Towards a Solution for
On-the-fly Device Failures in
Context-Aware Device Ecology Workflows

Thesis submitted to Caulfield School of Information Technology
in partial fulfilment of the Degree of
Master of Information Technology
(Minor Thesis)

By
Chii Chang
(Student ID: 19603274)

Supervisor:
Dr. Chris Ling
Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signed: …………………………………………………………………………………
Date: ……………………………
Acknowledgements

I would like to thank my supervisor Dr. Chris Ling. I really appreciate his guidance and advices on my research. He has been very patient and helpful in supervising and guiding me throughout this research, even he had a busy schedule.

I would also like to thank to my parents for supporting me during my study life. :-P
Abstract

Device Ecology, a digital ecosystem describes the interoperability of the user, the devices and the environment. In the previous work, the Device Ecology systems are implemented based on service-oriented workflow systems called - Decoflow. As a service-oriented workflow system which mainly performs tasks by executing devices through the services, the device failure detection and recovery before workflow execution was interesting research area which has been covered in previous research. On-the-fly device failures, which occur during workflow execution, is an issue that was not covered in the previous work. Our research aims to resolve the on-the-fly device failures as the primary approach.

Device Ecology aims to be a context-aware system which is capable of interacting with the user based on user’s activity, available resources and the environment. This research takes into consideration of context-awareness, aiming to use context to enhance the recovery algorithm to discover a best substitute based on the influence of contexts.

The research proposes a framework towards resolving on-the-fly device failures in Device Ecology workflow. The framework provides the resolution of the run-time device failure detection, the recovery algorithm and the rule-based substitution approach influenced by context. A prototype is implemented and evaluated in this research as proof-of-concept.
# Table of Contents

Chapter 1. Introduction ........................................................................................................................................... 11  
1.1. Preamble ......................................................................................................................................................... 11  
1.2. Research Significance ...................................................................................................................................... 12  
1.3. Research Objectives ......................................................................................................................................... 14  
1.4. Research Scope ................................................................................................................................................ 14  
1.5. Research Limitation .......................................................................................................................................... 15  
1.6. Thesis Structure ................................................................................................................................................. 15  

Chapter 2. On-the-fly Device Failures in Context-Aware Device Ecology ................................................................. 17  
2.1. Introduction ....................................................................................................................................................... 17  
2.2. Service Oriented Device Ecology ..................................................................................................................... 17  
2.2.1. Service Oriented Computing ......................................................................................................................... 17  
2.2.2. Workflow ....................................................................................................................................................... 18  
2.2.3. BPEL4WS ..................................................................................................................................................... 20  
2.2.4. Device Ecology ............................................................................................................................................ 20  
2.3. Workflow Systems and Device Failures ......................................................................................................... 21  
2.3.1. Adaptive Workflow ...................................................................................................................................... 21  
2.3.2. On-the-fly Device Failures Example ........................................................................................................... 22  
2.3.3. On-the-fly Exception Handling .................................................................................................................. 23  
2.4. Applying Context-Awareness to Workflow Systems ...................................................................................... 25  
2.4.1. Definition of Contexts ............................................................................................................................... 25  
2.4.2. Context-Aware Systems .............................................................................................................................. 25  
2.4.3. Run-time Context-Aware Decision Making ............................................................................................... 32  
2.5. Summary ......................................................................................................................................................... 36  

3.1. Introduction ....................................................................................................................................................... 37  
3.2. Run-time Failure Detection and Recovery .................................................................................................. 37  
3.2.1. Analysis of Run-time Device Failures in Device Ecology Workflow ...................................................... 37  
3.2.2. Workflow Adaptation for Devices .............................................................................................................. 39  
3.3. Context Ontology .............................................................................................................................................. 41  
3.3.1. Context Categories .................................................................................................................................... 42  
3.3.2. The Goal of Workflow Tasks and Contexts Utility ................................................................................... 44  
3.4. A Framework for Context-Aware Adaptive Workflow .................................................................................. 46  
3.4.1. Framework Requirements ........................................................................................................................ 46
3.4.2. Architecture Overview
3.4.3. Context-Aware Components
  3.4.3.1. Overview
  3.4.3.2. Context Sensors
  3.4.3.3. Context Manager
  3.4.3.4. Substitution Manager
3.5. Behaviour of the Engine
  3.5.1. Detecting Device Failures at Run-time
  3.5.2. Context-Aware Workflow Adaptation
3.6. Summary

Chapter 4. Implementation Description
  4.1. Introduction
  4.2. Overview of the Prototype Simulation
    4.2.1. Required Components
    4.2.2. Scenarios
  4.3. Implementation
    4.3.1. Technologies used
    4.3.2. System Design
    4.3.3. Context Manager and Corresponding Components
      4.3.3.1. Overview
      4.3.3.2. Widgets and Aggregator
      4.3.3.3. Interpreter Functions
    4.3.4. Substitution Manager
      4.3.4.1. Overview
      4.3.4.2. Substitution Function
    4.3.5. Adaptive Device Manager
    4.3.6. Workflow
    4.3.7. Database
  4.4. Summary

Chapter 5. Evaluation
  5.1. Overview
  5.2. Overall Testing of Scenario 1
  5.3. Overall Testing of Scenario 2
  5.4. The Individual Component Testing of Substitution Manager
  5.5. Summary and Discussion

Chapter 6. Conclusion
  6.1. Research Summary and Contributions
  6.2. Research Directions
References ................................................................................................................. 113
Appendix A ................................................................................................................ 118
Figures

Figure 1.1  Wakeup scenario workflow ................................................................. 13
Figure 2.1  The wake up scenario workflow ....................................................... 19
Figure 2.2  Workflow terminologies [14] .............................................................. 19
Figure 2.3  The wake up scenario workflow ....................................................... 23
Figure 2.4  Sample configuration of context components [32] ............................. 26
Figure 2.5  Components of context-aware system [33] ....................................... 27
Figure 2.6  Architecture [34] ............................................................................. 28
Figure 2.7  Interaction with context-aware systems [35] ...................................... 29
Figure 2.8  CE’s context architecture [36] ............................................................ 30
Figure 2.9  Syntax of a rule file [36] .................................................................... 30
Figure 2.10 Context and ontology use in Web services composition [37] ............. 31
Figure 2.11 Context collection and dissemination [37] ...................................... 32
Figure 2.12 Original workflow ........................................................................... 33
Figure 2.13 The workflow with decision engine ............................................... 33
Figure 2.14 Example calculation utility for in a low temperature, save energy profile during a windy day [9] .......................................................... 34
Figure 2.16 A demand process algorithm to adopt changes of contexts to a DITree [38] ............................................................................................................... 35
Figure 3.1  Workflow description and task execution ......................................... 38
Figure 3.2  Task manager detects device failures at run-time .......................... 39
Figure 3.4  Run-time device failure detection .................................................... 41
Figure 3.5  Context categories .......................................................................... 43
Figure 3.6  Relationship between goal and activities ........................................ 44
Figure 3.7  Context influences goal and activities ............................................. 45
Figure 3.8  Utility function ................................................................................ 45
Figure 3.9  The architecture .............................................................................. 47
Figure 3.10 Context manager example .............................................................. 50
Figure 3.12 Decoflow with the run-time device failure detection processes ..... 53
Figure 3.13 Recovery algorithm part 1 ............................................................... 54
Figure 3.14 Recovery algorithm part 2 ............................................................... 55
Figure 4.1  Architecture of prototype simulation ............................................... 59
Figure 4.2  Scenario 1 workflow ....................................................................... 60
Figure 4.3  Scenario 2 workflow ....................................................................... 60
Figure 4.4  Components relationship ................................................................. 62
Figure 4.5  Components in package: edu.deco.contextmanager ....................... 64
## Tables

Table 2.1  Information of context components [37] ................................................. 32
Table 4.1  Activity table ......................................................................................... 80
Table 4.2  Context table ......................................................................................... 80
Table 4.3  Context category table .......................................................................... 81
Table 4.4  Context element table .......................................................................... 81
Table 4.5  Proposed valid context element values .................................................. 82
Table 4.6  Device table ............................................................................................. 83
Table 4.7  Device element table .............................................................................. 83
Table 4.8  Device element type table ...................................................................... 83
Table 4.9  Goal table .................................................................................................. 83
Table 4.10  Interpreter rule table ............................................................................. 84
Table 4.11  Interpreter rule element table ............................................................... 84
Table 4.12  Sample high-level context element data ............................................... 84
Table 4.13  Sample low-level context element data 1 ............................................ 85
Table 4.14  Sample low-level context element data 2 ............................................ 85
Table 4.15  Sample Interpreter Rule Element data 1 ............................................. 85
Table 4.16  Sample Interpreter Rule Element data 2 ............................................. 85
Table 4.17  Sample Interpreter Rule data ................................................................. 85
Table 4.18  Profile table ............................................................................................ 86
Table 4.19  Rule table ................................................................................................ 86
Table 4.20  Sample rule table .................................................................................. 87
Table 4.21  Daytime element table ......................................................................... 87
Table 4.22  Location element table ........................................................................ 87
Table 4.23  User action element table .................................................................... 88
Table 4.24  Weather element table ......................................................................... 88
Table 5.1  Goal data used in scenario 1 ................................................................. 91
Table 5.2  Activity data used in scenario 1 .............................................................. 91
Table 5.3  Rule data used in scenario 1 ................................................................. 92
Table 5.4  Goal data used in scenario 2 ................................................................. 95
Table 5.5  Activity data used in scenario 2 .............................................................. 95
Table 5.6  Rule data used in scenario 2 ................................................................. 97
Chapter 1.

Introduction

1.1. Preamble

Today, people are surrounded by devices that are capable of communicating, cooperating, and performing tasks for users. Such devices can be in the home environment (television, microwave, bedroom light etc.) and they have been embedded with small processors to increase their capabilities to communicate and interact with each other. Moreover, some devices can be programmed to sense users’ movements and perform different actions or interact with users (e.g. turn on the television and switch to user’s favourite channel when user comes back from work) in a smart home environment.

Previously, numerous significant technologies have been developed to support networked devices to be able to interact and cooperate with each other, in order to achieve the goal of business processes. Popular frameworks such as UPnP [1], SIDRAH [2] and Jini [3] provide infrastructure for devices to be interactive, discover each other and utilise each other’s capabilities. Embedded Web Servers [4] was developed to enhance the mechanism of networked devices to operate as Web services. In other words, devices can be modelled as software components. As these technologies were developed for the purpose of implementing device interoperability, they have been categorised as low-level device aggregation frameworks [5].

Device Ecology [6] is such a framework, which implements the concept of Embedded Web Services at a higher level of abstraction, describes the interaction of user, devices and environment. It has three primary entities: the user, central server and devices. In Device Ecology, devices can be programmed to interact and participate in tasks depending on contexts such as time, environment conditions or user’s responses. The underlying contract behind device ecology is a workflow system – Device Ecology workflow (Decoflow) [6], which is based on Business Process Execution Language for Web Services (BPEL4WS) [7], a language used to specify a collection
of device tasks. The Decoflow is controlled by a central engine, known as Device Ecology workflow engine (Decoflow engine), which is the ‘brain’ of Device Ecology system.

One area of interest is in the area of failure detection and recovery of devices. In previous research, device failures in Decoflow have been categorized. Failure detection and the recovery methods were proposed before run-time [5]. However, one type of failure, which is on-the-fly failure (failure occurring during the task execution at run-time) detection, was not covered in previous research. This project aims to investigate relevant works that will contribute to possible solutions for on-the-fly device failures in Decoflow system.

1.2. Research Significance

Previously, Kurniati [5] has investigated the approach towards resolving device failures in Device Ecology. She analysed device failures based on failure identification principles in workflow management system. Device failures have been categorised into two types – partial device failure and total device failure. The recovery algorithm is implemented to find best substitute device for the failed device and create substitute tasks to replace failed task(s) after the failure is identified. Although Kurniati [5] provides solution to resolve the issue of device failure in Device Ecology, the recovery algorithm is only done before workflow execution. In other words, Decoflow engine can only discover the fault(s) before the workflow execution, and it will not be able to recognise the on-the-fly device failure. Below is the wake-up scenario based on previous researches [8] [9], which describes a wake-up routine. Note that some changes are made to fulfil the need of this research.

The scenario begins with an alarm clock in the morning. Once the workflow engine receives the notification from the alarm clock, it executes the workflow to perform tasks based on the user’s setting. Involved tasks are: (1) Switch on TV; (2) Turn on heater if temperature is below on the specified degree; (3) Switch on bathroom light; (4) Open window; (5) Brighten the bedroom, which can be completed by various choices such as ‘switch on bedroom light’ or ‘raise curtain’ depending on the situation/context. After the execution is completed, workflow engine will generate the result report to the user (on the computer monitor or PDA). Figure 1.1 is the workflow which represents this scenario:
In the above scenario, contexts of the environment, such as brightness and time of day are involved in the decision making of the workflow. We firstly, describe where the on-the-fly failure can occur and how the failure should be detected in the above scenario. Assume the on-the-fly failure occurs in task 3 (switch on bathroom light, as Figure 1.2 shows) due to a sudden occurrence of an electrical surge during task execution. The main device involved in this task is the bathroom light. From the user’s perspective, a possible solution is to switch on another light which is located closest to the bathroom (e.g. the corridor light). Another solution is the user can just walk into bathroom without the need to turn on another light if the brightness of the bedroom is enough to let him/her see the environment (if the
bedroom is next to the bathroom, and the door of bathroom is facing to the door of bedroom, and, the brightness of bedroom light is capable of reaching the bathroom when it is turned on).

We want to detect this failure at run-time and provide a substitution for the engine at run-time. We hope to make the decision for device substitution based on the current environment, situation and available entities, which are considered as ‘context information’. To resolve the bathroom light failure, we can possible add extra processing components to perform similar decision making as human does based on the available contexts.

1.3. Research Objectives

This research aims to:

- Discover a way to detect on-the-fly device failures
- Determine how context-awareness can support the decision making in Decoflow at run-time
- Provide a generic run-time context-aware decision making framework in order to resolve on-the-fly device failures

1.4. Research Scope

This research will analyse relevant contributions include adaptive workflow, run-time workflow adaptations and context-aware systems before narrowing down to on-the-fly device failures in Device Ecology workflow.

Base on the investigation of related contributions. A proposed framework will be introduced to resolve the on-the-fly device failure by applying context-awareness in Device Ecology workflow system. A prototype simulation will be developed as proof-of-concept.

In previous research related to Device Ecology, Lee [9] has proposed a run-time decision engine based on context-aware model. The features of Lee’s work are the system is capable of observing task nodes in workflow and performing decision making at runtime, which is applicable to enhance the capability of Decoflow in the area of detecting and recovery failed tasks at run-time. Hence, Lee’s work [9] will be specifically studied in this research and the design of his context-aware model will be
1.5. Research Limitation

Investigate relevant literatures is one of the major parts of this research, resources in this research such as relevant articles and documents will be retrieved from the available academic resources such as Monash university libraries, electronic academic database such as ACM digital library, IEEE Xplore, ProQuest computing etc.¹ Subjects relative to this research but not available from retrievable resources will not be considered.

The simulation in this research is based on the concept of Decoflow system [5] which is mainly developed in BPEL4WS and Java. The notion of this research is applicable to other platform but due to time constraint, this research will not deal with other programming language.

The contribution in this research focuses on the run-time device failure detection and recovery. Hence, the simulation is built to proof the concept of the run-time failure detection and recovery algorithm rather then implementing a completed Device Ecology system. Previous work [5] which deals failure recovery before the workflow execution will not be included in the work of this thesis since the previous work [5] has no functionality to handle events at run-time.

1.6. Thesis Structure

This thesis is divided into five chapters:

- **Chapter 1** introduces the background, significance, objective and the scope of this research.
- **Chapter 2** provides a discussion of related literatures which includes: the background technologies such as service-oriented system, workflow system and Device Ecology; adaptive workflow and on-the-fly failure detection in workflow; context-aware workflow system.
- **Chapter 3** analyse the on-the-fly device failure in Decoflow and introduces our proposed framework to resolve the issue of on-the-fly device failure.
- **Chapter 4** describe the implementation of prototype simulation which bases on the proposed framework resolution and followed by an evaluation in **Chapter 5**.

¹ Databases and electronic resources: http://www.lib.monash.edu.au/databases/
Chapter 6 concludes the research project, revisiting the contribution of this research and the direction of future research.
Chapter 2.

On-the-fly Device Failures in Context-Aware Device Ecology

2.1. Introduction

Embedded Web services enhance the capabilities of devices which can be programmed to interact with each other and the user. Device Ecology is a service-oriented system, which was introduced to realise the interaction of user, devices and environment bases on the workflow technology – BPEL4WS. Previous research has partially resolves the device failure issues in Device Ecology. The on-the-fly device failures were mentioned but have not yet been resolved. In order to resolve the issue of on-the-fly device failure, we have studied relevant contributions of adaptive workflow, and clarified the needs to solve on-the-fly device failures by analysing a workflow scenario. Context-awareness is one of the elements in the initial design of Device Ecology. Contexts (such as physical devices, time or condition) need to be analysed by workflow systems in order to discover the best substitute device to replace the failed device. We have considered existing research in context-aware systems to understand the design of context-aware workflow system and the required context information for our work.

2.2. Service Oriented Device Ecology

2.2.1. Service Oriented Computing

Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications/solutions [10]. Services represent functions that can be requested by simple messages through the standard protocols such as TCP/IP or HTTP. In early distributed computing technologies, RPC, DCOM or CORBA are representatives of service-oriented. In the 2000s, an evolution
of distributed computing has been introduced – the web service.

Web services are service-oriented based components that represent mechanisms/functions which can be requested from clients by sending Simple Object Access Protocol (SOAP) messages which are Extensible Markup Language (XML) formatted document that can be transferring through HTTP protocol. Clients of a web service are not only applications; they can be other web services cooperating together, increase their capabilities and provide more functionality for end-point user applications. Elements in web services such as methods, parameters, return types of methods are described by Web Service Definition Language (WSDL) which is also XML formatted document. Web services can be registered and discovered by Universal Description Discovery and Integration (UDDI). Due to the communication between web services and their clients, based on SOAP messages, cross-platform became one of the benefits of web service.

Popular web service implementations like Microsoft .NET XML Web service [11] Java EE [12] have been widely implemented in industries today. Moreover, web services can be implemented in devices such as PDA, mobile phone or any other devices in different platforms.

2.2.2. Workflow

Workflow was introduced to support manufacturing processes to identify involved resources such as well-defined tasks, roles, rules and procedures. Georgakopoulos et al. [13] define a workflow as a collection of tasks organized to accomplish some business process. Workflow technology is implemented to describe automation of business processes which involved documents, information or task participants based on defined business rules for the achievement or contribution of an overall business goal. The difference between workflow and the sequence diagram, which has been used widely in software development life cycle, is workflow represents the theory and logical operation chain of business processes instead of describing information passed between resources in sequence diagram. Hence, workflow has the feature of technology independence. Figure 2.1 represents an example of workflow which describes the process of a wakeup scenario.
Figure 2.1 The wake up scenario workflow

In the wake up scenario workflow above, we have a series of activities that work in sequence. To achieve the goal, Activity 1 is executed first followed by Activity 2. Activities 3 and 4 represent optional events in which one of them can be selected as a chosen path to reach the final activity (Activity 5) to complete its goal (of working up).

**WfMC**

The Workflow Management Coalition (WfMC) [14] was introduced as an organisation to provide workflow technology development through common terminology and standards in industrial workflow products. WfMC proposes that all workflow management products need to follow the common characteristics and standards to achieve the interoperability among heterogeneous workflow products. Also, it will improve the interaction of workflow applications with various I.T. services such as electronic document management systems or electronic commerce. Figure 2.2 shows the Relationships between basic terminologies.

![Workflow terminologies](image)

**Figure 2.2** Workflow terminologies [14]
2.2.3. BPEL4WS

Business Process Execution Language for Web Service (BPEL4WS) was mainly developed by IBM and Microsoft. It is a combination of WSFL from IBM and XLANG from Microsoft. Gathered with features from WSFL (support for graph oriented processes) and XLANG (structural constructs for processes), BPEL4WS is designed to be a new standard for Web service composition.

BPEL4WS supports two type processes descriptions – executable business processes and non-executable abstract processes. Executable business processes work as common software development frameworks, describes the actual behaviour (such as methods, parameters, types etc.) of participants in business interaction. Abstract processes mainly define the business protocol rules, publicly observable behaviours, and identify protocol-relevant data as message properties by implementing nondeterministic data values to hide private aspects of behaviour [15].

The BPEL4WS process is based on the workflow process mentioned earlier. A process step is called the activity. Each activity may require one or more service invocation depending on the complexity of the activity. BPEL4WS is implemented by using XML-like tags, for example: <invoke> for invoking web service, <receive> for receiving message from the service, <reply> for generating input/output operation etc. It also contains an idea called partner which represents cooperated entity. There are two types of partner: invoked partners – to be invoked as an integral part of algorithm from a service or a process; client partners – to invoke the process.

2.2.4. Device Ecology

“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.” [16]

In a Device Ecology, devices have been modelled as software components, which are coordinated by workflow processes in BPEL4WS. A central engine, the device ecology workflow (Decoflow) engine, exists as the “brain” which manages the workflow process and provides proper decision making.

In Device Ecology, small processors are imbedded in devices, each device is available
to be implemented Web services in order to allow Decoflow engine to communicate with it through Web service standards (WSDL, SOAP message etc.). We can consider Device Ecology as a service oriented system. As the control between the Decoflow engine and the devices is based on a low level programming language BPEL4WS, it is considered not user-friendly enough for the user to input his/her workflow settings in Device Ecology. Therefore, a high level language is developed to fulfil the users’ needs – the Eco Language. It is an English-like interaction language, which allows user to define a workflow [16]. An example of a process defined in the Eco Language is:
Switch on TV; wait for TV; display News on TV.

2.3. Workflow Systems and Device Failures

2.3.1. Adaptive Workflow

The concept of adaptive workflow was introduced to address the limitations of Workflow Management Systems (WfMSs). As the complexity of business processes increased, existing WfMS will need to be modified; extra algorithms, functionalities, rules or requirements may need to be added to fulfil the required task achievement. Researchers recognised that traditional WfMSs were unable to support such flexibilities. Hence, adaptive workflow has been introduced to support changes in workflow processes.

Kammer et al. [17] defined three general goals of an adaptive workflow system.

- **Detecting exceptions**: An adaptive workflow should support the exceptions discovery mechanism. Sources of the exception need to be clearly indicated. Active data stores and event driven architectures should be used to support detecting exceptions.

- **Avoiding exceptions**: Kammer et al. [17] have suggested that implementing open systems, support for incremental adoption, flexible execution approaches, reusable process components and integrated support for communication between participants may all help to avoid exceptions.

- **Handling exceptions**: This goal consists of three sub goals.
  - **Tolerating Minor Deviations**: Minor deviation, also known as “noise” in workflow execution, can either be accommodated or cause consequences. Karmmer et al. [17] have suggested providing flexibility in workflow to support the toleration minor deviations.
• **Handling Changes to the Process Instance**: It involves on-the-fly/runtime changes. Workflow exceptions may occur during runtime execution. They can be caused by service failure, network disconnection, or any other issues. It was described that a dynamic instance model can help to solve this problem. Alternative approach is not to specify activities in a model but only provide multiple available tasks to the user. This technique is called Freeflow, contributed by Dourish et al. [18].

• **Evolution and Optimisation of the Process Model**: Elements of workflows (such as tasks, priorities, responsibilities etc.) change over time, representing evolution of workflow processes. Adding, removing or redefining process activities in workflow may affect the workflow model resulting in an improvement of the previous work model results. This is so called optimisation. A workflow sometimes needs to be changed to apply new situations which have been unanticipated in the original design. It is recommended that the workflow infrastructure should allow such changes to occur before and after workflow deployment.

The second subgoal of adaptive workflow above is related to the on-the-fly/runtime changes for our research. Literatures that involved runtime changes/exception and dynamic changes in workflow system will be discussed in Section 2.3.3. We firstly review the on-the-fly device failure in Section 2.3.2.

### 2.3.2. On-the-fly Device Failures Example

Below is the wake-up scenario based on previous researches [8], [9], which describes a wake-up routine. Note that some changes are made to fulfil the need of this research. The definition of on-the-fly device failure will be made using this scenario.

The scenario begins with an alarm clock in the morning. Once the workflow engine receives the notification from the alarm clock, it executes the workflow to perform tasks based on the user’s setting. Involved tasks are: (1) Brighten the bedroom, which can be completed by various choices such as ‘switch on bedroom light’ or ‘raise curtain’ depending on the situation/context; (2) Turn on heater if temperature is below on the specified degree; (3) Switch on bathroom light; (4) Open window; (5) Switch on TV. After the execution is completed, workflow engine will generate the result report to the user (on the computer monitor or PDA). Figure 2.3 is the workflow which represents this scenario:
In the above scenario, contexts of the environment, such as brightness and time of day are involved in the decision making of the workflow. We firstly describe where the on-the-fly failure can occur and how the failure should be detected in the above scenario. Assume the on-the-fly failure occurs in task 3 (switch on bathroom light, as Figure 2.3 shows) due to the device failed to operate during task execution. Since the failure detection and recovery were done before the actual execution of tasks (refer to previous work [5]), workflow engine will not realise the failure will occurred. The main device involved in task 3 is the bathroom light. From the user’s perspective, a possible solution is to switch on another light which is located closest to the bathroom (e.g. the corridor light). Another solution is the user can just walk into bathroom without the need to turn on another light if the brightness of the bedroom is enough to let him/her see the environment (if the bedroom is next to the bathroom, and the door of bathroom is facing to the door of bedroom, and, the brightness of bedroom light is capable of reaching the bathroom when it is turned on).

This scenario raised two questions:

- How to detect on-the-fly device failures?
- How to recover device failures after detection is made?

### 2.3.3. On-the-fly Exception Handling

Workflow failures detection and recovery is a research area which have been covered by many literatures, such as Eder and Liebhart’s ‘Workflow Recovery’ [19], Aalst and Jablonski’s ‘Dealing with workflow change identification of issues and solutions’ [20], Klein and Dellarocas’s ‘A Knowledge-Based Approach to Handling Exceptions in Workflow Systems’ [21] and many others [22], [23], [24]. These researchers...
provide techniques to resolve the issue of workflow failures. However, these techniques are only dealing failures before the workflow execution, the on-the-fly failure described in Section 2.3.2 could not be resolved.

Kamath and Ramamrithan [25] have introduced the runtime workflow mechanism in their research. The technique called Language for Workflow Specification (LAWS) was developed as a rule based approach which has been used to manage the workflow execution at runtime. In their design, each workflow step has been identified with a name of a node, each node is observed by an agent which is capable of executing the step and return the result to the engine. Note that the agent is not necessary a mobile software agent, but can be a class or a component with specific behaviours. Chen et al. [26] in their research also use the concept of agent to enhance the workflow processes. The difference is the tasks in workflow were handled by agents (agents representing workflow steps) in Chen et al.’s work. The notion of using agent technology to support adaptive workflow can also be found in Buhler et al.’s research [27] which uses multi-agents to enhance workflow system. From the notion of relevant researches, we understand that on-the-fly failures can be detected by agent techniques. However, the problem is how to handle failures effectively and properly.

Kamath and Ramamrithan [25] emphasised the importance of rule specification. They have defined nine specifications in their research. One of the specifications involved failure handling when it occurs in an individual step of a workflow. In the description of the specification, when workflow step is going to be executed, the step event will occur to notify the engine. Workflow engine will be informed the status of the workflow step whether the step can be completed or need to be partially rollback and re-start from the step with another rule (substitute) replaced. Alternatively, if the recovery mechanism does not exist in the specification of the step, the step will be aborted as a default action in the specification. This notion can be applied to Device Ecology to resolve the on-the-fly device failure detection. However, our requirements of recovery can be more complicated because we are incorporating the concept of context in our work. The on-the-fly workflow failure handling in [25], [26], [28] or [29] are using ECA\(^2\) to execute in-built functions to recover failures, which may be unsatisfactory in Device Ecology because our substitution depends on available contexts in the environment.

\(^2\) ECA = Event-condition-action = ON event IF condition DO action
2.4. Applying Context-Awareness to Workflow Systems

2.4.1. Definition of Contexts

“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”.

Loke et al. [31] defined contexts by dividing them into four categories for Device Ecology:

- **Computing context**: This includes any information that relates to the computing facilities and environments such as workstations, printers, scanners, network connection and bandwidth.

- **User context**: This refers to any information about the user which includes user’s profiles, intentions, current location, and current social activity, for example, user is watching the television; user just came back home from work; or user is having a meeting with friends.

- **Physical context**: This involves any physical entities and properties in the environment such as the brightness of the room, temperature, and noise levels.

- **Time context**: This context refers to the current time and the user’s schedule. For example, the living room light will be switched on automatically at night time.

Device Ecology was aiming to be a context-aware system consisting of automatic devices working collaboratively to achieve the goal of workflow activities based on the workflow setting and contexts. This aim matches Dey’s definition of context-aware system.

“A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.” [30]

2.4.2. Context-Aware Systems

Dey et al. [32] have developed a framework for context-aware application, so called Context Toolkit, which has been applied to much research that relates to context-awareness. The Context Toolkit initially consists of three abstractions:

- **Widgets**, which encapsulate information about contexts. Context widgets are independent components, and can be accessed by applications through traditional poll and subscribe methods.

- **Aggregators**, which can be considered as meta-widgets that describe all
capabilities of widgets. They act as gateways between applications and elementary widgets and contain context-sensing mechanisms.

- **Interpreters** are used to analyse individual context information (low-level context information) to realise the current situation (high level context information) in the environment. For example, system may interpret that a user is having a meeting by analysing context information such as identity, location and sound volume.

Dey et al. [32] have mentioned that historical information is an important part of Context Toolkit. Hence, the support of context storage was provided. All the context information that has been sensed by Context widgets will be stored and can be invoked by applications for future needs. This functionality is handled by Interpreters as well. Figure 2.4 describes the relationship between components in Dey’s Context Toolkit.

![Sample configuration of context components](image)

Figure 2.4  Sample configuration of context components [32]

The Context Toolkit was written in Java programming language and the message passing is through the HTTP with XML formatted data. This means that the Toolkit can be implemented on different platforms. However, Henricksen et al. [33] mentioned that the Toolkit does not specifically address scalability, privacy, component failures, or deployment/configuration.

Henricksen et al. [33] have introduced their context-aware system model, which consists of five layers components as Figure 2.5 shows.
The responsibility of each layer is described below:

Layer 0: Contains physical sensors which produce context information
Layer 1: Processes low-level context information to high-level context information
Layer 2: Stores context information and advanced query facilities
Layer 3: Supports applications to select proper decision and adaptations based on the available contexts
Layer 4: Support interoperability between applications and other components of the context-aware system

The detailed description of their system architecture (see Figure 2.6) was provided in the earlier work of Henricksen [34].
Although Henricksen et al.’s work [34] is not meant for a workflow system and they did not cover the area of failure detection and recovery for their context components, their design of the component layered model provides us with a better understanding of what needs to be considered in the design of context-aware system.

Capilla [35] has introduced a context-aware architecture for service-oriented systems. The service-oriented system in Capulla’s work is based on BPEL4WS as well. The architecture is depicted in Figure 2.7.
As shown in Figure 2.7, services are registered and can be discovered by UDDI (which is the common approach in BPEL4WS). The context data is stored in a database and can be retrieved through a context manager. The objective of Capilla’s work [35] is to sense different available services (contexts) from other service-oriented systems and continue to update context information automatically at runtime. The system will provide the updated information to the system user or the server itself to perform interoperability.

One of the problems mentioned is the change of URL for new services. URLs for services were specified in WSDL before workflow execution. Although BPEL4WS provides the ability to invoke partners dynamically, Capilla [35] mentioned that the potential problem of applicability may occur when a dynamic port is assigned. Alternatively, the context manager can be implemented as a separate component from the BPEL4W. However, it takes extra processes and efforts to establish the interaction between clients, services and context manager.

Although this work may not be relevant to our work as it does not deal directly with the workflows, it can provide us with the basic idea on how to implement context-aware system in BPEL4WS.

Context Engine is a project developed by Goh et al. [36]. It is a context-aware workflow system which focuses on sensing environment contexts in order to determine the current situation. The system consists of following components:

- **Core engine**, which is capable to interact with other web services;
- **Reasoning services**, which process context information;
- **Rules repository** stores the pre-defined logic-based rules; and
- **Graphical User Interface** (GUI) allows system administrator to register physical entities such as user or devices manually into the system.
The architecture of Context Engine is shown in Figure 2.8:

![Figure 2.8 CE’s context architecture [36]](image)

Figure 2.8 shows a component called Reasoner included in the Core Engine. The purpose of the Reasoner is to execute custom inference rules and to handle updates. “It plays a vital role in providing the ‘intelligence hub’ of the contextually aware smart environment.” [36] Basically, it serves the same function as the Interpreters in Dey et al.’s Context Toolkit [32]. It interprets the low-level context information to high-level context information as the described in Section 4.2. Goh et al. [36] have provided a sample rule file (see Figure 2.9) which gives us a better understanding on how rules are defined.

```
# Example of a rule file
@prefix eg: <http://somewhere/else#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
[@preserve @prefix @iri]

[rule1]: (?u eg:LocatedIn eg:Pantry)
  (eg:CoffeeMaker eg:LocatedIn eg:Pantry)
  (eg:CoffeeMaker eg:Status eg:On) -> (?u eg:Event eg:MakingCoffee)
[rule2]: (?u eg:LocatedIn eg:MeetingRoom)
  (eg:Projector eg:LocatedIn eg:MeetingRoom)
  (eg:Light eg:LocatedIn eg:MeetingRoom)
  (eg:Projector eg:Status eg:On) (eg: Light eg:Status eg:Off) -> (?u eg:Event eg:InMeeting)
[rule3]: (?u eg:LocatedIn eg:MeetingRoom) (?u eg:Event eg:InMeeting) -> (eg:Phone eg:Status eg:Busy)
```

Figure 2.9 Syntax of a rule file [36]

Although this work was related to workflow system [36], they did not mention any adaptation or fault handling in their work. They only focused on the design of Interpreters.
Narendra [37] has introduced the technique of modelling adaptation for BPEL4W by context ontology. He described the objective of context-aware Web service is “to possess mechanisms for collecting contextual information and disseminating it to other Web services, as appropriate.” [37] Figure 2.10 depicts Narendra’s design.

Narendra’s work covers clear description of the information needed to be stored in each context components (see Table 2.1).

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Information Stored</th>
</tr>
</thead>
</table>
| Composite service     | • Goals of the workflow, consists of a version number which needs to be updated during adaptation. The version number serves the purpose of tracking the goals when they are changed.  
                        • The overall workflow schema derived from the goals; again, this workflow definition also needs to be version numbered, for use during adaptation. |
| Web service provider  | • Sub-goals for the individual component Web Services, derived from the overall goals; this can also be version controlled for identification during adaptation.  
                        • Sub-workflow schema, derived from the overall workflow schema stored at the C-context, again version controlled for identification during adaptation. |
There are two main features of context ontology in Narendra’s design. The first feature is that contexts can be continually tracked by the mechanism of context collection and dissemination. Hence the appropriate action will be performed for adaptation when changes occur. The second feature is based on the collected context data and additional information (see Table 2.1) that enable workflow to be changed dynamically by invoking adaptation strategies on-the-fly and the contextual information which pertains to the Web service components will be updated automatically. Figure 2.11 shows the design of components.

Narendra’s work [37] focused on the design and specification. The description of its implementation was not provided, and also the detail implementation of adaptation was not described. However, the design may be applicable to our project with regards to the information collection and storage of context components.

### 2.4.3. Run-time Context-Aware Decision Making

The Decision Engine [9] is a component which was designed to support a workflow system to utilise available contexts in the environment and is used to determine the workflow execution path based on context. It is a component of the context-aware model in Lee’s work [9] which was originally developed to support non-determinism in Device Ecology workflow. Non-determinism is not within the scope of our research project. However, the decision making algorithm based on contexts is highly relevant to our project.
Base on the concept of Dey’s Context Toolkit [32], Lee has implemented an additional component to Decoflow in order to support provide a mechanism for the Decoflow engine to choose the best execution path at run-time.

Figure 2.12 shows the original workflow (in Petri net notation), which has a choice of executing one of the three activities: Raise curtain, switch on bedroom light and switch on bathroom light. In Figure 2.13, the Decision Engine has been placed before the execution of the (non-deterministic) choice. The Decision Engine will decide which activity is to be executed based on context.

The decision making is based on the utility value of context. Lee [9] has introduced a mathematical function to calculate the utility value for each activity. In order to calculate the value, context information has to be converted to numeric value and stored in profile. Values are stored in a standardised range from 0 to 10 or 0 to 1. For example, to make a decision between ‘open window’ and ‘turn on air conditioner’, the calculation is shown in Figure 2.14.
Utility(open window) = \( w_d \times (\Sigma w_c \times v_c) \)
\[ = w_d \times (w_{\text{day}} \times v_{\text{day}} + w_{\text{night}} \times v_{\text{night}} + w_{\text{sunny}} \times v_{\text{sunny}} + w_{\text{windy}} \times v_{\text{windy}}) \]
\[ = 1 \times (-0.2 \times 1 + 0.5 \times 0 + -0.3 \times 0 + 0.8 \times 1) \]
\[ = 0.6 \]

Utility(switch on air-conditioner) = \( w_d \times (\Sigma w_c \times v_c) \)
\[ = w_d \times (w_{\text{day}} \times v_{\text{day}} + w_{\text{night}} \times v_{\text{night}}) \]
\[ = 0.5 \times (0.8 \times 1 + 0.3 \times 0) \]
\[ = 0.4 \]

As Figure 2.14 shows, Utility (open window) has a better result. Hence, it is the chosen decision.

Referring to the scenario in previous section, when device failure occurs, the workflow engine will need to make a decision on selecting the best substitute based on available contexts. Although the Decision Engine will be designed to serve a different purpose in our project, Lee’s run-time context-aware decision making mechanism [9] can be adapted for our work to choose the best substitute at run-time.

Choi et al. [38] have introduced a dynamic context-aware workflow system which is capable of performing adaptation based on the contexts without interrupting the on-going workflow. In order to change the workflow path dynamically, such as add a new workflow path into the on-going workflow, they have developed a component called DItree to handle the adaptation process. DItree is implemented as a component in the context-aware workflow engine. It has been designed to support the adaptation of workflow as shown in Figure 2.15.
A scenario was given to describe how the workflow system handles the change. According to the scenario, the system will execute the workflow activities of ‘warm bath’ when John comes back home. After John finishes his bath and sits on his couch in living room, the next step will be to execute ‘turn on television’ followed by ‘switch on favourite channel’ as initial setting. The adaptation occurs when John comes back home with Tom for a meeting. John will interrupt the workflow for the meeting preparation. A new service will be downloaded from John’s workstation in the office and be adapted into the existing workflow. The change is shown as Figure 2.15. Due to the interruption, the initial activities will be postponed until the new activity is completed. Figure 2.16 shows a clear description of the process for adaptation.

<table>
<thead>
<tr>
<th>Input</th>
<th>An initial DItree and a user's new demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>A DItree reconstructed against the changes</td>
</tr>
</tbody>
</table>

Method:

start demandProcess()
1. Checks whether the demand is for a new service or for context modification of services, which exist already in the DItree.
2. In the case of the former, calls makeSubtree() function, in the case of the latter, calls modifySubtree() function.

In makeSubtree(param1:DItree, param2:service, param3:contexts)
1. Makes a service node for the new service.
2. Makes a RDF-based subtree for execution conditions of the new service through the structural context model in Figure 2.
3. Calls a function findPlaceholder() to find a exact position that the subtree can be attached in the DItree.
4. Attaches the subtree onto the placeholder, that the findPlaceholder() returns.

In findPlaceholder(param1:DItree, param2:subtree)
1. Traverses the DItree to find a position in which the subtree has to be inserted among service nodes of the DItree.
2. When the position is found, return the point of it.

In modifySubtree(param1:service, param2:contexts)
1. Moves the root node of subtree for a service, which will be modified.
2. Modifies the structures or contents of the subtree according to the contexts, which a user inputs to change.

Figure 2.15  A demand process algorithm to adopt changes of contexts to a DItree [38]

The Subtree in Figure 2.16 provides external workflow activities to support the existing workflow. The process of Figure 2.16 may be applicable to Decoflow for runtime adaptation. However, Choi et al. did not describe what will happen if the workflow already executes the activity of a ‘warm bath’ before John’s interruption is performed. A roll-back or re-start algorithm might need to be considered in their design, but these issues were not covered.
2.5. Summary

From the study of relevant existing work, we intuitively feel that resolving on-the-fly device failure in Device Ecology involves mechanisms for workflow adaptation and context-awareness.

In order to define elements for the workflow adaptation, relevant works have been studied. As described in Section 2.3.1, we have explored the general goals of adaptive workflow. Referring to the goal of Handling Changes to the Process Instance in adaptive workflow, and analysis of the scenario of on-the-fly device failure, the problem statements have been further clarified. To solve on-the-fly device failure, the workflow system must be able to detect failure during workflow execution and perform recovery algorithm such as replacing failed device by a substitute.

Agent techniques can be used to support the capability of workflow in order to detect on-the-fly failures. To do so, agents can be placed in each step of the workflow as observers. Workflow engine can then be notified of any failure by the agents.

In previous research on adaptive workflow, ECA is the common way to handle exceptions. However, such in-built exception processes are unable to fulfil the context-awareness requirements in Device Ecology when handling on-the-fly device failures.

Device Ecology aims to be context-aware. Activities in Decoflow can not only be defined by administrator’s setting, but also depend on the dynamic changes in context. To realise context-awareness in Decoflow, existing context-aware techniques and context-aware workflow systems have been explored. One such context-aware model consists of a number of components, from bottom layer component – context sensor to upper layer component – the application layer. In addition, the calculation of context utility values for the best may be needed.

A potential difficulty of designing a context-aware adaptive workflow system might be the definitions of profile and rule. Most research does not provide much information on how profiles and rules should be defined. It is understandable because the absolute rules for workflow are nonexistent. The task of defining profiles and rules can be a very time consuming.
Chapter 3.

Solving On-the-fly Device Failures in Context-Aware Device Ecology Workflows

3.1. Introduction

In previous chapter, we have investigated relevant contributions which involved run-time adaptation in workflow system and using context-awareness to enhance the capability of workflow. In this chapter, we will introduce our proposed framework toward resolving run-time device failures in Device Ecology workflow (Decoflow) by using context-awareness.

In this chapter, we first discuss the run-time device failures in Decoflow in detail and propose a possible resolution for the device failure detection at run-time. Secondly, we describe how context-awareness can be implemented to enhance the capability of Decoflow to discover and identify a best substitute during the run-time device failure recovery algorithm. Our proposed framework will be introduced which includes the details of the behaviour of workflow, how recovery algorithm can be performed and components that will be used in this framework. The primary objective is to realise a device failure recovery algorithm at run-time.

3.2. Run-time Failure Detection and Recovery

3.2.1. Analysis of Run-time Device Failures in Device Ecology Workflow

An example of the run-time device failure has been described in the previous chapter. In this section, we analyse the reason which causes the run-time device failure in Decoflow and propose a possible resolution. Firstly, we briefly summarise the scenario in the run-time device failure example. The workflow task process in this example is a refinement of the task – turn on bathroom light in the wake-up scenario
of previous work [8], [9]. In the process of this workflow task, when the workflow reaches the node of activity – turn on bathroom light, the device – bathroom light should be turned on as the workflow description has defined. The workflow description can be either a document file such as BPEL or any form which the workflow engine is able to read. Note that developers may implement different parser program to enable the workflow engine to read different workflow descriptions. Figure 3.1 represents the relationship between the workflow description and the workflow engine; it also describes how Decoflow activates a device. As the figure shows, the workflow engine loads the workflow description and executes the task which was defined in the workflow description. At run-time, the workflow requests the bathroom light service to activate the bathroom light.

![Workflow Description and Task Execution](image)

Figure 3.1 Workflow description and task execution

In this sample scenario, there are three potential issues that may occur to cause the device failure at run-time.

1. **Service failed** – the bathroom light service is unable to control the bathroom light. This issue may occur due to the service provider failure (e.g. the embedded micro-processor on bathroom light, which provides the service, is failed to operate).

2. **Network connection unavailable** – Workflow engine is unable to communicate with the bathroom light service due to the network connection lost.

3. **The target device failed** – it may occur due to a sudden occurrence of a device crashed. The first two issues are hardware failures which can be considered as Decoflow system failure. Although they cause device failures, they are not within the scope of this research. Our research mainly deals with the third issue.

Base on the relevant contributions [25], [26], [27] which have been discussed in the previous chapter. Agent technology can be implemented in a workflow system to
support the workflow to discover failed tasks. An agent is not necessarily a mobile software agent [39], but it can be a service of any form (e.g. web service, RMI [40] or socket [41] etc.) which is capable of observing a task or handling a task in the workflow. The objective to implement an agent to handle a task is to enable the system capable of observing its task node and detecting the allocated task is applicable or not at run-time, instead of just activates a task when the workflow reaches the node. In our work, we will use the same agent concept to manage task. We call it a task manager. Figure 3.2 represents a process which implemented a task manager to handle a workflow task.

![Diagram](image)

Figure 3.2 Task manager detects device failures at run-time

As Fire 3.2 shows, when workflow execution starts, it will pass the task to the corresponding task manager when the task node is reached at run-time. The task manager will then examine whether the task is applicable or not by checking the status of the corresponding device via the device service. Hence, the workflow engine is capable of detecting device failures at run-time.

3.2.2. Workflow Adaptation for Devices

In the previous research by Kurniati [5], device failure can be detected before the workflow execution. As described by Loke et al. [6] and Kurniati [5], device tasks were implemented by Web services. Since devices are Web services, they are capable of communicating with the workflow engine through the wireless network. Information and status of devices can be retrieved by the workflow engine, and the workflow engine is able to identify the availability of devices. If a device, which is
utilised in a workflow process, is unable to be activated, it will be considered as device failure. A recovery algorithm will then be executed in order to discover a substitute device and modify the original workflow path to activate the substitute device as substitution for the failure device. Kurniati’s algorithm can be summarised in the flowchart shown in Figure 3.3.

The adaptation is done before run-time. Kurniati’s work [5] is capable of detecting device failures before the workflow execution by checking the status of devices. If a device failure occurs, the workflow engine will assign a replacement from pre-defined substitutes depending on the type of failure. The similar method can be implemented to the task manager (which has been described in last section) in order to perform an adaptation. The difference is we will implement the feature of context-awareness to identify a substitute at run-time instead of choosing one from the pre-defined substitutes. We will describe the feature of context-awareness in later sections.

Figure 3.4 represents the refinement of task execution of the modified workflow adaptation for run-time failure detection and recovery. In our proposed system, each workflow task will be checked before the task is executed at run-time. If a device failure is detected, the workflow adaptation algorithm in Figure 3.4 will be executed to discover substitute device and activate it. However, substitute device sometimes may not exist. Hence, if no accurate device is found, the alternative path will be executed in which the failure will be reported to the administrator and possibly relied onwards to the user to perform the substitution manually.
3.3. Context Ontology

Oxford English dictionary explains the word ontology as “Philosophy the branch of metaphysics concerned with the nature of being”. The term – ontology is a philosophy word to describe the existence of being. In computer science, ontology has been widely used in context-aware systems. Various works [42], [43], [44] have used the term “context ontology”. The general meaning of context ontology is to define the relationship of contexts, retrieving and categorising context information in a systematic way in order to transfer context information to programmable data. Context ontology can be a vast topic to researchers due to different people from different background has their own opinion and explanation about contexts. Based on different view and notion, contexts can be identified and categorised very differently. What we proposed are definitions which focus on fulfilling the needs of smart home environments.
3.3.1. Context Categories

Context-aware system has the capability of collecting and managing context information. Our proposed run-time workflow adaptation in Device Ecology relied on the current environment context. As different types of activities require different context data, we categorise context data into different types in order to let workflow system to retrieve relevant context data.

Referring to previous work [6], context information can be divided into four categories – computing, user, physical and time contexts. The concept of context information in the work is general, but we need to categorise context into a more detailed model in order to fulfil the need of our work. Therefore, we have categorised context into object-oriented structure consisting of top level context (supertype context) and secondary level context (subtype context). A subtype context can also be the supertype context of the other context types. For example, the computing context type is the subtype of the supertype context – device, and it is also the supertype context of hardware, software and networking. (see Figure 3.5)

In our proposed design for Device Ecology, primary supertype context includes environment, device, user and date/time context types. Each supertype context may contain various subtype contexts. For example, location and weather are subtype contexts of environment. Figure 3.5 presents the context types in detail. As the Figure shows, environment, device, user and data/time are supertype contexts while location and weather are subtype contexts of environment; computing and facility are subtype contexts of device; hardware, software and networking are subtype contexts of computing. Note that subtype contexts are not limited as Figure 3.5 shows; other context types can be included to fulfil different needs. In this thesis, our system mainly deals with a smart home environment. Therefore, context types in Figure 3.5 are satisfactory.
We describe two context types: Location and User context types. Figure 3.5 shows that three attributes have been specified for location context type – temperature, brightness and noise. As these three attributes can be determined depending on the location, therefore, each location context should contain its own data value for temperature, brightness and noise. For the user context type, since the goal of context-aware system is to assist user’s activities, the system needs to be able to detect user’s current activity and location in order to provide proper assistances.

Attributes for each context type is not limited to Figure 3.5 shown. The objective to present Figure 3.5 is to propose a guideline to categorise context. In a real world system, additional context attributes need to be considered. For example, in order to detect user’s current location, each location sensor should have the ability to sense user is currently in the specific location or not. Hence, a location type should have an attribute to define whether the user is currently at that location or not.
3.3.2. The Goal of Workflow Tasks and Contexts Utility

We define a goal for each individual workflow task for run-time device failure recovery. When the failure has been detected by the system, the recovery algorithm needs to be executed, a substitute device needs to be found and activate. Logically, activating device is an activity, and each activity is executed for a purpose (goal). A goal can be achieved by one activity or many different activities. However if an activity cannot achieve its goal by 100%, it cannot achieve the goal successfully. Therefore, randomly choosing an available activity to be the substitute will not be a satisfactory decision. Effectively, the system should compare the available substitutive activities to discover which one will be the best choice. Figure 3.6 represents a simple example of the relationship between goal and activities.

As Figure 3.6 shows, both A₁ and A₂ can achieve G₁. However, the efficiency of their achievement is actually affected by environment context which cannot be ignored. For example, the activity “raise curtain” will not be able to achieve the goal “brighten bedroom” at night or another: during a cloudy and raining day. To emphasise this concept, we proposed Figure 3.7: relationship between goal and activities taking context into consideration.
In Figure 3.7, weather context has been considered in the process. Let each activity has initial utility value as 1, which indicates the activity is able to achieve the goal by 100% efficiency. When $A_1I_{c1}$ (Switch on bedroom light in sunny day) occurs, the utility value of $A_1$ will be reduced due to the efficiency has been reduced. Note the dependency is defined in the rules and profile. It is possible that $A_1I_{c1}$ results as 1 if the rule was not defined to reduce the value of activity efficiency when bedroom light is switched on during the sunny day, but in the example of Figure 3.7, $A_2$ will be a better choice then $A_1$.

There can be more then one context dependency for each activity. Therefore, they need to be considered when utility value is calculated. The proposed utility function is defined below where the utility value equals “value of activity efficiency to goal” plus sum of “value of context influence to activity”.

$$V_u = A_g + (\sum AI_c)$$

$V_u$ = utility value

$A_g$ = value of activity efficiency to goal

$AI_c$ = value of context influence to activity

The calculation of context utility was introduced in Lee’s work [9] to compare the utility values of various workflow tasks in order to resolve the non-determinism. In our work, we refine the utility calculation to suit our goal. Generally, Lee’s utility
function calculation involves the importance of an activity. Our proposed utility function focuses on the goal of an activity. A more detailed description of our utility function can be found in Section 3.4.3.4.

3.4. A Framework for Context-Aware Adaptive Workflow

3.4.1. Framework Requirements

To realise the run-time context-aware adaptation for Device Ecology workflow system, we propose the following requirements.

- **Detect failure** - Device failure detection function needs to be executed before each workflow task execution. Intuitively, if the workflow system is capable of communicating with each single device, it should be able to detect the device failure before activates the device for a task.

- **Collect information** - The purpose of workflow adaptation is to discover available substitute device to replace the failed device. To do so, relevant information needs to be collected such as the current user profile and rules, the goal of activity and current context information.

- **Make decision** - Once the relevant information has been collected, various available substitutes can be found by system. The decision making mechanism needs to be used to choose a best substitution for the failed task.

- **Report task failure** - If substitute device is not found, the workflow system should be able to report the device failure to the system administrator in order to inform the system administrator to perform further action manually.

3.4.2. Architecture Overview

To incorporate the above requirements in our project, Figure 3.9 represents our proposed framework architecture.
The framework contains the following main components:

- **Workflow description** is the document which describes the workflow processes and the relevant service connection for each task. It can be any XML formatted document such as Petri Nets Markup Language (PNML) [45] or Business Process Execution Language (BPEL). As long as the function in the workflow engine is capable of parsing the workflow description, the workflow description can be in any format.

- **Workflow execution** is the main operator of the entire framework. The workflow description document will be received by the workflow execution component and the workflow tasks defined in workflow description will be executed.

- **Adaptive Device Manager** directly communicates with the device services through the network to control devices. As mentioned in Section 3.2.1, a task manager can be implemented as a task node in workflow to support the workflow engine to perform the failure detection and recovery at run-time, therefore, we propose the Adaptive Device Manager as the task manager of workflow tasks. The Adaptive Device Manager is capable of handling tasks in each workflow task node. If the task device is unavailable, it will perform the failure recovery algorithm to define a substitute device. There are two components associated with the workflow adaptation component – the Substitution Manager and the Context Manager.

- **Substitution Manager** and **Context Manager** are components of the context-aware model which supports the failure recovery algorithm to find a best substitute according to the current context. The details of the context-aware model will be
described in next section.

- **Context Sensors** are external hardware devices for collecting current context information. The collected context information will be retrieved by the Context Manager.

- **Data Storage** stores three main groups of data: devices, context and rule/profile data. The detail of data storage will be described in detail later.

- **Devices** are controlling by device services which are capable of communicating with the workflow engine. Hence, the workflow engine is capable of detecting the availability of devices and controlling devices.

### 3.4.3. Context-Aware Components

#### 3.4.3.1. Overview

Our proposed run-time workflow adaptation relies heavily on context information. Therefore, we will explain the objective of context-awareness components and how they have been implemented in the following sections. There are four components in the context-aware model:

- **Context sensors** are physical hardware components that are capable of collecting context information such as current environment situation, room temperature or room brightness. Since context sensors are external devices from the workflow engine, context manager will be the bridge between the sensors and the workflow engine.

- **Context Manager** is capable of retrieving context information from the sensors, categorises the context data collection and stores the context data into database after process. It is also capable of redefining context information according to the corresponded rules.

- **Substitution Manager** serves as a substitution information provider. It cooperates with the Context Manager and provides the information of available substitution for the workflow adaptation component.

- **Context database** stores current context information such as date, time, room temperature, brightness etc.

Following sections provide the detailed description of each component that is involved in the context-awareness.
3.4.3.2. Context Sensors

Sensors are hardware components which can be formed as small processors that can be imbedded in different devices or specific location of environment to collect current context information. For example, a sensor device can be attached outside the bedroom window to collect the value of current weather context. The Context Manager component of workflow system is capable of interacting with sensors through the wireless network. The collected context information will be retrieved by the Context Manager. After context information has been processed, the context data will be stored in the context database component.

3.4.3.3. Context Manager

Context Manager has similar capabilities as Context Toolkit [32] which collects context information from external sensor devices and transforms them into programmable data. Such context data will be stored in the database component and will be updated continually while the system is active. There are three major elements in the Context Manager:

- **Widget** directly communicates with its corresponding external context sensor device to retrieve context information and transform it into programmable data.

- **Aggregator** acts as a bridge between Widget and the application. Aggregator’s role is to process information from a collection of the same-type Widgets and transform them into specific data for applications. For example, by collecting all information from location widgets, user’s current location will be traceable by the system.

- **Interpreter** transforms low-level context information into high-level information. For example: having the following information: the user’s location is in the living room, the number of persons is 2 or more, and noise is achieved at the specified level, the Interpreter will conclude the current activity of the user as “in a meeting”.

Figure 3.10 shows the relationship between components in the Context Manager and the external components.
In the figure, each Widget component communicates with its corresponding sensor device. The location aggregator contains a collection of location data which allows system to trace user’s movement. The User Widget simply defines user’s current location. Together with the user movement (traced by the location aggregator) and the current user location, the system is capable of analysing the current user activity from the user activity interpreter. The external application from the Context Manager is allowed to retrieve context data via the interpreter components or the aggregator components.

### 3.4.3.4. Substitution Manager

The Substitution Manager component in the workflow engine plays an important role during the adaptation process. It is one of the adaptation components associated with the Context Manager. To invoke the Substitution Manager, the following tasks need to be performed:

1. **Define the goal of activity** – Each workflow activity has its goal, which has been described in Section 3.3.2. When the adaptation process is executed, the goal should also be passed to the Substitution Manager to know which substitute action is valid.

2. **Retrieve current context data** – Context is the major influence in the decision making of the Substitution Manager. Hence, the Context Manager will be invoked to provide the current context data.

3. **Retrieve alternative activities** – Since the initial defined activity is unable to be achieved due to device failure, the system should retrieve other alternative
activities that can achieve the same goal as the failed one. Substitution Manager will choose one of them as the best alternative for substitution.

Once the above information has been collected, the Substitution Manager will firstly eliminate invalid activities according to the current user profile and rules. The rest of the activities will be compared by calculating their utility values. The following describes an example of utility value calculation.

Before we describe the utility value calculation example, we should explain the values used in the example. When the utility value calculation needs to be performed, the system will retrieve the AIc value from the rule record in database. A rule defines the AIc value of an activity. Hence, the validity of an activity depends on its relevant rules. Defining the AIc value can be a challenging task. We propose a systematic way to define the AIc value as below:

The valid AIc value is between -1 to 1 which includes -1, -0.5, 0, 0.5, 1 where:
- -1 represents the activity is very unacceptable;
- -0.5 represents the activity is unacceptable;
- 0 represents the influence of activity is limited;
- 0.5 represents the activity is encouraged;
- 1 represents the activity is very encouraged.

Now we describe how the utility values be calculated by using an example. The example refers to the scenario which has been described in previous chapter. In the scenario, device failure has occurred when bathroom light needs to be turned on. Assume there is only one substitution has been found which is “turn on the corridor light”. As the activity – turn on the corridor light can only brighten the environment of bathroom by 50%, therefore, the value of $A_g$ (turn on the corridor light) will be 0.5. There are two contexts influencing $A_g$ which are $I_{c1}$ (bedroom light is on) and $I_{c2}$ (energy saving). As the “bedroom light is on” can also achieve the goal of “brighten bathroom” by 50%, the value of $AI_{c1}$ will be -0.5, and the activity “turn on the corridor light” will causes extra energy, hence, the influence value affected by the “energy saving” will be -0.5. Note that all these values have been predefined in the workflow system setting by the administrator. The calculation is shown below.

$$V_u = A_g + (\sum AIc)$$
$$= 0.5 + ((-0.5) + (-0.5))$$
$$= -0.5$$
As the result is below 0, the Decision engine will generate the result as “Do nothing” which means the substitution is out of ideal. Since the message “Do nothing” might confuse the user that system is not working properly, a notification must be generated and inform the user for further action in an ideal situation.

3.5. Behaviour of the Engine

3.5.1. Detecting Device Failures at Run-time

In the early design of Device Ecology workflow, task processes were quite straightforward. Figure 3.11 represents the refinement of a workflow task. As the figure shows, the activity which was defined in the workflow description is immediately executed when workflow reaches the activity node. The activity node consists of information about what action will be performed and which device is involved. As mentioned in Section 3.2.1, the workflow engine is capable of detecting the device failure within the workflow engine try to invoke the service which represents the corresponding device. However, the challenge is how to implement the run-time device failure recovery algorithm in the workflow.

We propose to identify a proxy service which represents a task manager (see Section 3.2.1) in the workflow description instead of identifying the actual device service for the task. In the task node, workflow will execute the proxy service and further process will be handled by the proxy service. Once the proxy is executed, it receives relevant information such as the activity ID, default activity and the default corresponding device.

In our proposed system, the Adaptive Device Manager (see Section 3.4.2) represents the proxy service. The Adaptive Device Manager is a service which is capable of checking the availability of the device. If the device failure occurs (either the device status shows that it is unavailable or the workflow engine is unable to communicate with the device), the Adaptive Device Manager will perform the failure recovery algorithm by invoking the relevant components. Figure 3.12 represents the process of
failure detection and the partial recovery process.

![Diagram of Decoflow with the run-time device failure detection processes]

Figure 3.11 Decoflow with the run-time device failure detection processes

3.5.2. Context-Aware Workflow Adaptation

As mentioned in the previous section, the failure recovery algorithm process is executed once the device failure has been detected. The primary objective of the recovery algorithm is to retrieve potential substitute activities from the database component and discover substitute device as the replacement for the failed device base on the current context influence. The secondary objective, which will be chosen if the primary objective cannot be achieved, is to report the device failure to the system administrator and relied onward the manual recovery action to be performed. Figure 3.13 represents the refinement process P1 of the recovery algorithm.
P1 represents the first part of recovery algorithm process. The current context data will be retrieved by the Context Manager in the first step. The purpose of retrieving context data is to ensure the invalid substitute activities will be eliminated during the adaptation process according to the current context. For example, the task to brighten bedroom failed, the following factors have to be considered:

- Time context presents night time
- Profile rule defines raise curtain is a negative action at night

Hence, the activity “raise curtain” will be an invalid substitute and it will be eliminated from the substitutive activity list. Once the activity list is generated, there is a possibility that the list could be empty which will generate the result of failure recovery algorithm as “no substitute available”. A failure report will be sent to the system administrator to notify the failure could not be resolved. If the result contains one or more available substitute activities, the process P2 (see Figure 3.14) will be executed.
Process P2 (see Figure 3.14) represents the process of the Substitution Manager. The feature of Substitution Manager is to calculate the utility values for each substitute activity as described in Section 3.4.3.4 (Substitution Manager). There should be only two types result returned from the Substitution Manager.

1. The best substitution has been defined after the utility calculation process is completed. This result will lead the recovery algorithm process to activate the substitute device for the failed device and complete the recovery algorithm.

2. Substitute activity cannot be found. This result will be generated by certain conditions. For example, when all substitute activities results in the negative utility value after the calculation, this result will be generated. (see Section 3.4.3.4 example)

If the second result is generated by Substitution Manager, then a proper report need to be sent to the administrator to notify the situation. Otherwise it may cause user’s confusion about system which does not work properly.

Figure 3.13 Recovery algorithm part 2
In other research that involves workflow failure recovery, “roll back” is a common action when failure cannot be recovered. However, the roll-back action cannot be performed in our case at run-time. Once the workflow is executed (at run-time), the possibility of roll-back is extremely low due to certain tasks have already been executed previously. For example, an action involves “turn on coffee boiler” which makes coffee for the user. If the coffee is made, it is impossible to roll-back to the task when coffee is still in the coffee bean stage.

3.6. Summary

Our proposed framework aims to provide a solution toward resolving on-the-fly device failures in Decoflow. We have described the components in our proposed framework including the Adaptive Device Manager, the Substitution Manager and the Context Manager, as well as the details of the failure recovery algorithm.

The Adaptive Device Manager behaves as a task manager which is capable of detecting the task is executable or not at run-time and performing recovery algorithm when the task fails. The Context Manager is capable of collecting current context data; moreover, it is able to transform the low-level context data to the high-level context data. The Substitution Manager assists with the Adaptive Device Manager to perform the utility calculation for each potential substitute activities based on current context, and defines the best substitute for the failed task.
Chapter 4.

Implementation Description

4.1. Introduction

In the previous chapter, we have introduced our proposed resolution for on-the-fly device failures in Device Ecology. In this chapter, we describe the implementation of the prototype simulation as proof-of-concept. In this prototype, we emphasise on the methods of the run-time device failure detection and the failure recovery algorithm which is the primary contribution of this research.

In a real-world Device Ecology system, various physical devices and external sensor devices are involved. Due to the limitation of this research, we will not implement the physical devices and sensors. A database will be implemented to represent the sensor data which will be retrieved by the Context Manager of our system. Although the external physical devices will not be connected with our system, the system still includes numerous Web services which are the controllers of devices.

Different from previous work [5], which toward controls the operation on performing ‘pre-recovery’ before the workflow execution, the prototype in our work implements the failure recovery algorithm at run-time. Therefore, the BPEL workflow is actually implemented as a complete executable framework which is capable of handling requests from the client-side application.

The prototype aims to evaluate the low-level implementation of the run-time recovery algorithm. Therefore, the high-level end-point application and graphical user interfaces are not covered.
4.2. Overview of the Prototype Simulation

4.2.1. Required Components

Based on the framework in the last chapter, the following components have been implemented in our prototype:

- **Workflow execution** – It is capable of receiving requests from client and response the result to the client. In this prototype, the workflow description is implemented in BPEL and the workflow execution is done in Netbeans IDE [46].

- **Adaptive Device Manager** – It is a task manager which handles workflow tasks. When the workflow reaches the task node, the Adaptive Device Manager will be invoked as a partner in BPEL. It receives the information of the task and performing failure detection and recovery processes.

- **Substitution Manager** – When a task failure has been detected, the Adaptive Device Manager will execute the Substitution Manager to discover a substitute activity based on the available resources and the rules of context influences.

- **Context Manager** – It collects current context data from sensors. It also contains some extra features such as Aggregator and Interpreter which will be described in later section. The Context Manager is a group of classes associating together to realise the functionalities of Context Manager, it consists with a central class – ContextManager, various Widgets to collect different sensor data, aggregator to provide specific context information form a collection of Widgets, and the feature of Interpreter which is capable of defining a high-level context information from a collection of low-level context information.

- **Device Ecology Database (decodb)** – It stores data that will be used by the system.

- **Sensor Database (sensordb)** – Since we don’t have real sensor devices, we have implemented an extra database to represent sensor data.

- **Device Services** – They are web services which control devices in a Device Ecology system. Each service represents one device. They are invoked by the Adaptive Device Manager.

- **Client** – The client in this simulation is a simple composite application which is able to send a request (a SOAP message) to the workflow to start the workflow execution and receive a result (also is a SOAP message) when the workflow finish.
The workflow, Adaptive Device Manager, Substitution Manager and Context Manager are the major components in this prototype. Hence, they will be described in detail later on. Moreover, the structure of two databases will also be covered. Figure 4.1 represents the architecture of this prototype.

**4.2.2. Scenarios**

As simulation can be affected by different contexts, evaluating the prototype in different situations is necessary. We developed two main scenarios; each of them will be tested by a number of test cases as sub-scenarios.

**Scenario 1:**

The first scenario is based on previous work [9], known as the ‘wake up scenario’. The workflow of this scenario starts when the alarm clock activating the workflow execution. Tasks included in this workflow are: ‘turn on heater’, ‘turn on bedroom light’ and ‘turn on bathroom light’. The task which will cause device failure is – ‘turn on bathroom light’. The system has to be able to detect this failure and performs failure recovery algorithm in order to find a substitute at run-time. The workflow below briefly describes this scenario.
Scenario 2:
The second scenario is also based on the ‘wake up scenario’ but with a different placement of tasks. In this scenario, the alarm clock task has been designed to be activated with other tasks concurrently and the failed device is the alarm clock. The system has to be able to detect this and also discover a substitute. There are two alternative actions which are capable of waking up the user: (1) Activate the alarm which is embedded in the mobile phone if the mobile phone is available. (2) Increase the volume of the television which is located in the bedroom if the television is activated. However, if the user is already awake, substitution is unnecessary. Hence, the device context data and the user context data are the primary factors that influence the result of recovery algorithm in this scenario. The workflow below briefly describes this scenario.
4.3. Implementation

4.3.1. Technologies used

Our prototype was implemented in Netbeans IDE v6.0 Beta 1 [46] which has the following features:

- Since version 6.0, Netbeans is not only a tool for developing Java applications, but it can be considering as a platform which is capable of executing service-oriented systems. For example, a web service built in Netbeans, can be activated directly in Netbeans without any external server running.
- Support BPEL processes. Developers can implement BPEL in either coding mode or graphical mode.
- It contains an in-built deployment server – GlassFish v2.0 [47]. Web services built in Netbeans can be directly deployed on GlassFish.
- It also contains an in-built database – Derby [48].

The benefit of using Netbeans IDE v6.0 Beta 1 is that everything we need for this prototype is included. It helps developers to save a lot of time on the environment setting.

4.3.2. System Design

The prototype consists of three main modules (in Netbeans [46], modules are called “projects”) which are:

- **AdaptiveDeviceManager module (EJB)** – It contains all packages that involve the run-time device failure recovery algorithm:
  - Package: edu.deco.adaptivedevicemanager contains a single web service – the task manager - AdaptiveDeviceManager.
  - Package: edu.deco.contextmanager contains numerous classes that involve the functionalities of collecting and processing context data. The detail of this package will be described later in this chapter.
  - Package: edu.deco.substitutionmanager contains a single class – SubstitutionManager which is capable of identifying a substitute activity according to current context data.
  - Package: edu.deco.data contains classes which are used to represent corresponding data which includes ActivityData, RuleData, ProfileData, DeviceData and DeviceElement.
• **Package:** `edu.deco.database` contains a single class – `DbManager` which is used to access database records.
• **Package:** `edu.deco.decodevice` contains various device Web services that used in the scenario of our prototype simulation.

**Decoflow module** – It is a SOA-BPEL module which contains the BPEL workflow of the testing scenarios. It also contains the corresponding XML Schema and WSDL file for the request and response SOAP message of client.

**Composite application module** – It is a module which imports the Decoflow module for the purpose of testing. It is capable of deploying the BPEL workflow on GlassFish and performing the testings.

Figure 4.4 is a diagram which represents the relationship of above components.

As Figure 4.4 shows, there are two modules will be deployed on the GlassFish web server – the composite application project and the EJB module – `AdaptiveDeviceManager` (EJB project). The BPEL workflow – Decoflow will be imported in the composite application before the deployment. The Web service –
AdaptiveDeviceManager (class) will be invoked as a partner during the workflow execution. The other components, which will be deployed together with the AdaptiveDeviceManager Web service, are the Web services of devices. The Web services of devices are invoked by the AdaptiveDeviceManager Web service. Hence the AdaptiveDeviceManager Web service is capable of interacting with the Web services of devices. The other packages (database, data, substitutionmanager etc.) have been imported in the AdaptiveDeviceManager Web service to assist the functions of failure detection and recovery algorithm.

4.3.3. Context Manager and Corresponding Components

4.3.3.1. Overview

A Context Manager is capable of collecting and processing context data by its assistant components. Figure 4.5 describes the relationship of the Context Manager and its components.
When a Context Manager is created, it directly executes its function `updateAllContextData()` to retrieve context data from sensors by using Widgets and Aggregator, and then update the context data of Device Ecology database (decodb). The Context Manager contains a method which enable the completed context data collection to be retrieved – the `getDecoContextCollection()` method. A collection of context data can also be retrieved according to its category by calling the `getContext()` method which requires a parameter – category ID.

As Figure 4.5 shows, the Context Manager uses Widgets to collect context data which includes ‘weather’, ‘time of day’ (DayTimeWidget), ‘user’s activity’ (UserWidget) and ‘location’ contexts. Interpreter is also included. The description of Widget, Aggregator and Interpreter components will be introduced in following sections.
4.3.3.2. Widgets and Aggregator

In our prototype, we have implemented four Widgets:

- **DayTimeWidget** – It retrieves the current time of day (morning, afternoon, evening etc.) from the sensor. Although the data of the time of day doesn’t need to be retrieve from external components. This design is to make the process of data retrieval concisely.

- **UserWidget** – It contains the data of user’s activity which includes:
  - User currently is moving or not
  - User’s current action (we have defined three actions – standing, sitting, lie down, to fulfil the need in our scenario).
  - User’s location: The UserWidget is unable to retrieve the data of user’s current location directly. The UserWidget will import the LocationAggregator to achieve this function.

- **WeatherWidget** – It simply collects the status of current weather such as sunny, cloudy or raining.

- **LocationWidget** – Each location data contains four attributes which are brightness, temperature, noise and the variable which represents whether the user is currently at that location or not.

All Widgets are subtype classes of the Widget. Figure 4.6 shows the relationship of Widgets which are used in our scenario.
Figure 4.6 Widgets for scenario
Figure 4.7 shows an example of how a Location Widget retrieves data from the sensor database. Each Location sensor contains four data, and each of them will be stored in the `sensordb` with an ID to define which location it belongs to. The Location Widget will retrieve all of them and updates the value of each corresponding Context Element data in `decodb`.

In our prototype, we only implemented one Aggregator which provides the current location ID of the user. As Figure 4.8 shows, the `LocationAggregator` has a collection of location context which is enable it to provide the location ID of current user’s location.

```
LocationAggregator
private ArrayList<LocationWidget> locations
private String userCurrentLocationId

public LocationAggregator()
public String getUserCurrentLocationId()
```

Figure 4.8 A simple Aggregator

### 4.3.3.3. Interpreter Functions

In order to realise the function of Interpreter, we have implemented the Interpreter Rule table and the Interpreter Rule Element (IR_ELEMENT) table in database. As Figure 4.9 shows, An `INTERPRETER_RULE` can have numerous `IR_ELEMENT(s)`; each `IR_ELEMENT` data describes a Context Element data.
The method to process Interpreter rules has been implemented in `ContextManager`. There are three steps in this process:

**Step 1:** Retrieve Interpreter rules which belong to the current user profile by calling the method `getInterpretersFromDB()`.

**Step 2:** Matching Interpreter Rule Elements to Context Elements. If all the Interpreter Rule Elements of the Interpreter Rule are matching to their corresponding Context Elements, the Interpreter Rule will be considered as 'requirement achieved'.

**Step 3:** Update the corresponding Context Element if the requirements of the Interpreter Rule are achieved by calling the method `updateContextElementFromInterpreter()`.

Figure 4.10 shows the source code of the process.
In the scenario 2 (see Section 4.2.2) of our simulation, we need to identify the activity of the user is sleeping or not. Hence, we will use Interpreter rules to define this high-level context (user is sleeping) and identify its occurrence by checking low-level context data such as ‘user is moving’, ‘user is lying down’ or ‘user is standing’ etc. For example, when the value of Context Element – ‘user is lying down’ is 1 (true) and the value of Context Element – ‘user is moving’ is 0 (false) and both of them were defined in IR_ELEMENT(s) of an Interpreter Rule (which updates the value of Context Element – ‘user is sleeping’ is true), then the value of Context Element – ‘user is sleeping’ will be updated to 1 (true).

4.3.3.4. Comments
The Context Manager in our prototype is a light-weight context data processor. The primary objective of this implementation is to evaluate our proposed Context Ontology and fulfil the needs of our testing scenarios. Basically, a completed Context Manager in Decoflow should contain more context Widgets and Aggregators.

4.3.4. Substitution Manager

4.3.4.1. Overview
Substitution Manager is used to find a best substitute activity based on rules, available activities and context influences. The primary function in this class is getSubstitutionResult() which requires two parameters – a collection of rules and a collection of activities. There are two possible results can be returned from this method – an Activity or a null which represents the substitute action is not needed based on a reason. The reason will be generated during the process and can be retrieved by calling the getDoNothingReason() method. There is an inner-class included in this class (see Figure 4.11) which is for temporary storing the Utility values of activities.
4.3.4.2. Substitution Function

In this section, we describe the detail of the `getSubstitutionResult()` method. Following are the processing steps:

**Step 1: Eliminate invalid activities according to their corresponding devices.**

The activity collection passed to this method may contain numerous invalid activities. For example, it may contain an activity which was defined to activate a device which is currently failed. Hence, following process needs to be performed.

```java
ArrayList<ActivityData> validActs1 = new ArrayList<ActivityData>;
for (int i = 0; i < this.activities.size(); i++) {
    if (checkDeviceStatus(this.activities.get(i).getDeviceId()).equals("ok")) {
        System.out.println("Activity: " + activities.get(i).getActivityId() + " is valid");
        validActs1.add(activities.get(i));
    } else {
        System.out.println("Activity: " + activities.get(i).getActivityId() + " is invalid");
        System.out.println("Due to device - " + activities.get(i).getDeviceId() + " is unavailable.");
    }
}
```

As Figure 4.12 shows, a new collection – `validActs1` has been created to store valid activities.

**Step 2: Retrieve context data according to rules.**

After a collection of valid activities is retrieved, a collection of Context Elements will be retrieved for the rules validation (`rules` is one of the parameters passed to this method). A Context Element ID list will be retrieved from the rules first. Then a collection of Context Elements (`ces`) can be retrieved from the Context manager (`ctxMgr`).
Figure 4.13  Step 2 of substitution

**Step 3: Validate rules with Context Elements.**

Figure 4.14 summarised the data structures of rule and Context Element.

As Figure 4.14 shows, a rule data has defined its corresponding Context Element data. When the Context Element value (CE_VALUE) of rule’s corresponding Context Element is matched to the Context Element value defined in the rule, the rule can be considered as a valid rule, and it can be added in validRules (see Figure 4.15).

Figure 4.15  Step 3 of substitution

**Step 4: Retrieve valid substitutive activities according to the valid rules.**

In this step, we validate the activities again by checking rules. If an activity has been defined in a rule with an AIC value of −999, it means the activity is an invalid action due to some effects (e.g. the Context Element - ‘current weather is raining’ is true and the activity is ‘open window’). As Figure 4.16 shows, the valid activities will be added to the validSubActs.
Figure 4.16 Step 4 of substitution

It is possible that all the activities are invalid. If so, the process will not be continue and the value of the doNothingReason will be assigned, and return a null value directly.

```
if (validSubActs.size() < 1) {
    this.doNothingReason = "No valid substitute.";
    return null;
}
```

Figure 4.17 No valid substitute

**Step 5: Calculate the utility values for each substitutive activity.**

The ActivityUtility is an inner-class in SubstitutionManager, it is only used to temporary restores an ActivityData together with an Utility value. In this step, a collection of ActivityUtility (actUtils) will be created and stores each valid activity and the utility valid of the activity (see Figure 4.18). The utility value is retrieved from the getUtilityValue() method (see Figure 4.19).

```
ArrayList<ActivityUtility> actUtils = new ArrayList<ActivityUtility>();
for (int i = 0; i < this.validSubActs.size(); i++) {
    ActivityUtility au = new ActivityUtility();
    au.activity = validSubActs.get(i);
    au.utilityValue = getUtilityValue(validSubActs.get(i));
    actUtils.add(au);
}
```

Figure 4.18 Retrieve utility values
In the `getUtilityValue()` method (see Figure 4.19), we first create a base value (`uv`) which is the Goal Achievement value (`GAValue`) of the activity. Then we check each data in `validRules`. If the activity ID in a rule data is matched to the activity ID of the parameter `act`, the `AIc` value of the rule will be add up to the base value (see Chapter 3: Section 3.4.3.4 for the description of the utility calculation).

```java
private double getUtilityValue(ActivityData act) {
    // utility calculation
    System.out.println("Calculate utility value for activityId: " + act.getActivityId());
    double uv = act.getGAValue();
    for (int i = 0; i < validRules.size(); i++) {
        RuleData rd = validRules.get(i);
        if (rd.getActivityId().equals(act.getActivityId())) {
            System.out.println("Rule ID: " + rd.getRuleId());
            System.out.println("AIc value: " + rd.getAIcValue());
            System.out.println("UV + " + rd.getRuleId());
            uv = uv + rd.getRuleId();
        }
    }
    System.out.println("Current AIc value for activity: " + act.getActivityId() + " is " + uv);
    return uv;
}
```

**Figure 4.19** The `getUtilityValue()` method

**Step 6: Compare the utility values of activities to define the best substitutive activity.**

Finally, we compare the utility value of each valid activity to identify which activity has the highest utility value. We assume there can be a number of activities have same utility value, hence, we use a collection (`resultAUs`) to store the activity which has the highest utility. Initially, we defined the first activity in the `actUtils` (generated in last step) has the highest utility value. Then we check each activity in the `actUtils`, if one of them has higher utility value then the first activity in `resultAUs`, we will remove all activities in the `resultAUs` and add the activity which has the highest utility value into `resultAUs`. However, if the activity of `actUtils` has the same utility value as the first activity in `resultAUs`, then we will add the activity directly into `resultAUs`.

Since it is possible to have more than one activity in the `resultAUs`, we will check if one of them has already been executed by matching the Device Element value. (see Figure 4.21 for the data structure and the relationship of Device Element and Activity). If the activity has been executed, the result of this process will be a `null` and the `doNothingReason` will be defined. If none of the activities in `resultAUs` have been activated, the result of substitute activity can be any activity in the `resultAUs` since they have the same utility value.
Figure 4.20  Compare utility values to find a best substitute activity

```java
ArrayList<ActivityUtility> resultActs = new ArrayList<ActivityUtility>();
resultActs.add(actUtils.get(0));
for (int i = 0; i < actUtils.size(); i++) {
    ActivityUtility au = actUtils.get(i);
    System.out.println("ActivityId: "+ au.activity.getActivityId() + "; Utility Value:" + au.utilitiyValue);
    if (au.utilitiyValue > resultActs.get(0).utilitiyValue) {
        System.out.println("ActivityId: "+ au.activity.getActivityId() + "; has the highest utility value.");
        resultActs.clear();
        resultActs.add(au);
    } else if (au.utilitiyValue == resultActs.get(0).utilitiyValue) {
        System.out.println("ActivityId: "+ au.activity.getActivityId() + "; has the same utility value as the highest one.");
        resultActs.add(au);
    } else {
        System.out.println("ActivityId: "+ au.activity.getActivityId() + "; has a lower utility value.");
    }
}
System.out.println("** Find best substitute activity.**");
ActivityData resultAct = resultActs.get(0).activity;
System.out.println("Current best substitute activity: "+ resultAct.getActivityId());
for (int i = 0; i < resultActs.size(); i++) {
    ActivityData ad = resultActs.get(i).activity;
    // check activity is already done or not
    DeviceElement de = emMgr.getDeviceElement(ad.deviceElementId());
    if (de.getValue() == ad.deviceElementValue()) {
        System.out.println("The activity "+ ad.getActivityId() + "; has been executed.");
        System.out.println("The result substitute activity in:" + ad.getActivityName()+" has been executed already.");
        return null;
    } else {
        System.out.println("The best substitute activity can be: "+ ad.getActivityId() + "; " + ad.getActivityName() + " has been executed.");
        resultAct = ad;
    }
}
System.out.println("The final result of the best substitute activity Id: "+ resultAct.getActivityId() + "; " + resultAct.getActivityName()+" ");
return resultAct;
```

Figure 4.21  Data structure and relationship between Device Element and Activity

<table>
<thead>
<tr>
<th>DEVICE_ELEMENT</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE_ID</td>
<td>ACT_ID</td>
</tr>
<tr>
<td>DE_NAME</td>
<td>ACT_NAME</td>
</tr>
<tr>
<td>DEVICE_ID</td>
<td>GOAL_ID</td>
</tr>
<tr>
<td>DET ID</td>
<td>DE_ID</td>
</tr>
<tr>
<td>DE_VALUE</td>
<td>DE_VALUE</td>
</tr>
</tbody>
</table>
4.3.5. Adaptive Device Manager

The Adaptive Device Manager is a Web service which is capable of handling the task of workflow. The main method is the `adaptiveDeviceOperation()` which receives three parameters – the ID of the current Decoflow user, the ID of the current applied user profile and the activity ID which defines the task. All the Web services of devices have been defined in the Adaptive Device Manager. Hence, the status of devices can be checked by the `isDeviceAvailable()` method. If the device, which performs the task, is available, the Adaptive Device Manager will pass the information of the task (ActivityData) to the Web service of device to execute the task by the `executeTask()` method and return the result. Otherwise, the Adaptive Device Manager will perform the recovery algorithm by using the Substitution Manager component (see Figure 4.23).

```java
if (isDeviceAvailable(activity.getDeviceId())) {
    String result = executeTask(activity);
    if (!DB.equals(result)) {
        return "Activity " + activity.getActivityId() + ": " + activity.getActivityName() + " completed."
    } else {
        return result;
    }
} else {
    SubstitutionManager substitutionManager = new SubstitutionManager(decoDecl, profileId);
    ArrayList<ActivityData> acts = retrieveActivities(activity.getOnId());
    if (acts.size() < 1) {
        return "Device Failure occurred. No substitution available."
    }
    ArrayList<RuleData> rules = retrieveRules(profileId);
    ActivityData substituteActivity = substitutionManager.getSubstitutionResult(rules, acts);
    if (substituteActivity == null) {
        return "Device Failure occurred. Recovery result is: " + substitutionManager.getNothingReason();
    } else {
        String finalResult = executeTask(substituteActivity);
        if (!DB.equals(finalResult)) {
            return "Device Failure occurred. Substitution device " + substituteActivity.getDeviceId() + " activated."
        } else {
            return "Device Failure occurred. Substitution device " + substituteActivity.getDeviceId() + " was found... Unable to activate substitute device. Please contact system administrator."
        }
    }
```

Figure 4.22 AdaptiveDeviceManager

Figure 4.23 Failure detection and recovery
As Figure 4.23 shows, a collection of activities will be retrieved from database by the `retrieveActivities()` method based on the goal of the task (see line 101 in Figure 4.23). If there are no activities found, a message will be returned to notify the failure (see line 102 and 103 in Figure 4.23).

A collection of rules will also be retrieved from database base on the profile ID (see line 105 in Figure 4.23). These two collections (activities and rules) will be passed to the Substitution Manager to find a best substitute activity (see line 106 in Figure 4.23).

If the result of the substitution process (the `getSubsitutionResult()` method of the Substitution Manager) is a null (see line 108 in Figure 4.23), a message will be retrieved by the `getDoNothingReason()` method of the Substitution Manager and this message will be the return data of current method (the `adaptiveDeviceOperration()`). On the other hand, if an activity ID has been returned from the substitution process, the substitute activity will be executed and the result message will be returned.

### 4.3.6. Workflow

We have implemented two workflows to test our scenarios. In scenario 1 (see Figure 4.24), the workflow starts with receiving a SOAP message from the client (DecoflowOperator). The message contains the information of user ID, profile ID and activity IDs. There are three tasks need to be executed concurrently, therefore, there are three sequences have been included in the concurrent flow. Each sequence represents one task of the scenario and there are three sub-tasks in each sequence. As Figure 4.24 shows, there are two ‘assign’ tasks that have been placed before and after the task ‘TurnOnHeater’. The ‘assign’ before the ‘TurnOnHeater’ is to copy the information (user ID, profile ID and Activity ID) from the request message and assign the information to the ‘TurnOnHeater’. The ‘TurnOnHeater’ is a task to invoke the task manager (the Adaptive device manager Web service) to execute the task adaptively. The ‘assign’ task after the ‘TurnOnHeater’ is to copy the result from the operation of executing task, and assign the result to the final task ‘Reply’ which will return a SOAP message which contains all task results to the client.
Figure 4.24  Scenario 1 workflow
Figure 4.25 shows the workflow which is implemented for scenario 2 based on the description of previous section (see Section 4.2.2). The tasks – ‘SwitchOnTV’, ‘TurnOnBedroomLight’ and ‘TurnOnAlarm’ are executed concurrently. The task – ‘IncreaseVolume’ will be executed followed with the task – ‘SwitchOnTV’.

Figure 4.25  Scenario 2 workflow
4.3.7. Database

Two databases have been implemented in our prototype – the decodb and the sensordb. We describe the decodb first.

As mentioned previously, the context utility function is heavily depending on rules. As Figure 4.26 shows, the $A_{Ic}$ value ($AIC\_VALUE$: value of context influence to activity; see Chapter 3: Section 3.3.2) has been defined in the RULE table. Each rule belongs to its corresponding profile.

The workflow system may have multiple user profiles which have been stored; however, only one user profile is allowed to be activated in a time. Following are the description for each database table.

---

**Figure 4.26** decodb database tables relationship

---

79
**Activity table**

Table: **ACTIVITY**

An activity data consists of its unique identification, name and device ID which defines the device that is involved to this activity. The **GOAL_ID** defines which goal will be achieved by this activity. The last element is the **GA_VALUE** (the Goal Achieve Value) which represents the percentage of the goal achievement by this activity. For example, the activity of “turn on the bathroom light” can achieve the goal of “brighten bathroom” by 100% which leads to the **GA_VALUE** as 1. The activity of “turn on corridor light” can achieve the goal of “brighten bathroom” by 50% which leads to the **GA_VALUE** as 0.5.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT_ID</td>
<td>Character</td>
<td>True</td>
<td>Activity ID</td>
</tr>
<tr>
<td>ACT_NAME</td>
<td>Character</td>
<td></td>
<td>Activity name</td>
</tr>
<tr>
<td>GOAL_ID</td>
<td>Character</td>
<td></td>
<td>Goal ID</td>
</tr>
<tr>
<td>GA_VALUE</td>
<td>Double</td>
<td></td>
<td>Goal achievement value</td>
</tr>
<tr>
<td>DE_ID</td>
<td>Character</td>
<td></td>
<td>Device element ID</td>
</tr>
<tr>
<td>DE_VALUE</td>
<td>Double</td>
<td></td>
<td>Device element value</td>
</tr>
</tbody>
</table>

Table 4.1  Activity table

**Context table**

Table: **CONTEXT**

A context data simply contains its unique identification, name and the context category it belongs to.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTEXT_ID</td>
<td>Character</td>
<td>True</td>
<td>Context ID</td>
</tr>
<tr>
<td>CONTEXT_NAME</td>
<td>Character</td>
<td></td>
<td>Context name</td>
</tr>
<tr>
<td>CC_ID</td>
<td>Character</td>
<td></td>
<td>Context category ID</td>
</tr>
</tbody>
</table>

Table 4.2  Context table

**Context Category table**

Table: **CONTEXT_CATEGORY**

Referring to Chapter 3: Section 3.3.1, context information can be categorised into various types and hence a general context category table is needed. A supertype context category may contain many subtype context categories. Therefore, when a context category is a subtype context category to another context category, it will contain its supertype context category ID.
As Chapter 3: Section 3.3.1 described, a context may have multiple attributes. For example, a location context has brightness value, temperature value and noise value. However, the weather context may only have one value which is the current weather (e.g. sunny). Therefore, we propose to separate the attributes of context to an individual table. The context element data contains its unique identification, name, value and the context it belongs to.

Context element values are defined in a standard way. The common values are 0 and 1 which representing true or false. For example, to define current weather, there will be three records stored in context element:
- Current weather is sunny: 0;
- Current weather is cloudy: 1;
- Current weather is raining: 0;

The above detect respectively that the current weather is not sunny, the current weather is cloudy and the current weather is not raining. This example illustrates the simplest definition of context values. However, it is not applicable to any kind of context. For example, to define the temperature context, negative numbers will be involved. We propose the following standards for defining context values in Decoflow. Note that we are unable to cover all types of context in the entire world but the table below provides a guideline which is applicable to the current Decoflow system.

Table 4.3  Context category table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC_ID</td>
<td>Character</td>
<td>True</td>
<td>Context category ID</td>
</tr>
<tr>
<td>CC_NAME</td>
<td>Character</td>
<td></td>
<td>Context category name</td>
</tr>
<tr>
<td>SUPERTYPE_ID</td>
<td>Character</td>
<td></td>
<td>Supertype category ID</td>
</tr>
</tbody>
</table>

Table 4.4  Context element table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE_ID</td>
<td>Character</td>
<td>True</td>
<td>Context element ID</td>
</tr>
<tr>
<td>CE_NAME</td>
<td>Character</td>
<td></td>
<td>Context element name</td>
</tr>
<tr>
<td>CE_VALUE</td>
<td>Double</td>
<td></td>
<td>Context element value</td>
</tr>
<tr>
<td>CONTEXT_ID</td>
<td>Character</td>
<td></td>
<td>Context ID</td>
</tr>
</tbody>
</table>

Context Element table

Table: CONTEXT_ELEMENT

As Chapter 3: Section 3.3.1 described, a context may have multiple attributes. For example, a location context has brightness value, temperature value and noise value. However, the weather context may only have one value which is the current weather (e.g. sunny). Therefore, we propose to separate the attributes of context to an individual table. The context element data contains its unique identification, name, value and the context it belongs to.

Context element values are defined in a standard way. The common values are 0 and 1 which representing true or false. For example, to define current weather, there will be three records stored in context element:
- Current weather is sunny: 0;
- Current weather is cloudy: 1;
- Current weather is raining: 0;

The above detect respectively that the current weather is not sunny, the current weather is cloudy and the current weather is not raining. This example illustrates the simplest definition of context values. However, it is not applicable to any kind of context. For example, to define the temperature context, negative numbers will be involved. We propose the following standards for defining context values in Decoflow. Note that we are unable to cover all types of context in the entire world but the table below provides a guideline which is applicable to the current Decoflow system.

Table 4.3  Context category table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC_ID</td>
<td>Character</td>
<td>True</td>
<td>Context category ID</td>
</tr>
<tr>
<td>CC_NAME</td>
<td>Character</td>
<td></td>
<td>Context category name</td>
</tr>
<tr>
<td>SUPERTYPE_ID</td>
<td>Character</td>
<td></td>
<td>Supertype category ID</td>
</tr>
</tbody>
</table>

Table 4.4  Context element table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE_ID</td>
<td>Character</td>
<td>True</td>
<td>Context element ID</td>
</tr>
<tr>
<td>CE_NAME</td>
<td>Character</td>
<td></td>
<td>Context element name</td>
</tr>
<tr>
<td>CE_VALUE</td>
<td>Double</td>
<td></td>
<td>Context element value</td>
</tr>
<tr>
<td>CONTEXT_ID</td>
<td>Character</td>
<td></td>
<td>Context ID</td>
</tr>
</tbody>
</table>
### Context Table

<table>
<thead>
<tr>
<th>Context</th>
<th>Valid Context Element Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0 to 23</td>
<td>24 hours</td>
</tr>
</tbody>
</table>
| Current weather is ?     | 0 or 1                       | True or False
|                          | ? = sunny or cloudy or raining|                                                 |
| Temperature              | -100 to 100                  | Minimum -100 degree
|                          |                               | Maximum 100 degree                               |
| Brightness               | 0 to 1                       | 0 = non brightness
|                          |                               | 0.5 = 50% brightness                             |
|                          |                               | 1 = 100% (maximum) brightness                    |
| Device power             | 0 or 1                       | 0 = device is deactivated
|                          |                               | 1 = device is activated                          |
| TV volume                | 0 to 1                       | 0 = non volume
|                          |                               | 0.5 = 50% volume                                 |
|                          |                               | 1 = 100% (maximum) volume                        |
| TV channel               | 0 to 999                     | Value represents the channel number              |

Table 4.5  Proposed valid context element values

### Device table, Device Element Type table and Device Element table

The objective to have a group of device data (device, device element type and device element) is to refer the concept of ‘device-oriented’ in Kurniati’s work [5]. A device may have various child-devices. For example, the device ‘television’ has child-device ‘channel tuner’ and ‘volume tuner’.

Each device may have various entities and each entity should contain a value. For example, the device ‘television’ has two entities which are ‘power’ and ‘status’, the ‘power’ represents the television is on or off and the status represents the television is capable to be activated or not. Hence, we have the Device Element Type data to define the entity type within the Device Element data.

**Table: DEVICE**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE_ID</td>
<td>Character</td>
<td>True</td>
<td>Device ID</td>
</tr>
<tr>
<td>DEVICE_NAME</td>
<td>Character</td>
<td></td>
<td>Device name</td>
</tr>
<tr>
<td>SDEVICE_ID</td>
<td>Character</td>
<td></td>
<td>Supertype/parent device ID</td>
</tr>
</tbody>
</table>
Table 4.6  Device table

Table: DEVICE_ELEMENT

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE_ID</td>
<td>Character</td>
<td>True</td>
<td>Device element ID</td>
</tr>
<tr>
<td>DE_NAME</td>
<td>Character</td>
<td></td>
<td>Device element name</td>
</tr>
<tr>
<td>DEVICE_ID</td>
<td>Character</td>
<td></td>
<td>Device ID</td>
</tr>
<tr>
<td>DET_ID</td>
<td>Character</td>
<td></td>
<td>Device element type ID</td>
</tr>
<tr>
<td>DE_VALUE</td>
<td>Double</td>
<td></td>
<td>Device element value</td>
</tr>
</tbody>
</table>

Table 4.7  Device element table

Table: DEVICE_ELEMENT_TYPE

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DET_ID</td>
<td>Character</td>
<td>True</td>
<td>Device element type ID</td>
</tr>
<tr>
<td>DET_NAME</td>
<td>Character</td>
<td></td>
<td>Device element type name</td>
</tr>
</tbody>
</table>

Table 4.8  Device element type table

Goal table

Table: GOAL

The definition of a goal has been described in Chapter 3: Section 3.3.2. The goal table simply contains its unique identification number and its name.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOAL_ID</td>
<td>Character</td>
<td>True</td>
<td>Goal ID</td>
</tr>
<tr>
<td>GOAL_NAME</td>
<td>Character</td>
<td></td>
<td>Goal name</td>
</tr>
</tbody>
</table>

Table 4.9  Goal table

Interpreter Rule table and Interpreter Rule Element table

Referring to Chapter 3: Section 3.4.3.3 in Context manager, the Interpreter is capable of defining a high-level context information from a collection of low-level context data. To realise such feature in our prototype, the Interpreter Rule table (INTERPRETER_RULE, see Table 4.10) and its associated table – Interpreter Rule Element (IR_ELEMENT, see Table 4.11) table have been introduced. Each Interpreter rule data contains its ID, the profile it belongs and the information of its target context data which will be updated when all Interpreter Rule Elements are matched. Note that the low-level context data is retrieved by context widgets (see Chapter 3: Section
Table: **INTERPRETER_RULE**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERPRETER_ID</td>
<td>Character</td>
<td>True</td>
<td>Interpreter rule ID</td>
</tr>
<tr>
<td>UPDATE_CE_ID</td>
<td>Character</td>
<td></td>
<td>The ID of the context element which will be updated by this Interpreter rule</td>
</tr>
<tr>
<td>UPDATE_CE_VALUE</td>
<td>Double</td>
<td></td>
<td>The value of the context element which will be updated by this Interpreter rule</td>
</tr>
<tr>
<td>PROFILE_ID</td>
<td>Character</td>
<td></td>
<td>Profile ID</td>
</tr>
</tbody>
</table>

Table 4.10  Interpreter rule table

Table: **IR_ELEMENT**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRE_ID</td>
<td>Character</td>
<td>True</td>
<td>Interpreter rule element</td>
</tr>
<tr>
<td>INTERPRETER_ID</td>
<td>Character</td>
<td></td>
<td>Interpreter rule ID</td>
</tr>
<tr>
<td>MATCH_CE_ID</td>
<td>Character</td>
<td></td>
<td>The ID of the context element which this data needs to be matched</td>
</tr>
<tr>
<td>MATCH_CE_VALUE</td>
<td>Double</td>
<td></td>
<td>The value of the context element which this data needs to be matched</td>
</tr>
</tbody>
</table>

Table 4.11  Interpreter rule element table

Below is an example to show how it works.

Example:

**Objective:** Define user’s current activity as sleeping.

High-level context data in Context Element table which defines whether the user’s current activity is sleeping or not:

<table>
<thead>
<tr>
<th>ContextElementID</th>
<th>ContextElementName</th>
<th>ContextElementValue</th>
<th>ContextID</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERSLEEP</td>
<td>User is sleeping</td>
<td>0</td>
<td>USERACT</td>
</tr>
</tbody>
</table>

Table 4.12  Sample high-level context element data
Low-level context data in Context Element table:

<table>
<thead>
<tr>
<th>ContextElementID</th>
<th>ContextElementName</th>
<th>ContextElementValue</th>
<th>ContextID</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERONBED</td>
<td>User is on the bed</td>
<td>1</td>
<td>USERONT</td>
</tr>
</tbody>
</table>

Table 4.13  Sample low-level context element data 1

<table>
<thead>
<tr>
<th>ContextElementID</th>
<th>ContextElementName</th>
<th>ContextElementValue</th>
<th>ContextID</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERMOVIN</td>
<td>User is moving</td>
<td>0</td>
<td>USERMOV</td>
</tr>
</tbody>
</table>

Table 4.14  Sample low-level context element data 2

Data in Interpreter Rule Element table:

<table>
<thead>
<tr>
<th>IReID</th>
<th>InterpreterRuleID</th>
<th>ContextElementID</th>
<th>ContextElementValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRE001</td>
<td>IR001</td>
<td>USERONBED</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.15  Sample Interpreter Rule Element data 1

<table>
<thead>
<tr>
<th>IReID</th>
<th>InterpreterRuleID</th>
<th>ContextElementID</th>
<th>ContextElementValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRE002</td>
<td>IR001</td>
<td>USERMOVE</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.16  Sample Interpreter Rule Element data 2

Data in Interpreter Rule table:

<table>
<thead>
<tr>
<th>InterpreterRuleID</th>
<th>ProfileID</th>
<th>ContextElementID</th>
<th>ContextElementValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR001</td>
<td>Prof001</td>
<td>USERSLEEP</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.17  Sample Interpreter Rule data

Hence, when the interpreter is activated in the system, the value of Context Element ‘User is sleeping’ will be updated as ‘1’ which represents true.

Profile table

Table: PROFILE

Since a Decoflow system can have multiple users (although the system can only activates one user’s profile at a time), and each user may have multiple profile to define different collections of rule to fulfil different needs (for example, a collection of rule was defined to be suit winter may not be the suitable rules for summer), the
profile table contains three elements – its unique identification, the corresponding user ID of the Decoflow system user and the CURRENT_LOAD value which defines whether the profile is currently loaded or not.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROF_ID</td>
<td>Character</td>
<td>True</td>
<td>Profile ID</td>
</tr>
<tr>
<td>DECOUSER_ID</td>
<td>Character</td>
<td></td>
<td>Decoflow user ID</td>
</tr>
<tr>
<td>CURRENT_LOAD</td>
<td>Integer</td>
<td></td>
<td>Is this profile the current activated profile?</td>
</tr>
</tbody>
</table>

Table 4.18  Profile table

**Rule table**

Table: RULE

When the context utility function needs to be performed, the system will retrieve the AI<sub>c</sub> value from this table. Ideally, all relevant data should be included in this table, such as the PROF_ID (that identifies which profile this rule belongs to), activity ID (ACT_ID), Context Element ID (CE_ID) and the Context Element value (CE_VALUE).

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULE_ID</td>
<td>Character</td>
<td>True</td>
<td>Rule ID</td>
</tr>
<tr>
<td>PROFILE_ID</td>
<td>Character</td>
<td></td>
<td>Profile ID</td>
</tr>
<tr>
<td>ACT_ID</td>
<td>Character</td>
<td></td>
<td>Activity ID</td>
</tr>
<tr>
<td>CE_ID</td>
<td>Character</td>
<td></td>
<td>Context element ID</td>
</tr>
<tr>
<td>CE_VALUE</td>
<td>Double</td>
<td></td>
<td>Context element value</td>
</tr>
<tr>
<td>AIC_VALUE</td>
<td>Double</td>
<td></td>
<td>The value of activity influenced by context</td>
</tr>
</tbody>
</table>

Table 4.19  Rule table

Below we explain how a rule can be defined by using an example.

Example:

Objective: activity ‘open bedroom window’ is very unacceptable when current weather context is raining.

ActivityID: ‘ACT032’ = ‘open bedroom window’

ContextElementID: ‘CE00032’ = ‘current weather is raining’

ContextElementValue = ‘1’ (represents the Context Element is true)

The rule data will be:

<table>
<thead>
<tr>
<th>RuleID</th>
<th>ProfileID</th>
<th>ActivityID</th>
<th>ContextElementID</th>
<th>ContextElementValue</th>
<th>AIC&lt;sub&gt;c&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0001</td>
<td>P001</td>
<td>ACT032</td>
<td>CE00032</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Sensor data
Following are the tables of sensordb. Each table represents a type of environment context data which retrieved by sensors. Each data below is corresponding to a Context Element data. The type is used to define which attribute the data is representing. For example, a LOCATION_ELEMENT data has ‘LOBRIGH’ as its LE_TYPE and ‘CORRIDO’ as its LOCATION_ID. When a Location Widget (ID = ‘CORRIDO’) retrieves this data, the Widget is capable of realising the data is representing the value of its brightness.

Table: DAYTIME_ELEMENT

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTE_ID</td>
<td>Character</td>
<td>True</td>
<td>Daytime element ID</td>
</tr>
<tr>
<td>DTE_VALUE</td>
<td>Double</td>
<td></td>
<td>Daytime element value</td>
</tr>
<tr>
<td>DTE_TYPE</td>
<td>Character</td>
<td></td>
<td>Daytime element type (e.g. ‘CURRENT’ = current daytime)</td>
</tr>
<tr>
<td>DAYTIME_ID</td>
<td>Character</td>
<td></td>
<td>Daytime ID</td>
</tr>
</tbody>
</table>

Table 4.21  Daytime element table

Table: LOCATION_ELEMENT

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE_ID</td>
<td>Character</td>
<td>True</td>
<td>Location element ID</td>
</tr>
<tr>
<td>LE_VALUE</td>
<td>Double</td>
<td></td>
<td>Location element value</td>
</tr>
<tr>
<td>LE_TYPE</td>
<td>Character</td>
<td></td>
<td>Location element type (e.g. ‘LOBRIGH’ = brightness, ‘LOTEMPE’ = temperature, ‘LONOISE’ = noise)</td>
</tr>
<tr>
<td>LOCATION_ID</td>
<td>Character</td>
<td></td>
<td>Location ID</td>
</tr>
</tbody>
</table>

Table 4.22  Location element table
### Table: USERACTION_ELEMENT

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAE_ID</td>
<td>Character</td>
<td>True</td>
<td>User action element ID</td>
</tr>
<tr>
<td>UAE_VALUE</td>
<td>Double</td>
<td></td>
<td>User action element value</td>
</tr>
<tr>
<td>UAE_TYPE</td>
<td>Character</td>
<td></td>
<td>User action element type (e.g. ‘ISMOVIN’ = is moving)</td>
</tr>
<tr>
<td>USERAVTION_ID</td>
<td>Character</td>
<td></td>
<td>User action ID</td>
</tr>
</tbody>
</table>

Table 4.23  User action element table

### Table: WEATHER_ELEMENT

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Primary Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WE_ID</td>
<td>Character</td>
<td>True</td>
<td>Weather element ID</td>
</tr>
<tr>
<td>WE_VALUE</td>
<td>Double</td>
<td></td>
<td>Weather element value</td>
</tr>
<tr>
<td>WE_TYPE</td>
<td>Character</td>
<td></td>
<td>Weather element type (e.g. ‘CURRENT’ = current weather)</td>
</tr>
<tr>
<td>WEATHER_ID</td>
<td>Character</td>
<td></td>
<td>Weather ID</td>
</tr>
</tbody>
</table>

Table 4.24  Weather element table

The sensor data in our prototype is only for testing. In reality, the structure of the sensor data can be very different.

### 4.4. Summary

In this chapter, we have implemented a prototype to simulate our proposed framework. The prototype includes the BPEL workflow module implemented in Netbeans IDE v6.0 beta 1, the Adaptive Device Manager Web service, Substitution Manager, the Context Manager component group and a number of web services for devices.
Chapter 5.

Evaluation

5.1. Overview

In this chapter, we evaluate our prototype based on the scenarios which have been described in the previous chapter (see Chapter 4: Section 4.2.2).

In order to test the system effectively, we have implemented two types of testing. The first type is the overall system testing which focuses on the workflow execution for each scenario. The second type focuses on the individual component testing which includes the primary function testing of the Substitution Manager. Note that in the following scenarios, the bedroom will be identified with a number (e.g. bedroom3) due to we assumed there are more than one bedroom in the house.

We used same scenario and test cases for the overall testings and the individual component testings. Below, we defined the context used in the test cases.

Scenario 1 – Test Case 1:
Current weather is raining;
Current time of day is morning;
Bedroom3 light is on;
Bedroom3 brightness is 100%;
Bedroom3 curtain is lowered;

Please refer to the user guide in Appendix A for the installation and testing of the prototype.
Corridor light is off;
Bathroom light status is failed;

Scenario 1 – Test Case 2:
Current weather is cloudy;
Current time of day is morning;
Bedroom3 light is off;
Bedroom3 brightness is 50%;
Bedroom3 curtain is raised;
Corridor light is off;
Bathroom light status is failed;

Scenario 2 – Test Case 1:
User is sleeping;
Bedroom3 TV is on;
Bedroom3 TV volume is 8;
Mobile phone 1 status is ok;
Mobile phone 1 is on;
Mobile phone 1 alarm status is ok;
Alarm clock status is failed;

Scenario 2 – Test Case 2:
User is sleeping;
Bedroom3 TV is on;
Bedroom3 TV volume is 8;
Mobile phone 1 status is failed;
Mobile phone 1 is off;
Mobile phone 1 alarm status is failed;
Alarm clock status is failed;

Scenario 2 – Test Case 3:
User is awake;
Bedroom3 TV is on;
Bedroom3 TV volume is 8;
Mobile phone 1 status is failed;
Mobile phone 1 is off;
Mobile phone 1 alarm status is failed;
Alarm clock status is failed;
5.2. Overall Testing of Scenario 1

In the overall testing of scenario 1, we have defined numerous common data for goal, activities and rules. There are two test cases have been implemented for scenario 1 and each of them represents a sub-scenario with its specific context data. Following are the common data of scenario 1:

In scenario 1, the failed task is the ‘turn on bathroom light’. Hence, we have defined the following data to test the recovery algorithm.

**Goal:**

<table>
<thead>
<tr>
<th>GOAL_ID</th>
<th>GOAL_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIBGTH</td>
<td>Brighten bathroom</td>
</tr>
</tbody>
</table>

Table 5.1 Goal data used in scenario 1

**Activities:**

<table>
<thead>
<tr>
<th>ACT_ID</th>
<th>ACT_NAME</th>
<th>GOAL_ID</th>
<th>GA_VALUE</th>
<th>DE_ID</th>
<th>DE_VALUE</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTHLION</td>
<td>turn on bathroom light</td>
<td>BRIBGTH</td>
<td>1</td>
<td>BTHLONF</td>
<td>1</td>
<td><strong>BTHLONF</strong> = bathroom light on or off</td>
</tr>
<tr>
<td>BR3LION</td>
<td>turn on bedroom3 light</td>
<td>BRIBGTH</td>
<td>0.5</td>
<td>BR3LONF</td>
<td>1</td>
<td><strong>BR3LONF</strong> = bedroom3 light on or off</td>
</tr>
<tr>
<td>CORLION</td>
<td>turn on corridor light</td>
<td>BRIBGTH</td>
<td>0.5</td>
<td>CORLONF</td>
<td>1</td>
<td><strong>CORLONF</strong> = corridor light on or off</td>
</tr>
</tbody>
</table>

Table 5.2 Activity data used in scenario 1

**Rules:**

<table>
<thead>
<tr>
<th>RULE_ID</th>
<th>PROF_ID</th>
<th>ACT_ID</th>
<th>CE_ID</th>
<th>CE_VALUE</th>
<th>AIC_VALUE</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULE040</td>
<td>PROF001</td>
<td>BTHLION</td>
<td>BTHLIST</td>
<td>0.0</td>
<td>-999.0</td>
<td><strong>BTHLION</strong> is invalid when the status of bathroom light is ‘failed’</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>BTHLIST</strong> = bathroom light status</td>
</tr>
<tr>
<td>RULE041</td>
<td>PROF001</td>
<td>BTHLION</td>
<td>BTHLIST</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>RULE042</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>BTHLIST</td>
<td>1.0</td>
<td>-999.0</td>
<td><strong>BR3LION</strong> is invalid when the status of bedroom3 light is ‘failed’</td>
</tr>
</tbody>
</table>


There are two test cases in scenario 1; both of them are using the same input message (see Figure 5.1). The input message defines the user ID as ‘USER001’, profile ID as ‘PROF001’ and three activity IDs have been defined which represents ‘turn on bedroom3 heater’ (BR3HEON), ‘turn on bedroom3 light’ (BR3LION) and ‘turn on bathroom light’ (BTHLION).

**Table 5.3** Rule data used in scenario 1

<table>
<thead>
<tr>
<th>Rule</th>
<th>Prof ID</th>
<th>Act1 ID</th>
<th>Act2 ID</th>
<th>Act3 ID</th>
<th>Act4 ID</th>
<th>Act5 ID</th>
<th>Act6 ID</th>
<th>Act7 ID</th>
<th>Act8 ID</th>
<th>Act9 ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULE043</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>BTHLIST</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE044</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>CUWTHSU</td>
<td>1.0</td>
<td>-1.0</td>
<td>CUWTHSU</td>
<td>current weather is sunny</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE045</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>CUWTHCL</td>
<td>1.0</td>
<td>0.0</td>
<td>CUWTHCL</td>
<td>current weather is cloudy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE046</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>CUWTHRA</td>
<td>1.0</td>
<td>1.0</td>
<td>CUWTHRA</td>
<td>current weather is raining</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE047</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>CUWTHSU</td>
<td>1.0</td>
<td>-1.0</td>
<td>CUWTHSU</td>
<td>current weather is sunny</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE048</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>CUWTHCL</td>
<td>1.0</td>
<td>0.0</td>
<td>CUWTHCL</td>
<td>current weather is cloudy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE049</td>
<td>PROF001</td>
<td>BR3LION</td>
<td>CUWTHRA</td>
<td>1.0</td>
<td>1.0</td>
<td>CUWTHRA</td>
<td>current weather is raining</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE050</td>
<td>PROF001</td>
<td>CORLION</td>
<td>BTHLIST</td>
<td>1.0</td>
<td>-999.0</td>
<td>CORLION</td>
<td>is invalid when the status bathroom light is ‘ok’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE051</td>
<td>PROF001</td>
<td>CORLION</td>
<td>BTHLIST</td>
<td>0.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE052</td>
<td>PROF001</td>
<td>CORLION</td>
<td>BR3BRIG</td>
<td>1.0</td>
<td>-1.0</td>
<td>BR3BRIG</td>
<td>bedroom3 brightness</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE053</td>
<td>PROF001</td>
<td>CORLION</td>
<td>CUWTHSU</td>
<td>1.0</td>
<td>-0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE054</td>
<td>PROF001</td>
<td>CORLION</td>
<td>CUWTHCL</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE055</td>
<td>PROF001</td>
<td>CORLION</td>
<td>CUWTHRA</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE056</td>
<td>PROF001</td>
<td>CORLION</td>
<td>CUWTHSU</td>
<td>1.0</td>
<td>-0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE057</td>
<td>PROF001</td>
<td>CORLION</td>
<td>CUWTHCL</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE058</td>
<td>PROF001</td>
<td>CORLION</td>
<td>CUWTHRA</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE059</td>
<td>PROF001</td>
<td>CORLION</td>
<td>BR3BRIG</td>
<td>0.5</td>
<td>-0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Input of Scenario 1:**

```xml
<soapenv:Envelope
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/"
```
Figure 5.1 The input message of scenario 1

**Test Case 1**

Output:

```xml
<xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
 http://schemas.xmlsoap.org/soap/envelope/"
>
 <SOAP-ENV:Body>
 <responseElement xmlns="http://xml.netbeans.org/schema/Scenario1">
  <ns0:task1result
   xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
   xmlns:ns0="http://xml.netbeans.org/schema/Scenario1"
   xmlns:ns2="http://adaptivedevicemanager.deco.edu/">
   Activity 'BR3HEON: turn on bedroom3 heater' completed.</ns0:task1result>
  <ns0:task2result
   xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
   xmlns:ns0="http://xml.netbeans.org/schema/Scenario1"
   xmlns:ns2="http://adaptivedevicemanager.deco.edu/">
   Activity 'BR3LION: turn on bedroom3 light' completed.</ns0:task2result>
 </responseElement>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Device failure occurred. Recovery result is: The substitute activity: 'turn on bedroom3 light' has been executed already.

Figure 5.2  Output of the test case 1 of scenario 1

Figure 5.2 shows the output of the test case 1. The first two activities have been executed successfully. The third activity was the failed activity – ‘turn on bathroom light’ which has been replaced by a substitute activity – ‘turn on bedroom3 light’. Since the bedroom3 light was activated already, the output simply notified the substitute activity has been executed.

Test Case 2
Output:

<?xml version="1.0" encoding="UTF-8"?>
  <SOAP-ENV:Body>
    <responseElement xmlns="http://xml.netbeans.org/schema/Scenario1">
      <ns0:task1result xmlns:msgns="http://adaptive-devicemanager.deco.edu/"
xmlns:ns0="http://xml.netbeans.org/schema/Scenario1"
xmlns:ns2="http://adaptive-devicemanager.deco.edu/">Activity 'BR3HEON: turn on bedroom3 heater' completed.</ns0:task1result>
      <ns0:task2result xmlns:msgns="http://adaptive-devicemanager.deco.edu/"
xmlns:ns0="http://xml.netbeans.org/schema/Scenario1"
xmlns:ns2="http://adaptive-devicemanager.deco.edu/">Activity 'BR3LION: turn on bedroom3 light' completed.</ns0:task2result>
    </responseElement>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Device failure occurs. Substitutive activity 'turn on corridor light' executed.

Figure 5.3 Output of the test case 2 of scenario 1

Figure 5.3 shows the output of the test case 2. It is similar as the result of the test case 1 but the substitute activity for the third activity is ‘turn on corridor light’.

5.3. Overall Testing of Scenario 2

For the testing of scenario 2, we have implemented three test cases. We first defined following common data for goal, activities and rules:

Goal:

<table>
<thead>
<tr>
<th>GOAL_ID</th>
<th>GOAL_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAKUPUS</td>
<td>waking up the user</td>
</tr>
</tbody>
</table>

Table 5.4 Goal data used in scenario 2

Activities:

<table>
<thead>
<tr>
<th>ACT_ID</th>
<th>ACT_NAME</th>
<th>GOAL_ID</th>
<th>GA_VALUE</th>
<th>DE_ID</th>
<th>DE_VALUE</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALARMON</td>
<td>turn on alarm sound</td>
<td>WAKUPUS</td>
<td>1.0</td>
<td>ALCSONF</td>
<td>1.0</td>
<td>ALCSONF = Alarm clock sound on or off</td>
</tr>
<tr>
<td>MP1ALON</td>
<td>turn on mobilephone1 alarm</td>
<td>WAKUPUS</td>
<td>1.0</td>
<td>MP1AONF</td>
<td>1.0</td>
<td>MP1AONF = mobile phone 1 on or off</td>
</tr>
<tr>
<td>B3TVV28</td>
<td>bedroom3 TV volume raise up to 28</td>
<td>WAKUPUS</td>
<td>1.0</td>
<td>R3TVVVA</td>
<td>28.0</td>
<td>R3TVVVA = bedroom3 TV volume value</td>
</tr>
</tbody>
</table>

Table 5.5 Activity data used in scenario 2
Rules:

<table>
<thead>
<tr>
<th>RULE_ID</th>
<th>PROF_ID</th>
<th>ACT_ID</th>
<th>CE_ID</th>
<th>CE_VALUE</th>
<th>AIC_VALUE</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULE001</td>
<td>PROF001</td>
<td>ALARMON</td>
<td>USERSLE</td>
<td>1.0</td>
<td>1.0</td>
<td>USERSLE = user is sleeping</td>
</tr>
<tr>
<td>RULE002</td>
<td>PROF001</td>
<td>ALARMON</td>
<td>USERSLE</td>
<td>0.0</td>
<td>-999.0</td>
<td>USERSLE = user is sleeping</td>
</tr>
<tr>
<td>RULE003</td>
<td>PROF001</td>
<td>ALARMON</td>
<td>ALCLSST</td>
<td>0.0</td>
<td>-999.0</td>
<td>ALCLSST = alarm clock sound status</td>
</tr>
<tr>
<td>RULE004</td>
<td>PROF001</td>
<td>ALARMON</td>
<td>ALCLSST</td>
<td>1.0</td>
<td>1.0</td>
<td>ALCLSST = alarm clock sound status</td>
</tr>
<tr>
<td>RULE005</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>USERSLE</td>
<td>1.0</td>
<td>1.0</td>
<td>USERSLE = user is sleeping</td>
</tr>
<tr>
<td>RULE006</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>USERSLE</td>
<td>0.0</td>
<td>-999.0</td>
<td>USERSLE = user is sleeping</td>
</tr>
<tr>
<td>RULE007</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>ALCLSST</td>
<td>1.0</td>
<td>-999.0</td>
<td>ALCLSST = alarm clock sound status</td>
</tr>
<tr>
<td>RULE008</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>ALCLSST</td>
<td>0.0</td>
<td>1.0</td>
<td>ALCLSST = alarm clock sound status</td>
</tr>
<tr>
<td>RULE009</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>MP1ALST</td>
<td>0.0</td>
<td>-999.0</td>
<td>MP1ALST = mobile phone 1 alarm status</td>
</tr>
<tr>
<td>RULE010</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>MP1ALST</td>
<td>1.0</td>
<td>1.0</td>
<td>MP1ALST = mobile phone 1 alarm status</td>
</tr>
<tr>
<td>RULE011</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>MPHO1ST</td>
<td>0.0</td>
<td>-999.0</td>
<td>MP1ALST = mobile phone 1 status</td>
</tr>
<tr>
<td>RULE012</td>
<td>PROF001</td>
<td>MP1ALON</td>
<td>MPHO1ST</td>
<td>1.0</td>
<td>1.0</td>
<td>MP1ALST = mobile phone 1 status</td>
</tr>
<tr>
<td>RULE013</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>USERSLE</td>
<td>1.0</td>
<td>1.0</td>
<td>USERSLE = user is sleeping</td>
</tr>
<tr>
<td>RULE014</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>USERSLE</td>
<td>0.0</td>
<td>-999.0</td>
<td>USERSLE = user is sleeping</td>
</tr>
<tr>
<td>RULE015</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>ALCLSST</td>
<td>1.0</td>
<td>-999.0</td>
<td>ALCLSST = alarm clock sound status</td>
</tr>
<tr>
<td>RULE016</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>ALCLSST</td>
<td>0.0</td>
<td>1.0</td>
<td>ALCLSST = alarm clock sound status</td>
</tr>
<tr>
<td>RULE017</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>R3TVVST</td>
<td>0.0</td>
<td>-999.0</td>
<td>R3TVVST = bedroom3 TV volume tuner status</td>
</tr>
<tr>
<td>RULE018</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>R3TVVST</td>
<td>1.0</td>
<td>1.0</td>
<td>R3TVVST = bedroom3 TV volume tuner status</td>
</tr>
<tr>
<td>RULE019</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>BR3TVST</td>
<td>0.0</td>
<td>-999.0</td>
<td>BR3TVST = bedroom3 TV status</td>
</tr>
<tr>
<td>RULE020</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>BR3TVST</td>
<td>1.0</td>
<td>1.0</td>
<td>BR3TVST = bedroom3 TV status</td>
</tr>
<tr>
<td>RULE021</td>
<td>PROF001</td>
<td>B3TVV28</td>
<td>MPHO1ST</td>
<td>1.0</td>
<td>-1.0</td>
<td>MPHO1ST = mobile phone 1</td>
</tr>
</tbody>
</table>
In scenario 2, the primary context influences that affect the testing results are user and device contexts. Figure 5.4 shows the input message of scenario 2 which defines the user ID is ‘USER001’, profile ID is ‘PROF001’ and four activity IDs which represent ‘turn on bedroom3 TV’ (BR3TVON), ‘increase bedroom3 TV volume to 12’ (R3TVVV12), ‘turn on bedroom3 light’ (BR3LION) and ‘turn on alarm’ (ALARMON).

**Input of Scenario 2:**

```xml
<soapenv:Envelope
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:scen="http://xml.netbeans.org/schema/Scenario2">
  <soapenv:Body>
    <scen:requestElement>
      <scen:decouserId>USER001</scen:decouserId>
      <scen:profileId>PROF001</scen:profileId>
      <scen:activityId1>BR3TVON</scen:activityId1>
      <scen:activityId2>R3TVVV12</scen:activityId2>
      <scen:activityId3>BR3LION</scen:activityId3>
      <scen:activityId4>ALARMON</scen:activityId4>
    </scen:requestElement>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 5.4  Input of scenario 2
Test Case 1
Output

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/envelope/"/>
<SOAP-ENV:Body>
    <responseElement xmlns="http://xml.netbeans.org/schema/Scenario2">
        <ns0:result1 xmlns:msgns="http://adaptivevicemanager.deco.edu/
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivevicemanager.deco.edu/">Activity 'BR3TVON:turn on bedroom3 TV' completed.</ns0:result1>
        <ns0:result2 xmlns:msgns="http://adaptivevicemanager.deco.edu/
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivevicemanager.deco.edu/">Activity 'R3TVV12:increase bedroom3 TV volume to 12' completed.</ns0:result2>
        <ns0:result3 xmlns:msgns="http://adaptivevicemanager.deco.edu/
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivevicemanager.deco.edu/">Activity 'BR3LION:turn on bedroom3 light' completed.</ns0:result3>
        <ns0:result4 xmlns:msgns="http://adaptivevicemanager.deco.edu/
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivevicemanager.deco.edu/">Device failure occurs. Substitutive activity 'turn on mobilephone1 alarm' executed.</ns0:result4>
    </responseElement>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.5 Output of the test case 1 of scenario 2

Figure 5.5 shows the output of the test case 1. The first three activities were executed successfully and the fourth activity has failed due to the device failure. The substitute activity – ‘turn on mobile phone 1 alarm’ has been found and replaced the failed activity.
Test Case 2

Output

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/"

  http://schemas.xmlsoap.org/soap/envelope/>

  <SOAP-ENV:Body>
  <responseElement
    xmlns="http://xml.netbeans.org/schema/Scenario2">
    <ns0:result1
      xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
      xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
      xmlns:ns2="http://adaptivedevicemanager.deco.edu/"
      Activity 'BR3TVON: turn on bedroom3 TV' completed.</ns0:result1>
    <ns0:result2
      xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
      xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
      xmlns:ns2="http://adaptivedevicemanager.deco.edu/"
      Activity 'R3TVV12: increase bedroom3 TV volume to 12' completed.</ns0:result2>
    <ns0:result3
      xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
      xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
      xmlns:ns2="http://adaptivedevicemanager.deco.edu/"
      Activity 'BR3LION: turn on bedroom3 light' completed.</ns0:result3>
    <ns0:result4
      xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
      xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
      xmlns:ns2="http://adaptivedevicemanager.deco.edu/"
      Device failure occurs. Substitutive activity 'bedroom3 TV volume raise up to 28'
      executed.</ns0:result4>
  </responseElement>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.6 Output of the test case 2 of scenario 2

Figure 5.6 shows the output of the test case 2. The failed activity was replaced by the substitute activity – ‘bedroom3 TV volume raise up to 28’. Note that the activity ‘bedroom3 TV volume raise up to 28’ will also switch on its parent device – ‘bedroom3 TV’ even the TV was switched off, as long as the TV is available.
Test Case 3

Output

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <responseElement xmlns="http://xml.netbeans.org/schema/Scenario2">
      <ns0:result1 xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivedevicemanager.deco.edu/">Activity 'BR3TVON:turn on bedroom3 TV' completed.</ns0:result1>
      <ns0:result2 xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivedevicemanager.deco.edu/">Activity 'R3TVV12:increase bedroom3 TV volume to 12' completed.</ns0:result2>
      <ns0:result3 xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivedevicemanager.deco.edu/">Activity 'BR3LION:turn on bedroom3 light' completed.</ns0:result3>
      <ns0:result4 xmlns:msgns="http://adaptivedevicemanager.deco.edu/"
xmlns:ns0="http://xml.netbeans.org/schema/Scenario2"
xmlns:ns2="http://adaptivedevicemanager.deco.edu/">Device failure occured. Recovery result is: No valid substitution.</ns0:result4>
    </responseElement>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.7 Output of the test case 3 of scenario 2

The output of the test case 3 shows that the fourth activity has failed but no valid substitute activity is available. This is because the user is already awake. The substitute activity is unnecessary.
5.4. The Individual Component Testing of Substitution Manager

The overall testings in previous section can only shows the request message from the client and the result output from the workflow execution. In order to evaluate the detail processes of recovery algorithm, we have to test component individually. Hence, we are able to track the processes.

The Substitution Manager is the component which is capable of finding a best substitute activity according to the activities and rules it received. In order to test its primary function – the `getSubstitutionResult()` method, we have implemented numerous test cases based on the scenario 1 and 2.

There are two common testing methods have been implemented for the scenario 1 – test case 1 and 2, and the scenario 2 – test case 1, 2 and 3. The context data used in each test case is referred to the test cases in last section.

For the first two test cases, we defined a collection of activity IDs and retrieved all rules that under the profile ID – ‘PROF001’. We passed these two collections as the parameters of the `getSubstitutionResult()` method.

```java
public static void main(String[] args) {
    ArrayList<ActivityData> acts = new ArrayList<ActivityData>();
    ArrayList<RuleData> rds = new ArrayList<RuleData>();
    DBManager dbm = new DBManager();
    try {
        acts.add(dbm.getActivityData("SRILICK"));
        acts.add(dbm.getActivityData("SRILICK"));
        acts.add(dbm.getActivityData("DORLICK"));
        rds = dbm.getRules("PROF001");
        SubstitutionManager sm = new SubstitutionManager("USER001", "PROF001");
        ActivityData ad = sm.getSubstitutionResult(rds, acts);
        if (ad != null) {
            System.out.println("Substitution result is null.");
            System.out.println("The reason is: " + ad.getDeNothingReason());
        } else {
            System.out.println("The result activity is:" + ad.getActivityId());
        }
    } catch (Exception e) {
        System.out.println(e.toString());
    }
}
```

Figure 5.8 Method of the individual component testing 1
Following are the outputs of the test cases for scenario 1.

Scenario 1 – Test Case 1:

Output:

---

Step 1: Eliminate invalid activities according to their corresponding devices.

Activity: BTHLION is invalid due to Device - BTHLIGH is unavailable.  
Activity: BR3LION is valid  
Activity: CORLION is valid  
**  
** Number of valid activities = 2 **
---

Step 2: Retrieve context data according to rules.

Add Context Element ID: BTHLIST to list.  
Add Context Element ID: CUWTHSU to list.  
Add Context Element ID: CUWTHCL to list.  
Add Context Element ID: CUWTHRA to list.  
Add Context Element ID: CUDATMO to list.  
Add Context Element ID: CUDATAF to list.  
Add Context Element ID: CUDATNI to list.  
Add Context Element ID: BR3BRIG to list.  
Add Context Element ID: BR3CONF to list.  
Add Context Element ID: USERSLE to list.  
Add Context Element ID: ALCLSST to list.  
Add Context Element ID: MP1ALST to list.  
Add Context Element ID: MPHO1ST to list.  
Add Context Element ID: R3TVVST to list.  
Add Context Element ID: BR3TVST to list.  
---

Step 3: Validate rules with Context Elements.

Rule: RULE040 is valid.  
Rule: RULE043 is valid.  
Rule: RULE046 is valid.  
Rule: RULE047 is valid.  
Rule: RULE051 is valid.  
Rule: RULE052 is valid.  
Rule: RULE055 is valid.  
Rule: RULE056 is valid.  
Rule: RULE060 is valid.  
Rule: RULE001 is valid.  
Rule: RULE003 is valid.  
Rule: RULE005 is valid.  
Rule: RULE008 is valid.  
Rule: RULE020 is valid.  
Rule: RULE022 is valid.  
Rule: RULE023 is valid.  
Rule: RULE026 is valid.  
Rule: RULE028 is valid.  
Rule: RULE030 is valid.  
Rule: RULE031 is valid.  
---

Step 4: Retrieve valid substitutive activities according to the valid rules.

Activity: BR3LION is valid.  
Activity: CORLION is valid.  
**  
** Number of valid activities after Step 4: 2 **
---

Step 5: Calculate the utility values for each substitutive activity.

*Calculation utility value for ActivityId: BR3LION  
GA value: 0.5  
Rule Id: RULE043  
Aic value: 1.0  
0.5 + 1.0  
Current Aic value for Activity: BR3LION is 1.5  
Rule Id: RULE046  
Aic value: 1.0  
1.5 + 1.0  
Current Aic value for Activity: BR3LION is 2.5  
Rule Id: RULE047  
Aic value: -1.0  
2.5 + -1.0  
Current Aic value for Activity: BR3LION is 1.5  
Rule Id: RULE060  
Aic value: -0.5  
1.5 + -0.5  
Current Aic value for Activity: BR3LION is 1.0  

*Calculation utility value for ActivityId: CORLION  
GA value: 0.5  
Rule Id: RULE051  
Aic value: 1.0  
0.5 + 1.0  
Current Aic value for Activity: CORLION is 1.5  
Rule Id: RULE052  
Aic value: -1.0  
1.5 + -1.0  


Current AIC value for Activity: CORLION is 0.5
Rule ID: RULE055
AIC value: 1.0
0.5 + 1.0
Current AIC value for Activity: CORLION is 1.5
Rule ID: RULE056
AIC value: -0.5
1.5 + -0.5
Current AIC value for Activity: CORLION is 1.0

Step 6: Compare the utility values of activities to define the best substitutive activity.

ActivityID: BR3LION; Utility Value: 1.0
ActivityID: BR3LION has the same utility value as the highest one.
ActivityID: CORLION; Utility Value: 1.0
ActivityID: CORLION has the same utility value as the highest one.

*Find best substitute activity:
DE_ID: BR3LION
*The activity BR3LION - 'turn on bedroom3 light' has been executed.
The result substitute activity is:'turn on bedroom3 light'.
Substitution result is null.
The reason is: The substitute activity:'turn on bedroom3 light' has been executed already.

Figure 5.9  Output of scenario 1 test case 1

Scenario 1 – Test Case 2:

Output:

Step 1: Eliminate invalid activities according to their corresponding devices.

Activity: BTHLION is invalid
due to Device - BTHLIGN is unavailable.
Activity: BR3LION is valid
Activity: CORLION is valid

** Number of valid activities = 2 **

Step 2: Retrieve context data according to rules.

Add Context Element ID: BTHLIST to list.
Add Context Element ID: CUWTHSU to list.
Add Context Element ID: CUWTHCL to list.
Add Context Element ID: CUWTHRA to list.
Add Context Element ID: CUDATMO to list.
Add Context Element ID: CUDATAF to list.
Add Context Element ID: CUDATNI to list.
Add Context Element ID: BR3BRIG to list.
Add Context Element ID: BR3CONF to list.
Add Context Element ID: USERSLE to list.
Add Context Element ID: ALCLSST to list.
Add Context Element ID: MP1ALST to list.
Add Context Element ID: MPHO1ST to list.
Add Context Element ID: R3TVVST to list.
Add Context Element ID: BR3TVST to list.

Step 3: Validate rules with Context Elements.

Rule: RULE040 is valid.
Rule: RULE043 is valid.
Rule: RULE045 is valid.
Rule: RULE047 is valid.
Rule: RULE051 is valid.
Rule: RULE054 is valid.
Rule: RULE059 is valid.
Rule: RULE060 is valid.
Rule: RULE001 is valid.
Rule: RULE003 is valid.
Rule: RULE005 is valid.
Rule: RULE008 is valid.
Rule: RULE020 is valid.
Rule: RULE022 is valid.
Rule: RULE023 is valid.
Rule: RULE026 is valid.
Rule: RULE028 is valid.
Rule: RULE030 is valid.
Rule: RULE031 is valid.

Step 4: Retrieve valid substitutive activities according to the valid rules.

Activity: BR3LION is valid.
Activity: CORLION is valid.

** Number of valid activities after Step 4: 2 **

Step 5: Calculate the utility values for each substitutive activity.

*Calculation utility value for ActivityID: BR3LION
GA value: 0.5
Rule ID: RULE043
AIC value: 1.0
0.5 + 1.0
Current AIC value for Activity: BR3LION is 1.5
Rule ID: RULE045
AIC value: 0.0
1.5 + 0.0
Current AIC value for Activity: BR3LION is 1.5
Rule ID: RULE047
AIC value: -1.0
1.5 + -1.0
Current AIC value for Activity: BR3LION is 0.5
Rule ID: RULE049
AIC value: -0.5
0.5 + -0.5
Current AIC value for Activity: BR3LION is 0.0

*Calculation utility value for ActivityId: CORLION
GA value: 0.5
Rule ID: RULE051
AIC value: 1.0
0.5 + 1.0
Current AIC value for Activity: CORLION is 1.5
Rule ID: RULE054
AIC value: 0.0
1.5 + 0.0
Current AIC value for Activity: CORLION is 2.5
Rule ID: RULE056
AIC value: -0.5
2.5 + -0.5
Current AIC value for Activity: CORLION is 2.0
Rule ID: RULE059
AIC value: -0.5
2.0 + -0.5
Current AIC value for Activity: CORLION is 1.5

Step 6: Compare the utility values of activities to define the best activity.

ActivityID: BR3LION; Utility Value: 0.0
ActivityID: BR3LION has the same utility value as the highest one.
ActivityID: CORLION; Utility Value: 1.5
ActivityID: CORLION has the highest utility value.

*Find best substitute activity.
*Current best substitute activity is: CORLION
DE_ID: CORLION
The best substitute activity can be: CORLION - 'turn on corridor light'
The final result of the best substitute activity is: CORLION - 'turn on corridor light'
The result activity is: CORLION

Figure 5.10 Output of scenario 1 test case 2

In the scenario 2 test cases, we also defined a collection of activities and a collection of rules (see Figure 5.11). The process is same as the test cases in last scenario.

```java
public static void main(String[] args) {
    ArrayList<ActivityData> acts = new ArrayList<ActivityData>();
    ArrayList<RuleData> rds = new ArrayList<RuleData>();
    ThManager dm = new ThManager();
    try {
        acts.add(dm.getActivityData("ALARMSW"));
        acts.add(dm.getActivityData("XP16L0W"));
        acts.add(dm.getActivityData("B3TUZ48"));
        rds = dm.getRules("PROFO01");
        SubstitutionManager sm = new SubstitutionManager("USERC01","PROFO01");
        ActivityData ad = sm.getSubstitutionResult(rds, acts);
        if(ad!=null) {
            System.out.println("Substitution result is null.");
            System.out.println("The reason is: "+sm.getBoothinReason());
            return;
            System.out.println("The result activity is:"+ad.getActivityId());
        }
    } catch (Exception e) {
        System.out.println(e.toString());
    }
}
```

Figure 5.11 Method of the individual component testing 2
Following are the outputs of the test cases in scenario 2.

### Scenario 2 – Test Case 1:

**Output:**

```
Step 1: Eliminate invalid activities according to their corresponding devices.

Activity: ALARMON is invalid due to Device - ALACLSO is unavailable.
Activity: MPIALON is valid
Activity: B3TVV28 is valid

** Number of valid activities = 2 **

Step 2: Retrieve context data according to rules.

- Add Context Element ID: BTHLIST to list.
- Add Context Element ID: CUWTHSU to list.
- Add Context Element ID: CUWTHCL to list.
- Add Context Element ID: CUWTHRA to list.
- Add Context Element ID: CUDATMO to list.
- Add Context Element ID: CUDATAF to list.
- Add Context Element ID: CUDATNI to list.
- Add Context Element ID: BR3BRIG to list.
- Add Context Element ID: BR3CONF to list.
- Add Context Element ID: USERSLE to list.
- Add Context Element ID: ALCLSST to list.
- Add Context Element ID: MP1ALST to list.
- Add Context Element ID: BR3TVST to list.

Step 3: Validate rules with Context Elements.

- Rule: RULE040 is valid.
- Rule: RULE043 is valid.
- Rule: RULE045 is valid.
- Rule: RULE047 is valid.
- Rule: RULE051 is valid.
- Rule: RULE054 is valid.
- Rule: RULE056 is valid.
- Rule: RULE059 is valid.
- Rule: RULE060 is valid.
- Rule: RULE001 is valid.
- Rule: RULE003 is valid.
- Rule: RULE005 is valid.
- Rule: RULE008 is valid.
- Rule: RULE020 is valid.
- Rule: RULE022 is valid.
- Rule: RULE023 is valid.
- Rule: RULE026 is valid.
- Rule: RULE028 is valid.
- Rule: RULE030 is valid.
- Rule: RULE031 is valid.

Step 4: Retrieve valid substitutive activities according to the valid rules.

- Activity: MPIALON is valid.
- Activity: B3TVV28 is valid.

** Number of valid activities after Step 4: 2 **

Step 5: Calculate the utility values for each substitutive activity.

\[
\text{GA value: 1.0} \\
\text{Rule Id: RULE005} \\
\text{AIC value: 1.0} \\
1.0 + 1.0 \\
\text{Current AIC value: 2.0} \\
\text{Rule Id: RULE008} \\
\text{AIC value: 1.0} \\
2.0 + 1.0 \\
\text{Current AIC value: 3.0} \\
\text{Rule Id: RULE010} \\
\text{AIC value: 1.0} \\
3.0 + 1.0 \\
\text{Current AIC value: 4.0} \\
\text{Rule Id: RULE022} \\
\text{AIC value: 1.0} \\
4.0 + 1.0 \\
\text{Current AIC value: 5.0}
\]

\[
\text{GA value: 1.0} \\
\text{Rule Id: RULE023} \\
\text{AIC value: 1.0} \\
1.0 + 1.0 \\
\text{Current AIC value: 2.0} \\
\text{Rule Id: RULE026} \\
\text{AIC value: 1.0} \\
2.0 + 1.0
\]
```
Step 6: Compare the utility values of activities to define the best substitutive activity.

ActivityID: MP1ALON; Utility Value: 5.0
ActivityID: MP1ALON has the same utility value as the highest one.
ActivityID: B3TVV28; Utility Value: 4.0
ActivityID: B3TVV28 has a lower utility value.

*** Find best substitute activity.
*Current best substitute activity is: MP1ALON
DE_ID: MP1ALON
*The best substitute activity can be: MP1ALON - ‘turn on mobilephone1 alarm’
DE_ID: MP1ALON
*The best substitute activity can be: MP1ALON - ‘turn on mobilephone1 alarm’
The final result of the best substitute activity is: MP1ALON - ‘turn on mobilephone1 alarm’
The result activity is: MP1ALON

Figure 5.12 Output of scenario 2 test case 1

Scenario 2 – Test Case 2:
Output:

Step 1: Eliminate invalid activities according to their corresponding devices.
Activity: ALARMON is invalid due to Device - ALACLSO is unavailable.
Activity: MP1ALON is invalid due to Device - MPHO1AL is unavailable.
Activity: B3TVV28 is valid
**
** Number of valid activities = 1 **

Step 2: Retrieve context data according to rules.
Add Context Element ID: BTHLIST to list.
Add Context Element ID: CUWTHSU to list.
Add Context Element ID: CUWTHCL to list.
Add Context Element ID: CUWTHRA to list.
Add Context Element ID: CUDATMO to list.
Add Context Element ID: CUDATAF to list.
Add Context Element ID: CUDATNI to list.
Add Context Element ID: BR3BRIG to list.
Add Context Element ID: BR3CONF to list.
Add Context Element ID: USERSLE to list.
Add Context Element ID: ALCLSST to list.
Add Context Element ID: MP1ALST to list.
Add Context Element ID: MPHO1ST to list.
Add Context Element ID: R3TVVST to list.
Add Context Element ID: BR3TVST to list.

Step 3: Validate rules with Context Elements.
Rule: RULE040 is valid.
Rule: RULE043 is valid.
Rule: RULE045 is valid.
Rule: RULE047 is valid.
Rule: RULE051 is valid.
Rule: RULE054 is valid.
Rule: RULE056 is valid.
Rule: RULE059 is valid.
Rule: RULE060 is valid.
Rule: RULE001 is valid.
Rule: RULE003 is valid.
Rule: RULE005 is valid.
Rule: RULE008 is valid.
Rule: RULE009 is valid.
Rule: RULE022 is valid.
Rule: RULE023 is valid.
Rule: RULE026 is valid.
Rule: RULE028 is valid.
Rule: RULE030 is valid.
Rule: RULE031 is valid.
**
** Number of valid activities after Step 4: 1 **

Step 4: Retrieve valid substitutive activity.
Activity: B3TVV28 is valid.
**

Step 5: Calculate the utility values for each substitutive activity.
Figure 5.13 Output of scenario 2 test case 2

Scenario 2 – Test Case 3:

Output:

Step 1: Eliminate invalid activities according to their corresponding devices.

Activity: ALARMON is invalid due to Device: ALACLSD is unavailable.
Activity: MEDALGN is valid
Activity: B3TVV28 is valid

** Number of valid activities = 2 **

Step 2: Retrieve context data according to rules.

Add Context Element ID: BTH LIST to list.
Add Context Element ID: CUWTHSU to list.
Add Context Element ID: CUWTHCL to list.
Add Context Element ID: CUWTHRA to list.
Add Context Element ID: CUDATMO to list.
Add Context Element ID: CUDATAF to list.
Add Context Element ID: CUDATNI to list.
Add Context Element ID: BR3BRIG to list.
Add Context Element ID: BR3CONF to list.
Add Context Element ID: USERSLE to list.
Add Context Element ID: ALCLSST to list.
Add Context Element ID: MP1ALST to list.
Add Context Element ID: MPHO1ST to list.
Add Context Element ID: R3TVVST to list.
Add Context Element ID: BR3TVST to list.

Step 3: Validate rules with Context Elements.

Rule: RULE040 is valid.
Rule: RULE043 is valid.
Rule: RULE045 is valid.
Rule: RULE047 is valid.
Rule: RULE051 is valid.
Rule: RULE054 is valid.
Rule: RULE056 is valid.
Rule: RULE059 is valid.
Rule: RULE060 is valid.
Rule: RULE002 is valid.
Rule: RULE003 is valid.
Rule: RULE006 is valid.
Rule: RULE008 is valid.
Rule: RULE020 is valid.
Rule: RULE022 is valid.
Rule: RULE024 is valid.
Rule: RULE026 is valid.
Rule: RULE028 is valid.
Rule: RULE030 is valid.
Rule: RULE031 is valid.

** Number of valid activities after Step 4: 0 **
5.5. Summary and Discussion

The evaluation was done by two types of testing – the overall testing and the individual component testing. The overall testings were done based on the input and output of the workflow execution. In order to review the process of the substitution in the console window, we have tested the function of the Substitution Manager separately. The test cases were based on the two main scenarios which have been defined in the previous chapter (Chapter 4: Section 4.2.2). Each scenario contains a number of test cases as sub-scenarios. We have used different simulated context data to test each test case to ensure the recovery algorithm was influenced by the context.

Generally, the results of test cases were as we expected. The system was capable of detecting the failure at run-time and executing the recovery algorithm was done based on the influence of context and rules.

The Web services of device were defined in the Adaptive Device Manager since the beginning. We have realised that if the Adaptive Device Manager is capable of discovering the new device Web services dynamically at run-time, it can increase the capability of the recovery algorithm. However, it also generates additional questions. The first question is if the system discovers a new device Web service, how does the system knows what activities the device is able to perform? Another question is the system must have no rules have defined the validity of the new device’s activities; is it possible to generate new rules for the new device dynamically at run-time?

We have noticed the task of defining rules and AIc values is not user friendly either to administrator or the user in our proposed system. Theoretically, there need to be a system is needed to support the user/administrator to define rules and AIc values. For example, providing a user friendly application is helpful for the user to define rules based on various context influences. The end-point application is not in the scope of this research; hence, the application to support rule definition can be future work.

We have considered enabling the system to automatically generate rules for the user. However, it is not the focus at this stage. Since each user has different notion of context information, it is unsuitable to automatically generate the context based rules for the user. For example, user A may define the action “turn temperature to 15
degree” is a “very encouraged” action ($AI_c = 1$) when the current temperature context is 22 degree. However, for user B, the action can be a ‘very unacceptable’ ($AI_c = -1$) action. Consequentially, rules that generated by the system, are hardly to satisfy all users. To realise such kind functionality (auto-generate rules) successfully, implements Artificial Intelligence system is inevitable. However, it is out of the scope of this research.
Chapter 6.

Conclusion

6.1. Research Summary and Contributions

This research intends to investigate an approach towards resolving on-the-fly device failures in Device Ecology by using context. The approach was done by first, clarifying the on-the-fly device failures by analysing a scenario. Secondly, we have studied relevant literatures to understand how failed tasks can be detected in a workflow system at run-time. Thirdly, we have reviewed a number of context-aware systems to find out how context-awareness can be implemented to enhance the capability of decision making in Device Ecology workflow. The contributions made in this research are listed as follow:

- **Detect on-the-fly device failures:**
  We have proposed an approach to detect device failures at run-time in Decoflow. We have introduced the concept of task manager. A task manager is capable of handling tasks with the features of failure detection and recovery in workflow at run-time. The task manager has all information about devices. Hence, it is capable of ensuring the workflow task can be executed properly. Moreover, if the pre-defined device failed, the task manager can perform the recovery algorithm by executing the relevant components for substitution at run-time.

- **Enhance the decision making by using context:**
  We have studied a number of context-aware systems and proposed an approach to use context to enhance the Decoflow. Context ontology has been introduced to manage the context information in Device Ecology. Effectively, contexts have been categorised into four main supertypes – *environment, device, user* and *time*. Each supertype may contain a number of subtypes and each type may consist of numerous attributes. The substitute made by the system involves the current
context information.

- **Propose a framework to resolve on-the-fly device failures**
  A proposed framework was developed towards resolving the on-the-fly device failures in Decoflow. The framework consists of the following entities:
  - The Adaptive Device Manager, which has been implemented as the task manager, is capable of handling the failure detection and task execution
  - The Substitution Manager which is capable of finding a best substitute activity by using a rule-based decision making process based on the context influences
  - Rules, which are used in decision making, were defined by the user. Hence, the result of substitution can satisfy the user
  - The Context Manager is capable of performing the context data retrieval and managing contexts based on our Context ontology.
  - The device failure recovery algorithm in our proposed framework was described in detail as a guide of the recovery process. Components that are involved in the algorithm and the tasks of process were clearly explained.
  - The failure recovery can be done at run-time without interrupting the workflow execution. Moreover, the substitute activity can be found and executed automatically without inquiring the user to perform the substitute action.

6.2. Research Directions

*Context rule definition support*
Using context to support Decoflow to find a substitute activity at run-time is an effective approach. However, the process of substitution is done based on the rules which were defined by the user/administrator. Although we have proposed a systematic way to define the rules, but it is only applicable to our work since our prototype was focused on the run-time adaptation in the workflow execution. Effectively, The Decoflow system needs a system which is capable of aiding the user to define rules based on context influences. In other words, we need a rule-based context-aware decision making system.

*Context rule generator*
Although our proposed framework is capable of handling failure recovery and executing a substitute activity at run-time, but the process of substitution was done based on the rules of context influences which are defined by the user. This means
that the user has to define numerous rules to ensure the substitution can be performed properly. We have considered it an inconvenience to the user. Although it is possible to define the rules for the user as the default setting of the system, a very important issue must be considered when developing a rule-based context-aware system: context is identified by persons, different people have different notion of context. For example, person A thinks the air condition should be switched on when the temperature is 20 or over, but person B does not agree. For the person B who is still wearing a jacket when the temperature is 20, a rule which defines the air condition should be switched on when temperature is 20, could be an invalid decision. This simple example illustrates why defining the context rules for the user is usually not feasible. This is an unsolved issue by many rule-based context-aware decision making systems. Is it possible to implement a component which is capable of studying or recording user’s movement and activities in order to dynamically generate the context rules automatically based on various pre-defined rules and the user’s activity records?

**Dynamic discover new service of device for substitution**

Currently, the task manager (or Adaptive Device Manager in our prototype) has no capability to discover a new service of device dynamically at run-time. As mentioned in the discussion section of Chapter 5. If the task manager is capable of dynamically invoking a new service of device at run-time, it may enhance the system to discover more potential substitute devices. However, the data of activities and rules were defined in the database before run-time. If a new service of device is discovered, since there is no data about the new service stored in the database, how does system identify the capability of the new service of device and also, how can the rules be applied to the new service of device?
References


viewed 14 November, 2007

viewed 14 November, 2007


viewed 14 November, 2007

viewed 14 November, 2007


2000.


Appendix A

User and Developer’s Guide of Prototype

User Guide

Software Installation

In order to run the prototype simulation, you are required to install JDK 1.6 or above version and Netbeans IDE 6.0 Beta 1. The JDK 1.6 and the Netbeans IDE 6.0 Beta 1 have been included in the CD. You will be able to find them under the ‘software’ directory/folder. It is recommended to install the official version of the Netbeans IDE 6.0 if it is available at the time you are running this prototype simulation.

Please ensure you install the GlassFish web server. We are not sure the Apache Tomcat is compatible to our project or not.
Database Setting

The database we used in this prototype is called Derby which is bounded with the GlassFish. Please follow the steps below to complete the setting of the database environment.

Please start the Netbeans IDE first. Then:
(1) Target the location of your Java DB and the location which you want to use to create your database.

Select Tools -> Java DB Database -> Settings
Specify the location of the Java DB and the location of your database

(2) Create databases

Select Tools -> Java DB Database -> Create Database…

Please create two databases using the name: ‘decdb’ and ‘sensordb’.
You should be able to start the database server now by selecting Tools -> Java DB Database -> Start Server.

(3) Insert data
You can connect the database by right click the database name and select “Connect…” from the menu. When it is connected, right click the ‘Tables’ folder and select “Execute Command…” from the menu. The IDE will then open a SQL command window. We have provided the SQL command in the decodb.txt file and the sensordb.txt file. You can find these two text files under the SQL directory/folder. Copy all texts in decodb.txt and paste them in the SQL command window of the decodb, and also copy all texts in sensordb.txt and paste them in the SQL command window of the sensordb. To execute the SQL command, simply click the “Run SQL” icon as Figure 1.6 shows.

Create tables and insert records by SQL command
Project Setting

(1) Open projects:
First, please copy the Projects directory/folder to your hard disk (e.g. C:\deco).

To open a project in Netbeans, click File -> Open Project… on the main menu of IDE. You can find the projects under the Projects directory/folder. Open all of them which include a BJE project - AdaptiveDeviceManager, BPEL project - Scenario1, Scenario2 and ScenarioM, composite application project – Scenario1CompositeApp, Scenario2CompositeApp and ScenarioMCompositeApp, and the last one is a Java application project – TestScenarios.

(2) Project property setting:
By default, the project does not automatically find the library it uses at the compile time. You have to find the library for it. Please follow the steps below to complete the library setting.

i. Right click the AdaptiveDeviceManager project and select “Properties” from the menu.

ii. Select “Libraries” from the categories.

iii. Under the “Compile” tab, click “Add JAR/Folder” button. Target the database driver as following screenshots show.

Execute SQL command
Locate the library

Select derbyclient.jar and click “Open”
For the TestScenarios project, you also need another library which is the AdaptiveDeviceManager.jar, you can add the jar by click the “Add Project...” button to add the AdaptiveDeviceManager to the compile library of the TestScenarios project.

You should be able to deploy projects now. Right click a project and select “Undeploy and Deploy” from the menu to deploy projects. Please follow the order to deploy projects:

i. AdaptiveDeviceManager
ii. Scenario1CompositeApp
iii. Scenario2CompositeApp
iv. ScenarioMCompositeApp
v. TestScenarios

Overall Testing

The test cases we used to evaluate our prototype have been described in the Chapter 5 of our thesis. There are two scenarios; each of them has a number of test cases. Each test case requires different context data. Following sections provides the information guide you to run each test case

Scenario 1

You can find all the update commands in the update.txt file under the SQL directory/folder.

Test Case 1:

(1) Please execute following commands in sensordb:

```
UPDATE WEATHER_ELEMENT SET WE_VALUE=1 WHERE WE_ID='CUWTHRA';
UPDATE WEATHER_ELEMENT SET WE_VALUE=0 WHERE WE_ID='CUWTHSU';
UPDATE WEATHER_ELEMENT SET WE_VALUE=0 WHERE WE_ID='CUWTHCL';
UPDATE DAYTIME_ELEMENT SET DTE_VALUE=1 WHERE DTE_ID='CUDATMO';
UPDATE DAYTIME_ELEMENT SET DTE_VALUE=0 WHERE DTE_ID='CUDATAF';
```
UPDATE DAYTIME_ELEMENT SET DTE_VALUE=0 WHERE DTE_ID='CUDATEV';
UPDATE LOCATION_ELEMENT SET LE_VALUE=1 WHERE LE_ID='BR3BRIG';

(2) Please execute following commands in decodb:
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='BTHLIST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='BTHLONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3LIST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='BR3LONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='CORLIST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='CORLONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3CUST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3CONF';

(3) Please deploy the AdaptiveDeviceManager project.
If you haven’t deployed the AdaptiveDeviceManager project, please do so by right click the project folder and select “Undeploy and Deploy”.

(4) You may test the web service to ensure the deployment was completed successfully.
When you select “Test Web Service” as the screenshot above shows, the IDE will open the browser for you to test the Web service (see screenshot below).

(5) Please deploy the composite application for this test case. Please deploy the Scenario1CompositeApp project if you haven’t done it previously.
(6) Check the input message.

![Input XML](input.xml)

input.xml of test case in scenario 1

(7) Run the test case.

Right click the Test Case 1 and select “Run” from the menu. If you see a dialog window, just click the “Yes” button. Try to run the test case couple times if the output message comes out with a fault.

(8) Check the output message.

```
Activity 'BRIGHT(turn on bedroom) light' completed</ns0:taskresult>
```

![Output XML](output.xml)

Output of the scenario 1 test case 1

If the test case outputted a ‘failed’ message, do not be surprised because this Netbeans is just a beta version. If the test case **DOES** output a message even it shows ‘failed’, double click it to see the output. The output message should show the results of the workflow execution, and the test case can be considered as completed. If the test case is failed at first run, please try to run it again.

If your RAM is less than 2GB, the test case probably will fail to complete and there will be no output from the test case. In this case, please test our prototype by the **Minimal Testing** or the **Component Testing** in later sections.
Double click the red message icon to see the output, even it shows “Failed”

**Test Case 2:**

(1) Please execute following commands in sensordb:

```sql
UPDATE WEATHER_ELEMENT SET WE_VALUE=0 WHERE WE_ID='CUWTHRA';
UPDATE WEATHER_ELEMENT SET WE_VALUE=0 WHERE WE_ID='CUWTHSU';
UPDATE WEATHER_ELEMENT SET WE_VALUE=1 WHERE WE_ID='CUWTHCL';
UPDATE DAYTIME_ELEMENT SET DTE_VALUE=1 WHERE DTE_ID='CUDATMO';
UPDATE DAYTIME_ELEMENT SET DTE_VALUE=0 WHERE DTE_ID='CUDATAF';
UPDATE DAYTIME_ELEMENT SET DTE_VALUE=0 WHERE DTE_ID='CUDATEV';
UPDATE LOCATION_ELEMENT SET LE_VALUE=0.5 WHERE LE_ID='BR3BRIG';
```

(2) Please execute following commands in decodb:

```sql
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='BTHLIST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='BTHLONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3LIST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='BR3LONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='CORLIST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='CORLONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3CUST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3CONF';
```

(3) Please deploy the AdaptiveDeviceManager project.

(4) Run the test case.

(5) Check the output message.

```
<result>Activity 'BR3HEAT:turn on bedroom heater' completed.</result>
<result>Activity 'BR3LIC:turn on bedroom light' completed.</result>
<result>Device failure occurs. Substitute activity 'turn on corridor light' executed.</result>
```

Output of the scenario 1 test case 2
Scenario 2

You can find all the update commands in the update.txt file under the SQL directory/folder.

Test Case 1:

(1) Please execute following commands in sensordb:

```
UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERMOV';
UPDATE USERACTION_ELEMENT SET UAE_VALUE=1 WHERE UAE_ID='USERLIE';
UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERSTA';
UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERSIT';
```

(2) Please execute following commands in decodb:

```
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALACLST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCLSST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCSONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3TONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3TVST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='R3TVVST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=8 WHERE DE_ID='R3TVVVA';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='R3VVONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='MPHO1ST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='MP1ALST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='MP1AONF';
```

(3) Please deploy the AdaptiveDeviceManager project.

(4) You may test the web service to ensure the deployment was completed successfully.

(5) Please deploy the Scenario2CompositeApp project for this test case.

(6) Check the input message.
input.xml of test case in scenario 2

(7) Run the test case.
(8) Check the output message.

```
  <soapenv:Body>
    <soapenv:Envelope>
      <soapenv:EnvelopeId>USERMOV</soapenv:EnvelopeId>
      <soapenv:EnvelopeId>USERLIE</soapenv:EnvelopeId>
      <soapenv:EnvelopeId>USERSTA</soapenv:EnvelopeId>
      <soapenv:EnvelopeId>USERSIT</soapenv:EnvelopeId>
      <soapenv:EnvelopeId>BANENN</soapenv:EnvelopeId>
    </soapenv:Envelope>
  </soapenv:Body>
</soapenv:Envelope>
```

Output of the scenario 2 test case 1

**Test Case 2:**

(1) Please execute following commands in sensordb:

- UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERMOV';
- UPDATE USERACTION_ELEMENT SET UAE_VALUE=1 WHERE UAE_ID='USERLIE';
- UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERSTA';
- UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERSIT';

(2) Please execute following commands in decodb:

- UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALACLST';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCLONF';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCLSST';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3TONF';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3TVST';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='R3TVVST';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=8 WHERE DE_ID='R3TVVVVA';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='R3V0NF';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='MPH01ST';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='MPH1ONF';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='MP1ALST';
- UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='MP1AONF';

...
(3) Please deploy the AdaptiveDeviceManager project.
(4) You may test the web service to ensure the deployment was completed successfully.
(5) Please deploy the Scenario2CompositeApp project for this test case.
(6) Check the input message.
(7) Run the test case.
(8) Check the output message.

```
w"">Activity 'BRTTVON: turn on bedroom3 TV' completed.</ms0:result>
w"">Activity 'BRTTV12: increase bedroom3 TV volume to 12' completed.</ms0:result2>
w"">Activity 'BR3LON: turn on bedroom3 light' completed.</ms0:result3>
w"">Device failure occurs. Substitutive activity 'bedroom3 TV volume raise up to 35' executed.</ms0
```

Output of the scenario 2 test case 2

**Test Case 3:**

(1) Please execute following commands in sensordb:

```
UPDATE USERACTION_ELEMENT SET UAE_VALUE=1 WHERE UAE_ID='USERMOV';
UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERLIE';
UPDATE USERACTION_ELEMENT SET UAE_VALUE=1 WHERE UAE_ID='USERSTA';
UPDATE USERACTION_ELEMENT SET UAE_VALUE=0 WHERE UAE_ID='USERST';
```

(2) Please execute following commands in decodb:

```
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALACLST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCLONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='ALCLSST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3TONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='BR3TVST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='R3TVVST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=8 WHERE DE_ID='R3TVVVA';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='R3VVONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='MPH01ST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='MPH1ONF';
UPDATE DEVICE_ELEMENT SET DE_VALUE=1 WHERE DE_ID='MP1ALST';
UPDATE DEVICE_ELEMENT SET DE_VALUE=0 WHERE DE_ID='MP1AONF';
```

(3) Please deploy the AdaptiveDeviceManager project.
(4) You may test the web service to ensure the deployment was completed successfully.
(5) Please deploy the Scenario2CompositeApp project for this test case.
(6) Check the input message.
(7) Run the test case.
(8) Check the output message.

Output of the scenario 2 test case 3

**Minimal Testing**

In the minimal testing (the ScenarioMCompositeApp project), the BPEL workflow (SencrioM) only contains one task, which minimised the requirement of the RAM. You may modify the input message to test the workflow with different tasks (activityId). If you don’t know what tasks are available to the system, please check the Activity table in `decodb`. Each Activity record has an ID. The ID is the value which is allowed to be entered as the activityId of the input message.

Try to enter BTHLION or ALARMON for the activityId. These two activities were defined as the failed tasks due to the device failure.

**Component Testing**

We have implemented an extra project for the component testing. To run the class for each testing case, simply right click the file to pop up the menu and select ‘Run File’. You will see the output in the output window which located at the bottom side of IDE.
Developer’s Guide

To develop applications in Netbeans IDE, you can find various tutorials available on the Netbeans website: http://www.netbeans.org

To get start with the BPEL application development, we recommend you to visit the 99 Sec Demo [1] which provides a number of useful tutorials about developing BPEL in Netbeans.

If you would like to develop a BPEL Web service client application client to execute a BPEL, please visit the online tutorial [2].

References
