ADAPTIVE WORKFLOWS FOR SMART DEVICES
A Concrete Approach Towards Device Failures

Seng Loke
Department of Computer Science and Computer Engineering, La Trobe University, Australia

Sea Ling, Maria Indrawan, Suryani Kurniati
Faculty of Information Technology, Monash University, Australia

Keywords: Pervasive systems, smart devices, adaptive workflow, device failures.

Abstract: Smart devices in an environment (e.g., home, factory, military settings, in-vehicle, office, etc) can be programmed and coordinated by a workflow in advance to achieve a user's goal. No matter how advanced or smart the devices are, devices can fail during workflow execution. In this paper, we describe an approach to remedy such situations. We apply the existing concept of adaptive workflow management to a collection of devices, called a *device ecology*. Information about the devices are kept in a device hierarchy so that a suitable substitute device that can perform a similar task can be retrieved to replace a failed device in order to ensure the workflow can continue execution. Similarity is defined based on a device hierarchy in an ontology language. A prototype has been implemented as proof of concept.

1 INTRODUCTION

Technology developments in the past decade have resulted in "less for more", intelligent devices. Computers are built small enough and smart enough to be embedded into devices and human daily appliances, creating intelligent devices. Smart device or intelligent device is "any type of equipment, instrument, or machine that has its own computing capability" (SearchExchange.com, 2006).

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, 2000), SIDRAH (Durand et al., 2003) and Jini (Sun 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).

Previous work (Loke, 2003; Loke et al., 2005) provides a high-level device aggregation framework, adopting service-oriented computing and business process management in the form of device ecologies. A device ecology is defined as an environment consisting of collections of devices 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 (Loke, 2003). To program these devices, we make individual web service calls to these devices which are coordinated by a workflow in BPEL4WS. A central coordinator called device ecology workflow (*decoflow*) engine exists as the "brain" of the group of devices in this environment. Therefore, device ecology is characterised by the idea of service-oriented computing and workflow in a high-level device aggregation framework, which perceives devices as orchestrated web services taking part in a service-oriented workflow. Such coordinated execution of devices and Internet services (including appliances, and embedded processors and computers) have widepsread uses in the home, factories, military environments, office environments, within cars, etc, wherever there is a need for a collection of devices (hardware and soft-

Loke S., Ling S., Indrawan M. and Kurniati S. (2007).
ADAPTIVE WORKFLOWS FOR SMART DEVICES - A Concrete Approach Towards Device Failures.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - SAIC*, pages 191-197
DOI: 10.5220/0002368901910197

ware) and Internet resources to work together to fulfill a common user programmed task (e.g., in response to the task of cooking a recent dish talked about in a popular TV program such as the IronChef, suitable Web services are consulted to download the recipe and the fridge is consulted to determine if suitable ingredients are available before the appropriate kitchen utensils are prepared and instructions displayed on suitable devices for the user to take action and to take the user through the whole cooking process, including setting up suitable timings and alarms on appliances, in a coordinated workflow manner).

No matter how advanced or smart the devices are, dynamic changes can occur during workflow execution. A device is prone to faults and failures. Several realistic examples will be: accidental loss of electricity, electrical spike or surge, thrown exceptions in task failures, and loss of network connection. When one of these changes occur, decoflow execution will be affected. It will be halted or will totally fail, becoming inexecutable. Although changes like these can be resolved by manual effort, it is inefficient and cumbersome. As quoted from (Aalst, 2001), no matter how temporary or permanent changes are, workflow has to be able to support it by typically executing a "more or less idealised version of the preferred process", as an effort to achieve the original objective. In Workflow Management Systems, such effort is captured in the concept of the adaptive workflow.

Overall the aims of this project are two-fold:

1. applying existing concepts of adaptive workflow management to a collection of devices in a device ecology, in particular to remedy situations of device failure; and
2. providing a basis, for future research by investigating how device failures could be handled.

In this paper, we will report on the resulting concepts and algorithms for adaptive device ecology workflows that can still execute effectively despite device failures occurring during execution. Section 2 contains an overview of device ecology and the impact of task dependencies towards providing adequate workflow flexibility. Section 3 explains the hierarchy of devices used in our solution. This is followed by a description of the remedy algorithm in section 4. Section 5 describes our prototype implementation as a proof of concept and section 6 is the conclusion.

2 WORKFLOW AND TASK DEPENDENCIES

Workflows exist in areas from business management, information system to computing in general. A workflow consists of an arrangement of activities and tasks to achieve a business objective. It is a business process comprising operational logics for the coordination of resources, independent units that have the capability to perform specific tasks (Piccinelli et al., 2004). The term resources include entities that act autonomously and upon request, depending on its coordination. Resources can take the form of devices, applications and, most significantly, web services.

In general, there are two types of workflow coordination logic, orchestrated and choreographed. In an orchestrated environment, a single local entity is in charge of maintaining the state of the process, and to request the resources to perform tasks (Piccinelli et al., 2004). In a choreographed workflow, no central entity organises order and request services. Interaction occurs between entities dynamically. It usually occurs in web service based workflows as in a BPEL4WS specified workflow.

BPEL4WS (Microsoft et al., 2003) was developed to become the specification language to support the implementation of business processes with web services. It was designed to support business process specific issues like, the potential execution order of operations from a collection of web services, the data shared between these web services, which partners are involved, joint exception handling for collections of web services and issues involving how multiple services and organisations participate (Leymann et al., 2002). These capabilities are reflected in the BPEL4WS specification using XML-based tags like `<invoke>`, `<receive>`, `<reply>`, etc. BPEL4WS has to adopt, comply and able to make use of Web service specific standards like Universal Discovery Description and Integration (UDDI), Web Service Description Language (WSDL) and Simple Object Access Protocol (SOAP).

Our work attempts to develop a framework to allow users to program devices in terms of workflow using a suitable programming language undergirded by a formal model. At the highest level, we have developed *Eco*, an English-like interaction language, consisting of a collection of abbreviated commands to make end-user workflow programming simpler. This is mapped down to the lower-language BPEL4WS which is executed by the workflow engine that controls the devices. Figure 1 depicts the conceptual architecture of our framework.

Two decoflow scenarios in a smart home environ-

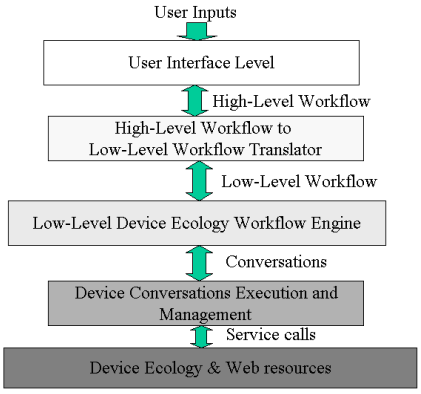

Figure 1: Multilayered Conceptual Architecture for Device Ecology Workflow Engine.

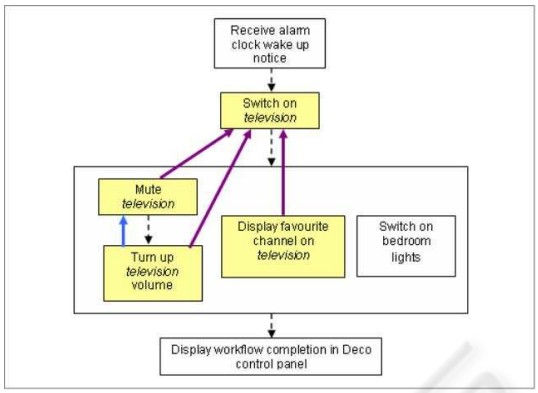

Figure 2: Wake Up Scenario.

ment are provided in Figure 2 and Figure 3. Following the dashed arrows which depict the flow of workflow execution and ignoring the dark arrows, we describe the scenarios as follows:

Wake-Up Scenario: Decoflow execution starts upon receiving a bed alarm clock notice, signifying the beginning of the wake up decoflow. A sequence of activities starts right after. The next activity initiated is switching on television. After television has successfully changed its state, the following activities take place concurrently: mute television; display favourite channel on television and, switch on bedroom lights. Following the muting of television, the volume is then increased. Once the television is set, and two other activities have successfully completed, a message of decoflow completion is displayed in a control panel.

Morning Kitchen Scenario: This scenario takes place in the kitchen area. It starts by receiving a user response on the alarm clock (setting the alarm to the off state). This is then followed by three concurrent activities: activating coffee maker; activating kitchen hall sensors and, retrieving 'to do' list from the PDA. After the kitchen hall sensors state turns to active, kitchen lights are switched on. After retrieving 'to do' list on the PDA, the list is displayed on the refrigerator monitor, to remind user of his/her duties for the day.

3 TASK DEPENDENCIES IN DECOFLOW

The dependency between tasks has an impact towards providing adequate workflow flexibility. Currently, most task dependency studies focus on task primitives and operations, particularly in the field of database management. Generalised views of dependency models has been reviewed to provide a background.

Kim (Kim, 2003) categorised workflow dependency into three different models: procedure-driven model; dependency-driven model and condition-driven model, while in (Ray et al., 2004; Zhu et al., 2005), task dependencies are classified based on task primitives (task's execution states: begin, abort, commit), task operations and task input/outputs perspectives, which resulted in three different types of dependencies: control-flow dependencies, data-flow dependencies and read-write dependency. Control-flow dependencies and read-write dependencies focus on task internal mechanism, particularly dependencies from one task's state to another in the same workflow. Hence, these dependencies do not apply to task dependencies for devices in our work. However, data-flow dependencies is applicable.

A task $T1$ produces an output x that will be used as the input to another task, say $T2$. Hence, the ability to execute $T2$ depends on the success of executing $T1$. If $T1$ and $T2$ are tasks performed by the same device, their relationship is considered as same device task dependency. The dark arrows in Figure 2 depict *same device task dependencies*, e.g. turning up TV volume depends on the successful execution of switching on the TV first.

The dark arrow in Figure 3 shows a *different device task dependency* relationship. PDA and refrigerator monitor are two different devices but they share the same data - the "to do" list.

To provide adaptiveness towards device failures in the above scenarios, the decoflow engine which executes the workflow needs to be aware of the device description, the types of failure and the available recovery techniques. Intuitively, when a task or a device is going to fail, a substitute or replacement needs to be found. A listing of the device and its description is maintained through a device hierarchy. Figure 4 shows how devices are modelled as object-oriented

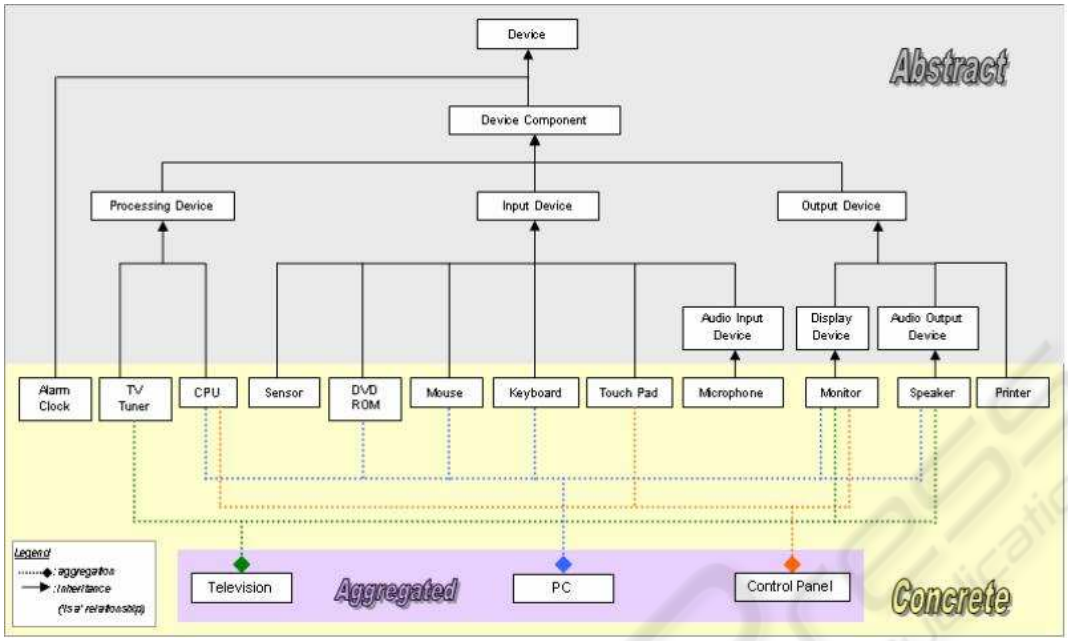

Figure 4: Device Hierarchy.

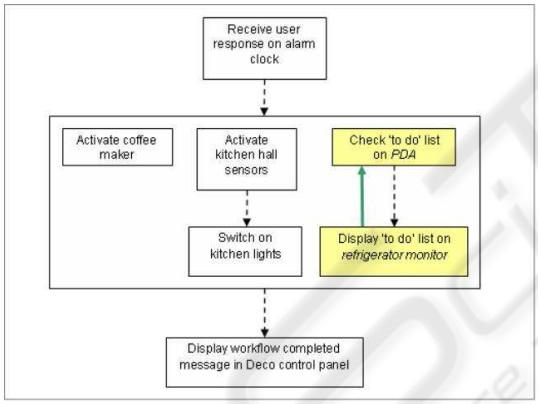

Figure 3: Morning Kitchen Scenario.

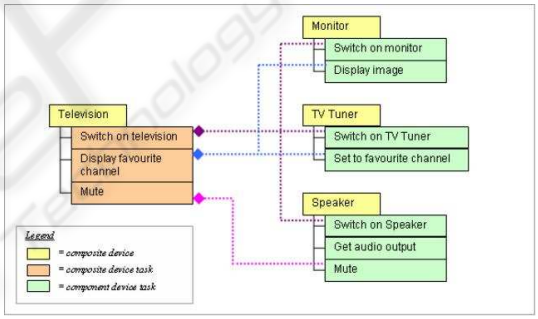

Figure 5: Composite Tasks of a Composite Device.

abstract and concrete classes in such a hierarchy.

A device can also be seen as a collection of atomic devices where appropriate. Although inspired by UPnP MediaRenderer (Intel Research & Development, 2003) which viewed device aggregation at low-level framework, the hierarchy also represents a high-level view of devices with the focus on tasks and subtasks. For example, in the hierarchy, television consists of the components: a monitor, a TV tuner and a pair of speakers. The composite task `switch on TV` means switching on the three component devices, i.e., a composition of three subtasks (child tasks) `switch on monitor`, `switch on TV tuner` and `switch on speakers`. The composite task and its component (atomic) tasks are shown in the task hierarchy in Figure 5.

4 REMEDYING DEVICE-ORIENTED TASK FAILURES

Previous works on task failures in workflows (Aalst et al., 1999; Aalst and Jablonski, 2000; Ellis and Keddara, 2000; Elder and Liebhart, 1996) have provided ideas on how to resolve device-oriented task failures in decoflow. While some works (Aalst et al., 1999; Aalst and Jablonski, 2000; Ellis and Keddara, 2000) generalise failures as process change including workflow changes as a result of handling modifications, other works (Elder and Liebhart, 1996), upon detecting a task failure, propose a recovery mechanism which includes a combination of *forward execution* and *backward recovery*. Forward execution is a deci-

sion to ignore failed task and proceed to the next task. This can only be carried out if failed task has no vital relationship (i.e., no dependencies) with other tasks, especially for consistency measures (Elder and Liebhart, 1996). Backward recovery involves effort to roll back the whole workflow, undoing successful terminated tasks, which in the end resulted in unsuccessful execution of the workflow as a whole. Hence, we adopt a combination of forward execution and backward recovery. When a potential failure is detected, this means to proceed to the following tasks where no consistency measures need to be satisfied and undo tasks that have vital relationships.

By applying the above mechanism to device ecology, the steps for forward recovery by the decoflow engine are:

1. To eliminate previous successfully terminated tasks that utilises the same failed device;
2. To substitute the failed device with a working device, available for the job (not utilised by other decoflow executing in the same time frame) and capable of executing the task;
3. To make necessary changes towards the following tasks that use the same device and other tasks that has consistency measures (dependency) with the changed and rolled back tasks.
4. If no substitute device found, skip the task and move on.

We categorised device failures into two types: total device failure and partial device failure. Depending on the type of failed device, single device can only result in total failure whilst composite device may result in either partial or total failure.

Total Device Failure. This type of failure exists when all device components of a single device instance are unable to carry out any function. For the case of atomic device (with no components), it can only exhibit this type of failure. Ultimately these are device instances whose classes reside on the top level of the concrete part of device hierarchy in Figure 4. They include devices such as lights, sensor, printer, monitor and speaker. In a composite device situation, total device failure can only occur when all device component instances refuse to function.

Partial Device Failure. Partial device failure occurs when a device is able to execute only some of its listed operations while the rest are not executable due to some fault. An example will be failed television's tuner while television's monitor and speakers are still functioning. This type of failure can only affect composite devices.

By considering the type of device failure, the number of tasks affected in decoflow, the dependencies between them and the number of component devices affected (if any), we have developed an algorithm to recover from failures. Essentially, before executing the algorithm, a device failure status needs to be determined. This is done by a `checkDeviceStatus` procedure, initiated just before the task is executed, (in sequence), or before a series of tasks is executed concurrently, (in flows).

The purpose of the algorithm is to look for a substitute device or a combination of devices for a failed device in a failed task, if any. In summary, the algorithm consists of the following steps:

1. Identify device failure type (total or partial).
2. Identify task failure (single or composite).
3. Get all available substitute device.
4. Obtain the best substitute devices. The criteria of selection is based on the number of tasks its component devices can cover.
5. Replace executable task, either single or composite.
6. Replace prior tasks that use the same failed device.
7. Replace following tasks that use the same failed device.

5 DECOFLOW ENGINE IMPLEMENTATION

In our previous work, we have developed an engine which accepts a decoflow specification in BPEL4WS and simulates the execution of the specification. An execution manager reads the necessary tasks to form a decoflow execution script, based on the task's script template whilst a graph manager forms the graphical representation of the process elements derived from the decoflow analysis by the engine.

We extend the decoflow engine to facilitate adaptiveness towards device failures. Hence, to implement the algorithm described in the previous section, the engine needs to incorporate the following functionalities: failure detection, task manipulation, decoflow manipulation and device recording and tasks configuration. The extension is shown in Figure 6. It accepts *Deco Hierarchy* as an additional input to keep track of devices that exist in the ecology as well as the variety of tasks that the devices can perform. It consists of the device hierarchy and the task hierarchy, mentioned previously.

The engine itself consists of:

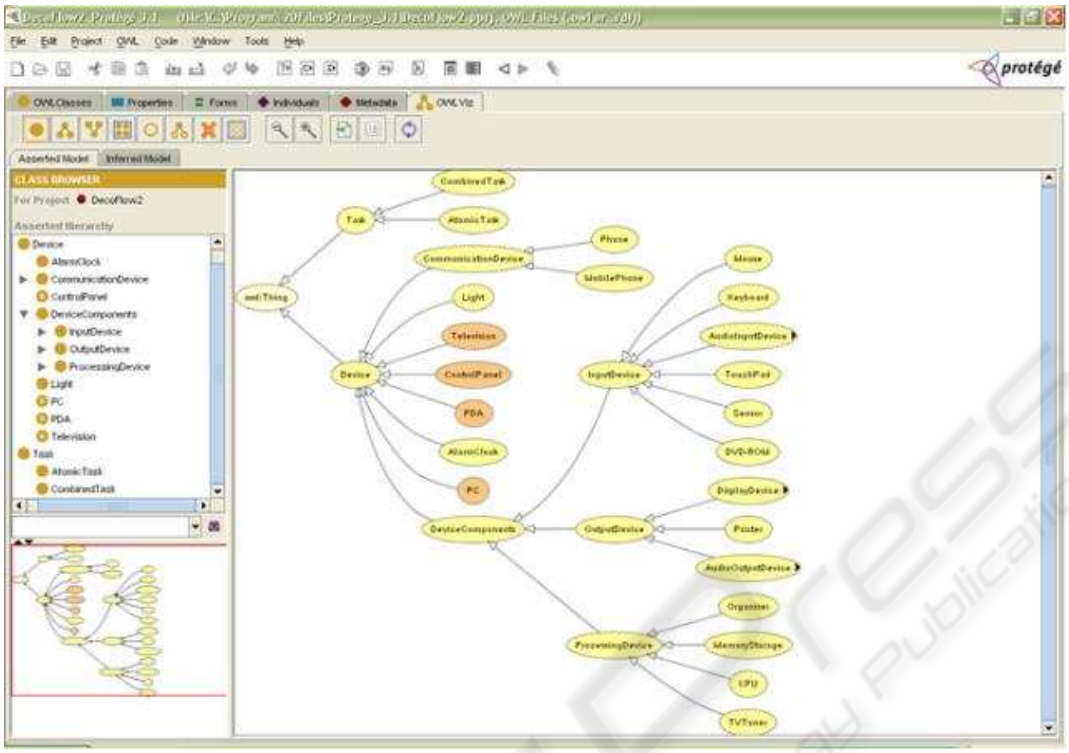

Figure 7: Deco Hierarchy Snapshot.

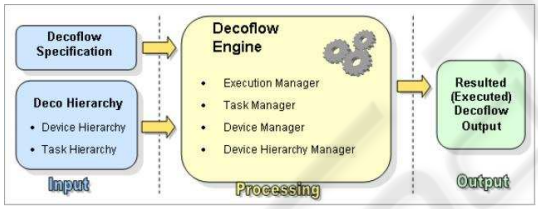

Figure 6: Decoflow Engine.

- **Execution Manager**. The manager checks tasks for their abilities to execute. If a device is not capable of carrying out the designated task, the recovery algorithm is called.
- **Task Manager**. This entity deals with failure firsthand. Once a task is detected as incapable of execution, the task needs to be manipulated in accordance with the best possible resolution suggested by recovery algorithm. Specifically, the task manager alters the decoflow executable script, not the original decoflow specification in BPEL4WS (Loke et al., 2005).
- **Device Manager**. Its main function is to find a substitute device, using the device hierarchy through the device hierarchy manager. The device manager queries and searches the device hierarchy, for failed device substitutes. Similar to the task manager, the device manager is designed to support proposed failure recovery.
- **Device Hierarchy Manager**. This serves as the interface between the device hierarchy and the decoflow engine. Its main function is to query the hierarchy. It relies heavily on the structural design of the device hierarchy in carrying out its activities.

As proof of concept, a prototype for the engine was developed in Java with the deco hierarchy defined by the Web Ontology Language OWL. Access to the hierarchy from the engine is provided by Jena (Seaborne, 2004) utilising RDQL (RDF Data Query Language) for the query statements. A snapshot of the deco hierarchy using Protégé and *OWL Viz* plug-in is shown in Figure 7. The prototype has been tested using various workflow scenarios on device failures.

6 CONCLUSION

Pervasive systems have seen intelligent devices communicating and interacting with one another. Our work attempts to develop a framework to program these devices in terms of workflow which is called a device ecology workflow. Problems set in when one

or more devices fail. We have described an approach by which we can remedy such situations by searching for substitute devices before the workflow is being executed.

While other work exists in modelling, developing and configuring smart home systems (Norbisrath et al., 2006), our work focuses on the workflow execution and control of these devices and providing a concrete solution to remedy device failures by adopting existing concepts in the literature on adaptive workflows.

REFERENCES

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

Aalst, W., Basten, T., Verbeek, H., Verkoulen, P., and Voorhoeve, M. (1999). Adaptive workflow: On the interplay between flexibility and support. In *Proc. 1st Int'l conference on Enterprise Information Systems, Vol 2*, pages 353–60.

Aalst, W. and Jablonski, S. (2000). Dealing with workflow change: Identification of issues and solutions. *Int'l Journal of Computer Systems Science and Engineering*, 15(5):267–76.

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

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

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.

Elder, J. and Liebhart, W. (1996). Workflow recovery. In *Proc. 1st IFCIS Int'l Conference on Cooperative Information Systems*, pages 124–34.

Ellis, C. A. and Keddara, K. (2000). A workflow change is a workflow. In *Business Process Management: Models, Techniques and Empirical Studies*, pages 201–17.

Intel Research & Development (2003). Designing a UPnP AV MediaRenderer. Available at http://cache-www.intel.com/cd/00/00/21/87/218761_218761.pdf, accessed on 30 Nov 2006.

Kim, K. (2003). Workflow dependency analysis and its implications on distributed workflow systems. In *Proceedings of the 17th Intenational Conference on Advanced Information Networking and Applications (AINA'03)*.

Leymann, F., , and Roller, D. (2002). *Business Processes in a Web Services World, IBM developerWorks*.

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., Ling, S., Butler, G., and Gillick, B. (2005). Levels of abstraction in programming device ecology workflows. In *Proceedings 7th International Conference on Enterprise Information Sytems (ICEIS2005)*, pages 137–44, Miami, 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/.

Norbisrath, U., Armac, I., Retkowitz, D., and Salumaa, P. (2006). Modeling eHome systems. In *Proc. 4th Int'l Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006)*.

Piccinelli, G., Finkelstein, A., and Williams, S. (2004). Service-oriented workflow: The dysco framework. In *Proceedings of the 29th EUROMICRO conference (New Waves in System Architecture)*, pages 291–7.

Ray, I., Xin, T., and Zhu, Y. (2004). Ensuring task dependencies during workflow recovery. In *Proc. 15th Int'l Conference on Database and Expert Systems (DEXA 2004)*, pages 24–33.

Seaborne, A. (2004). Jena tutorial: A programmer's introduction to rdql. Available at http://jena.sourceforge.net/tutorial/RDQL/, accessed on 14th November 2006.

SearchExchange.com (2006). Definition of intelligent devices. Available at Whatis.com - http://whatis.techtarget.com/definition/0,,sid9_gci812508,00.html,accessed on 10 Oct 2006.

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

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

Zhu, Y., Xin, T., and Ray, I. (2005). Recovering from malicious attacks in workflow systems. In *Proc. 16th Int'l Conference on Database and Expert Systems (DEXA 2005)*.