Formal Mirror Models: an Approach to Just-in-Time Reasoning for Device Ecologies

Seng W. Loke,[1] Sucha Smanchat,[2] Sea Ling,[3] Maria Indrawan[3]
[1]*La Trobe University, Department of Computer Science and Computer Engineering, Australia*
[2]*Faculty of Information Technology, King Mongkut's Institute of Technology North Bangkok, Thailand*
[3]*Monash University, Faculty of Information Technology, Australia*
s.loke@latrobe.edu.au, ssmanchat@acm.org, Chris.Ling@infotech.monash.edu.au, Maria.Indrawan@infotech.monash.edu.au

Abstract

Pervasive computing calls for new ways of thinking about software and new ways of applying software development techniques. In pervasive computing environments, users will need to interact with collections of devices surrounding them (such as the "system" of smart devices in a user's home), which we metaphorically term device ecologies. A user can interact with these devices with embedded software and hardware, either directly via the device's own interface, or automated through scripts executed by a central coordinator. For the latter, we employ a workflow abstraction for the collective operation of devices, which we call decoflow. While executable formal models are typically used at specification time, we maintain and use such models, calling them mirror models, to record the on-going states of devices, their relationships, and the effects that such devices have on the environment. Users effectively update a mirror model of the devices s/he interacts with. The model can be used to predict the effects of decoflows just before execution. Generalizing from mirror models for device ecologies, we propose the novel paradigm of continually updated mirror models for on-going tracking and reasoning about pervasive computing systems which cohabitates with the user.

1. Introduction

Living spaces contain an increasing number of electronic appliances and devices of different form factors. Everyday objects might also be embedded with computers and sensors. These appliances can interact with each other, as well as with their environment. 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* 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 communication infrastructures that range from Internet-scale to very short range wireless networks [1]. Elsewhere, collections of devices have been termed *device ensembles* [5].

For example, consider a bedroom environment. When the alarm clock rings, the light will be automatically switched on and the heater automatically adjusts the temperature settings.

However, one would like to reason with the effects on the environment which the operation of these devices may have, such as whether a "comfortable" environment will result for the user. For example, a device ecology can be designed to monitor and control the brightness and temperature level in the house to satisfy user's comfort specifications. The automation of household activities requires the clear modeling of the interactions among the devices and their environment. Since it is important in the device ecology to model the sequence of the devices' interaction, we investigate a workflow based model as an appropriate choice in modeling the interactions [1]. A mechanism to control the device operations and the interactions among devices is called a device ecology workflow, or *decoflow* for short. Such a workflow can be executed by a workflow engine (which we call *DecoFlow Engine*) which receives a specification of the workflow from the user and then executes it by issuing commands to devices.

The user can interact with a collection of devices as mediated by the DecoFlow Engine in four stages, as outlined in Figure 1: (1) specify the workflow, (2) do a what-if analysis by simulating execution of the workflow, (3) after the analysis, either (3.1) or (3.2) is carried out, depending on whether the user wishes to modify the workflow or to execute it, and (4) the workflow is executed, which may result in feedback to the user (4.1) or the workflow might be interrupted by the user or due to errors (4.2).

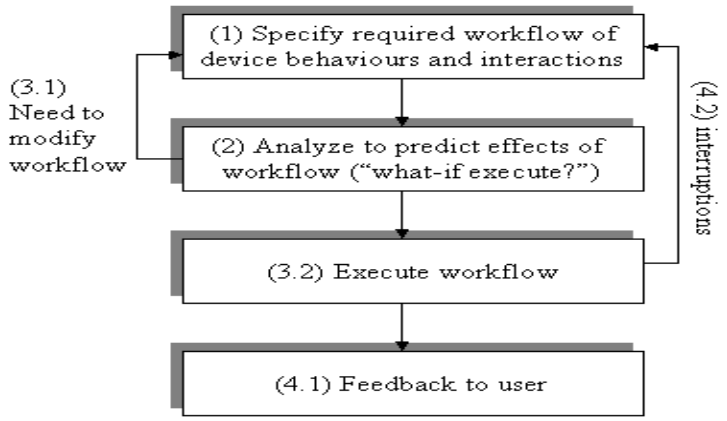


Fig. 1. Interaction model of user with DecoFlow Engine

Alternatively, the devices mentioned above can be operated directly by the user (e.g., the user switches off the alarm clock).

There needs to be some means to (i) keep track of the states of the devices (i.e., observable states) and (ii) predict the effects of a decoflow before execution (e.g., for safety reasons, user requirements, or to estimate the resource consumption of a decoflow). We propose the use of mirror models for this purpose. Figure 2 illustrates the devices and their mirror models. The device models are updated when the user operates directly on the device or as a result of a decoflow execution which changes the states of devices. The operation of the devices in a device ecology might have a direct effect on the user's environment. The environmental effect, in particular within the context of creating a "comfortable" user environment, can be characterized using a set of variables, such as brightness level, noise level, and temperature level.

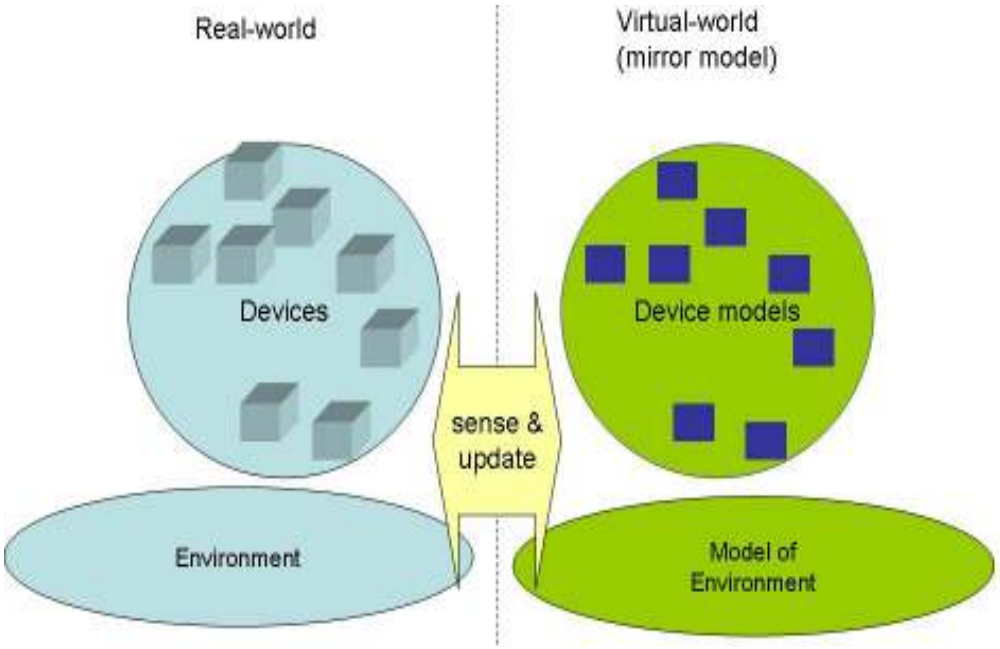


Fig. 2. Mirror models for devices in the real world.

We use *executable formal models* as the mirror model in order to conveniently reason with the mirror models and use the mirror models in order to predict the effects of decoflows. Based on the mirror model, one could simulate the execution of a decoflow before actually executing the decoflow to see what environmental effects a decoflow is likely to yield, or the simulation can happen concurrently with the execution of the decoflow but with several steps ahead at each point in time.

For our executable formal models, we require a formalism with the following properties:

a. modular: a full specification being composable from disparate specifications, enabling our model to be modifiable, extensible and scalable as new devices are added or removed;
b. has well-defined formal properties for representing state and state transitions, and concurreny, with state reachability analysis,
c. well-known and comes with tools for simulation (so that we can leverage on existing tools), and
d. easy to understand semantics and can be depicted visually.

A number of formalisms can satisfy the above properties, including process-calculi, and Petri Nets.

While any formal model with the above properties can be employed, we illustrate our approach using a modular Petri Net model of decoflow, device behaviours and environmental effects, such models being intuitive, graphical (visual), formal yet executable, and compositional (the reader is referred to [2,3] for more information on Petri nets). Petri nets are also more general than finite state machines, enabling concurrency to be modeled if needed.[1] In the rest of this paper, we validate our approach by

[1] We note that the metaphor of workflow is useful here since the same workflow can be viewed using different formalisms or abstractions suited for different purposes: the user can specify a decoflow using a graphical tool (with diagrams as Figure 4) or English like scripts such as the following:
(in parallel turn on lights and turn on heater) then show news on tv then raise curtain

i). demonstrating how the modular Petri net formalism enables different aspects of devices to be captured independently at first, and later linked, and
ii). providing a tool to enable simulation-based analysis of decoflow behaviours, prior to their actual execution.

2. Background

2.1. Architectural Overview

The analysis of workflows is done by a component of the DecoFlow Engine as depicted in Figure 3. The user composes workflows and requires them to be analyzed before execution. Overall execution is managed by the Decoflow Manager which initiates conversations with one or more devices at a time, such conversations being managed by the Device Conversations Manager. The rest of this paper focuses on the decoflow analysis component, whose outcome is a validated decoflow. We also note that environmental sensors are attached to the DecoFlow Engine which provides actual initial conditions for decoflow analysis.

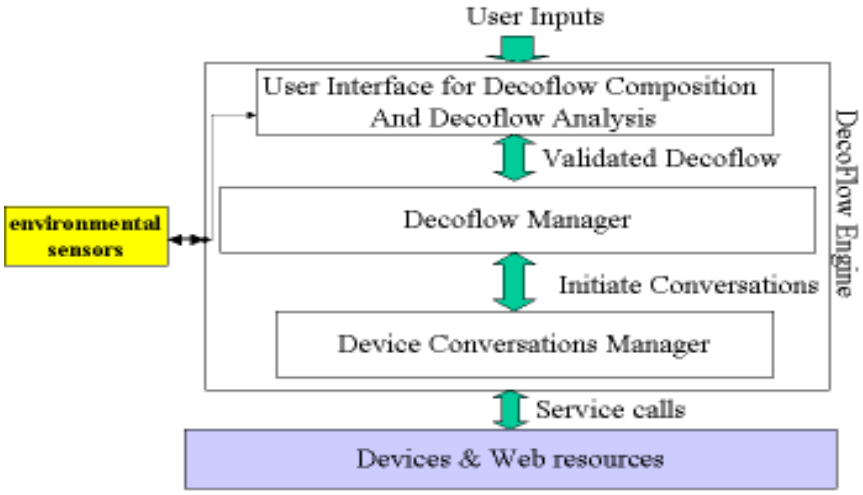


Fig. 3. Components of the DecoFlow Engine

2.2. Decoflow Representation and Example

A decoflow describes the interactions and tasks to be performed on devices, from the perspective of a central coordinating engine. Consider a decoflow implementing a wakeup routine which is adapted from [1]. The scenario comprises six concurrent tasks: opening window, turning on the TV and adjusting the volume, raising curtain, turning on heater and increasing the temperature setting, and switching on the bedroom light and bathroom light (see figure 5 (box diagram view)). In Figure 5, once the wakeup notice is received, the six tasks are executed concurrently. After all tasks are completed the wakeup routine is considered completed. This scenario involves five types of device: window, TV, curtain, heater, and light. There are six instances of devices: two instances of light (bedroom light and bathroom light), and one instance for each of the other device types. There are some

and then such a script or graph is translated into a BPEL XML document and into a Petri Net for analysis. We do not describe in detail the English like language or the graphical BPEL tools in this paper.

assumptions made to this scenario to clarify the environmental effect of the devices. The tasks "Open Window", "Turn on TV", "Raise Curtain", "Switch on Bedroom Light", and "Switch on Bathroom Light" are assumed to affect the brightness level. The task "Increase Volume" is assumed to affect the noise level and the task "Turn On Heater" and "Increase Temperature Setting" are assumed to affect the temperature level.

In our work [1], we model such an example in terms of workflow or business process. At the user level, we use a high-level, English-like language to represent workflow and at the low level, we use BPEL4WS [10]. The specification can then be executed by a BPEL like engine. Our modelling of devices as a collection of Web services implies issuing commands to devices is invoking calls on Web services. We model the observable and controllable aspects of devices as Web services as done in [11]. Such device modeling is not inconsistent with emerging standard models for appliances such as the AHAM Appliance Models,[2] where each appliance (such as clothes washer, refrigerator, over, room air conditioner, etc) is modelled as a collection of objects categorized according to subsystems. We note that there will be aspects of the device which are not exposed as Web services. Such an approach also enables uniform modeling of devices and Internet resources in the same workflow (e.g., a decoflow can not only involve tasks among devices (or invocations of Web services on devices) but also invocations of Web services over the greater Internet to check the weather report or book a hotel room). We can then write a decoflow which downloads the weather report and show it on the television, for instance, or make a purchase for eggs over the Internet (with appropriate authorization) after asking the fridge and discovering no eggs left. BPEL4WS is expressive, but is not by itself amenable to formal analysis. Petri net semantics has been given to BPEL4WS [12], which provides a means to translate a BPEL4WS specification to a Petri net for analysis using Petri net tools [2].

3. A Mirror Model for Device Ecologies: an Example Using Petri Nets

The decoflow, the devices involved in the decoflow and their environmental effects can be modeled as Petri Nets. Three Petri Nets are shown for comfort level analysis. The DecoFlow Net is the Petri Net that represents the decoflow. The Device Net is the Petri Net that models observable behaviors of the devices in a decoflow [8]. In the Device Ecology project, we use RENEW as our modeling tool for Petri Net. RENEW [6,9] is developed based on one type of Petri Net called Reference Net [7] which enables composition of a set of separate Petri Nets via a downlink and uplink synchronization mechanism. In RENEW, the synchronous communication is achieved by using the idea of *system-net* and *object-net* from the Reference Net. In a Reference Net, a system-net represents the main workflow and the object-net represents sub-workflows. For the system-net and object-net to communicate, two communication channels can be used, *downlink* and *uplink*. Downlink is the inscription at the transition where there is the need to reference to other net instance i.e. system-net or object-net. Uplink is the inscription at the transition which serves the call from the downlink. An example is figure 4.

[2] AHAM. Connected Home Appliances – Object Modelling, AHAM CHA-1-2002, 2002.

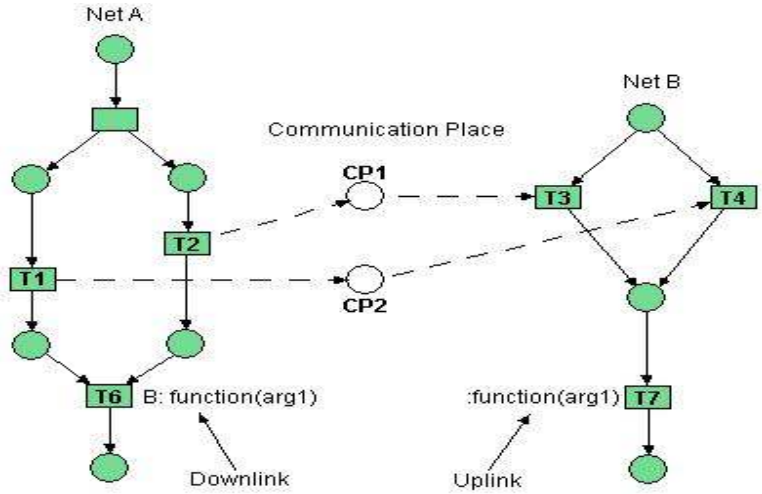


Fig. 4. Example of Synchronous Communication in RENEW

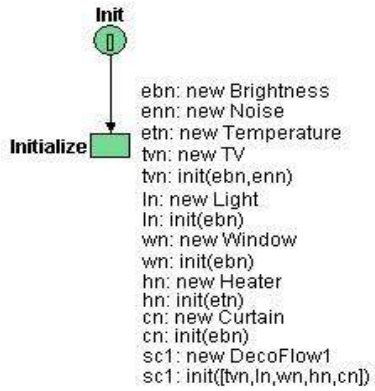
The Effect Net is the Petri Net that models each environment variable that characterizes the environmental effect of the devices. The requirement of the Petri Nets modeling for comfort level analysis in the Device Ecology is that the DecoFlow Net, the Device Nets, and the Effect Nets can be analysed at run-time. Thus, users, or the device ecology designer, can run the Petri Net simulation to predict environmental effects before (or while) a decoflow is (being) executed. While we do not show how in this paper, for an application at hand, the state and the current values of tokens in the DecoFlow Net, the Device Nets, and the Effect Nets can be captured accurately at run-time by interrogating the devices and appropriate sensors. We use synchronous communication among the DecoFlow Net, the Device Nets, and the Effect Nets in our model, as modeled using uplinks and downlinks in RENEW.

In the device ecology simulation, the DecoFlow acts as the system-net. The Device Net and the Effect Net act as the object-nets. The execution of the decoflow as system-net influences the firing of the object-nets representing the devices and the environmental effects. The system-net representing the decoflow is inscribed with the downlinks which call the uplinks inscribed in the object-net representing the devices and the environmental effects. By the use of synchronization links, the environmental effect of the execution of DecoFlow Net can be synchronized through to Effect Net in a single firing. In other words, the firing of the transition in the DecoFlow Net, the firing of transition in Device Net, and the firing of the transition in Effect Net happen synchronously. Therefore, run-time comfort level analysis can be performed correctly given that, in the real world, environmental effects by device operations would happen immediately.

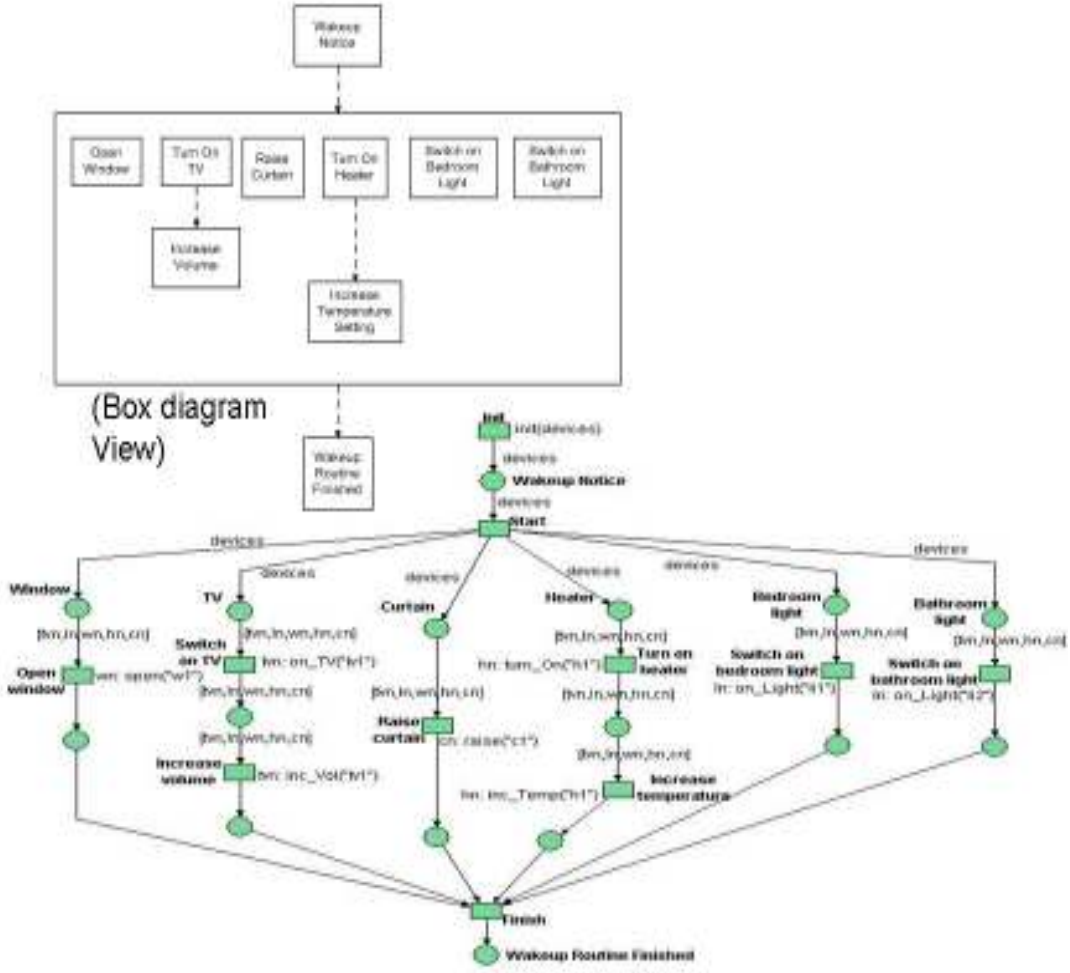
Initialization Petri Net. In RENEW, which is based on the reference nets concept, in order to use or refer to a Petri Net, an instance of that Petri Net must first be created. As for comfort level analysis in Device Ecology, instances of the DecoFlow Net, Device Nets, and Effect Nets must be created and made known to each that needs to send messages to or to communicate with [6, 7]. Figure 5(a) shows the Initialisation Petri Net that is used to create the instances of Petri Net used in the comfort level analysis. From the initialisation in figure 5(a), all the Petri Nets for the comfort level analysis are created. The DecoFlow Net *sc1* has the references of all the Device Net instances with the use of the 5-tupled token "[tvn, ln, wn, hn, cn]" passed as argument of the synchronous channel *init*. Each Device Net instance also has the references of all the Effect Net instances representing the environment variables

which are affected by the device represented by that Device Net by the use of argument of the synchronous channel *init*. Below, we describe how to model the decoflow given as a box diagram in figure 5(b) with a Petri Net.

DecoFlow Net. The DecoFlow Net is the Petri Net that models a decoflow. A task in the decoFlow is modelled as a single transition and the concurrent execution is modelled by AND-Split and AND-Join [4]. The additional transition "Init" at the top in figure 5(b) is created to receive the call from the Initialization Petri Net through the synchronous channel *init* with the *devices* as argument to initialize the DecoFlow Net. The argument *devices* is the 5-tupled token which contains the instances of Device Nets initialized in the Initialization Petri Net. Each transition that represents a task in Device Ecology Workflow is inscribed with the downlink using the Device Net instance in the 5-tuple token to communicate with the Device Net (to change the state of the device). The downlinks at these transitions must match with the uplink of the synchronous channel in the Device Net that they need to communicate with [6, 9].



(a) Initialization Petri Net



(b) DecoFlow Net which models a decoflow (wakeup routine)

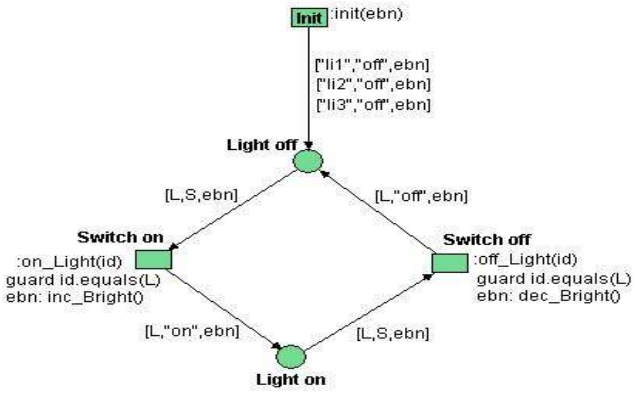
Fig. 5(a) and (b) DecoFlow representation

Device Net. Device Net models the observable behavior of the device involved in the decoflow – we do not model the details of internal device states. Each Device Net represents a device type; a device (or an instance of a device type) is represented as a token in the Device Net. Each place in the Device Net represents the observable state of a device and each transition represents the change of observable state of the device. The initialization transition "Init" at the top of each Device Net in figure 6(a)-(e) is added to receive the initialization message from the Initialization Petri Net. In the Device Net Light, *ebn*, or the instance of the Effect Net Brightness which represents the environment variable that is affected by the device Light, is passed in as the argument from the Initialization Petri Net. The light can be in two states, "Light off" and "Light on", and can be switched on and off. Three 3-tupled tokens indicate that there are three light instances (though only two are used in the DecoFlow Net in figure 5(b)). If the device would affect more than one environment variable, the 3-tupled token can be extended to 4-tupled or more to accommodate the Effect Net instances. The transitions "Switch on" and "Switch off" are inscribed with the downlink using the instance

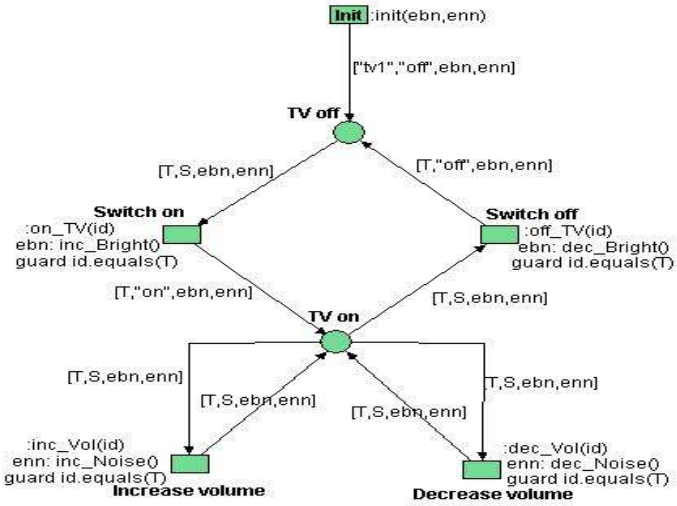
of the Effect Net Brightness to communicate about the effects of the operations of the device. The guard conditions are also inscribed to determine which light instance to be switched on or off.

Effect Net. Effect Nets models (figure 6(f)-(h)) the environment variables. Each Effect Net receives the messages from Device Nets to adjust the value of each environment variable. There are only two transitions "Increase" and "Decrease" in each Effect Net to increase and decrease the level of each effect.

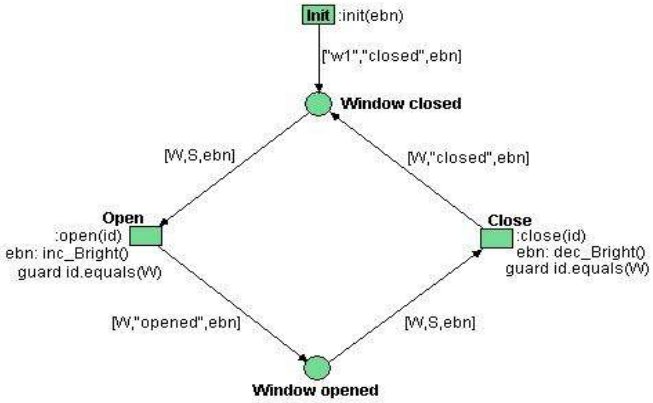
These two transitions are also inscribed with the uplinks of the synchronous channel to receive the messages from the downlink inscribed at the transitions in the Device Nets (see figure 6). The only Petri Net place in Effect Net is given the alias as "Effect Place"; it holds the token used in the Effect Net. The token used in the Effect Nets is simply an integer which is initialised when the Effect Nets are created by the Initialization Petri Net as the initial marking. This integer value is increased or decreased as the Effect Net receives messages from Device Nets.



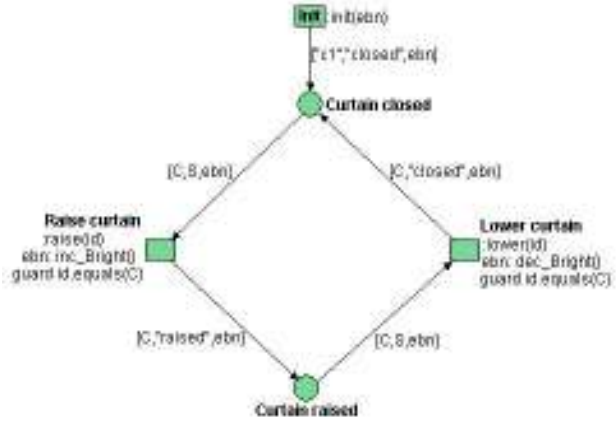
(a) Device Net for Lights



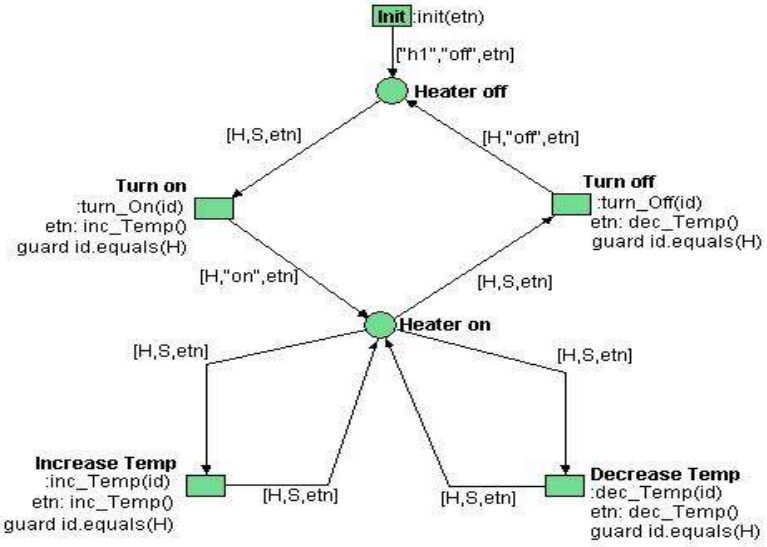
(b) Device Net for TVs



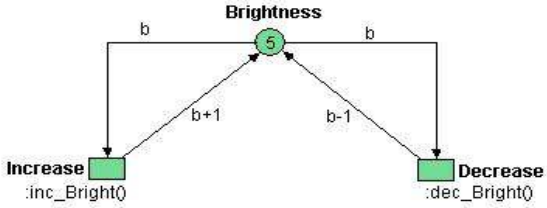
(c) Device Net for Windows



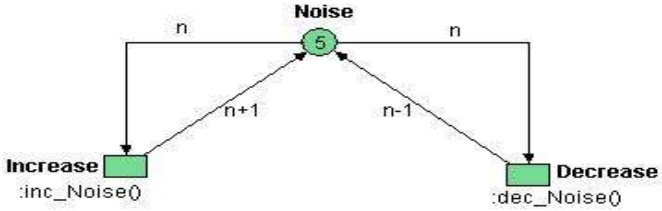
(d) Device Net for Curtains



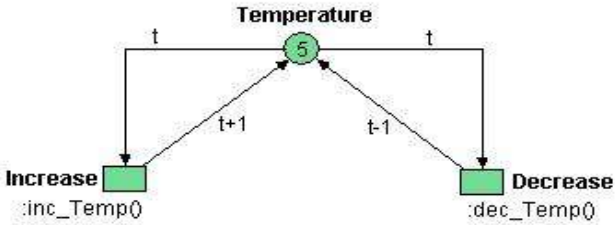
(e) Device Net for Heaters



(f) Effect Net for Brightness



(g) Effect Net for Noise



(h) Effect Net for Temperature

Fig. 6(a)-(e). Device Nets for five device types, **(f)-(h).** The Effect Nets for three environment parameters.

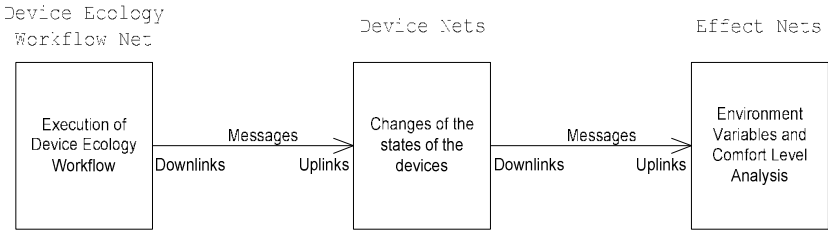
4. Reasoning With Decoflows: an Example with Comfort Levels

With the DecoFlow Net, Device Nets, and Effect Nets, the environmental effect of the execution of the tasks in the decoflow can be represented in the Effect Nets. Effect Nets can receive messages to increase or decrease the environmental effects from the Device Ecology Workflow Net indirectly via the Device Nets by the mechanism of the downlinks and uplinks of the synchronous channels. Since the comfort level can be characterised as a set of environmental effects and each is in turn represented as the Effect Net, the comfort level analysis can be performed on these Effect Nets.

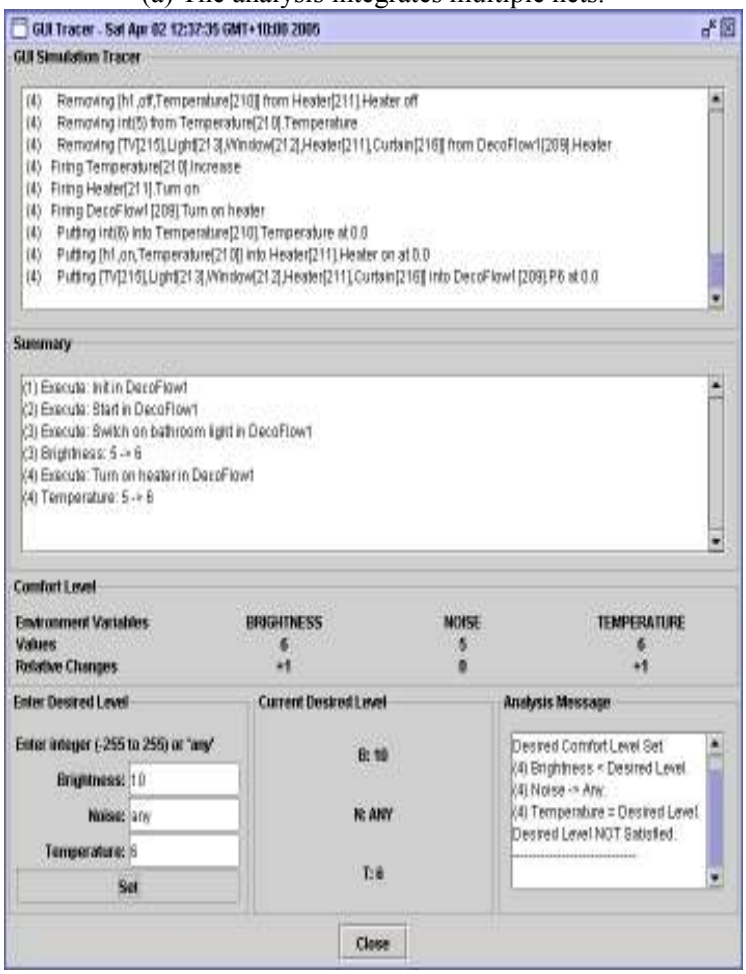
Since the marking at the Effect Place (the specific Petri Net place that is marked with the integer representing the value of the environmental effect) is the integer that holds the value of the environmental effect, the comfort level analysis can be performed by applying the reachability concept of Petri Nets on the Effect Nets [2]. Reachability of a Petri Net indicates whether a certain state of Petri Net is reachable in firings. The state of a Petri Net is represented by the position of tokens (or marking) present in the Petri Nets. State *B* is reachable from state *A* if there is a sequence of firing that moves the tokens whose positions represent state *A* to the new positions that represent state *B*. For the Effect Nets, the reachability should take into account the value of the token as well; the state of Effect Net is represented by the value of the integer token at the Effect Place [2]. The current comfort level from (partial) execution of a decoflow so far (and also, the desired comfort level) is characterised as a set of integers, each representing the value of an environment variable. The desired comfort level is satisfied when the value of the token in the Effect Place of every

Effect Net is equal to the value of the set of integers that characterize the desired comfort level. Schematically, the comfort level analysis is done as in Figure 7(a).

Figure 7(b) shows the interface to the simulator build on top of RENEW. Trace messages from RENEW are analysed to detect the changes in the value of the token in the Effect Places of the Effect Nets. In figure 7(a), the trace messages are analysed and the current values of the environment variables are initialized (e.g., 6 for Brightness, 5 for Noise, and 6 for Temperature) – which can be obtained via sensors or interrogating devices.



(a) The analysis integrates multiple nets.



(b) Interface to the Petri Net decoflow simulation

Fig. 7. Analysis of decoflows

The summary text area displays the summary of the execution of the Decoflow Net. The "Comfort Level" part displays the *current* value of each environment variable and the relative changes compared to the initial values. The three text fields at the bottom left part are used for specifying the desired comfort level (in terms of values for brightness, noise and temperature) to be compared with the comfort levels resulting from the decoflow execution as seen under the "Comfort Level" part. The value of each environment variable can be set between -255 and 255 (inclusive). The summary of the comfort level analysis is the comparison of the value of the environment variable representing current comfort level and the desired comfort level. If both set of values are equal then the message will show that the desired comfort level is satisfied, otherwise the message will show that the desired comfort level is not satisfied as in figure 7(b).

5. Related Work

There has been much work on 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[3] and Jini[4] provide infrastructure for devices to be inter-connected, find each other, and utilize each other's capabilities.

Getting devices connected via Web services is the aim of the proposed Device Profile for Web Services (DPWS)[5] which can be viewed as the next major version of UPnP (UPnP v2) with closer alignment to and taking advantage of standardized Web Service protocols. The various prototypes of DPWS led to embedded devices hosting Web services. Hence, the idea of being able to "talk to" and control devices by invoking their Web services is not new and we can expect to see more advanced versions of these in time to come and more of such devices. The gap that this paper fills is, given a collection of such devices, each exposing Web services, how one can utilize a mirror model for just-before-execution understanding (via simulation) of a decoflow.

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 [1] 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. VRDK [14] is a graphical tool for programming home automation tasks. It also uses a notion of processes for device control. With underlying workflow notions, VRDK scripts can be programmed to accomplish tasks involving devices. The user drags and drops devices to compose event-driven processes (e.g., wait for a movement or certain time before turning the lights on). A code generator transform scripts to executable code. VRDK also employs a location model to position devices. Our work employs a well-known formal model to mirror executable decoflow programs in order to reason about the decoflow before execution or during execution.

In [15], UPnP itself was formalized using an abstract state machine model in AsmL. The aim of the formalization was to resolve ambiguities, incompleteness, loose ends, or inconsistencies. Key entities used are agent, communicator, control point and device, and an application is modelled as combinations of these key entities. UPnP communication protocols are formalized in the paper. Our work formalizes device ecology workflows and device

[3] http://www.upnp.org/
[4] http://www.sun.com/software/jini/
[5] http://specs.xmlsoap.org/ws/2005/05/devprof/devicesprofile.pdf

models via Petri Nets to enable practical reasoning with the whole system, before or after runtime.

Work by Bellotti and Edwards [16] has noted the importance of technology being intelligible and our work, while not specifically aiming to tackle all those issues (but to provide decoflow analysis prior to execution), does help in the user understanding and making sense of the effects of a decoflow (which they have composed themselves or pre-programmed) before it is even executed. Indeed, making sense of script written to automate a home or certain activities within a home is important. With our formal approach, one could even answer questions regarding whether a particular decoflow can accomplish a given effect (i.e., the query is translated in our model into a reachability analysis - is this particular state reachable?). Our approach helps in reasoning about a decoflow before execution or midway during its execution (to understand the effects of the remaining part of the decoflow to be executed).

Instead of scripts which are centrally executed, one can also specify possible device-to-device interactions separately as recipes [17]. Such pairwise or n-wise recipes for how two devices can interact are useful though they might not provide a bird's eye-view that a centrally controlled approach can provide.

6. Conclusion

We have proposed an approach to reason with decoflow effects on a formal basis, articulating mirror models as run-time counterparts (in the sense of [13]) of device ecologies and their immediate environment parameters, and showed that this provides the advantage of reasoning with device behaviours. A user may compose a decoflow, which is then translated into a decoflow net and reasoned about in combination with current (up-to-date) device and effect nets, at any time just when they are about to be executed.

Our notion of decoflow was inspired in part by the notion of "automation scripts" of the late Michael Dertouzos.[6] We are also aware of other programmatic solutions to device collective behaviour such as Jini,[7] UPnP[8] and AutoHan.[9] Users might compose decoflows using some high level (e.g., graphical or speech based) tools to manage the devices at home, in public spaces, in offices, in museums, in gardens, or in factories, and mirror models for the device ecologies maintained in these different places enable reasoning any time users need to compose and run decoflows that meet requirements, not only comfort levels but perhaps also safety levels, noise levels, energy consumption levels, water consumption levels, and engagement levels (e.g., entertainment applications).

We illustrated our ideas using modular Petri Nets as mirror models of device ecologies. The Petri Net models can be fed with environmental parameters obtained via real sensors and/or interrogating devices for their observable states (as noted in Figure 3). The Petri net models are scalable as increasing the number of devices results in more tokens and not more nets (e.g., the Device Net for Lights is for all three lights, rather than for only one), and can be linked with newly introduced nets for new devices or effects. Nuances of concurrent actions and effects are also captured.

[6] In the book, "The Unfinished Revolution: Human-Centered Computers and What They Can Do For Us", Harper Collins, 2001.
[7] http://www.sun.com/software/jini/
[8] http://www.upnp.org/
[9] http://www.cl.cam.ac.uk/Research/SRG/HAN/AutoHAN/

In summary, we see that

(i) an explicit and formal model of the on-going devices' collective behaviour is useful, and this can be done via a combined collection of executable formal models of devices' observable behaviours - these models can be updated when devices operate either due to the user's action directly on the device or as a result of a decoflow execution, and new devices added will be modeled by adding device nets to the existing model, and

(ii) predicting and validating (via simulation) end-user programs before execution is useful for the user to understand the effects of his/her programs (even if this cannot be done perfectly, due to uncertainties at run-time – the DecoFlow Engine still needs to be able to deal with run-time exceptions), and sensing can provide initial conditions for prediction, as real-time inputs into the formal executable models maintained throughout the lifetime of the devices.

Our approach has not incorporated probabilities measures of certain events occurring that might affect decoflow execution (e.g., the user's direct intervention with a device might occur with a given probability), though the theory on Petri Nets supports this (e.g., stochastic Petri Nets). One reason is due to the engineering effort to provide probabilities in this setting as well as the arbitrary nature in which probabilities in this case might be estimated. Such issues can be explored for future work.

Also, the GUI for the system we have presented is a prototype and more work can be done to provide a more end-user friendly view of results – the end-user does not even need to know that the underlying model is based on Petri Nets.

Finally, we note that our approach of using formal models to accompany home automation scripts is novel, and general in that it is

(i) applicable to automation in other environments such factories and vehicles, and

(ii) not restricted to the use of Petri nets alone or decoflows, but amenable to other formal models and home automation in other scripting languages, as long as a mapping exists between the constructs of the scripting language and the formal model.

Acknowledgements. We acknowledge the Australian Research Council for support of the Device Ecology Project under the ARC Discovery Grant scheme.

7. References

[1] S. W. Loke, "Service-Oriented Device Ecology Workflows," Proceedings of the International Conference on Service-Oriented Computing, pp. 559 - 574, 2003, Springer-Verlag.

[2] T. Murata, "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, vol. 77, pp. 541-580, 1989.

[3] High-level Petri Nets - Concepts, Definitions, and Graphical Notation, Final Draft International Standard ISO/IEC 15909, Version 4.7.1, 2000, http://www.petrinets.info/standard.php.

[4] W. M. P. v. d. Aalst, "The Application of Petri Nets to Workflow Management," The Journal of Circuits, Systems and Computers, vol. 8, pp. 21-66, 1998.

[5] B.N. Schilit and U. Sengupta. Device Ensembles. *IEEE Computer*, pp. 56- 64, December, 2004.

[6] O. Kummer, F. Wienberg, and M. Duvigneau, Renew, Version 2.0.1, 2004, www.renew.de.

[7] W. M. P. v. d. Aalst, D. Moldt, R. Valk, and F. Wienberg, "Enacting Interorganizational Workflows using Nets in Nets," Proceedings of the 1999 Workflow Management Conference, pp. 117-136, Germany, November, 1999.

[8] S. W. Loke and S. Ling, "Analyzing Observable Behaviours of Device Ecology Workflows," Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS2004), pp. 78-83, Porto, Portugal, 2004.

[9] O. Kummer, F. Wienberg, and M. Duvigneau, Renew - User Guide, Release 2.0.1, 2004, http://www.informatik.uni-hamburg.de/TGI/renew/renew.pdf.
[10] Microsoft, IBM, Siebel, BEA, and SAP (2003). Business Process Execution Language for Web Services Version *1.1*. http://www-106.ibm.com/developerworks/library/ws-bpel/.
[11] 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 Systems (WSTFES'03), pp. 71-74, Japan, May 2003.
[12] K. Schmidt and C. Stahl (2004). "A Petri net Semantic for BPEL4WS – Validation and Application". Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN), pp. 1 – 6, Paderborn, Germany, 2004.
[13] K. Römer, F. Mattern, T. Dübendorfer, and J. Senn Infrastructure for Virtual Counterparts of Real World Objects. Technical Report ETHZ, 2002. http://mics.epfl.ch/getDoc.php?docid=253&docnum=1
[14] M. Knoll, T. Weis, A. Ulbrich, and A. Brandle. Scripting Your Home. In Proceedings of LoCA, Lecture Notes in Computer Science 3987, pp. 274-288, 2006.
[15] U. Glasser, Y. Gurevich, N. Veanes. High-Level Executable Specification of the Universal Plug and Play Architecture. In Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSSS), vol 9., 2002.
[16] V. Bellotti and K. Edwards, "Intelligibility and accountability: Human Considerations in Context-Aware Systems," *Human-Computer Interaction*, vol. 16, no. 2-4, pp. 193-212, 2001.
[17] M. Newman, T. Smith, and W.N. Schilit. Recipes for Digital Living. IEEE Computer, vol 39, no. 2, pp. 104-106, 2006.

Authors

Dr. Seng Wai Loke is a Senior Lecturer at the Department of Computer Science and Computer Engineering at La Trobe University. He is the author of "Context-Aware Pervasive Systems: Architectures for a New Breed of Applications" (December 2006). His research interests are in pervasive computing including smart homes and device ecologies.

Sucha Smanchat is a lecturer at the Faculty of Information Technology, King Mongkut's Institute of Technology North Bangkok, Thailand. He completed his Master of Information Technology at Monash University, Australia.

Dr. Chris Ling is a lecturer at the Caulfield School of Information Technology in Monash University. He completed his PhD in Petri Nets, and works with Dr. Loke on the Device Ecology project.

Dr. Maria Indrawan is the Associate Head of School at the Caulfield School of Information Technology in Monash University. Her research interests include device ecologies and information retrieval.