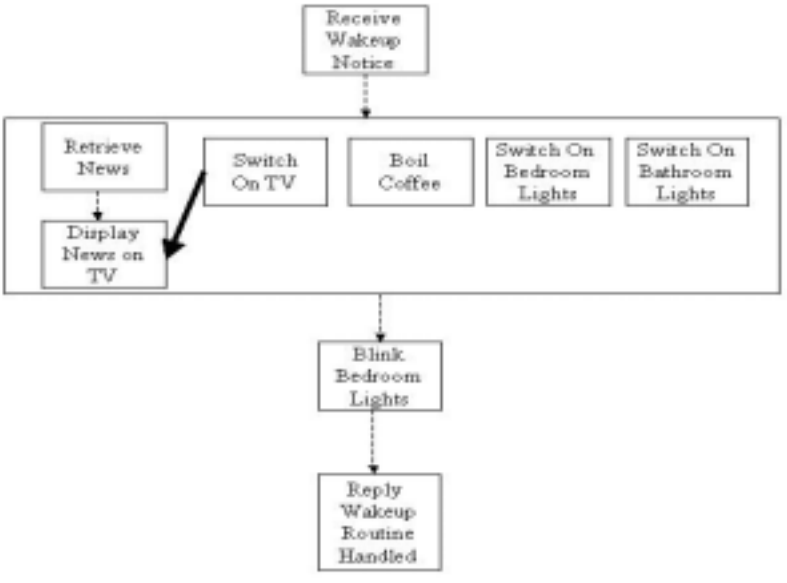

Fig. 1. A Device Ecology Workflow

3.2 The Decoflow in BPEL4WS

The above decoflow can be described using BPEL4WS. The process is given in outline as follows in Figure 2:

```
<sequence>
        <receive partnerLink="wakingUp"
                        portType="lns:wakeUpNoticePT"
                        operation="sendWakeUpNotice"
                        variable="WN">
        </receive>
        <flow>


                <links>
                        <link name="tv-to-news"/>
                </links>
                <sequence>

                   <invoke partnerLink="newsRetrieval"
                       portType="lns:newsUpdatePT"
                       operation="requestNews"
                          inputVariable="newsRequest"
                                  outputVariable="newsInfo">
                   </invoke>

                   <invoke partnerLink="tv"
                          portType="lns:tvControlPT"
                       operation="displayOnTV"
```

```
                    inputVariable="newsInfo"
                    <target linkName="tv-to-news"/>
                </invoke>
            </sequence>

            <invoke partnerLink="tv"
                    portType="lns:tvControlPT"
                    operation="sendTVCommand"
                    inputVariable="switchOnTVRequest">
                    <source linkName="tv-to-news"/>
                   </invoke>

            <invoke partnerLink="coffeeBoiling"
                    portType="lns:boilerControlPT"
                    operation="sendCoffeeBoilerCommand"
                    inputVariable="boilCoffeeRequest"">
            </invoke>

            <invoke partnerLink="bathroomLighting"
                    portType="lns:bathroomLightControlPT"
                    operation="sendLightCommand"
                    inputVariable="switchOnLightRequest">
            </invoke>

            <invoke partnerLink="bedroomLighting"
                    portType="lns:bedroomLightControlPT"
                    operation="sendLightCommand"
                    inputVariable="switchOnLightRequest">
            </invoke>
        </flow>

        <invoke partnerLink="bedroomLighting"
                portType="lns:bedroomLightControlPT"
                operation="sendLightCommand"
                inputVariable="blinkLightRequest">
        </invoke>
        <reply partnerLink="wakingUp"
                portType="lns:wakeUpNoticePT"
                operation="sendWakeUpNotice"
                variable="WNHandled"/>
        </reply>
    </sequence>
```

Fig. 2. Outline of the process in BPEL4WS.

We first note the structure of the decoflow indicated by the following tags in Figure 2. The outermost <sequence>...</sequence> tags sequences the inner four activities encapsulated within the tags <receive...>...</receive>, <flow>...</flow>, <invoke...>...</invoke>, and <reply...>...</reply>:

1. The first <receive...>...</receive> activity and the final <reply...>...</reply> activity represents the DecoFlow Engine receiving the wake-up notice, and informing the alarm clock that the wake-up notice has been handled.
2. The <flow>...</flow> tags indicate that the five immediately nested activities, i.e., the <sequence>...</sequence> activity and the four <invoke...>...</invoke> activities are to be carried out concurrently.

3. The second last <invoke…>…</invoke> blinks the bedroom lights before replying the alarm clock.
4. The <link>…</link> from tv to news represents the dependency of the display news operation on the switch on television operation. Note that the switch on television operation is the source and the display news operation is the target, since the target depends on the source, and so, the source must complete before the target begins.

BPEL4WS has the notion of partner link, where a partner has a relationship with the business process, and represents both a consumer of a service provided by the business process and a provider of a service to the business process. In this context, a partner relationship between a decoflow and a device signifies that the device is used in the decoflow. A partner link can be defined by two roles, one to be played by the partner and the other by the business process. Sometimes only one role is given in a partner link definition implying that the partner expresses a willingness to link with the business process without placing any requirements on the business process, or conversely, i.e. the business process links with other partners without requirements on the partner. In this context, such roles provide semantics as to the role played by a device for that decoflow. For example, the following partner link is between the decoflow and the television, where the television is viewed as a service.

```
<plnk:partnerLinkType name="tvLT">
     <plnk:role name="tvService">
            <plnk:portType name="pos:tvControlPT"/>
     </plnk:role>
</plnk:partnerLinkType>
```

There is only one role in this definition, and it is to be taken up by the television. The port type given is a communication endpoint for accessing television controls. Note that only one role is given and is taken up by the device. The decoflow need not provide any services in this partnership with the television.

Associated with each port type is one or more operations, corresponding to Web service methods. For example, the tvControl port type has the displayOnTV operation.

```
<portType name="tvControlPT">
     <operation name="displayOnTV">
            <input message="pos:newsInfoMessage"/>
            <fault name="cannotCompleteWN"
     message="pos:WNFaultType"/>
     </operation>
</portType>
```

If there is fault with the invocation of this operation, the fault handler is invoked as explained later.

Using such partner link types, actual partner links can be defined. For example, the following partner link is an instance of the tv partner link type, where the partner or the device's role is specified as one offering the television service. Since there is

only one role, no requirements are placed on the decoflow (or the DecoFlow Engine) in this partner relationship between the decoflow and the television.

```
<partnerLink name="tv"
       partnerLinkType="lns:tvLT"
       partnerRole="tvService"/>
```

But consider the following partner link type and its instance partner link.

```
<plnk:partnerLinkType name="wakingUpLT">
     <plnk:role name="wakingUpService">
            <plnk:portType name="pos:wakeUpNoticePT"/>
     </plnk:role>
</plnk:partnerLinkType>

     <partnerLink name="wakingUp"
            partnerLinkType="lns:wakingUpLT"
            myRole="wakeUpService"/>
```

The partner link type defines only one role but unlike the partner link type definition for the television, this role is taken up not by a device but by the decoflow instead. Also, since there is only one role, no requirements are placed on a partner device in this partner relationship between a device (which we have assumed is an alarm clock) and the decoflow. In other words, the decoflow offers this wakingUpService to any device, or any device can issue a wake up notice to initiate an instance of this decoflow, but only because we have defined the decoflow in this way.

Partner relationships with the other devices and their port types are similarly defined. We have not assumed an ontology for describing partner relationships and port types. In practice, device standards such as [1] can be used to specify the operations and port types for the respective devices. Through the use of such standards, such operations can be expected of different appliances (even of different manufacturers) as long as they adhere to the standards. We can view defining such partner relationships between a device and a decoflow as defining an abstraction (of some aspects) of the device which will be utilized in the decoflow, where this abstraction of the device is a subset of the device's capabilities (as accessed by their corresponding operations).

3.3 A Library of Decoflows

The above decoflow is only one example of how a wake-up routine can be captured. There could be other routines such as come-home-from-work routines or entertain-guests routines, etc, which can be captured in decoflows. A library of decoflows can be constructed, indexed on particular situations.

We also note that some of these decoflows might not be invoked by the user (or through the user's initiative) but by devices' own initiative. For example, a device might initiate a decoflow to further secure a home if it detects unwanted intruders when the owners are out for dinner, or initiate a decoflow to replenish (e.g. to order) certain household items based on a provided budget. A decoflow initiated by a device can be programmed to seek human approval for more critical tasks.

4 Decoflow Execution via the Device Ecology Workflow Engine

The above workflow can be executed by the DecoFlow Engine, as Figure 3 illustrates.

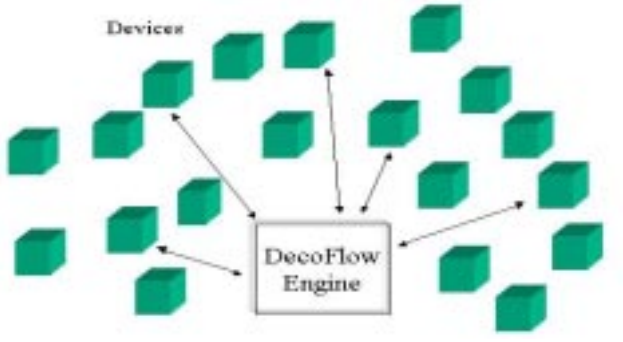

Fig. 3. DecoFlow Engine interacting with Device Ecology

The cubes represent devices and the bi-drectional arrows indicate messages exchanged with these devices while executing a decoflow specification. We assume the devices are described and connected to the DecoFlow Engine via some underlying networking and service infrastructure such as UPnP. The DecoFlow Engine is in charge of invoking the appropriate Web services on the devices, in order to request resources to perform tasks and maintaining the execution state of the process.

The execution state of the process are recorded in one or more declared variables such as the following, whose values can be used as inputs to operations or to store outputs of operations.

```
<variables>
       <variable name="WN" messageType="lns:WNMessage"/>
       <variable name="newsRequest"
              messageType="lns:newsRequestMessage"/>
       <variable name="newsInfo"
              messageType="lns:newsInfoMessage"/>
       <variable name="boilCoffeeRequest"
              messageType="lns:boilCoffeeRequestMessage"/>
       <variable name="switchOnLightRequest"

      messageType="lns:switchOnLightRequestMessage"/>
       <variable name="blinkLightRequest"
             messageType="lns:blinkLightRequestMessage"/>
       <variable name="switchOnTVRequest"
              messageType="lns:switchOnTVRequestMessage"/>
       <variable name="WNHandled"
              messageType="lns:WNHandledMessage"/>
       <variable name=
           "WNFault" messageType="lns:WNFaultType"/>
</variables>
```

It is too simplistic to assume that the decoflow will complete without any problems. BPEL4WS has constructs for defining fault handlers and for throwing exceptions. For example, the following description states that if a fault occurs in the invo-

cation of an operation, i.e., the wake up routine (as defined by the decoflow) cannot be completed, a reply is sent to the alarm clock that there has been a fault.

```
<faultHandlers>
      <catch faultName="lns:cannotCompleteWN"
            faultVariable="WNFault">
            <reply partnerLink="wakingUp"
                  portType="lns:wakeUpNoticePT"
                  operation="sendWakeUpNotice"
                  variable="WNFault"
                  faultName="cannotCompleteWN"/>
      </catch>
</faultHandlers>
```

By including the following declaration in port type definitions

```
<fault name=
    "cannotCompleteWN" message="pos:WNFaultType"/>
```

operations can be associated with the fault handler by means of the common fault name.

5 Formal Modelling of Devices' Decoflow and Changes in Observable States

To the author's knowledge, BPEL4WS has not yet been given a formal semantics. But a basic formalization for workflow is given in a process algebraic manner for the DySCo framework [17], where the basic tasks in these workflows are Web service invocations. By modelling the controllable and observable aspects of devices as Web services, we can employ this formalisation in specifying decoflows.

From a given device's point of view, assuming that the device has an observable state, we would also like to model the changes to the device's observable state as the decoflow executes. Some of the activities of the decoflow can affect the device's observable state (for example, if the activity is one which invokes a Web service of the device) but there will be activities of the decoflow which does not affect the device's observable state (e.g., if the activity invokes a Web service of some other device). A device can also, with respect to external observers, spontaneously move from one observable state to another, that is, the device might change its observable state without sending or receiving any messages. Such *spontaneous moves* has been modelled in [20], where agents are modelled with observable states and the observable state of an agent can spontaneously change, spontaneous in the sense that it changed without the agent receiving any messages. The assumption there is that the spontaneous change is due to the agent's own internal processes, and represents the proactive nature of these agents. We can similarly model devices with such spontaneous changes, where spontaneity is with respect to a given decoflow and is in the sense that the change is not due to any of the activities of the decoflow being considered. In reality, such changes, thought spontaneous, might be due to some other decoflow that is concurrently executing, or due to actions of other users directly on the device – for example, before the decoflow instructs the bathroom lights to turn on, the user might have already turned it on.

Below, we present a basic formalization of decoflow including synchronization between concurrent activities based on the DySCo formalization in [17]. Other for-

malizations for service-oriented workflows might also be used. We need only consider global state in decoflows. We also provide transition rules that can be used to model changes in a device's observable state.

5.1 Decoflows with DySCo

The grammar for decoflows are given as follows.

$W ::= \varepsilon$	empty decoflow	
$\mid T$	task	where
$\mid W.W$	sequence	$T ::= t_d(\sigma).\lambda(n)$
$\mid W +_c W$	choice	$\mid \lambda'(n).t_d(\sigma)$
$\mid W \mid\mid W$	concurrency	$\mid t_d(\sigma)$
$\mid !W$	loop	

where W represents a decoflow, λ(n) and λ'(n) represents the source of a synchronization link named n and the target of the synchronization link, respectively, $t_d(\sigma)$ represents a task utilising device d with actual parameters given by σ: N –>V (where N is a set of variable names and V is a set of values), T is a task suffixed by a source of a synchronization link, a task prefixed by a target of a synchronization link, or a task without any synchronization links. The condition c for choice is a binary function with domain N. This formulation of workflows is the same as that in DySCo but with a definition of task that includes synchronization links, and resource is, in our case, a device. The task, in our case, assuming we use BPEL4WS operations, is an operation on a device, which is either an invoke, receive or reply.

The labelled transition system comprises the following rules of the form $(label)\frac{premises}{conclusion}$ taken from [17]:

$$(step)\frac{\sigma' = \rho(t, d, \Omega \triangleright \sigma)}{\Omega :: t_d(\sigma) \xrightarrow{\varphi} \Omega \triangleleft \sigma' :: \varepsilon}$$

where φ represents information to be made externally visible as a result of the action. In decoflows, we can interpret this to mean a signal given to the user that the task has been carried out. Ω represents the global state of the workflow. The functions (ω) and ($\overline{\omega}$) are used to extract information from the global state and to feed information into the global state respectively. ρ is a function representing the results of the execution of a task.

$$(loop)\frac{}{\Omega :: !W \xrightarrow{\tau} \Omega :: W(!W)}$$

where τ represents null visible information.

Sequence is represented as follows:

$$(seq1)\frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1'}{\Omega::W1.W2 \xrightarrow{\alpha} \Omega'::W1'.W2}$$

$$(seq2)\frac{}{\Omega::\varepsilon.W2 \xrightarrow{\tau} \Omega::W2}$$

The symbol α represents visible information. Choice depends in the binary function c whose evaluation determines which alternative to execute. Two rule are given which represents the non-determinism.

$$(choice1)\frac{eval(c)=1}{\Omega::W1 +_c W2 \xrightarrow{\tau} \Omega::W1}$$

$$(choice2)\frac{eval(c)=2}{\Omega::W1 +_c W2 \xrightarrow{\tau} \Omega::W2}$$

Concurrency is given by the following four rules:

$$(conc1)\frac{}{\Omega::\lambda(n).W1 || \lambda'(n).W2 \xrightarrow{\alpha} \Omega::W1||W2}$$

$$(conc2)\frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1'}{\Omega::W1||W2 \xrightarrow{\alpha} \Omega'::W1'||W2}$$

$$(conc3)\frac{\Omega::W2 \xrightarrow{\alpha} \Omega'::W2'}{\Omega::W1||W2 \xrightarrow{\alpha} \Omega'::W1||W2'}$$

$$(conc4)\frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1' \quad \text{and} \quad \Omega::W2 \xrightarrow{\alpha} \Omega''::W2'}{\Omega::W1||W2 \xrightarrow{\alpha} \Omega' \nabla \Omega''::W1'||W2'}$$

The four rules represent the different possibilities in concurrent execution, either we resolve a synchronization link, W1 executes first, W2 executes first, or both executes at the same time. The operator σcombines the two resulting states, perhaps serializing the updates. Note that concurrent access to Ω has not been represented explicitly. The first rule (conc1) is not in DySCo, but the rest are. (conc3) is not necessary since the concurrency operator is commutative (but is added for comparison with [17]). In addition, as observed from (conc1), execution cannot continue if the workflow is such that there is no matching source and target for each synchronization link.

The example decoflow given earlier can be expressed using the following expression:

$$rcv_a . ((inv_b.(\lambda'(tn).inv_c)) \;||\; (inv_d.\lambda(tn)) \;||\; inv_e \;||\; inv_f \;||\; inv_g) \;.\; inv_h . rep_a$$

where rcv_a is a receive via the sendWakeUpNotice operation,

inv_b is an invocation of the requestNews operation,

inv_c is an invocation of the displayOnTV operation prefixed by the link target,

inv_d is an invocation of the switchOnTVRequest operation
suffixed by the link source,
inv_e is an invocation of the sendCoffeeBoilerCommand operation,
inv_f is an invocation of the sendLightCommand operation for the bathroom,
inv_g is an invocation of the sendLightCommand operation for the bedroom,
inv_h is an invocation of the sendLightCommand operation
to blink the bedroom lights,
and rep_a is a reply to the sendWakeUpNotice operation. $\lambda(tn)$ and $\lambda'(tn)$ corresponds to the source and target of the "tv-to-news" link.

The execution proceeds as follows:

$rcv_a . ((inv_b.(\lambda'(tn).inv_c)) \parallel (inv_d.\lambda(tn)) \parallel inv_e \parallel inv_f \parallel inv_g) . inv_h . rep_a$

$\longrightarrow$ (using seq)

$((inv_b.(\lambda'(tn).inv_c)) \parallel (inv_d.\lambda(tn)) \parallel inv_e \parallel inv_f \parallel inv_g) . inv_h . rep_a$

$\longrightarrow$ (using conc repeatedly, and by commutativity of $\parallel$)

$(\lambda(tn) \parallel (\lambda'(tn).inv_c) \parallel \varepsilon \parallel \varepsilon \parallel \varepsilon) . inv_h . rep_a$

$\longrightarrow$ (using conc1)

$(\varepsilon \parallel inv_c \parallel \varepsilon \parallel \varepsilon \parallel \varepsilon) . inv_h . rep_a$

$\longrightarrow$ (using seq, and since $\varepsilon \parallel W = W$)

$inv_h . rep_a$

$\longrightarrow$ (using seq)

rep_a

$\longrightarrow$ (using seq)

ε

5.2 Modelling Changes in Devices' Observable States

Now we turn to the modelling of a device's observable state. Given a device d, we can model d's possible changes in observable states as a decoflow is being executed. Suppose D represents d's initial observable state. Then, after executing a decoflow W, we would like to compute the series of observable states that d goes through when W is executing, i.e. D^1, D^2, D^3, D^4, …, D^n, where $D \xrightarrow{W} D^n$. Note that not all the operations in W will affect d. Suppose that only two operations in W affects d, namely $t_d(\sigma)$ and $t'_{d'}(\sigma')$, and that these two operations are sequenced in W, i.e. W = W' . $t_d(\sigma)$ .W'' . $t'_{d'}(\sigma')$. W''', for some W', W'', and W'''. Also, suppose we have $D \xrightarrow{t_d(\sigma)} D'$ and $D' \xrightarrow{t'_{d'}(\sigma')} D''$. Then, $D \xrightarrow{W} D''$.

If we now consider a set of devices (d1, d2, d3, …, dk), we can work out the effect of a decoflow W on the collective observable state of the devices:

$(D_1, D_2, ..., D_k) \xrightarrow{W} (D_1', D_2', ..., D_k')$

If we perform analysis on $(D_1', D_2', ..., D_k')$, we can predict particular effects of the decoflow W. For example, we can define a function *num_lights_on* on the collective observable states of the devices, where *num_lights_on* $(D_1', D_2', ..., D_k')$ is the number of lights which have been switched on in the given state $(D_1', D_2', ..., D_k')$. By simulating the execution of W on the devices, we can determine if W will produce a collective observable state where *num_lights_on* $(D_1', D_2', ..., D_k') > 20$. A similar analysis technique can also be used to determine if a decoflow will ever switch a device on or off, and what circumstances will prevent a device from being switched off.

In reality, the problem is more complex. One complexity is due to two avenues by which a device can be controlled. One is via the system, and the other is direct control by a human user. For example, a light can be switched on manually by the user or automatically by the system in one or more decoflows. Thus, with respect to a decoflow, not all the changes in a device's observable state is due to actions of that decoflow. A user or another concurrently executing decoflow might have changed the device's observable state. To model changes in state not due to a given decoflow, we introduce the idea of *spontaneous state changes* analogous to the spontaneous moves mentioned earlier. With such spontaneous state changes, using the example of the effect of W on D above, we might not have $D \xrightarrow{W} D''$ anymore, but $D \xrightarrow{W} D'''$ for some $D''' \neq D''$, due to spontaneous state changes apart from the actions of W on d. Because it is practically impossible to predict when such spontaneous state changes might occur, when we analyse the effect of W, we do so with an assumption about the set of possible spontaneous moves that might occur during the execution of W, i.e. this set of spontaneous moves become a parameter to the analysis.

For example, suppose we have the following transition rule that defines only one possible spontaneous move of a device, which in this example, is a lamp with two observable states ("off", denoted by D, and "on", denoted by D'): $D' \xrightarrow{\text{user switch off}} D$. The user can only switch off the lamp but the system can switch it on or off automatically. The device can be switched on or off via a Web service invocation: $D \xrightarrow{\text{sendLightCommand(on)}} D'$ and $D' \xrightarrow{\text{sendLightCommand(off)}} D$. Given a decoflow W which includes these invocations, the transition rule for the user's possible action, and that the initial state of the lamp is "off", we can determine what possible states the lamp can be in at any point in the execution of W. Suppose, at a given point in time, W has partially executed, and so far, has not invoke any services on the lamp, then the lamp must be in the "off" state at this point, since the user cannot switch it on.

The above example is intentionally simple. More generally, analysis can be done to determine if a given decoflow will cause undesirable effects on the device ecology as a whole, i.e. to verify *safety properties* of the decoflow with respect to a device ecology, or whether the decoflow will result in intended effects, i.e. to ensure *liveness properties*. Liveness properties include the property of a decoflow completing, but

also that it completes with intended effects. The user can pose "what-if" questions to the DecoFlow System situated between the user and the device ecology to examine properties of decoflows before executing them.

6 Conclusions and Further Issues

We have shown by example how BPEL4WS can be used to model device ecology workflows where devices are modelled as a collection of Web services. Note that these means that any other Internet based Web service can be integrated into the same workflow with these devices and existing Web service technology can be applied to interact with these devices. BPEL4WS is a practical language but has not been given a formal semantics. Hence, we have also adapted the DySCo algebraic workflow model for device ecology workflows or decoflows. Lastly, we have outlined how to analyse the effect of a decoflow on a collection of devices' observable states. We note that state space explosion can potentially occur and will require further experimentation to determine what size of device ecologies and device complexity can be tractably tackled. The result of this work is a basis for a workflow based programming model for device ecologies. Another contribution of this work is the proposal of a technique to analyze (e.g., by simulation) or to prove properties of decoflows, where a property of a decoflow is about an effect that a decoflow can have on a device or a collection of devices. We are currently investigating a prototype implementation over UPnP. A number of outstanding issues need to be addressed:

1. *When should a decoflow be triggered and stopped?* A decoflow may start a process involving a number of devices and have long lasting effects. Hence, there should not only be facilities to terminate a decoflow but perhaps also to compensate for a cancelled decoflow.
2. *How can the user perceive the progress of the workflow, and intervene freely?* This is as much a Human-Computer Interaction issue as it is a distributed computing one. Even non-technically minded users should be able to flexibly intervene, e.g., to terminate or alter, a decoflow.
3. *How can the decoflow deal with faults?* This is related to the problem of defining and executing compensations as stated above.
4. *How can we integrate event reporting with decoflows?* A spontaneous state change might result in the emission of an event message which should be handled by the DecoFlow Engine.
5. We have also not considered preconditions for tasks in a decoflow, in order to deal with situations where the task to perform is redundant.

Some devices might also exhibit more autonomous, proactive, and intelligent behaviour – we would like to see if such devices can be adequately modelled with spontaneous state changes. Also, we plan to identify further properties of decoflows and employ Petri net analysis techniques to analyze properties of decoflows as often used for analyzing workflows and multiagent interactions.

We have so far considered only a centralized engine for executing decoflows. Multiagent distributed workflows using a decentralized peer-to-peer model provides greater flexibility and facilitates on-the-fly just-in-time ad hoc workflows directly between devices without the mediation of a heavy-weight central engine. This would

be another avenue of investigation. Lastly, recent work have considered Semantic Gadgets [11], using the Semantic Web framework to describe the semantics of devices, to discover new devices and to dynamically compose device coalitions. It would be interesting to consider semantics based extensions to dynamically compose decoflows, or less ambitious is, given an abstract description of a decoflow, dynamically bind the tasks mentioned in the decoflow to devices' Web services, thereby reducing the burden of composing a detailed decoflow prior to runtime, and providing flexibility by allowing alternative devices or resources to be selected and integrated into the decoflow at runtime.

References

[1] AHAM. Connected Home Appliances – Object Modelling, AHAM CHA-1-2002, 2002.
[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. Business Process Execution Language for Web Services (version 1.1), May 2003.
[3] J. Bentham. TCP/IP Lean: Web Servers for Embedded Systems (2nd Edition), 2002, CMP Books.
[4] S. Berger. Intelligent Appliances Give Automation A New Home, 2002. Available at http://www.aarp.org/computers-features/Articles/a2002-07-10-computers_features_appliances.html
[5] E. Bergman. Information Appliances and Beyond, 2000, Morgan Kaufmann Publishers.
[6] C. Carabelea and O.Boissier. Multi-agent Platforms for Smart Devices: Dream or Reality? In Proceedings of the Smart Objects Conference (SOC'03), Grenoble, May 2003. Available at http://turing.cs.pub.ro/~cosminc/papers/grenoble03.pdf.
[7] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1, March 2001. Available at http://www.w3.org/TR/wsdl.html.
[8] Y. Durand, S.P.J.-M. Vincent, C. Marchand, F.-G. Ottogalli, V. Olive, S. Martin, B. Dumant, and S. Chambon. SIDRAH: A Software Infrastructure for a Resilient Community of Wireless Devices. In Proceedings of the Smart Objects Conference (SOC'03), Grenoble, May 2003.
[9] J.H. Jahnke, M. D'Entremont, and J. Stier. Facilitating the Programming of the Smart Home, IEEE Wireless Communications, pp. 70–76, December 2002.
[10] O. Kasten and M. Langheinrich. First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network. In Proceedings of the Workshop on Ubiquitous Computing and Communications at PACT 2001, October 2001.
[11] O. Lassila and M. Adler. Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web, in D. Fensel et al. (eds.), Spinning the Semantic Web, pp. 363–376, 2003, MIT Press.
[12] Microsoft Corporation. Understanding UPnP™: A White Paper. Available at http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc
[13] D. Marples, P. Kriens. The Open Services Gateway Initiative: An Introductory Overview, IEEE Communications Magazine, pp. 2–6, December 2001.
[14] K. Matsuura, T. Hara, A. Watanabe, and T. Nakajima. A New Architecture for Home Computing, In Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Ssytems (WSTFES'03), pp. 71–74, Japan, May 2003.
[15] T. Nakajima. Pervasive Servers: A Framework for Creating a Society of Appliances. In Proceedings of the 1AD: First International Conference on Appliance Design, pp. 57-63, May 2003.
[16] D. Norman. The Invisible Computer, 1999, MIT Press.

[17] G. Piccinelli, A. Finkelstein, and S.L. Williams. Service-Oriented Workflows: the DySCo Framework. In Proceedings of the Euromicro Conference, Antalya, Turkey, 2003. *(to appear)* Available at http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/euromicro2003.pdf
[18] F. Ramparany, O.Boissier, and H. Brouchoud. Cooperating Autonomous Smart Devices. In Proceedings of the Smart Objects Conference (SOC'03), Grenoble, May 2003.
[19] U. Saif, D. Gordon, and D.J. Greaves. Internet Access to a Home Area Network, IEEE Internet Computing, pp. 54–63, January-February, 2001.
[20] M. Viroli and A. Omicini. A Specification Language for Agents Observable Behaviour, in H.R. Arabnia and Y. Mun (eds.), International Conference on Artificial Intelligence (IC-AI'02), volume I, pp. 321–327, Las Vegas, NV, USA, 24–27 July 2002. CSREA Press.
[21] J. Waldo. The Jini Architecture for Network-Centric Computing, Communications of the ACM, pp. 76–82, July 1999.
[22] S. Weerawarana and F. Curbera. Business Process with BPEL4WS: Understanding BPEL4WS, Part 1, August 2002. Available at http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/.