Remote control service system architecture and dynamic web user interface generation

LOUGHBOROUGH UNIVERSITY

Remote Control Service System Architecture and Dynamic Web User Interface Generation

Xi GUO

Thesis submitted for the degree of Doctor of Philosophy

Loughborough University

Department of Computer Science

May 2011

For Mum and Dad

ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor P.W.H. Chung. I appreciate his guidance and support for my research. I also thank him for patiently correcting my papers and thesis.

I would also like to thank Zhining Liao, Moi hoo, Lee Booi, Lin Guan and Sheryl William for their support, understanding and friendship throughout my PhD.

Finally, I would like to thank my parents and my husband Jonathan Hansford for their unconditional support and help for my PhD study. Thank you for always being there for me.

ABSTRACT

According to current development of internet technology, remote control over the internet becomes a heated discussed topic. Some recent technologies such as Service-Oriented Architecture (SOA), web service and ontology offer great opportunity for remote control over the internet and a lot of research has been done into this topic. However, there are still many challenges in architecture design and dynamic user interface generation.

Architectures in this research field lack clear description of controlled machine model as well as related knowledge support. Also, there is little system support further control service development. There is little research on web user interface design for remote control system over the internet. The design of web user interface has the challenge of overcoming the limitations of web technology to satisfactorily support different machines, users and control process requirements.

This work overcomes the limitations on architecture by offering a SOA based design which allows both multiple users and distributed machine access. The system applies a machine model for the description of the machine structure and functions, which help the system to reason about machine components and their relationships with instructions. With a web service based design, different machines can be connected via the system and execute user commands. Using semantic description and ontology based methods the system can automatically retrieve machine information and generate the structure and function descriptions for different machines. Moreover, the system provides services which can support further development in remote control services. The proposed architecture improves on former designs, and offers a flexible architecture for remote control services over the internet.

An intelligent web user interface is also introduced in this work. The design separates the interface data structure from the data representation. Supported by the remote control service development environment, the user interface could adapt to the control sessions. Semantic descriptions are used to describe the page data structure, session context as well as control command. Therefore, control page content can be adapted by the computer to real time control session. At the same time, a session dependent navigation is designed to resolve the problem

of changeable requirements for multiple-machine web user interface. With a message driven model and a session dependent semantic data structure, the required machine data can be analysed by the system and categorised according to user's requirements. Compare with other designs, this service offers a method for web user interface generation for different machines, users and can adapt to different control processes.

The design is demonstrated in five evaluation scenarios aimed at testing different aspects of the system. Evaluation demonstrates the design proposed in this thesis is feasible. It also shows the design can be applied to different areas and adapted to different control related requirements well.

Keywords

Remote control, SOA, Web service, web user interface, ontology, semantic, session dependent navigation, machine model.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACH	Adapter Control Handler
AL	Adapter Layer
API	Application Programming Interface
CML	Control Management Layer
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CPL	Control Page Logic
CWUI	Control Web User Interface
DLL	Dynamic-Link Library
DOM	Document Object Model
DPWS	Devices Profile for Web Services
DRH	Device Resources Handler
FRH	Function Record Handler
GUI	Graphical user interface
HCP	Hardware Control Program
HLL	Hardware Logic Layer
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IT	Information Technology
IWUIC	IWUI Controller
IWUIM	IWUI Model
IWUIV	IWUI View
J2EE	Java 2 Platform, Enterprise Edition
JDOM	Java-based document object model
JSON	JavaScript Object Notation
KSE	Knowledge Search Engine
MAA	Machine Adapter Application
MFD	Machine Function Description
MSD	Machine State Description
MVC	Model-View-Control
MWSD	Machine Web Service Data
OLE	Object Linking and Embedding
OOHDM	Object-Oriented Hypermedia Design Model
OOWS	Object-Oriented Web Solution
OPC	OLE Process Control
OPC UA	OPC Unified Architecture
OSGi	Open Services Gateway initiative
OWL-S	Ontology Web Language for Services

PUC	Personal Universal Controller
RMI	Remote Method Invoke
SDL	Service Development Layer
SOA	Service-Orianted Architecture
SOAP	Simple Object Access Protocol
SRH	State Record Handler
UAL	User Application Layer
UDDI	Universal Description, Discovery and Integration
UI	User Interface
UIML	User Interface Markup Language
UPnP	Universal Plug and Play
WebML	Web Modeling Language
WMAAPI	Web-service-based Machine Access API
WSRCSS	Web Service-based Remote Control Service System
WSDL	Web Services Description Language
WSMCG	Web-services-based monitoring and control gateway
WUI	Web User Interface
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

CHAPTER 1. INTRODUCTION

1.1. Overview

With the internet, all kinds of information and documents can be accessed by anyone from anywhere without travelling. The potential benefit of remote control and monitoring using the internet is also attracting increasing attention. The idea of using the internet for remote control is not new, but work in this area is still at an early research stage.

This chapter begins with the motivation for the research then highlights some of the problems with current remote control system architectures and client user interface generation. Relating to these problems, the aims of the research are made clear. The final section describes the thesis organisation.

1.2. Motivation

Since Leonardo Torres-Quevedo laid down the modern wireless remote-control operation principles in 1901, the development of this technology has not stopped [IEEEGHN, 2007]. In World War I, the German navy used radio controlled motorboats to ram enemy ships [IEEEGHN, 2007; Yang, 2002] and then remote controlled televisions appear [TVhistory, 2001; IEEEGHN, 2007; 1-800-Remotes, 2010]. Today, wireless technologies are used in many fields, such as industrial applications, emergency services, space [IEEEGHN, 2007] and house facilities. Various ways have been used to help people access their appliances like lights, heaters, robots and security systems [Burger and Frieder, 2006; Wang, Liu and Wang, 2006; Ito and Murai, 2007; Kim, *et al.*, 2007].

There are a lot of applications in the real world that could benefit from remote monitoring and control. For example, shift operators may have difficulties with operating a machine due to an unexpected event. An experienced engineer who is located far away from the factory can login remotely and assist with resolving the situation. A person can remotely switch on the heating system at home before leaving work to ensure they return to a warm home. A nurse could

monitor the condition of multiple patients without having to walk around a hospitality ward. The possibilities are endless.

Although various approaches for remote control and monitoring have been proposed, none of them dominates the field. Radio, infrared and ultrasonic signals are used for short distant control [TVhistory, 2001]. With the help of wireless technology, such as Bluetooth [Haartsen, et al., 1998] and Zigbee [ZigBee Alliance, 2011], signals can be sent over longer distances. These technologies help to group the controlled machines in to local networks which can then be connected to a long distance controller via long distance communication technologies such as satellite [David and Jeffrey, 1998]. However, satellite remote control is expensive and is not widely used.

Control and monitoring machines over the internet becomes more feasible than before. First, the internet is widely available and used around the world in offices, factories and home for accessing information and communicating with friends and colleagues. It provides an obvious infrastructure for supporting remote control and easy access to users. Therefore, it is being investigated by a number of researchers [Chen and Luo, 1997; Yang, Tan and Chen, 2002; Dai, Yang and Tan, 2004; Hung, Chen and Lin, 2004; Yang, et al., 2005; Dai, Yang and Knott, 2006; Oltean, Abrudean and Dulau, 2006; Yang, Dai and Knott, 2007]. Also, existing middleware technologies such as OSGi [OSGi Alliance, 2011] (Open Services Gateway initiative), DPWS [OASIS, 2009] (Devices Profile for Web Services), and UPnP [UPnP Forum, 2011] (Universal Plug and Play), as well as industry standards such as OPC [Li and Nakagawa, 2002; OPC Foundation, 2011b] (OLE Process Control) have been improved gradually to make better seamless connection with low level device hardware with high level drive applications. Moreover, many concepts have been discussed recently in order to better integrate devices into IT (Information Technology) systems. For example, “Internet of Things” (IoT) [Gershenfeld, Krikorian and Cohen, 2004; Haller and Karnouskos, 2008] is the idea that future devices will be smart enough to access the internet and potentially be controlled and monitored over the internet. Similar to most of the research on “internet of things”, work in this thesis tries to integrate hardware objects to the internet. Nevertheless, most of current research on “internet of things” focuses on internet-ready embedded devices and the network for devices [Krikorian and Gershenfeld, 2004; Guinard and Trifa, 2009;

Vazquez, et al., 2010], while remote control over the internet discussed in this thesis complement this by focusing on control and monitor mechanism for devices.

Based on current internet technologies, it is relatively easy to build a specified purpose direct remote control system. However, according to current literature review, there is seldom a remote control architecture existing that provides a service for flexibly adding users and machines. Also, there is very little research on web based user interface generation for such a remote control service system. The following section will explain the details of the problems related to remote control over the internet.

1.3. Problems related to remote control over the internet

A general problem with the internet is control link security. The internet offers convenient access to resources from anywhere, but at the same time has the risk of exposure to malicious attack. This is a major topic for the internet security research community. Another problem is reliability. Messages transferred across the internet can be lost or corrupted. This is a topic for the communication research community. The maintenance of a control session is more complicated than just recording current state and continuing the control session after the interruption. The influence of the tasks and command sequence should also be considered. However, these problems are already receiving considerable attention. Instead of focusing on these reasonably well established research areas, this thesis focuses on two topics which are important in establishing flexible and convenient control services. The details of these will be explained in the following sections.

1.3.1. Remote control system architecture

The system architecture for remote control is comparable to the structural framework in a house. The design of the remote control system will influence the future expandability of the system and the functions that can be provided. Current architectures [Chen and Luo, 1997; Walsh and Hong, 2001; Yang, Tan and Chen, 2002; Wang, et al., 2003; Dai, Yang and Tan, 2004; Hung, Chen and Lin, 2004; University Of Florida, 2004; Ha, Sohn and Cho, 2005, 2007; Helal, et al., 2005; Yang, et al., 2005; Dai, Yang and Knott, 2006; Oltean, Abrudean and Dulau, 2006; Ha, et al., 2007; Kim, et al., 2007; Yang, Dai and Knott, 2007] have limitations described below:

i. Lack of support for multi-access remote control

A number of remote control system architectures have been proposed [Chen and Luo, 1997; Walsh and Hong, 2001; Yang, Tan and Chen, 2002; Wang, et al., 2003; Dai, Yang and Tan, 2004; Hung, Chen and Lin, 2004; University Of Florida, 2004; Ha, Sohn and Cho, 2005, 2007; Helal, et al., 2005; Yang, et al., 2005; Dai, Yang and Knott, 2006; Oltean, Abrudean and Dulau, 2006; Ha, et al., 2007; Kim, et al., 2007; Yang, Dai and Knott, 2007]. However, these architectures are for developing individual remote control system and do not cater for adding or removing users and machines from the system, i.e. they are not based on a service approach. For an open environment like the internet, these architectures limit future system development and are not adaptable to changing user requirements. For example, some of the control applications allow different users to enter the system but only allow specified machines to be controlled. Some systems only allow machines on a specified area to be controlled, which results in difficulties when introducing new machines outside of the area. This particular issue will be further explained in Chapter 3.

ii. Difficulties in identifying machine structure and functions for multi machine control

Different machines have different actuators and sensors as well as different functions available to a system. Even if the system can establish a control link between the user and the target machines, the system needs to offer an appropriate interface for the user to access the machine information to be able to monitor and control any of the selected devices for any particular control session. This means that it is necessary to facilitate the exchange of appropriate commands and information between any user and any machines that the user has chosen to control. However, the controlled machines can vary and it is difficult to know the structure and functions for the target machines in advance.

iii. Inflexible for new services development

There is no remote control service system that supports the development of innovative services to support the control task. These services can be considered as extendible functionality for remote control ability. For example, services like dynamic control user interface generation need the support of basic control ability. So, these services have to build on the basic control functions. It is difficult to reuse or extend the functionalities if they are tightly coupled to the system. Some researchers have tried to make these abilities into services [University Of Florida, 2004; Helal, et al., 2005; Ha, Sohn and Cho, 2007; Helal, Yang and

Bose, 2007]. However, they have not discussed how to provide a framework for supporting service development for remote control.

1.3.2. Client user interface

Client user interface is a critical issue for almost any system and user interface development is a very resource-intensive task. A good user interface can help users achieve their goals efficiently. For an internet-based remote control system, a user interface needs to be designed to meet user needs and control requirements. This area is generally overlooked by researchers. It is normally assumed that a be-spoke user interface is developed for a control system. However, this would not work if different users can register and logon to control different machines of varying types. The problems in this area are analysed below.

i. Challenges in dynamic generation of user control interfaces

User interface design is a challenging problem when a remote control system allows multi-user access and machines can be added and removed dynamically from the system. For example, the user interface has to be modified when a new type of machine is added to the system. It is a significant problem for the system to offer an appropriate user interface according to the user's requirements in conjunction with the machines capability. For example, if a nurse needs to monitor the blood pressure and heart rate of fifty patients simultaneously, it would be confusing and inefficient if they were required to view the data for each patient on a separate page. Furthermore, the same nurse may be required to monitor different patients on different wards so the interface has to be designed to display the relevant information and structure them accordingly. Manual design of user interfaces is time consuming and limits the future extensibility of a system. Additionally, the traditional GUI approach and technologies used to implement user interfaces for remote control systems also have limitations. Therefore, in the internet driven world, a new innovative approach is required to overcome the limitations of traditional GUI technologies.

ii. Difficulties in dynamically organising and indexing mass information

For a user interface that has to display a large quantity of control information, it is difficult to provide a web user interface that is appropriate for a user's specified context in each session. For example, a nurse may monitor different patients on different shifts and if the number of devices and sensors involved is large then the information cannot be displayed on one screen.

It would be inconvenient for the nurse to scroll up and down the screen in order to seek target information. Therefore, research need to be done to find a dynamic way to organise and index the control data, which is available in real time, and related information for the user to locate easily the required data or information.

iii. Difficulties in adding session dependent information

Session dependent information cannot be added to a control session in an arbitrary manner. For example, a piece of equipment in a hospital may need information such as the patient who is using it and the nurse who is operating it. For some machines, for monitoring purposes, set points for raising alarms have to be set. Therefore, the user interface also needs to make provision for the user to enter any relevant session specified information.

1.4. Aims of the research

Instead of focusing on the traditional areas of security and reliability of the communication system, this research focuses on the following two problems. First, there is little architecture that support a web service-based approach for remote control over the internet and allow new users and new machines to be registered dynamically and become part of the system. Second, there is little research on web user interface generation for such a control service system. Therefore, this research aims at solving the architecture design problem and client user interface problem for remote control over the internet. The objectives are:

i. A novel architecture for a remote control and monitoring service system over the internet

- Design an architecture that would be able to support multi-access from both the user side and machine side through the internet;
- Support and offer control methods for different machines;
- Support the development of remote control services.

ii. Dynamic generation of web user interface

- Create a method that could dynamically generate the content for a web-based user interface, which can change automatically according to different user requirements, control machines and control processes;

- Organise information and navigation links appropriately for different control sessions;
- Offer methods to input session dependent information for each control session.

iii. Prototype implementation and evaluation

- Test if real hardware can be controlled using this architecture;
- Build an evaluation environment to demonstrate the architecture as well as dynamic generation of web user interface;
- Apply the prototype to a range of different machines and applications to demonstrate the architecture and web user interface generation.

1.5. Thesis organisation

The rest of the thesis is organised as follows:

Chapter 2 describes the enabling technologies that are related to this research. These include: Service Oriented Architecture, Web services and Ontology. The way these technologies influence the research will also be explained.

Chapter 3 reviews existing system architecture designs for remote control systems. Three basic architectures are discussed and compared. Applications based on different architecture are considered and conclusions presented.

Chapter 4 reviews work related to user interface generation for remote control. The challenges and features required for web user interface design are explained. Guidelines and models for web user interface are described. Research related to automatic generation of machine control user interfaces are analysed and conclusions drawn.

Chapter 5 introduces the system architecture and kernel design. This chapter explains the functions of six layers in the architecture. It also explains how these layers work together to achieve the design goals.

Chapter 6 introduces a service for the automatic generation of web user interface for the remote control service system. This chapter explains how the service is designed and how it can generate web user interfaces for different machines, users and control processes.

Chapter 7 introduces the design for generating flexible session-dependent navigation for control user interface. With the navigation, the web user interface for remote control service system can organise large amount of information and index among the information for easy access.

Chapter 8 explains the evaluation environment and preparation as well as five evaluation scenarios aimed at testing different aspects of the design. The results are analysed and conclusions are drawn.

Chapter 9 concludes with the contribution of the research. Limitations and potential future direction for research are identified.

CHAPTER 2. ENABLING TECHNOLOGIES

2.1. Overview

Designing remote control systems over the internet is a complicated research area. Challenges arise from both the internet communication field and machine control field. The proposed system will need technologies to achieve the operation of distributed machines over the internet and to organise machine related knowledge. The emergence of Service Oriented Architecture (SOA), web service and ontology makes it possible to design a remote control service system that can overcome some of the problems previously highlighted.

Three technologies are discussed in this chapter with an explanation of the benefits they bring to remote control research. Firstly, a general description of SOA is given and then further explanations are given for the advantages it could bring to remote control system design. Second, web service is introduced as a software solution of SOA with an explanation of the problems that web service could help to solve in a web–based remote control system. Finally, ontology as a technology is described and its benefits explained.

2.2. Service-Oriented Architectures (SOA)

There are several different definitions for SOA. The W3C glossary described it as *"A set of components which can be invoked, and whose interface descriptions can be published and discovered"* [Haas and Brown, 2004]. SOA is also described as *"a form of distributed systems architecture"* with six features: Logical view, Message orientation, Description orientation, Granularity, Network orientation and Platform neutral [Booth, et al., 2004]. While in [He, 2003], it is defined as *"an architectural style whose goal is to achieve loose coupling among interacting software agents"*. However, these definitions are not particularly easy to understand without further explanation. To understand SOA, it is necessary to understand the meaning of two keywords —"architecture" and "service".

Architecture is *"the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution."* [Maier, Emery and Hilliard, 2009] From the definition, it is clear that architecture

plays an important role in helping to organise the resources inside and outside the system so that the system can be constructed to meet the requirements of a target application. Due to the fast development of the internet, as well as the increasing functionality and complexity of components, IT systems are now more complex than before. With traditional architectures, functions are closely coupled to the system so even a small change will affect the whole system. This fact results in a lot of time and energy are required to build, upgrade, and maintain these systems due to the difficulties of combining different system components [Booth, et al., 2004; Endrei, et al., 2004; Sprott and Wilkes, 2004]. This traditional approach makes it particularly difficult to organise components and resources for an internet-based distributed system. A new approach to architecture design that offers a flexible way to organize components and resources and be able to adapt the system to new requirements is needed. SOA has been brought forward as a solution.

The word "service" highlights the key difference between SOA and other architectures. In W3C [Haas and Brown, 2004], service is defined as *"an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities. To be used, a service must be realized by a concrete provider agent."* There are three main ideas behind the definition:

- A service is an abstracted view of actual functions that are used to implement certain tasks. The group of functions could be actual programs, databases or business processes. They need to be well-defined and not tied to other components in the system.

- The services are described using a particular kind of interface so that they can be requested and invoked.

- There should be a particular kind of mechanism for accessing these interfaces allowing message exchange between providers and requesters.

There are many examples of services in real life. Transport between two cities could be considered as a kind of service. A person can choose to go to another city by bus, by train or by plane. Each service is independent from the others and will help the person to reach their destination. Each service has its own access station where a person can access and use the resources available.

A further definition to understand is “service-oriented”. It actually means to separate function entities from the system so that they can be packaged into a group of services and allow these services to be published, located and invoke by other applications. SOA is an approach for designing systems that utilise service components.

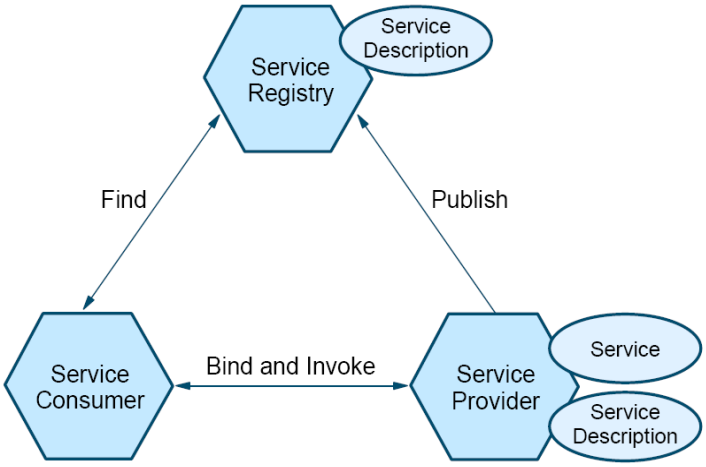


Figure 2. 1 Collaboration In A Service-Oriented Architecture
Adapted From [Endrei, et al., 2004]

The collaboration process in SOA is shown in Figure 2.1 [Endrei, et al., 2004]. Service Consumer, Service Provider and Service Registry are three basic elements of SOA. Service Consumer represents applications that request the service. Service Provider is the entity which offers actual services to the service consumer and Service Registry is the place where services are published. Each service in the Service Provider will have a Service Description to describe further details of the functions provided by the service. When the Service Provider publishes Service Descriptions on the Service Registry, these services and descriptions can be found by the Service Consumer. Once the Service Consumer obtains these details, the services interface can be bound to the application in the Service Consumer, so that the services can be invoked. In this way, the interface for accessing the functions in the services is independent from the application. Hence services can be deployed separately from the system that invokes them.

On the system with components built using SOA, the service components need to be well defined and made available to other system components to access. This helps to organise the system in a modular “plug-in” manner, which is also known as “loose coupling”, so that all the function components can be developed independently as services. With the triple mechanism of “publish-find-bind”, these functions can be “plugged” in to the system. Hence, future

changes in a component will not affect the system and the costs and risks introduced by the changes will be reduced.

SOA will also benefit the development of remote control system over the internet. As discussed in Chapter 1, for many years, remote control systems have been confined to be-spoke applications and limited machines. One of the aims of this research is to provide facilities for adding machines and users to the remote control system. However, it is difficult to build a system that has flexibility to cope with large groups of users with different requirements and also include a wide range of application machines. With the help of SOA, there are possibilities to solve these problems. Some industry standards, such as OPC UA (OPC Unified Architecture) [Mahnke, Leitner and Damm, 2009; OPC Foundation, 2011a], propose SOA based method to integrate different low level devices in to high level applications. Some middleware technology, such as OSGi [OSGi Alliance, 2011] and DPWS [OASIS, 2009] appears to be supporting SOA based control system. There are researchers already applying the SOA approach to remote control design [University Of Florida, 2004; Ha, Sohn and Cho, 2005, 2007; Helal, et al., 2005; Ha, et al., 2007]. However, most of the work focuses on building up the local device cooperation networks instead of internet based remote control systems. Also, current remote control architectures are still not flexible enough and lack support for developing remote control services. This will be further explained in Chapter three.

2.3. Web service

The core idea behind SOA, i.e. separating functional components from their interface and from its implementation, is not new. There are also other technologies that try to achieve similar goals, e.g. Java 2 Platform, Enterprise Edition (J2EE), Common Object Request Broker Architecture (CORBA) and Component Object Model (COM). The biggest problem for these technologies is that they require special execution environment to function. For example, some only work with a particular language, such as J2EE only works on the java platform, while COM only works on Microsoft operating systems. Even CORBA, which is aimed at cross-language and cross-platform operability, needs to be deployed and compiled in the same develop environment [Endrei, et al., 2004; Ray, 2010]. When it comes to an open environment such as the internet, problem occurs when the system needs to invoke services

built on different technologies. A distributed computing model, which separates service interface from the execution environment, is needed.

Web service provides a potential solution to achieve this goal. A definition of web service [Booth, et al., 2004] is:

> *"a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."*

According to the definition, there are two basic technologies upon which web services are built: XML (eXtensible Markup Language) and HTTP (Hypertext Transfer Protocol). XML is a flexible markup language used for formatting text and exchanging a wide variety of data [W3C, 2011]. HTTP is an application-level protocol used for the internet. Other important basic concepts of web services include: Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and Simple Object Access Protocol (SOAP). Although UDDI is not mentioned in the definition, it plays an important role in web services. According to [Booth, et al., 2004; Ray, 2010], the meanings of these terms are:

- WSDL (Web Services Description Language): an XML-based language used to describe the attributes of a web service, such as location and interface. From the description, the client will know how and where to invoke the actual services.

- UDDI (Universal Description, Discovery and Integration): is a directory service that provides a publicly accessible means to store and retrieve web services.

- SOAP (Simple Object Access Protocol): an XML-based protocol used to format and process data with web services.

Web services can be considered as technologies that help to achieve SOA and separate the service interface from the execution environment. As shown in Figure 2.2, there are three elements in the web service architecture, which can be mapped to the three basic components in SOA collaboration. Service provider is the services and their description. Service consumer

is the entity that requests the services. UDDI Directory is the service register centre where web services descriptions are registered and stored. These entities communicate using SOAP. Before the web services can be used, their descriptions need to be created using WSDL and sent to UDDI Directory. When the service consumer requests the services from UDDI Directory, the service description will be sent back. According to the WSDL service description, the service consumer will find the available services and send a request to the correct service provider to invoke the service. In this way, the link between service consumer and service provider is built up and the services can be used by a consumer. Web service technologies provides a way to achieve SOA and separate the service APIs (Application Programming Interface) and the executing environments as it does not require special software or environments to execute. As soon as SOAP is understood, web services can be published, found and invoked. This technology can smoothly deliver an SOA design to achieve the goal of "loose coupling".

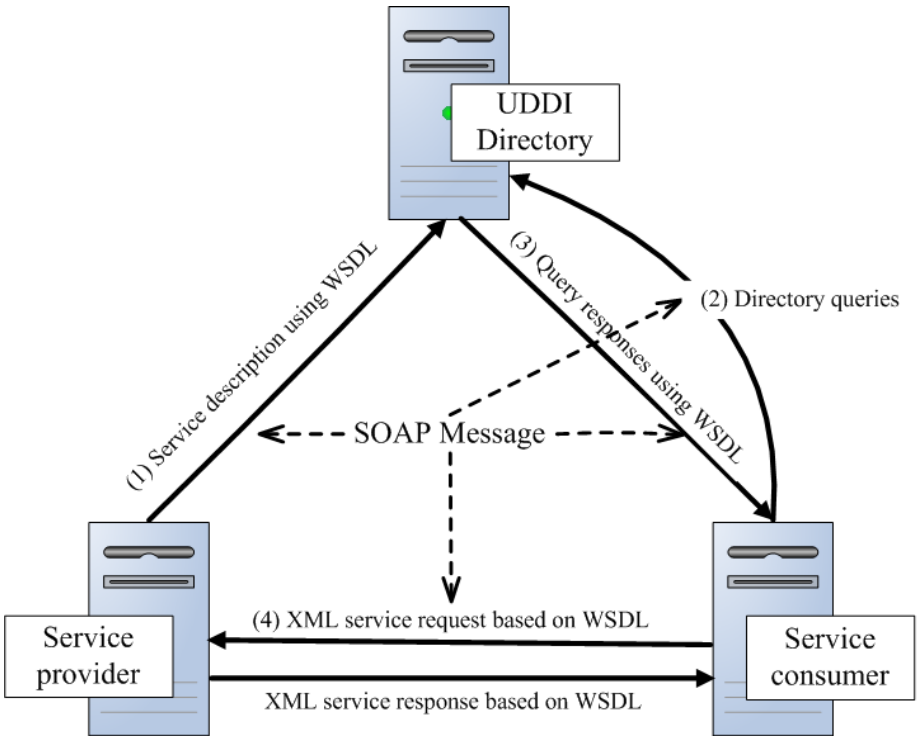


Figure 2. 2 Web Service Architecture
Adapted From [Barry and Gannon, 2003]

Web service is not the only way to achieve SOA, but is suitable for the internet based systems use distributed components build on different platforms. There is some work comparing SOA to other distributed technologies, such as CORBA and COM [Gisolfi, 2001; Gokhale, Kumar

and Sahuguet, 2002]. According to the literature, there are several advantages of web services which need to be highlighted.

Firstly, web service message is carried by SOAP which is an execution model that is language and platform independent. Therefore, there are no restrictions on development environments for web services.

Second, web service is firewall friendly since HTTP is associated with a well defined port that is accepted by all firewalls. Therefore, there are fewer difficulties with communication between different components over the internet.

Third, in a dynamic environment in which the address of the client or even the server may change, web service can adapt to the changes by simply resending the SOAP message.

Finally, web service does not demand much from the client side other than a SOAP message handling application. Although there are some criticisms regarding web services, such as security and efficiency, for internet-based remote control web service is most suitable for achieving a SOA design when compared with other available technologies.

It is not new to use web service in control systems. Some designs already use web services to help to connect machine application over certain networks to achieve SOA [Jammes, Mensch and Smit, 2005; Jammes and Smit, 2005; Karnouskos, et al., 2007; Mathes, et al., 2009; UPnP Forum, 2011]. New technology that adapts web service to the device level such as UPnP [UPnP Forum, 2011] and DPWS [OASIS, 2009] has been widely used in embedded device network systems. Industry standard OPC UA uses web services to make it possible to access the machine over internet [Mahnke, Leitner and Damm, 2009; OPC Foundation, 2011a]. With the help of these technologies, integrating devices and sensors into collaborating networks becomes possible. However, there are still some questions need to be solved. Firstly, most of the researches only use web service as middleware for invoking the distributed machine functions directly. The invoke process happened when low level devices hardware connected to high level application. These high level applications need to be further integrated in to IT system on enterprise level to support further remote control over the internet. Also, for a multi-access remote control system over the internet, it is hard to control these distributed machines unless their functions are already known. The remote control service system proposed in this thesis can operate different machines, and their functions are all unknown for

the system before they are registered. Moreover, the system needs to support changeable user's requirements during a control session. Therefore, presetting a fixed set of functions for users to invoke directly would not work. Hence, more work is needed to provide flexible access to different machines in a remote control service system.

2.4. Ontology

The term ontology has been around for many years. The concept of ontology originates from the philosophy field, and is referred to as the subject of existence. In computer and information science, ontology is an abstract model for the knowledge in a certain domain. In 1993, Tom Gruber defined that *"An ontology is a specification of a conceptualization."* [Gruber, 1993]. It is considered to be the first commonly accepted definition of ontology [Gruber, 1992]. One of the goals of ontology is to represent the information and relationships between different identities, so that they can be better understood by computers. [Gruber, 2009]

Different from other kinds of knowledge representations such as traditional databases, ontology consists of classes, properties, [Noy and McGuinness, 2001]. In [Xie and Shen, 2006], the differences between ontology, object models and knowledge base are analysed and the article summarised the use of ontology to be:

- Communication and shared understanding between people;
- Interoperability among computer systems and agents;
- Knowledge sharing and reusing.

In AI systems, what "exists" is that which can be represented. Ontology not only represent the domain knowledge and related vocabularies, but the knowledge has to be in a form that can be processed by computer programs [Gruber, 1992] to achieve automated reasoning.

One of the biggest problems for handling more than one type of machines is how the system knows what commands can be used to control a given machine. Ontology provides a way to solve this problem. Machine related knowledge can be represented using ontology. When new machines are added to the system their functions can be retrieved by referencing the ontology. Some work has been done in utilising ontology for machine control. Cho, et al. [Cho and

Kawamura, 2007] created a home automation robot called BlogAlpha. Using ontology, BlogAlpha could understand the hierarchy of the commodities in a home, as well as their functions and locations. Therefore, it could process commands in natural language intelligently by using the ontology. In [Joo, Park and Paik, 2007], ontology is adopted an intelligent home service framework called iHSF. The main idea is to use ontology to semantically describe concepts such as user requirements, places and devices. Similarly a device ontology is created in [Chrysoulas, et al., 2007] to describe and support different control services.

However, these ontologies only described some high level commands or tasks for the controlled machine and there is very little research using ontology to solve the problem of automatic machine structure and function descriptions. The structures and functions are key aspects which can map machine control knowledge to the real machine components. Therefore, research about ontology models for machine structures and functions still needs to be continued.

2.5. Summary and conclusions

Remote control over the internet is not new but existing systems are limited. Recent development of new technologies brings many opportunities. These technologies offer new ways to solve problems that exist in existing designs. This chapter considered three technologies that are particularly relevant — SOA, web service and ontology.

SOA is an approach that helps to achieve “loose coupling” in software architecture. The concept of separating function components from the system as services reduces the project costs and risks. It helps to organise the system in a flexible way and allows sharing of resources between different components or systems. Some remote control research based on SOA has already begun but existing designs are still not sufficiently flexible.

Web service is a technology to achieve SOA. It is based on XML and HTTP, and uses SOAP message to register, find and bind services. Web services helps projects to achieve better SOA structure compared with other middleware technology, because it separates the service interface from its execution environment. So far the focus has been on developing middleware that helps devices to connect to remote control systems via the internet in a SOA way, but more work needs to be done to make machine structure and function available for further use.

Ontology is a method that can be used to describe real world knowledge in a way that computers can process and reason with. Ontology has been widely used in remote control systems. However, there is still not a clear and unified way to apply ontology for machine structure and function descriptions.

This review has highlighted that these three technologies can be applied to the design of remote control system for the internet, which would help to overcome existing problems that have been identified.

CHAPTER 3. REVIEW OF CURRENT SYSTEM ARCHITECTURES FOR REMOTE CONTROL

3.1. Overview

Remote control technology through the internet is developing rapidly. Enabled development by new technologies there has been different architectures proposed over the years to improve the abilities and qualities of remote controls. The following chapter will analyse the development of remote control architecture over the internet. According to the different architectures, related works are summarised and analysed.

3.2. One-to- one remote control

Early research on remote control was done by setting up data links between two computers via the internet using communication ports such as the RS232 port [Cai, et al., 2004] or more recently other technologies, for example, RS485 and CAN (Control Area Network) [Castro, et al., 2002]. Control commands and signals are sent and received through the network. As shown in Figure 3.1, one-to-one remote control is usually based upon a special communication link. Sometimes even special protocols are required, so that signals can be transferred from a controller to a specified machine. The one-to-one architecture offers simple and direct control.

A lot of works has been done using this architecture. S.E. Oltean, et al. [Oltean, Abrudean and Dulau, 2006] developed a system which contains a local digital controller and server-client application. Using the TCP/IP protocol, a PIC16F628 microcontroller could monitor and control the local digital temperature controller. Yang, et al. [Yang, et al., 2005] considered the time delay problem with the internet transmission. In their initial system the connection to the internet was via a 33.6k bandwidth telephone modem. Later a network card was used to improve speed and reliability.

In their system one thread is used to read or write control messages and signals to and from the controlled machine in one second intervals. The data buffer is used to store any necessary information. The other thread is used to communicate with the remote controller. Again, any necessary information is stored in the data buffer. The remote controller receives data through

the network every two seconds and the data is displayed on the screen of the remote controller. The frequencies of one second and two seconds for the two threads are determined empirically to solve the problem of time delay due to the network.

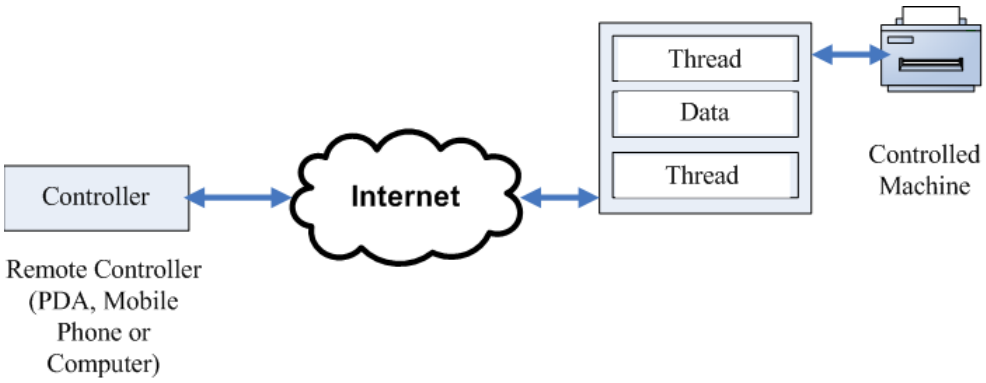


Figure 3. 1 One-to-one Remote Control Architecture

The limitations of the one-to-one architecture are obvious. It does not support multi-machines or simultaneous access multi-users. The architecture is not flexible to cope with changes relating to different needs.

3.3. Many-to-one remote control

To overcome the problems with one-to-one architecture, the many-to-one architecture has been proposed. The differences between the one-to-one architecture and the web-based architecture (many-to-one) for remote control are discussed in [Chen and Luo, 1997]. Compared with the one-to-one architecture, it overcomes the access limitation as it allows different users to access the machine at the same time. Usually, the controller and controlled machines are connected using the internet, as shown in Figure 3.2, this method is also called web-based remote control.

Generally, a server is located between the controlled machine and the internet, to provide an interface for the web browser. This allows different users to be able to control the machine anywhere via the internet. There has been significant research and development in applying this architecture to actual use.

Research has been done in using the internet as a control message transfer method [Chen and Luo, 1997; Dai, Yang and Tan, 2004; Kim, et al., 2007; Yang, Dai and Knott, 2007].

Sometimes a Local Area Network (LAN) based technology is used on the controlled machine side to link with more than one machine. However, it is still many-to-one remote control, because these machines are still limited to a special area, and the area is often a closed network, which also limits the number of the machines.

Figure 3. 2 Many-to-One Remote Control Architecture

Although this architecture allows different users to access the controlled machine, the machine side is limited to a certain kind of network. For example, K.S. Kim, et al. [Kim, et al., 2007] proposed a remote control system for home network electrical applications via the internet. Zigbee and UpNP are used to expand the control for the home network environment. It allows multi-user access but the problem is that this kind of method does not allow machines to move to other networks. S.H. Yang, et al. [Yang, Tan and Chen, 2002] proposed an approach to writing requirements specification for internet-based control systems and deriving architectures for this new type of control system according to the requirements specification. Later, Yang, et al. [Yang, Dai and Knott, 2007] discuss remote maintenance with the internet transmission in a web-based remote control system. Also, fault detection is discussed for such remote control system in [Dai, Yang and Tan, 2004] but the machines in this remote system are all confined to a LAN, which limits their expandability. Hung, et al. [Hung, Chen and Lin, 2004] propose a remote monitoring and control architecture, consisting of a Web-services-based monitoring and control gateway (WSMCG), distributed Ethernet-ready I/O modules, safety detection modules, web cameras, and networks. It not only provides general functions of remote monitoring and control but also possesses mechanisms of actively detecting an appliance's abnormal power consumption and ensuring appliance safety. However, this architecture is also limited to a certain controlled machine network. Also, the tight link between a complicated control system and controlled machine make it hard to move the machines somewhere else easily.

There is other research that offers control over the internet, but the controlled machines are limited. For instance, T.M. Chen, et al. [Chen and Luo, 1997] propose a method and architecture for remote control of a remote robot and the user can also supervise the robot via a CCD camera. Basic motion commands, such as “Set Motion Velocity” and “Move a Distance”, are available to the user, however only one machine can be controlled at a time.

In web-based architecture, control messages and values are passed between the remote controller and the controlled machine using an agreed protocol. Dai et al [Dai, Yang and Knott, 2006] proposes wrapping control messages as JDOM (Java-based Document Object Model) objects using XML technology and passing them as arguments using TCP socket. The data transfer mechanism is built on static Remote Method Invoke (RMI) of the object. Microsoft.Net framework and web service have also been used in this way for remote control and maintenance [Hung, Chen and Lin, 2004]. Other related work using the internet protocols is described in [Walsh and Hong, 2001; Marti, et al., 2004; Liu, Meng and Yang, 2005; Wang, Liu and Wang, 2006]. Industry standard should be used for message passing to provide a service platform that is hardware and software independent.

From an architectural point of view the many-to-one approach is still very limited. The effort is on the owner of each controlled machine designing and implementing all the facilities and services that they want to make available to the remote users. They can only control limited machines in the same closed network, and they can only recognise certain groups of machines or possibly only one machine.

3.4. Many-to-many remote control

In order to overcome the limitations of the many-to-one architecture, the many-to-many architecture has been proposed. This method enables a platform to support multi-user and multi-machines. The idea is to break the close link between the server and the controlled machines. Based on SOA, controlled machine functions are provided as services. The machines communicate with a central control server via the internet in a loose-coupled way.

The general framework of the many-to-many architecture is shown in figure 3.3. In this general framework the control structure is divided into three parts: Application layer, Service Logic Layer and Hardware Logic layer. The application layer offers a user interface for different users to access the service system. The service logic layer is the kernel part of the

system, not only because it connects users with machines, it is also designed for supporting all kinds of remote control services that can offer different controls according to users' needs. The service logic can be deployed on different servers, and there may be control data support these services. The actual control application is located in the hardware logic layer, where different machines can be accessed via different methods. As shown in Figure 3.3, the machine can be connected through a computer, a wireless router connection, a gateway or other internet devices.

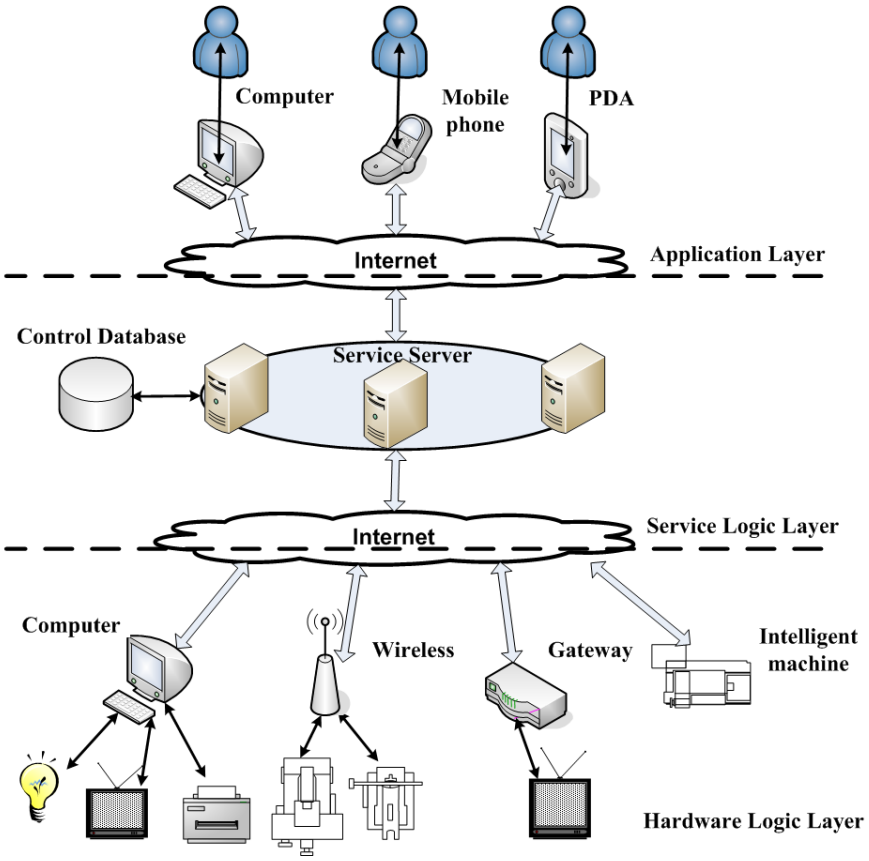


Figure 3. 3 Many-to-many Remote Control Architecture

The essential improvement of this design is separating remote control service logic with control application so that it supports multi-access from both the user side and the device side. According to SOA, the idea behind the approach is that a controlled machine is considered as an independent component or service that can be registered with the service server. The communication link between the controlled machine and the service server is established according to an agreed standard. Compared to the many-to-one method, the control logic no longer has to be located on the same closed network with actual control application. In this

way, different machines in different places can be accessed remotely by the control system from anywhere via the internet.

Another important change is that the focus of the architecture is no longer about establishing a communication middleware. Instead, the focus is on providing a better service development environment. Existing middleware technology such as web service is used to achieve the communication over the internet for the machines.

Currently, research related to this architecture is described in [Walsh and Hong, 2001; Wang, et al., 2003; University Of Florida, 2004; Ha, Sohn and Cho, 2005, 2007; Helal, et al., 2005; Ha, et al., 2007; de Souza, et al., 2008]. Increasingly researchers have put their focus on service-based systems. However, the results still have drawbacks so that they do not offer flexible control services. Three different many-to-many architecture designs will be analysed according their types of problems.

Design problem 1: Lack of standardised mechanism to communicate and understand device level applications.

SOA can be applied to two levels of a remote control system over the internet: the device level and the enterprise level. Device level SOA aims at integrating different machines in to a unified control application so that there is no need to develop different drivers to get data from the machines. Enterprise level SOA aims at integrating different machine applications in to high level services, so that machine data and functions can be used to achieve more complicated tasks. There are some explanations for the differences in these two levels in [de Souza, et al., 2008]. A lot of technology and research is focussed on device level SOA [University Of Florida, 2004; Ha, Sohn and Cho, 2005; Helal, et al., 2005; Ha, et al., 2007; Ha, Sohn and Cho, 2007]. However, there is not as much research into the SOA based mechanism to further link enterprise level applications with different device level applications. Also, there is a lack of research in the mechanisms to exchange the information on how the machines can be operated between the two levels.

Firstly, due to the lack of mainstream SOA middleware technologies, some designs try to separate controlled machine application from the enterprise level remote control services, however they fail to separate them clearly. The researchers need to create their own methods to build communication between servers and controlled machines over the internet. For

example, Wang, et al. [Wang, et al., 2003] proposes a web-based remote control service system that is supposed to support multiple services and is self-organising. From the architecture shown in Figure 3.4, the system is designed according to the many-to-many architecture as it allows different users to control different machines via a system built upon the internet.

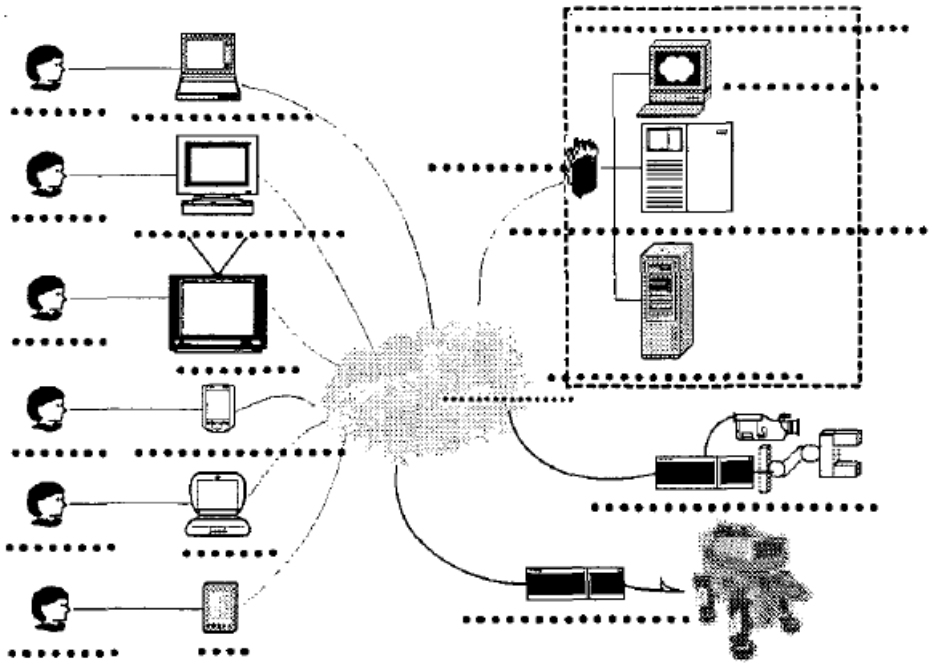


Figure 3. 4 Overview Of The Web-based Remote Control Service System
Adapted From [Wang, et al., 2003]

The main benefit of this design is to achieve a server that supports self-organised processes. Figure 3.5 shows a remote control factory would create the objects of service. A model called the component image factory will generate console image and device image, which can then communicate with the remote machine and console using XML-based messages. In this way, the machine side application can be separated from the main server and communicate with the server using messages. However, most of the work in the system has to be put into creating a "self-organisation" process, which is actually a method to achieve communication with distributed remote machines.

This kind of design lacks a communication standard over the internet, which will make the resulting system rely heavily on a particular platform, language or implementation environment. This kind of design also lacks consideration for offering control services,

because the focus is on solving the communication problem. If the services cannot be clearly separated from the execution environment then it will be difficult to organise and use components to generate new services.

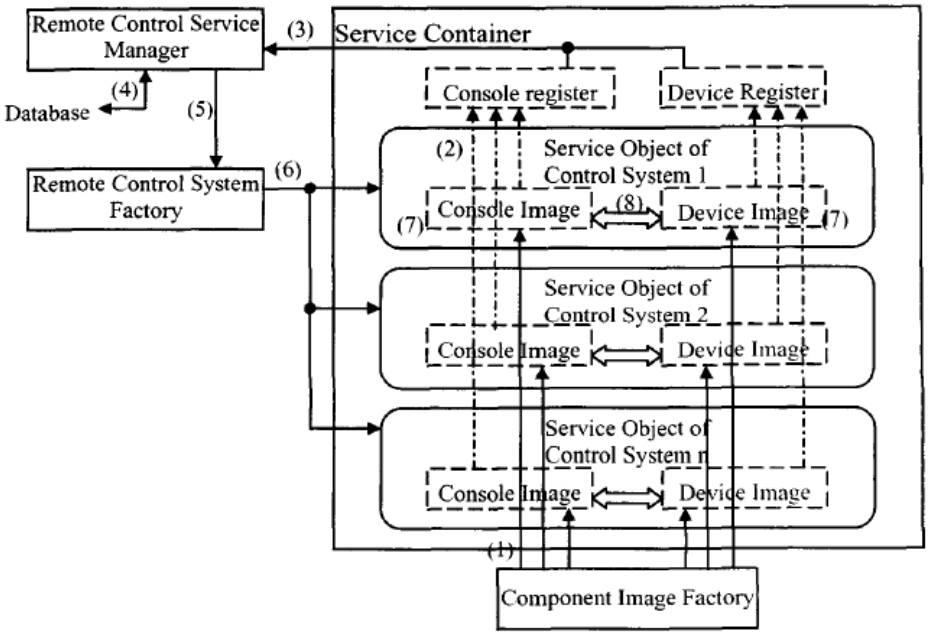


Figure 3. 5 Hierarchy Of Remote Control Server
Adapted From [Wang, et al., 2003]

As discussed in Chapter 2 , web services perform better on independent machines compared to other technologies that rely on more tightly integrated machines for enterprise applications. Recently, architectures have been proposed to use web services to integrate different device level applications to an SOA based IT system. However, there is lack of research on information exchange that allows the enterprise level application to understand how to operate lower level machine applications. For example, L.M.S. Souza, et al. propose a middleware "SOCRADES" in [de Souza, et al., 2008]. Using web services, this middleware integrates "shop floor" device applications in to enterprise level business logic services. Nevertheless, the middleware layer only passes the control APIs to the high level applications. It is impossible for computers to use these APIs without further definition. Therefore, it is not possible for the computer to automatically reason about how to operate the machines. Lack of a standardised mechanism, means it is complicated for multi-access remote control systems to operate the machines dynamically registered into the system.

Design Problem 2. Lack of clear description of controlled machine model.

Some architecture uses middleware or standard methods to solve the communication problem. With the help of middleware technologies, such as web services, the architecture achieves a better SOA design. However, some of the research lacks a clear description of the machine model, so it is not easy to maintain the relationships between different sensors and actuators within an actual control task from services.

Machine model which decomposes the machine component in to different sensors and actuators has been used a lot in the field of automate control. There are already researches consider machine as an object consisted of different sensor objects and actuator objects [Panne and Fiume, 1993]. There are also researches decompose control tasks into some atomic functions which can be executed by individual sensors or actuators [Ljungkrantz, et al., 2007]. Using a machine model, a machine's inner structure as well as its relationship with these control functions can be represented clearly.

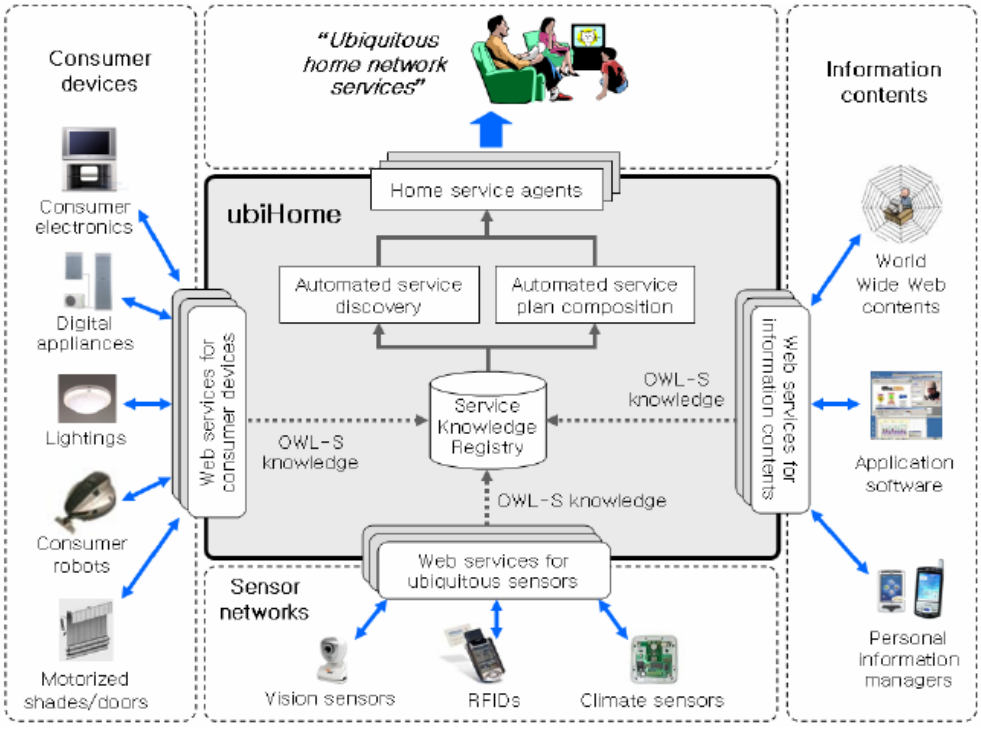


Figure 3. 6 Conceptual Architecture Of ubiHome Infrastructure
Adapted From [Ha, Sohn and Cho, 2007]

However, there is little research on remote control applying machine models to the internet based remote control system architecture design. Ha, *et al.* [Ha, *et al.*, 2007] proposed a Ubiquitous Robotic Service Framework and later discussed using semantic web service technology [Ha, Sohn and Cho, 2005] and applied it to build a Ubiquitous Home Network Service System [Ha, Sohn and Cho, 2007]. As shown in Figure 3.6, the main idea is to consider all the devices and web resources as a web service and using a home service agent to integrate all these resources so that can be used by the service control process.

The main components of the system (Figure 3.7) are built upon Web services, which allows different devices to access the system from different places via the internet. Ubiquitous Resources Service represents devices while Service Knowledge Registry stores semantic knowledge for domain & device relationship, so that services could be organised in Service Agent Platform according to different contexts (mainly based on location).

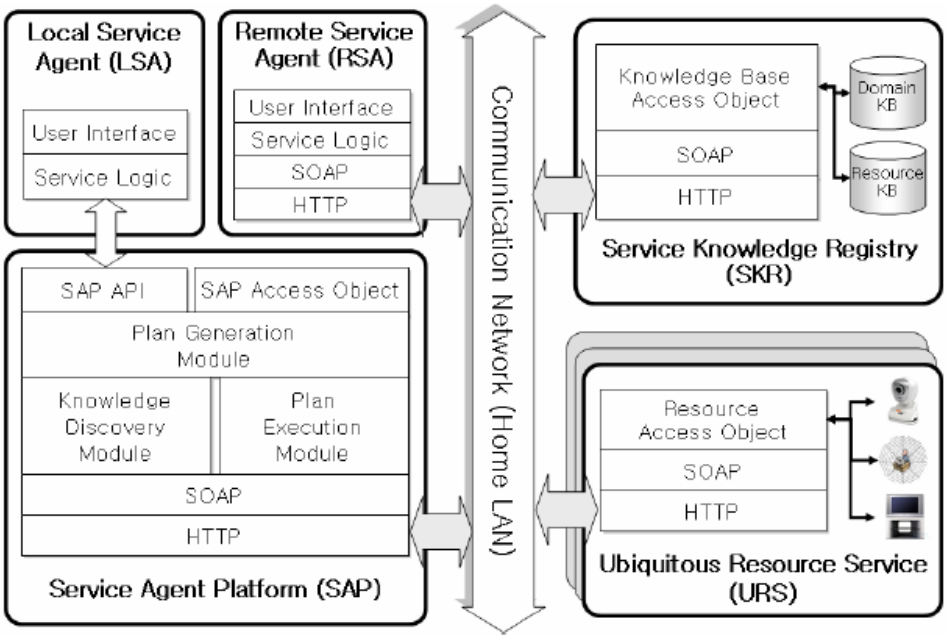


Figure 3. 7 Detail Architecture Of ubiHome
Adapted From [Ha, Sohn and Cho, 2007]

There are two problems that caused by unclear description for the machine model. One problem is that although the research mentioned Ontology Web Language for Services (OWL-S), service profile is used to describe each sensor service individually and there is insufficient information to describe each machine as a whole. A machine may contain a group of sensors and actuators, but there is not a clear description for these relationships between these

components in a machine. Another problem is that only high level services are described in the system, such as "RaiseWindowBlindService". No explanation is given to how the commands actually operated the machine components. The result of this kind of design is the services can only build upon some "pre-registered" services, and further development of these services will have problems in expanding or reorganising atomic actions, because it is not clear how the high level commands are put together using low level commands that executed by individual components.

The disadvantage for this type of design is that the system may have difficulty in understanding the relationships between the sensor and actuators within a machine, as well as the detailed atomic operations for the high level commands. This lack of information will result in difficulty for the system to reason about machine functions. Also, without the understanding of the machine structure, it will be difficult to change the high level commands once they are set or to introduce new ones. Therefore, it is not flexible for remote control service development, which may require changes to atomic operations when new hardware is introduced.

Design Problem 3. Lack of machine structure and function knowledge.

The University of Florida Mobile and Pervasive Computing Lab has been working a SOA based intelligent control system for a number of years. The Gator Tech Smart House in Gainesville, Florida, [University Of Florida, 2004; Helal, et al., 2005] is a program with federal funding from the National Institute on Disability and Rehabilitation Research (NIDRR). This program aims at developing an intelligent environment to assistant older people as well as the disabled. The design of a system that offers a smart-space middleware is given in [Allen, et al., 2009]. It is based on the Atlas Platform [Helal, Yang and Bose, 2007], which allows plug-and-play Atlas devices to access the system and be programmed into different intelligent services.

This design uses a standard middleware to help the integration of distributed machines. Figure 3.8 shows an early version of the architecture for the middleware [Helal, et al., 2005]. Using an Open Services Gateway Initiative (OSGi) framework, the system can control the devices using web resources.

Also, this design tries to propose a machine model to describe the relationship for the components within a machine. As shown in Figure 3.9, the design is improved later by developing a "node layer" so that individual sensors and actuators can be linked to the devices that contain them. In that way, the relationships between atomic sensors and actuators can also be better described and then mapped more logically to the services on high level application layer.

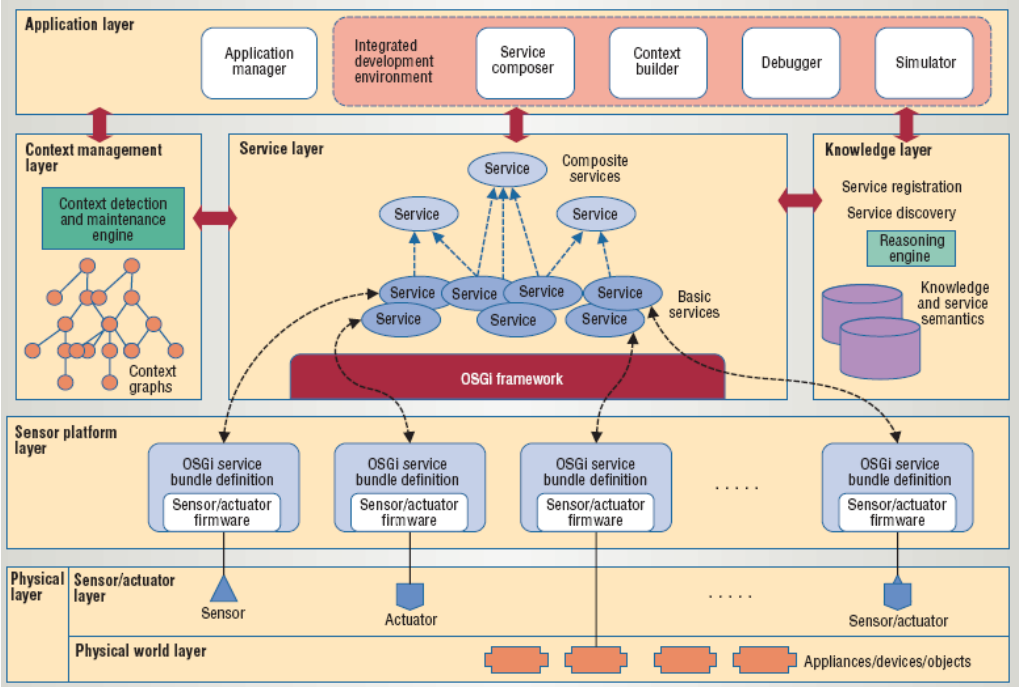


Figure 3. 8 Architecture Of The Smart Space Middleware
Adapted From [Helal, et al., 2005]

In this design of OSGi service, bundle definition is used to describe each machine as a Device Drive Service when the machine accesses the system. However, there is no machine structure and function knowledge to support automatic generation of these descriptions. As illustrated in Figure 3.9, although the system offers knowledge support for service development, and ontology support for context reasoning, there is no support knowledge to support automatically reasoning machine functions. Different machines have different structure and machine functions. For a system that is supposed to be able handle all the different kinds of machines, it will consume a lot of time and energy to generate description for each atomic service manually.

Another example architecture [Wang, Liu and Wang, 2006] is based on message transmission protocols Jabber/XMPP (Extensible Messaging and Presence Protocol). Using a transfer server, this architecture allows machines to be connected to the internet from anywhere. Although there are some XML based representation of the messages transfer between the machines and controller, there is no further machine structure and function knowledge designed to support the reasoning of these functions. Hence, the messages used for control the machine functions have to be designed manually for each type of machine. Therefore, this kind of architecture, it is not suitable for developing services for machines with various types.

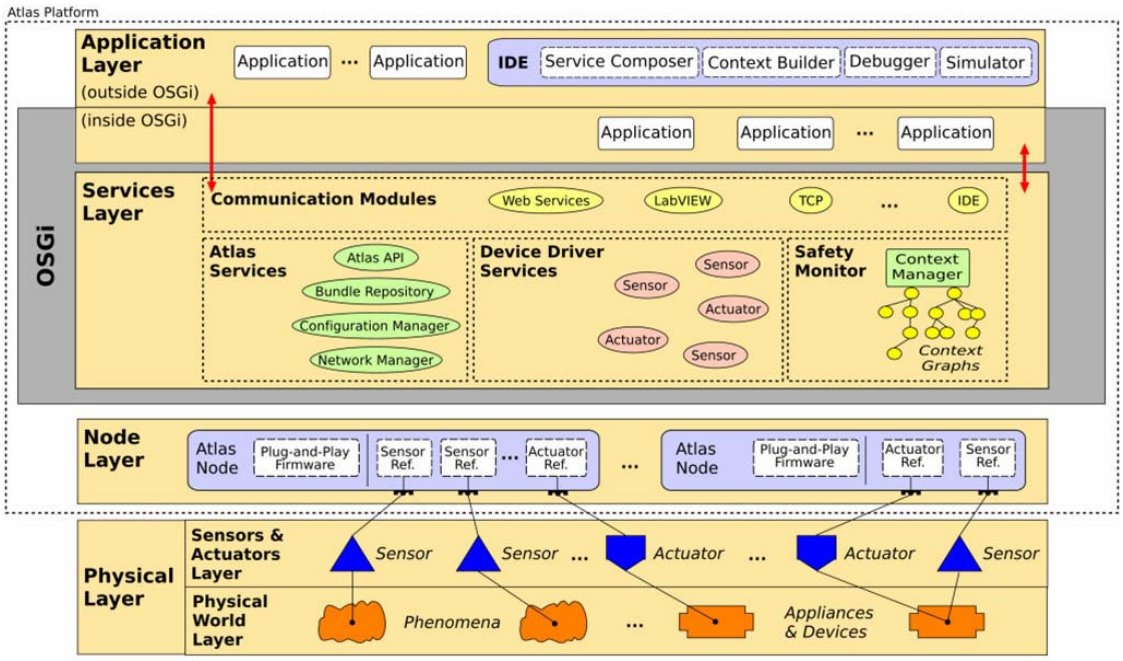


Figure 3. 9 Overall Architecture Of The Atlas Architecture
Adapted From [Helal, Yang and Bose, 2007]

The fact is, without machine structure and functions knowledge, it will be hard for the system to know how to use machine functions automatically. For the systems which use semantic descriptions to identify machine functions these semantic descriptions have be to manually generate. If the system needs to control large number of different machines, the manually generation of the descriptions will be a cost and time consuming work. In this way, the architecture is limited by the manually generated descriptions and is not flexible enough to cope with extensions when new machines are added.

3.5. Summary and conclusions

The development of the internet has promoted progress in architecture developments of remote control and monitoring systems. This chapter considered three different architectures: one-to–one remote control, many-to-one remote control and many-to-many remote control. Early designs apply direct control between the controller and the controlled machines; however, this kind of one-to-one control regime is based on a limited architecture that cannot be easily expanded to allow multiple users or multiple devices. To overcome the limitation, many-to-one remote control architecture was developed. This architecture allows multi-user - access to the controlled machines.

Later, many-to-many architecture is proposed to offer more flexible control service. Applying the internet access to both the controller side and controlled machine side, many-to-many architecture makes multi-access available for both users and controlled machines. Based on the many-to-many architecture, a system can have a flexible structure and information for developing control services. This chapter described the general architecture for many-to-many remote control over the internet. Some designs – a web-based Remote Control Service System, SOCRADES, ubiHome, Gator Tech Smart House and an architecture design based on Jabber/XMPP – that fit the general many-to-many architecture are described and analysed. However, these designs are not flexible enough to support the services development for a multi-access remote control system due to not utilising mechanism to communicate and understand device level applications, lack of clear description of controlled machine model, and lack of machine structure and function knowledge.

Nonetheless the many-to-many architecture offers the best support for multi-access from both the users and machines side. Usually, many-to-many remote control system is built based on SOA, which separates device services from the main system logic. This results in different devices being accessible to the system in a plug-in way. Using standard communication middleware, a system can be free from handling service integration problems and focus on the control logic. The development of further services requires a better description for device services. Without a clear description of machine models, it will be difficult to understand machine structure and develop new services flexibly. At the same time, machine structure and function knowledge is needed so that these descriptions as well as the relationships between

individual services can be reasoned automatically. There is a need for a new design based on the many-to-many architecture but addresses the problems identified in this chapter.

CHAPTER 4. REVIEW OF AUTOMATIC WEB USER INTERFACE GENERATION FOR REMOTE CONTROL

4.1 Overview

User interface is an important part of any system, not only because it offers users access to the system functions but also because it helps users to understand what the system is suppose to do and for the system to respond correctly to the users' intention. A well designed user interface is supposed to represent system functions effectively, and help users to carry out their tasks efficiently. However, good user interfaces can be difficult to design. As it is described in Chapter 1, there are many problems for user interface design for a web application like the remote control service system proposed in this research. Designing the user interface can be time consuming and costly and it is difficult to anticipate all the interfaces that are required for the different control scenarios. A solution is to generate the user interfaces automatically in response to a user's goals.

In order to understand the research trends related to this area, this Chapter is organised as follows. Firstly, the design challenges for Web User Interface (WUI) are explained by comparing it with traditional Graphical User Interface (GUI) design and web page design. Second, existing design principles and guidelines for general user interface and current web application design models are reviewed. These principles and guidelines offer foundation knowledge for WUI design. Third, existing methods related to automatic generation of web user interface for device control is analysed. The methods are reviewed according to their contribution on different aspects in interface usability for a control system – machine, user and process. Dynamic navigation technology for web user interface is then reviewed. The chapter ends with a summary and conclusions section.

4.2 Challenges from WUI design

When it comes to web user interface (WUI) design, people often mistakenly think that it is similar to web page design, or think that WUI can be designed using the same methods as traditional GUI. However, WUI design is neither the same as web page design nor GUI design.

It is because WUIs are developed in a different environment and have their own features. Challenges for WUI design include [Galitz, 2007]:

- HTML is limited in object representation and interaction styles.
- Browse navigation is often poor in WUI. Users can get lost and it is hard to keep track of a session.
- Information architecture and task flow are difficult to standardise.
- Diverse range of users with different characteristics.

If control User Interface (UI) is based on web pages then the task of designing the interfaces is subject to all of the problems inherent to web technology. Therefore, WUI design is different from general GUI design.

Research related to web page design is an ongoing hotly debated topic and it is easy to confuse the semantic web with WUI design. The semantic web receives growing attention for its ability to make web content available for computer reasoning [Lee, Hendler and Lassila, 2001]. Unfortunately, WUI for control is different from web pages for semantic-based information access. Jean Tillman [Tillman, 2003] compares the design of WUI with the design of web pages. The basic difference is that web page design is focused on the display of page content; WUI design is concerned about uses and tasks. In [Galitz, 2007] WUI is considered as a web application. In other words, instead of considering a web page as a content-based document, it should be considered as a part of an application system.

Although the dividing line between content and application is not always clear [Galitz, 2007], some challenges that make WUI design different from web page design are highlighted below:

- WUI should not only consider information representation but also helping users to achieve their tasks by invoking system functions.
- WUI needs to consider if the design structure is suitable for accessing system functions, instead of considering only the structure for content and graphic elements.
- WUI needs to handle data exchanges between the application and users, while web pages only need to represent the web information to users.

- WUI needs to maintain user sessions that keep the system running continuously while web pages can be stateless.

The kernel part of the semantic web is focussed on representing web content in a semantic way. Although WUI can use some methods from semantic web technologies, such as semantic representation and ontology, more focus needs to be put on the data flow and control logic behind the interface so that the information pages can be linked with the real control services.

Therefore WUI is different from either GUI design or information based web site design. Although related technologies from GUI or web site design can be useful, WUI is actually a functional component for a remote control system.

4.3 WUI design related guidelines and design models

As discussed above, WUI can be considered as part of a web application and is different from traditional GUI and content-based web pages. There are currently seldom guidelines for building WUI. However, WUI design can take advantage of existing guidelines from general user interface design as well as web site design, while addressing its own challenges.

4.3.1. General design principles and guidelines for user interface design

According to [Stone, Stone and Jarrett, 2005] and [Cato, 2001] the most important factors for user interface design are usability, which includes effectiveness, efficiency and satisfaction in a specified user context. There are different definitions for usability in different contexts. For example, in [Cato, 2001], usability is defined using five main features: learnability, efficiency, memorability, errors and satisfaction. These are similar to the design principles described in [Brinck, Gergle and Wood, 2002]: function correctly, efficient to use, easy to learn, easy to remember, error tolerant and subjectively pleasing. [Suh, 2004] considers the factors that impact on user experience: page layout, design consistency, accessibility, information content, navigation, personalisation, performance, security, reliability and design standards. These descriptions of usability offer designers some general factors that need to be considered when designing a user interface.

There are also some guidelines for user interface design. For example, the “Bull’s-Eye” [Beier and Vaughan, 2003] offers a frame work for web application user interface guidelines. It combines the guidelines for designing a web user interface into concentric circles. Starting

from the centre, the guidelines includes components or combination of components, page template, page flows, combination of page flows to create an interaction pattern/model and overarching features and principles. Although the guidelines do not provide details they do highlight what to do and what not to do to solve design problems. As the factors and guidelines are vague descriptions and are not connected with real problems, when applying usability to the design of remote control system specific details need to be worked out to achieve the required outcome.

4.3.2. Design models of web applications

Web application is different from web page design. Without a good model that represents and organises data and the functions behind them, it is hard for a system to carry out reasoning. Recent research has been done to help designers to build user interfaces in a structured way. For example, Object-Oriented Hypermedia Design Model (OOHDM) [Schwabe and Rossi, 1995], Object-Oriented Web Solution (OOWS) [Fons, et al., 2003], User Interface Markup Language (UIML) [Abrams, et al., 1999] and Web Modeling Language (WebML) [Ceri, Fraternali and Bonio, 2000] are all concept models for web application design. The latter two models use a semantic language like XML to describe the logic behind the web page contents.

UIML is a kind of description language that can be used to describe the representation of web user interface. The idea is to provide a universal user interface language free from assumptions about applications and interface technology. This language allows designers to represent user interfaces with an XML based language, which can subsequently be mapped to the real user interface [Abrams, et al., 1999]. However, this language can only be used as a representation of user interface elements; it cannot be used to show the relationships between the data on a page and its intended meaning.

WebML [Ceri, Fraternali and Bonio, 2000] is proposed as a high level model of web user interface by offering different models for describing four aspects of a web application model: structural model, composition model, navigation model, personalisation model. WebML supports a model-driven approach to Web site development. ToriiSoft [Ceri, Fraternali and Bonio, 2000] is a tool based on WebML for web site design. [Ceri, Fraternali and Bonio, 2000]

The WebML models help designers to build web site models using a general semantic representation that can be mapped to the actual pages. Although the models do not directly

deal with remote control systems, there are some good ideas that can be used to achieve a good WUI application design. The most important is that the content of a web page can be organised in a structured way using these models. However, more research is required to achieve automatic generation of user interfaces for web applications.

4.4 Automate generation of machine control user interface

Automatic generation of user interfaces for machine control application is not new. There are already some ideas proposed to generate control user interfaces according to different usability problems. This section reviews the research related to the three different aspects of WUI design for remote control: machine, user and control process.

4.4.1. Automatic generation of control user interface for machines

It is always a problem to design new interfaces for new machines especially in a multi-access remote control system where there are different users and different machines.

Some previous work offers methods for generating adaptive user interfaces for different machines by retrieving semantic data from the devices [Grooters, 2006; Namgoong, et al., 2006]. In [Grooters, 2006], the method is to use a database to store the page information. When a page is requested then the system will execute a predefined procedure which would retrieve the relevant data from the database to generate that specific page. Although the page content is generated dynamically and can be adapted to different HTML browsers, the method is simply retrieving the data and constructing a page without further organising and reasoning. Namgoong, et. al. [Namgoong, et al., 2006] proposes a more dynamic method which allows the page object to carry out reasoning based on a device ontology and then generate a control page dynamically according to a user's preset preference. Also, the user interface will adapt to the changes in locations. For example, when a user enters or leave a place, the system will refresh the user interface to include available devices for control in that location and remove those that are not present in the location. However, the method only supports a few high level commands and the interface is developed using GUI instead of WUI. There is still much to do on user interfaces that can adapt to different control processes.

Some researchers proposed using description languages for the generation of user interfaces for complicated applications. Nichols, et. al. [Nichols, et al., 2002; Nichols, et al., 2003;

Nichols and Myers, 2009] proposes a method to generate a PUC (Personal Universal Controller) by using a light weight user interface description language [Nichols and Myers, 2009]. All the functions, especially control commands, are pre-defined in a set of specifications. The user interface can then be generated according to the specifications. Using "group tree", the commands that are related are put into the same group in the user interface. However, the user interfaces generated using this method cannot be changed even when the state of a machine is altered.

Existing research offers ideas for generating UI for different types of machines. However, the UIs are not "designed" by the system. Instead, user interface templates are stored in a database or a kind of data resource. When a user wants to use them, the templates are retrieved and presented to the user.

4.4.2. Adapting control user interface for different user requirements

There is also some research done on user side adaptation. The general aim is to adapt the same control content to different users according to their needs so that users are presented with their own personalised control user interface. However, what needs to be emphasised is that this kind of personalisation is different from simple adaptations like changes in font, colour and background. Websites like MSN space (http://home.live.com) and Twitter (http://twitter.com) allows users to choose their own style. However, they use simple techniques in preset style sheets or simple frameworks that change the content of a page.

"Multi-device" user interface research tries to solve the problem of representing the same contents on different devices such as PDAs, PCs and mobile phones. WebSplitter [Han, Perret and Naghshineh, 2000] uses an XML-based representation to split information from the same web page content according to a user's needs and authorities. Also, using multi-device web browsing proxy, it can transfer user interfaces to different client machines. In [Baluja, 2006], Document Object Model tree (DOM-tree) based analysis is used to help to split a web page according to its structure, so that a large website can be browsed using devices with small screens like PDAs. SUPPLE [Gajos and Weld, 2004] uses an optimisation algorithm to support user interface generation across different display devices. C. Bruninx et. al. [Bruninx, et al., 2007] uses filters to select content which is important so that the content can be adapted to different screen sizes.

Existing research offers ideas for splitting the content of a web page or website and adapting it for different users. The focus is on fitting the content onto to target device screen instead of rearranging the selected material according to a user's control purpose. There is no research that allows users to choose their own control functions based on WUI.

4.4.3. Automatic generation of user interface for control processes

Apart from handling different machines and different users, the control processes themselves are dynamically changing. This is influenced by the different control tasks and the state change of the controlled machines.

There has been some research done to generate control user interface to adapt to different tasks. A task can be considered a high level description of a group of atomic commands for a certain purpose. Some researchers try to construct atomic commands from one or more machines and generate dynamic user interfaces with high level commands. For example, in [Namgoong, et al., 2006], a user interface can be generated automatically according to device services found by the controller. Available high level commands will be constructed according to ontology knowledge and then displayed on the user interface. Huddle [Nichols, et al., 2006] presents an approach for generating user interface for multiple control applications. Based on tasks, it offers two types of user interface for users. One is "Flow-Based interface". It helps users to build up high-level task user interfaces. The other is "Aggregate user interface". It allows functions from different devices to be combined onto the same control page. This design is more flexible compared with [Namgoong, et al., 2006] because it allows atomic commands to be reorganised and used by the user. The Roadie system [Lieberman and Espinosa, 2006] can also generate user interface for multiple devices according to a user's goals. Different from Huddle, Roadie uses a common sense knowledge base to map between user goals to actual devices functions instead of task flow.

However, there are three disadvantages in these approaches. Firstly, the user interfaces can not be changed after generation. All the function from a system, no matter whether they are relevant or not in a session context, are all represented on the UI. For example, if the device is a CD player, even when the state of the CD player is already "Play", the command "Play" is still available for the user. Second, the method for organising multiple devices is not flexible. Only high level command can be showed on the UI while users can not select and construct

command according to their own interests. All the information has to fit on the same interface. Third, GUI design is used, which cannot be used directly for WUI design.

Little research has been done to generate user interfaces that can synchronise with machine states. Nichols, et. al. [Nichols, et al., 2002] proposes a type-based method to solve this problem so that commands in the same group are not made available. However, there is no detailed explanation on how to synchronise the user interface with the controlled machine. Therefore, if the state on the machine side changes, which is not caused through the user interface, the state may not be changed on the user interface.

4.5 Dynamic navigation for web user interface

Another problem this research has to deal with is the representation of a large amount of information associated with different machines. If a user wants to control or monitor several machines together then there can be more information than a single screen can display. Navigation is needed for a website that has to present a large amount of information. Therefore an efficient navigation method is needed to solve the problems.

There are already methods to support the design of information-based websites and applications [Fowler and Stanwick, 2004; Kappel, et al., 2006; Rossi, et al., 2008]. Information can be hidden, folded or separated into multiple pages so that only the relevant information for the user is displayed on the available screen at any one time [Kappel, et al., 2006]. However, no matter how the information is shown, it is necessary to provide a link to the information.

Usually, for a website that needs to show a large quantity of information, navigation is used by categorising the information in a logical way and offering links to the users [Fowler and Stanwick, 2004; Kappel, et al., 2006]. Commercial websites, such as Amazon (www.amazon.com) and Ebay (www.ebay.com), offer categories for users to search for items of interest. Although there has been some research done on generating dynamic navigation facilities for websites, their focus is on tracking a user's browse history to assist information searching [Rucker, 1997; Anupam, et al., 2000], or try to use method like faceted navigation to filter search result [Tvarožek and Bieliková, 2007; Deutch, Milo and Yam, 2009; Harth, 2009], instead of offering different categories or structures for representing information. For example,

D. Deutch, et al. [Deutch, Milo and Yam, 2009] proposes a navigation system called ShopIT to assist a user in searching for items of interest more efficiently. However, it does not generate web pages with new navigation structures. There is currently no research reported on dynamic generation of web navigation interface applications for mass information, except our earlier work reported in [Guo and Chung, 2008].

4.6 Summary and conclusions

In order to get an understand of existing research about automatic generation of WUI for remote control systems, automated WUI generation related literature has been reviewed.

Firstly, to make WUI features understood, WUI design is compared to GUI design as well as web page design. WUI design is different from GUI design because it needs to overcome challenges inherent to web technologies. Also, WUI design is different from content-based web page design because a WUI is a kind of web application. Therefore, traditional GUI design or web pages design methods cannot be used directly for WUI design.

However, design knowledge can be helpful for WUI design. General user interface design guidelines and web application design models are reviewed. Research found that “usability” is an important fact as a design needs to cater for users’ needs. Semantic representation can be used to help the system understand user interface content and structure, so that automatic WUI reasoning and generation can be achieved.

Work on automating generation of machine control user interface is also reviewed. Existing design methods are analysed according to their contribution to different aspects of interface usability for control systems. These are machine, user and process. However, the review finds that current design methods have drawbacks in all of the three aspects. Firstly, the automated generation of UI for different machines are based on retrieving information from static databases, and will be hard to cope with new machine information. Second, existing work tries to adapt UIs for different end user devices, such as mobile phones and PDAs However, they do not allow users to reorganise machine functions freely and generate a personalised user interface with only functions they prefer. Third, some work has been done on constructing high level task-based commands automatically, while ignoring the independent ability of each controlled machine and individual functions that composite these tasks. Hence the design lacks flexibility for future changes and service developments. Also, the user interfaces do not

reflect machine state changes during control processes. Fourth, most design methods are based on GUI design and do not deal with issues related to WUI design for remote control system.

Finally, dynamic navigation related to designs for WUI is reviewed. The aim of navigation is to structure and index information to avoid displaying a large amount of machine information on one WUI page. Although there are some methods for supporting organising large quantity of information, there is little research in organising the category navigation structure for information pages, and there is no research reported on dynamic generation of web navigation interface applications.

WUI design could benefit from some of the design knowledge used in general UI design and web page design, but needs to adapt and further extend these methods according to its own requirements. New methods will also need to be developed.

CHAPTER 5. SYSTEM ARCHITECTURE AND DESIGN

5.1. Overview

As discussed in previous chapters, a number of remote control frameworks based on the internet have been proposed, and a significant amount of work has been done on improving the technology behind remote control processes & command transformation between the user's computer and the control machines. However, as analysed in Chapter 3, current architectures still have limitations. This project aims at offering a many-to-many architecture that overcomes the limitations by supporting the development of remote control services.

In this chapter, a novel web service-based architecture that supports many-to-many control is proposed. The architecture offers multi-access for both users and machines at anytime and anywhere; it also provides a better environment for developing remote control services. The chapter is organised as follows. Firstly, the assumption behind the system architecture is outlined. A general overview of the architecture is then given. This is followed by a description of the machine model and the machine integration strategy used in this research. Support for device knowledge and device description generation is then explained. After that, the support environment for service development is described. The chapter ends with a summary and conclusion section.

5.2. Research assumptions

Remote control over the internet is non-trivial. Besides the integration of machine applications and user interface design, many significant problems, such as security and communication maintenance, need to be solved. It is impractical to solve all of these in a single project. Therefore, for the purpose of this project, the architecture design is focused on the platform for general access for both users and machines. Furthermore, the design is to offer a support environment for developing different kinds of services. In order to achieve these goals and focus on the major design problems, the assumptions for the research are:

- The network is secure and all user access is legitimate.
- All network communication is reliable.

- The ontology used in the system is sufficiently developed to provide the relevant information.

However, it does not mean that these problems should be ignored. These problems are so large and significant that they warrant their own specific projects.

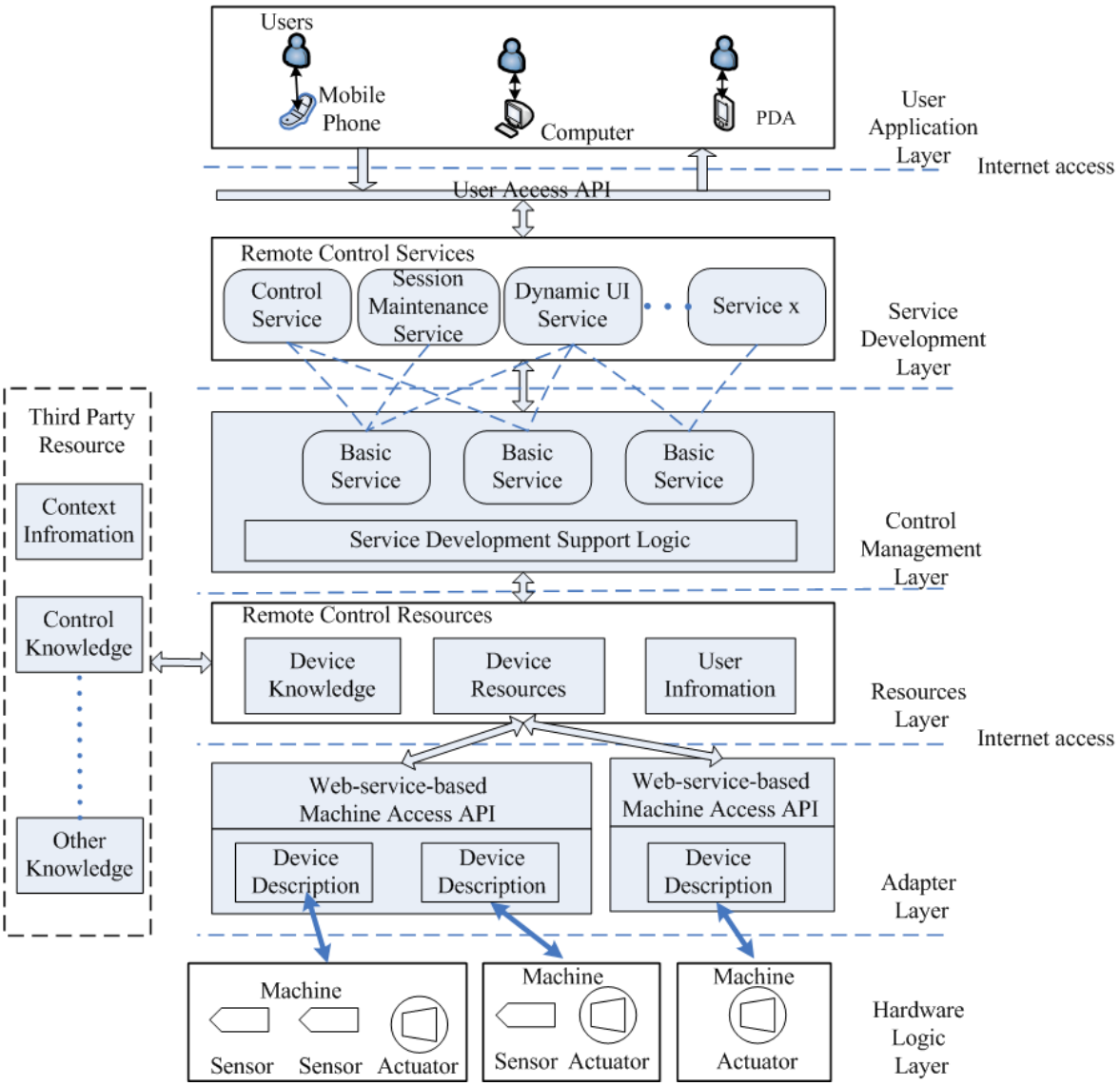


Figure 5. 1 System Architecture Diagram

5.3. System Architecture

Figure 5.1 shows that the architecture of the Remote Control Service System consists of six layers. HLL (Hardware Logic Layer) is where the controlled hardware actually runs. AL (Adapter Layer) adapts the different available machines for the internet access. RL (Resources

Layer) maintains the domain knowledge as well as the control and system related information. CML (Control Management Layer) manages machine control processes and control sessions. SDL (Service Development Layer) supports service composition and new service development. UAL (User Application Layer) implements the user applications. These layers are divided according to their different functions and roles in the system. It is actually a distributed system over the internet. That is, different layers are deployed on different servers over the internet. For example, the connection between UAL and SDL, and the connection between RL and AL are all based on the internet. RL, CML and SDL are located in system control service on the server side, which is also called the service server in the rest of the thesis. HLL and AL are located on the client side server, which links with the controlled machine. Third party resources that are independent from the system can be used to support the system as additional resources in RL, as well as service development in SDL. These are included in the architecture design to show the possibilities for extension for the system. A general introduction to these layers is given below.

5.3.1. Hardware Logic Layer (HLL)

HLL is the layer for the actual hardware machines to be controlled. A machine may be composed of different kinds of sensors and actuators. Actuators can perform certain actions, while sensors can be used to monitor different machine state variables. Some machines contain both sensors and actuators, and others may only have sensors or actuators. Machines may either be grouped via local networks or connected to the network individually.

5.3.2. Adapter Layer (AL)

Each machine can access the internet via an adapter application in AL, which is also called "adapter" in this thesis. Physically, an adapter may be embedded within a machine. It may also be located on an application gateway or a computer that links a group of machines. As shown in Figure 5.1, an adapter contains two parts: Web-service-based Machine Access API (WMAAPI) and Device Descriptions. WMAAPI is used to pack the machine access functions using web services. Therefore, the machine functions can be registered, found and invoked by the remote control service system. Device Description is used to record hardware states and control functions in a semantic way. Therefore, the machine information can be passed to the server and understood by the system.

5.3.3. Resource Layer (RL)

There is a lot of information needed to be used initiate and continue a remote control process. Before the control session starts, the information needs to be organised and made available for access. RL is dedicated to storing and maintaining remote control information as available resources. As illustrated in Figure 5.1, it contains three remote control resources: Device Knowledge, Device Resources, and User Information. Device Knowledge contains knowledge about device structures and functions. The knowledge helps the system to analyse how to control the machine and update the machine states. Device Resources is an abstraction of the actual devices and their control related information, which will be explained later. User Information is information registered by users. It contains user authorisation information and information related to user control requirements, such as the machines and functions that they want to access. Third Party Resources, such as Context Information and other Control Knowledge, can be considered as an extension of the RL. Third Party Resources are shown inside the dotted lines in the diagram because they are not provided as part of the core system. These resources offer APIs that CML that can be used to support the development of control services.

5.3.4. Control Management Layer (CML)

CML may be considered as a central control layer which organises control related resources using "Service Development Support Logic" to offer actual basic service implementations for use by the layer above. It also collects and maintains control related data. Some major services provided by this layer are:

- **User registration information service:** provides functions for registering, searching, maintaining and updating control related data registered by users.
- **Machine information initialisation service:** provides functions for handling machine pre-registration information and initialising machine related description;
- **Control session management service:** provides functions for handling machine control sessions.
- **Machine data management service:** provides functions for maintaining and controlling machines for control sessions.

This layer aims at offering basic service support for new service development. The details of the functions will be explained later.

5.3.5. Service Development Layer (SDL)

The system offers different kinds of control services that users can access via the internet. SDL plays the role of service development environment and allows different services to be developed. A difference between this proposed design from other existing designs is that services in this layer are separated from the basic services in CML. Services in CML are aimed at basic and commonly used functions. Services in this layer aims at offering intelligent services for users. Examples are session maintenance service and dynamic UI service. The role of session maintenance service is to supervise the transactions between a user and the machines under control by the user. In the event that the communication connection is broken the controlled machines will be brought to a safe state automatically and the session will be resumed when the communication connection is re-established. The dynamic UI service will allow different user interfaces to be created dynamically to suit specific user requirements. These services can be generated by service composition or direct creation. After publishing, services at this level can be used by UAL.

5.3.6. User Application Layer (UAL)

UAL is located on the user side. A user can access the system using devices such as mobile phones, computers or PDAs. For some services, there are no specific programs required, except the use of an internet browser for logging into the system and accessing the services. Using services via the internet, different users can access the same machines as long as they have the required permission.

These layers cooperate together to achieve remote control over the internet. Using semantic description based method as well as web services in AL, machine functions in HLL can be accessed via the internet and the control information can be understood by the system. The machines as well as other control related resources will be gathered in RL and later be organised and packed as basic services in CML. With support from CML, the system will offer better service development support for further services development in SDL. Therefore, the users will be able to access various remote control services via different methods in UAL.

5.4. Machine model

Given the architecture, an interesting question is "How can the layers work together to achieve the research goals?" A key point is for the system to reason about how the machine functions. A machine is made of sensors or actuators. It is difficult for a system to understand the machine structure and features if there does not exist a model that describes the machine components and functions clearly. As explained in Chapter 3, previous designs lack a clear machine model. A machine model is introduced in the system, which forms the basis for the whole design. This model is the link between the system and the hardware machines and enables the system to reason about the machine abilities.

Before the model is introduced, there are some terms that need to be defined. These definitions are abstracted using Object-Oriented analysis [Booch, 1994]. These definitions will be used to describe the machine model and in the rest of the thesis.

- Machine: a general term referring to plants, equipments, facilities and devices that can be controlled by the remote control system.
- Component: a part that makes up a machine. A component is operated by some actuators and sensors.
- Actuator: an independent unit in a machine component which has a group of atomic functions for moving or controlling the machine. The atomic functions will be executed by a single mechanism.
- Sensor: an independent unit in a machine component which has atomic functions for measuring a state variable and converting it into a signal.
- Instruction: an atomic low level command that can be executed by actuator hardware.
- Command: a high level statement that can be processed and executed by the machine. It can be made up of a sequence of instructions.

According to the definitions, the machine model is shown in Figure 5.2. A machine has different commands, components as well as the sensors for monitoring purpose. "0..*" means the number of the elements ranged from 0 to more than 0 while "1..*" means the number of

the elements ranges from 1 to more than 1. Each component may contain actuators or sensors or both. These actuators are the atomic elements which will invoke real hardware functions. There are different types of actuators, such as motor, switch, and valve. Also, an actuator has at least one instruction. Each instruction has a group of description attributes, such as command name, type, and current state. In the control process every attribute has a corresponding atomic control command. Meanwhile, these sensors are the atomic elements to capture state data of the machine. Each sensor has a sensor description including information such as sensor name, type, value and range. Commands can be preset in the hardware as atomic commands, or they can be added to the machine model as a kind of composited service later when needed. However, only atomic commands are discussed in this thesis. A module for command composition is not developed. The reason is that once a high level command is created, it can be treated in exactly the same way as an atomic command in terms of transformation, processing and execution

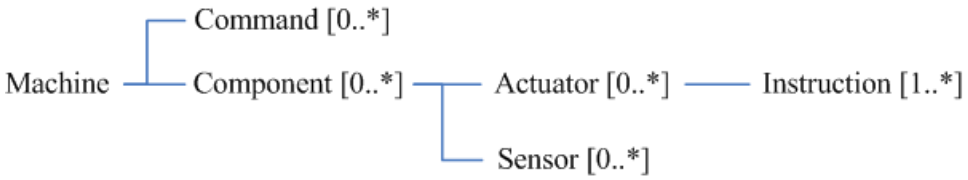


Figure 5. 2 Machine Model Diagram

Among these terms, instruction is the most dynamic one. Although it is defined as atomic functions that can be executed by machine hardware, it is hard to define how low level the instructions are. The reason is that it all depends on which control mode the system uses and how intelligent the controlled machines are. Some instructions can be processed by actuators or sensors directly, while some instructions are actually processed by computer embedded in the hardware. According to [Sheridan, 1992] there are three types of operator control: fully manual control, supervisory control and fully automated control. As it is illustrated in figure 5.3, the instructions can be as simple as atomic low level command that can directly control the actuators in “manual control”. At the same time, the instructions can be as complicated as some high level tasks that can be executed by the computer which controls the low level sensors and actuators in “supervisory control”. In “manual control”, there is no direct sensor data can be captured by the actuator so that normally there is no parameter contained in the instruction sending to the actuators. These instructions are supposed to change actuator states

directly. For example, turn on the light. When the instructions are activated, the result of machine state related data changes, such as temperature and light intensity, will be reflected through sensors. For instructions in "supervisory control", it is possible to contain some parameters in the instructions. These instructions can be used to request the computer to change the actuator to certain kind of conditions automatically. For example, turn up the brightness of a light to certain level. "Fully automatic control" is discussed in this thesis because it belongs to another research field. Command discussed in this thesis belongs to the situation in "supervisory control". The difference between command and instructions is command need break down to several actions executed by actuators and sensors.

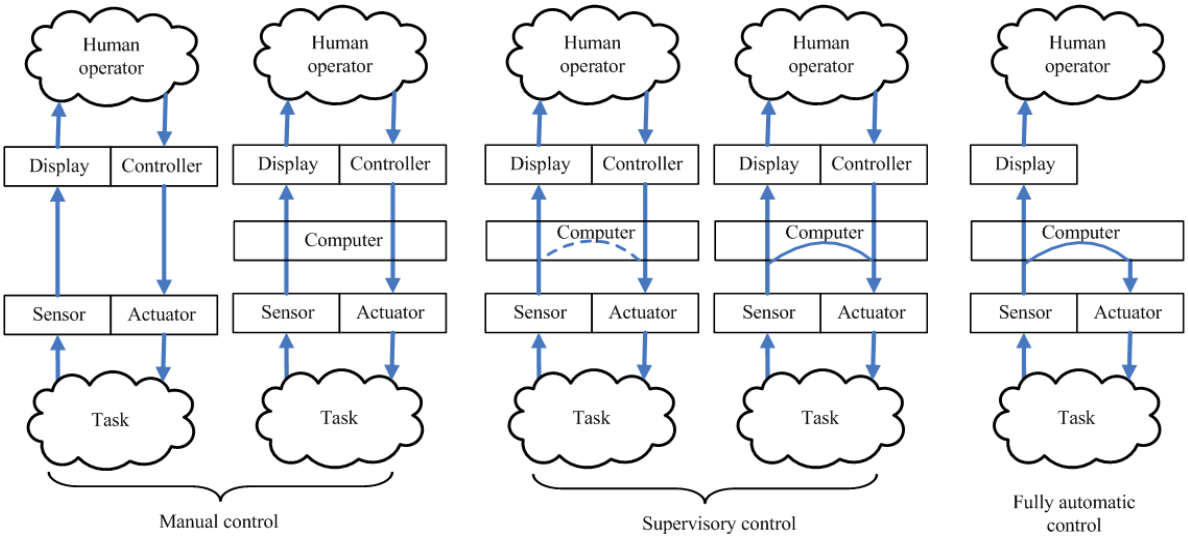


Figure 5. 3 The Spectrum Of Control Models
Adapted From [Sheridan, 1992]

To better describe the attribute of the instructions attributes called "type", "value" and "state" are used. "Type" indicates the type of the instructions. "Value" indicates the value of the instruction and "state" indicates whether current instruction is active or not. Basic instruction type discussed in this thesis is "state". "State" represents a group of instructions that are mutually exclusive. For example, the instruction "switch on" and "switch off" for a switch is a pair of instructions belonging to the type "state", with value "ON" and "OFF" correspondingly. There may be more than two instructions exists for the group of instructions. If they are of the type "state" only one instruction can be active at a time. The instruction active in the group has a state "true", while the others will have the state "false". Similarly, for instructions with parameters, the "type" will be used to indicate the data type such as "integer" or "float".

"Value" will indicate the target value of the instruction while "state" indicates whether the instruction is active or not. However, currently the instructions discussed in this thesis are only statements without any parameters, since the process of handling these instructions are all the same except the details to drive the actuators. The following content will focus on how these instructions are represented, processed and executed in the system.

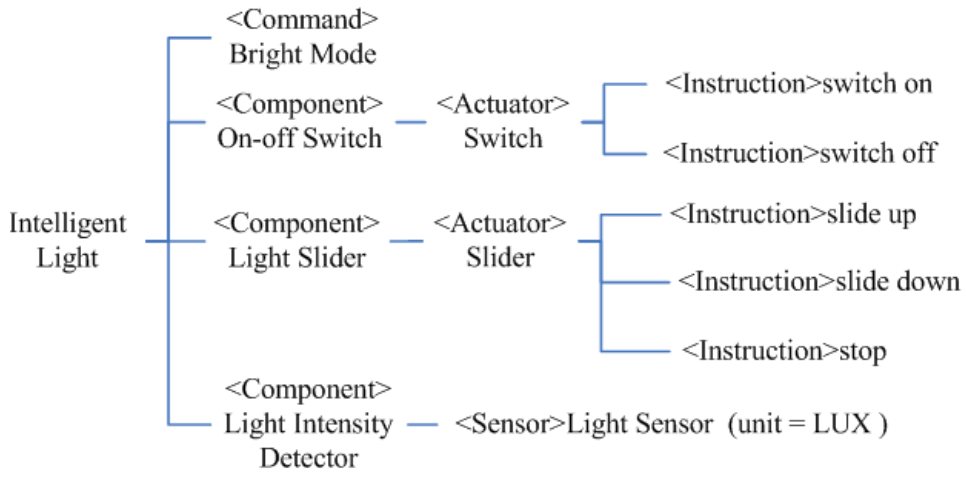


Figure 5. 4 Machine Model For An Intelligent Light

An example model for an intelligent light is shown in Figure 5.4. The intelligent light has three components, the first component is "On-off Switch", which is used to turn on and turn off the light. The actuator to execute the command is "Switch" with two instructions – "switch on" and "switch off". If the state of "switch on" is true, then the state of "switch off" is false. The second component is a "Light Slider" which can change the brightness of the light using an actuator called "Slider". It can be turned up to make the light brighter, turned down to make the light darker and stopped to maintain the brightness. Therefore, the "Slider" has three instructions – "slide up", "slide down", and "stop". The third component is a light sensor called "Light Intensity Detector" to check the brightness of the light. With a value represented using the unit of LUX [Rowlett and the University of North Carolina at Chapel Hill, 2008]. If the light is working correctly, when the slider is "slide up", the value of the light sensor will be increased and if it is "slide down", the value will be reduced. Finally, there is a command "Bright Mode" for the light. This command means switch on the Switch, and slide up the Light Slider to the highest value it allows.

This model focuses on machine attributes representation. It describes machine components and functions as well as the relations between them, so that the machine model can be understood by the system and the system can control the machine to carry out a task. For example, if the "Intelligent Light" is described separately as a "Slider", a "Switch", and a "Light Intensity Detector", it is not easy to understand that they belong to the same unit. With the machine model of the light, it is easier to understand that the "Intelligent Light" can be turned on and turned off using a "Switch". The brightness can be slide up and down using "Slider". The brightness of the "Intelligent Light" can also be obtained from the "Light Intensity Detector".

5.5. Semantic Description Based Machine Operation

The proposed system aims at controlling all kinds of machines distributed over the internet. The first problem for building such a distributed system is integrated: how to operate different machines from different places. The operation does not only mean make these machines accessible through the internet, it also means they can be controlled correctly, and return their states to the system when needed. Because different machines have different functions, it is not possible to invoke machine functions dynamically and correctly without knowing what they are. Also, different machines have different state variables it is impossible to get machine states variables back without knowledge of the machine state data.

A semantic description based method is introduced in this section to solve these problems. These semantic descriptions are created according to the machine model proposed in "Machine Model" section. As discussed, in this design, machine model will bring a convenient way to help the system understand machine components and functions. The following content in this section will explain how these semantic representations are used to achieve this.

To help the system to reason how to control and monitor a machine, there are three things that need to be described clearly. Firstly, the machine model which explains what kind of components the machine has, as well as what kinds of functions are available. With an understanding of the machine model, the system will know how to operate the machine.

Second, the system should know current machine state, as well as the attributes of these states. The attributes include state data type, range and other related information which describes the data as well as its relationships with the machine, so that the system will obtain machine state

data and handle it in an appropriate way. Thirdly, there should be a description about the hardware functions as well as its relation with machine model. It is important because to the instructions and command need to be mapped to the real hardware functions of the controlled machine.

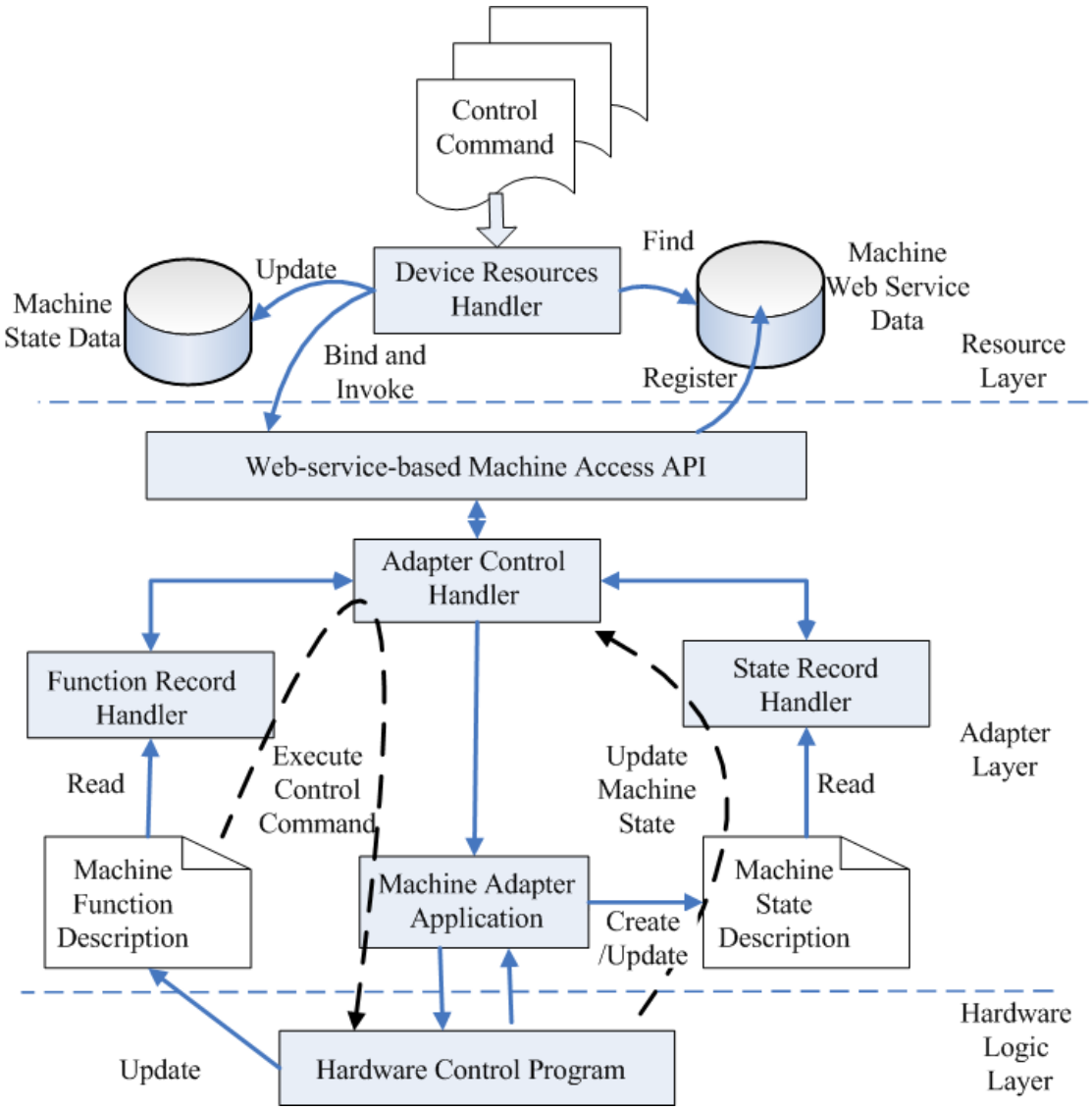


Figure 5. 5 Semantic Description Based Machine Operation Diagram

In proposed architecture, a Machine Function Description (MFD) is designed to describe the hardware functions as well as their relationships with corresponding machine model. Meanwhile, a Machine State Description (MSD) is designed to describe the detail machine model and machine state. MFD and MSD are deployed in AL. To maintain these descriptions and exchange the data between machine side and server side, the AL is designed as shown in

Figure 5.5. The AL application consists of some light weight modules. Web-service-based Machine Access API (WMAAPI) is the module that offers web service APIs that can be invoked by the remote server. Also, it contains APIs for some of the remote service functions that the machine application can use. Below it the Adapter Control Handler (ACH) performs as a central controller in AL and will invoke different modules to finish the task when it receives commands. Function Record Handler (FRH) is the module that can read information from MFD. State Record Handler (SRH) is the module that can read information from MSD. Finally, Machine Adapter Application (MAA) will be the module that executes the hardware functions dynamically according to the control message, and maintains machine state in MSD. Each machine will have an MAA. The aims of an AL application, also called an adapter in the following content, are listed follows:

- Initialise machine descriptions, and register control service information on the server.
- Process data exchange from the control server. Handling communication messages between the server and controlled machine.
- Exchange the control information with machine hardware program when a control command is received.
- Monitor the machine state change and maintain related description. When a state checking command is received, return state information back to the remote server.

As shown in Figure 5.5, on the service server side, applications in RL will maintain registered machine services in Machine Web Service Data (MWSD). A module called Device Resources Handler (DRH) will than find machine services description in MWSD so that later in the control session the corresponding machine services will be bind and invoked using web services technology and machine states stored in Machine State Data will be updated during the control process.

There are three important components in this structure that need to be highlighted: WMAAPI, MFD and MSD.

5.5.1.WMAAPI – APIs for handling semantic information exchange

WMAAPI functions as a web service adapter for a machine. It offers web service based APIs that allow messages to come into and go out of the machine. Using semantic data exchange method, WMAAPI is able to make the machine communicate with the service server. The rest of the section will explain the details of the design.

Using WMAAPI, instead of directly invoking the hardware functions, data exchange will be processed using messages. Compared with directly invoking functions message transmission will be more flexible and will keep machine side application logic independent from server side logic. The kernel of the method is to achieve semantic information exchange. First of all, the message needs to be semantic. Using a language like XML, the machine control commands as well as returned results can be represented in a semantic way, so that they can be understood by the system. Moreover, the messages need to be designed according to machine model. Using the same machine model, it is easier for the system to generate messages according to control sessions and find the target operations that the user needs. With the help of the message, the control and monitor process can be considered as the data exchange. For example, during the control process, the service server will first generate machine control command. Later, by using the message handling method available in WMAAPI, the result will be returned to the server as a message. Similarly, during the monitor process, the server will send the request messages to the WMAAPI. WMAAPI then forwards the requests to the MAA accordingly, gets the feedback data as a result of the request, and returns it to the server.

To receive and send these messages, WMAAPI will need to have the ability to communicate with the service server. Current designs choose web services as the middleware to link the hardware application to the system. As analysed in Chapter 2, compared with many other middleware technologies, web service can successfully separate the API description and its executing environment. Therefore, it will benefit the system by reducing the restrictions for hardware and server side development. That is, there is no need to develop the system on a certain development platform or language. Another advantage is each machine will be considered as an independent resource over the internet. Using WMAAPI, these machine services will be able to register to MWSD in RL. Users can register their machines to the service server from anywhere via the internet. These registered machines will then be

available for control. Machine information will be initialised during the registration process, which is also called “machine registration” in this thesis.

To handle these messages, WMAAPI can be designed as twe types of message handlers, which forward the messages to the corresponding modules and get the results back to the modules that request it. In WMAAPI, there are two kinds of APIs. One is designed to be used by the service server. This kind of APIs contains two types of message handlers. One is the control message handler, and the other is the monitor message handler. The control message handler will be invoked when the server needs to control the machine. As shown in Figure 5.4, in the “Execute Control Command” process, according to different control messages, the message will be passed to different machine applications by ACH. After checking MFD, the MAA will execute target the hardware functions and update MSD. Another handler is the monitor handler. When the sever wants to update a machine state, the monitor handler will be invoked. As shown in the “Update Machine State” process, the logic behind the handler will get the required machine state via MSD and return it to the server and update the Machine State Data on the server side. The other kind of APIs is the public remote APIs that are implemented on the service server, which can be used for machine initialisation. These APIs offer services for machines to check control related knowledge, and generate initial MSD automatically for the machine that the user wants to register. The purpose of this kind of APIs is to use the machine knowledge to identify newly registered machines. This will be further discussed in a later section.

Compared with other designs, the proposed design is more flexible and can cope with future changes. The reason is that the design separates the execution function from its logic description. That is, this design uses messages to describe the functions the system wants to achieve, while using a module which understand the messages to pass through to the corresponding services. Therefore, the communication between machines and the service server can be considered as a process of semantic data exchange. As soon as the message is generated according to the same machine model, the content can be changed to adapt to different machines and to achieve different control goals. When the control requirements change, the server only needs to create new messages to describe the task, instead of reprogramming the whole control process. In this way, different machines can communicate with the service server smoothly.

5.5.2.MFD – Hardware ability description for controlled machines

When control commands are sent to the adapter, it will be forwarded to MAA and then corresponding hardware functions can be invoked. Even when semantic messages are understood by the adapter but without the understanding of corresponding hardware functions, it is difficult to execute the correct commands. MFD is designed to solve this problem. This section will introduce how MFD works to solve this problem.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <machineID>000017</machineID>
  <machineType> Intelligent Light</machineType>
  <manufacturer>Panda</manufacturer>
  <function_maps>
    <function name="Brightmode">
      <hardwareFunction>
        <name>updateUWORD</name>
        <parameter>{0x00096857, 0}</parameter>
      </hardwareFunction>
    </function>
    <function name="LightSlider_Slider_state_UP">
      <component operatedby="Slider" value="UP">LightSlider</component>
      <hardwareFunction>
        <name>updateUWORD</name>
        <parameter>{0x00096820, 0}</parameter>
      </hardwareFunction>
    </function>
    <function name="LightSlider_Slider_state_DOWN">
      <component operatedby="Slider" value="DOWN">LightSlider</component>
      <hardwareFunction>
        <name>updateUWORD</name>
        <parameter>{0x00096820, 1}</parameter>
      </hardwareFunction>
    </function>
  …
```

Figure 5.6 Machine Function Description Example For An Intelligent Light

MFD can be considered as a “map” for the MAA to find the right function in the controlled machine. It is represented using semantic language, so that it can be understood and analysed by MAA. MFD is designed to be generated by manufactory that produces the machine so it is hard to find a unified structure for MFD because different machines have different hardware

functions. For example, some machine will have to write data on certain memory address while some machines will invoke certain kind of hardware functions. No matter what it is, the rules for creating MFD are the same. That is, there are three things need to be described: “function name”, “component” “hardware function”. “Function name” is the name of the function which can be referenced in the control message. “Component” indicates the actual actuator that operates the function, including its state. “Hardware function” is the description of actual hardware function that invoked. This description will be various according to different control machines.

An example of MFD is given in Figure 5.6. There is some general information which is initialised when the machine is produced: <machineID> represents a unique machine identifier; <machineType> represents what the machine type is; <manufacturer> represents the company that produces the machine. Additionally, there is important information which will help to map control hardware function to corresponding machine actuator. Each sub node <function> of the node <function_maps> will map hardware functionality using its attributes. The “name” attribute represents the hardware control function name that can be identified by MAA. The sub node <component> will specify the actuator or sensor and corresponding attributes it is aiming to change or get for the hardware. The node <hardwareFunction> will include the details of the hardware functions that the machine can execute.

In the example, the description represents the function mapping information of an intelligent light with ID 000017. From the basic information it is not difficult to ascertain that the intelligent light is produced by a company called “Panda”. There are five atomic hardware functions for the intelligent light control, such as “LightSlider_Slider_state_UP” and “LightSlider_Slider_state_DOWN”. From the function map, it is easy to find “LightSlider_Slider_state_UP” will be invoked when the component named “LightSlider” is activated and it aims to change the attribute of the actuator “LightSlider” value to “UP” with the hardware function name “updateUWORD” with a pair of parameters ”{0x00096820, 0}”. “0x00096820” represents the memory address and “0” is the value need to be set in this address when the instruction is activated.

With the help of MFD, the execution of control commands can be processed as shown in Figure 5.5. When a control command is received by the adapter, ACH will require FRH to analyse MFD to get the right hardware function description and send it to MAA. Then, MAA

will process the information in a message to Hardware Control Program (HCP). Once HCP get the message, it will invoke corresponding hardware functions, such as reading and setting certain memories, to finish the control task. Although the content of the node <function> can be changed according to different manufacturers, as soon as they are pointed to the right hardware function and can be understood by the hardware control program the command can be processed correctly.

This description is supposed to be generated by the manufacturer who produces the machine. As soon as it can be downloaded to local machines using the machine driver program, it will be invoked and updated by FRH. The hardware upgrade program will responsible for future updating. The fact behind it is the machine hardware function as well as their relationships with the machine components can be described semantically according to the machine model. In this way, no matter how the control logic changes the system is able to understand and dynamically invoke machine hardware. Even if the hardware function changes in the future, no more change is needed except updating the description.

5.5.3.MSD – Semantic description for machine attributes and states

Methods are required to capture the machine state changes for the system. It is difficult to collect state information from different machines without the understanding of machine model and state attributes. To solve these problems, MSD is used as a description for detail machine model and states. Similar to MFD, MSD is also generated according to the machine model proposed in this research. But MFD is more focus on the details of machine model and machine states. An example of MSD represented using XML for an intelligent light is given in Figure 5.7.

Each MSD represents a machine state in two parts. The first part includes some basic information, for example <machineID> represents machine ID, <machineType> represents machine type and <location> represents current machine location. An optional node <controlled_by> represents the user who is currently controlling the machine. The second part is a description of the machine state, which is included in the node <machineState>. According to the machine model, there are the following type of nodes: <command> records command information; <componentState> keeps a record of the component state information; <sensor> node records current sensor value and attributes; <actuatorState> represents the

actuator state; <attribute> represents the detailed attribute state inside each actuator, which includes available instructions as well as their active state. There is an attribute “state” that shows if the command is active or not. Currently, it shows “false” meaning this command has not been activated yet. The attributes inside the node <attribute> illustrate the state of machine instructions: “type” is the type of the attribute, which indicates the relationship for the attributes within the same group; “value” is the value of the machine actuator which will have semantic meaning for future ontology development; “state” represents if the attribute is currently active. If the “state” equals “true”, it means the attribute is currently active. If it is “false”, it means the attribute is not active. The node name of the <attribute> is the defaulted name of the action.

The code inside <sensor> is represented using JavaScript Object Notation (JSON) [JSON, 2011], a lightweight semantic language. It get value of the attributes: "unit" which means the unit of sensor value; "sensorValue" which is the value of the sensor; "stateType" which is the sensor value type; "sensorRange" which is the condition of the sensor value, including “min” (minimal value) and “max” (max value); "sensorType" which is the type of the sensor; "sensorName" which is the name of the sensor in the machine.

According to the example MSD, the intelligent light with machine ID 000017 is located in ROOM3. It is being controlled by a user “Kate”. The intelligent light include two sensors, one sensor is a kind of temperature sensor called “light temperature detector”. Another one is a light sensor called “light intensity detector”. Both of them have a value of type “INT_NUMBER” which means it is an integer. The value of the temperature sensor is 30C, which is actually means 30 degree centigrade. The value of the light sensor is 100 LUX. Also, the machine has two components “LightSlider” and “Switch”. “LightSlider” will be controlled using actuator “Slider”. The “Switch” will be controlled by “OnOffSwitch”. From the description of the detail attributes, it is not difficult to find the “Slider” is now in the state of “UP”, because the “state” of the node “turnUP” is “true”. At the same time, the “OnOffSwitch” is “ON”. The result is caused by the user sending commands to “turnON” the “OnOffSwitch” and “turnUp” the “Slider”. According to the machine model, the commands will be mutually exclusive with the commands in the same group. Therefore, only the attributes “turnDown”, “stop” and “turnOff” can be triggered by the next command.

As illustrated in Figure 5.5, MSD is created automatically by MAA when initiating the machine. The details for the generation will be explained in the next sections. Later, it will be updated by MAA. The process of updating machine state is actually reading data from MSD. Then the data will be sent back to the DRH, and update the Machine State Data server side.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
   <machineID>000017</machineID>
   <machineType>Intelligent Light</machineType>
   <location>ROOM3</location>
   <controlled_by> Kate</controlled_by>
   <machineState>
      <command state="false">Bright mode</command>
    <componentState>
        <sensor>{"unit":"C","sensorValue":"0","stateType":"INT_NUMBER","sensorRang
      e":"{\"min\":\"-5\",\"max\":\"800\"}","sensorType":"Temperature
      Sensor","sensorName":"light temperature detector"}</sensor>
      </componentState>
      <componentState>
        <sensor>{"unit":"LUX","sensorValue":"100","stateType":"INT_NUMBER","senso
      rRange":"{\"min\":\"0\",\"max\":\"10000\"}","sensorType":"Light
      Sensor","sensorName":"light intensity detector"}</sensor>
      </componentState>
      <componentState>
        LightSlider
        <actuatorState>
          Slider
          <attribute type="state" value="UP" state="true">turnUp</attribute>
          <attribute type="state" value="DOWN" state="false">turnDown</attribute>
          <attribute type="state" value="STOP" state="false">stop</attribute>
        </actuatorState>
      </componentState>
      <componentState>
        Switcher
        <actuatorState>
          OnOffSwitch
          <attribute type="state" value="ON" state="true">turnOn</attribute>
          <attribute type="state" value="OFF" state="false">turnOff</attribute>
        </actuatorState>
      </componentState>
   </machineState>
</root>
```

Figure 5. 7 Machine State Description Example For An Intelligent Light

Using MSD, the system can fetch the required data as well as related information easily without direct communication with the hardware every time a request is made. In that case, or even if multiple users want to access the same machine, the system is able to respond to the requests quickly by referencing the same MSD. The purpose behind the design is to separate the hardware level control as much as possible from the control system. For a remote control system facing hundreds of thousands machines, the separation will make the system more adaptable to different machine attributes. In this way, different machines can be operated via the system without worrying about their differences.

5.6. Knowledge-based machine information automatic retrieve

Semantic descriptions such as MFD and MSD are created according to a unified machine model. However, different machines can have different descriptions. It will be time inefficient if these descriptions need to be manually created every time. If the machines information can be automatically retrieved, it will be much easier to generate and maintain these descriptions later. In this section, a knowledge-based machine information retrieving method is introduced to solve this problem.

Domain knowledge is needed to support the system automatically retrieving structure and function information for different machines. Knowledge is needed to help the system understand individual machine components and functions so that the system will know how to control the machine. Also, the knowledge is needed to help automatically generate and maintain the machine description and support the development of intelligent services.

Figure 5.8 shows how the domain knowledge data is deployed in RL. The knowledge is needed when machines are initialised or updated in description models. Using this knowledge, the system will be able to identify different machines, reason about machine components and available functions according to the machine model and generate machine related descriptions. On the RL, there are three major kinds of data that are involved in the process:

- **Machine knowledge resource:** Machine Ontology, Sensor Ontology and Actuator Ontology can be considered as knowledge to help the system to understand and reason about machine model related attributes;

- **Context Information:** the information collected to describe the machine control environment, such as location;

- **User Information:** the user register and user control session related information.

On the AL, as analysed in the previous section, there are two types of descriptions (MSD and MFD) that need to use the knowledge as they are created, updated and reasoned. The data cooperates in the system and helps to achieve automatic machine structure and function retrieve.

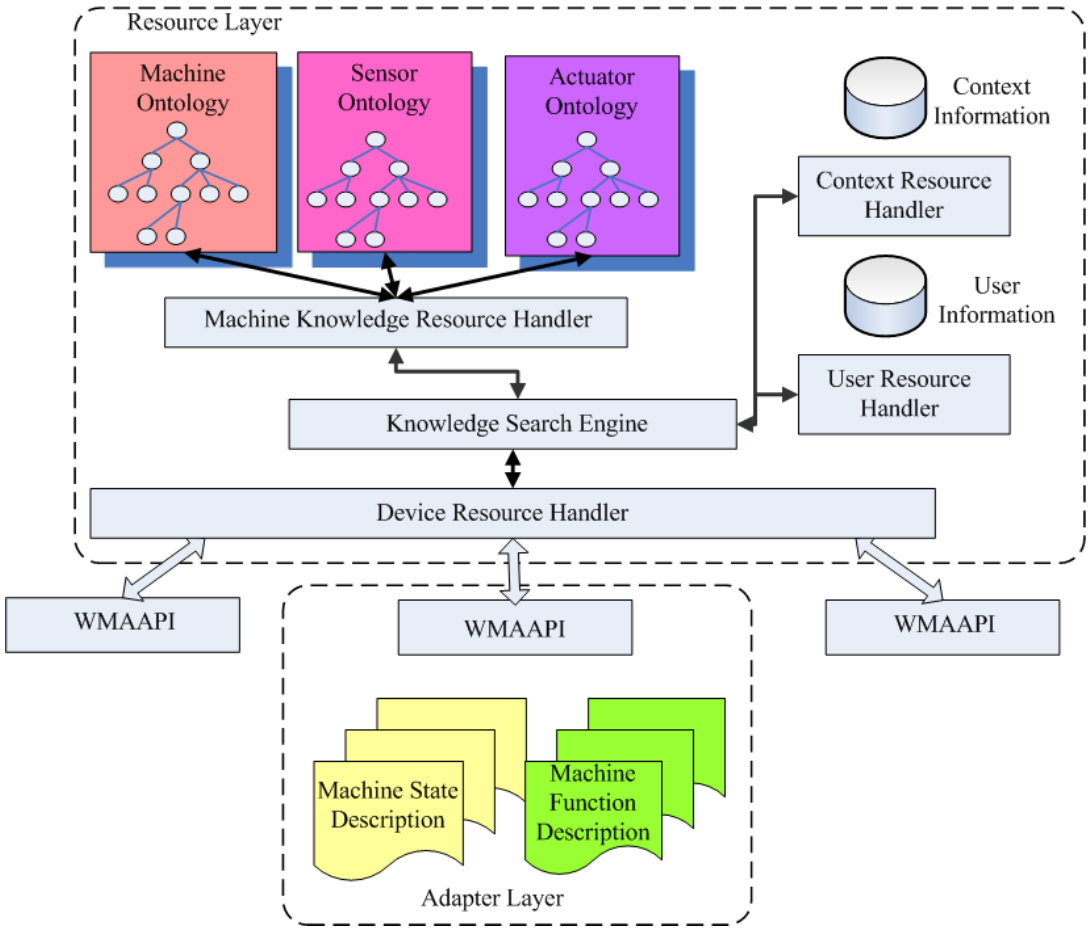


Figure 5. 8 Machine Knowledge Deployed In Resource Layer

Machine Knowledge Resource is the key to the method. The three ontologies in the resource have different roles in describing the machine data:

- Machine Ontology maintains the structure and hierarchy knowledge of the machine. It describes the machine type category. For example, “home facility” has a machine type of

"dish washer", and "Hot Point 365" is a kind of "dish washer". It also describes what kind of commands, components and sensors the machine will contain.

- Sensor Ontology maintains the knowledge of the sensor category. It also offers knowledge on detailed sensor attributes. These attributes describe sensor type, sensor name, and sensor value type, unit, initial condition and limitations.

- Actuator Ontology maintains the category for actuators. It also offers knowledge on attributes for each actuator. More over, it contains the knowledge of the detail instruction information of the actuators, as well as the attributes for the instructions.

The ontology knowledge can be understood and analysed by the computer. It offers support for further reasoning about the details of the machine components and functions. Therefore, it will be easier for the system to reuse the data and reason about the control related attributes for a machine when it accesses the system.

Figure 5.8 also shows that each data resource is controlled by a handler. For example, machine related ontology resources is managed by the Machine Knowledge Resource Handler, while context information and user information is controlled by the Context Resource Handler and the User Resource Handler correspondingly. These handlers offer APIs for outside modules to use the knowledge. Knowledge Search Engine (KSE) is an important module in RL as it is in charge of reasoning about machine knowledge as well as generating machine control related description. One thing needing to be highlighted is after the integration the machines, it can be considered as a special kind of resources available for RL to use. These machines are connected via Device Resource Handler. By using the knowledge analyse method in KSE, the accessed machine can be identified and descriptions of the machine will be generated and sent to the client machine application.

An example process of identifying and automatically retrieving machine information is illustrated in Figure 5.9. The diagram shows a process when a machine is accessing the system, how the machine structure and functions are automatically retrieved and how the machine control description is generated. Before machines are registered, there is some pre-generated information called Machine Initial Information set by the user or manufactory. When a machine is registered, the pre-generated Machine Initial Information will be sent to the KSE by invoking the web services in WMAAPI, which will obtain the basic information of the

machine, such as machine type and machine ID. KSE checks the Machine Ontology to find if there is any information on this type of machine and obtained basic machine component information from the Machine Ontology. It will then search in the Sensor Ontology and the Actuator Ontology to find sensor and actuator information for the machine. After that, combined with the User requirements from the user data, the required machine model is obtained. Now the system will know what kind of components, including sensors and actuators, the machine has and what kind of instructions can be used for machine control. After checking the Context Information, the necessary machine control context data, such as location, will be added in. Then, according to particular Description Rules, the new description is generated automatically.

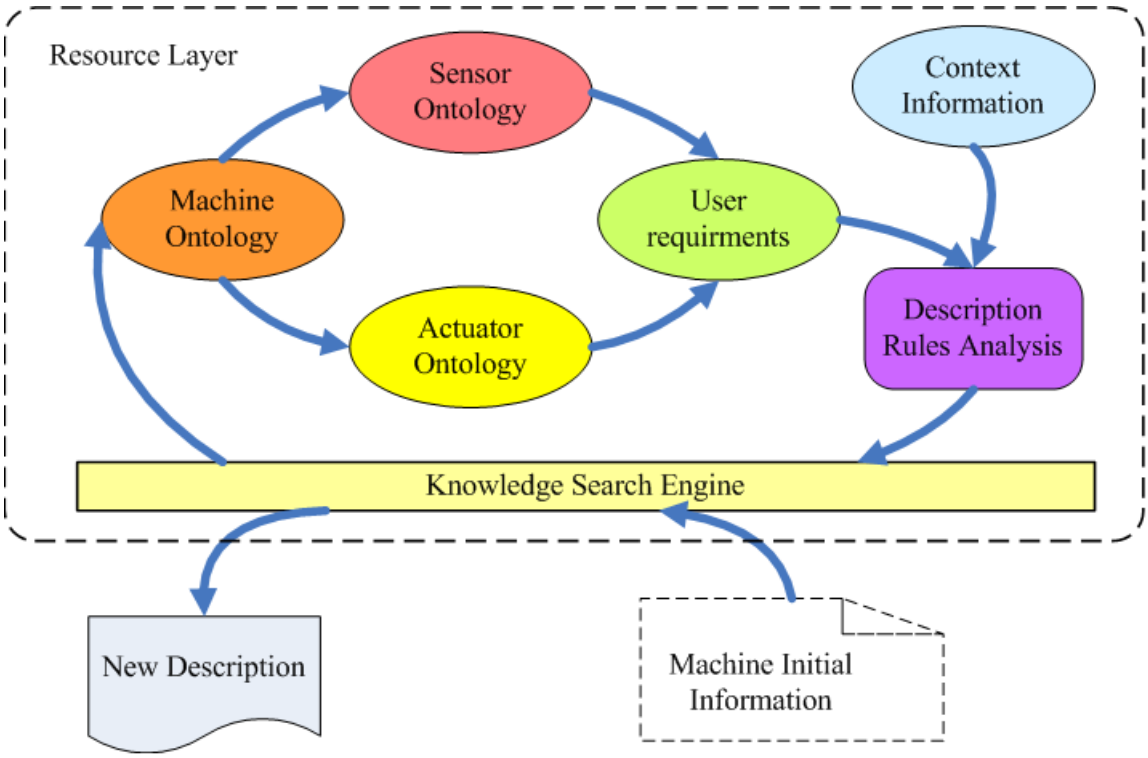


Figure 5. 9 Data Flow In The System During Service Developing

The idea behind this design is to reuse the machine knowledge to reason about individual machine models. This kind of design will benefit not only the system, but also future service developers. It reduces the adaption work for different kinds of machines so that makes the integration of new machines easier for the system. It also offers opportunities for better service development because the knowledge can be reused in other services to support the reasoning about machine control related attributes. Furthermore, the knowledge can be grown and

improved in the future without influencing existing work. Therefore, this kind of design is flexible enough to cope with future changes.

5.7. Control service development environment

It is important that services can be used by other services to develop new services for users. A simple control service could be just to turn on and turn off a light. Directly invoking the light switch actuator seems to be a good way to achieve this. However, if there is a need to offer a user interface to control a hundred lights with different brands, locations and brightness requirements, then the simple service would be clumsy to use. Therefore, the system should facilitate the putting together of high level services by combining basic services without having to worry about how to achieve the basic services.

5.7.1. Basic service support

The proposed architecture aims to offer such a service development environment. There are five basic services that almost all the remote control services need.

i. User registration information service

Normally, before using different services, it is necessary to check the user is legal. Also, the service needs to check what kind of machines and functions the users are allowed to use. User registration information service offers an API to solve these problems. User registration information service offers two groups of APIs. The first group of APIs is used to manage basic user register information. It is in charge of user authorisation, new user registration and maintaining basic user information. The second group of APIs will help to manage user register machine information. For example, it will help to manage what kind of machine can be controlled by the user and what kind of machine functions are required by user. Using this service, it will be much easier to register, search and maintain user data in the new service.

ii. Machine information initialisation service

To develop new control services, machine control knowledge is needed. The machine information initialisation service will offer APIs for retrieving machine information automatically, which was introduced in the previous section. New services can use this service as a search engine that can help the service logic to check machine model. Also, this service

will help new services generate machine control related semantic representation according to their needs. In this way, the necessary new machine data will be initiated using this service while the new service will just focus on the rules to generate the description and other new developments.

iii. Control session management service

Usually, for each control service, there will be a control session used to tracking user activities. The control session management service is a basic service but very important to the control process. A control session usually records control information related to the current task the user wants to achieve. For example, the target machine names, types and machine states will be recorded. To keep and maintain the control session, the control information needs to be recorded and synchronised by the system. Further control actions can be carried out based on current control sessions. Therefore, the control session records always need to be managed according to the session lifecycle: registered, maintained, updated and finally deleted when the control session finished. Control session management service offers services to help service developers finish these processes. Using this service directly, the control logic does not need to consider the details concerning the session lifecycle management any more.

iv. Machine control management service

A control service usually needs to control the machine and update machine control state. As explained in the previous section, the system has the ability to communicate with the machine and control the machine using commands. Using this ability directly, the services do not need to consider the details about communicating with the machines. The machine control management service offering APIs for services to send the correct messages to the target machine. It also offers APIs for the services to get machine state information, and the control logic will update the machine state automatically during the control session for the service. In this way, new control services can fetch machine state data directly from the system without being affected by the integration or any communication related issues. More focus can be put on developing better control logic.

5.7.2. Resource support for the services

To offer these basic services for developers, the function logic behind them is actually the process of managing and reusing different kinds of data resources in RL. There are basically four kinds of resources depending to where the resources come from:

- **Resources from local system:** the resource data comes from a preset local database. A typical example is the machine related ontology described in the previous section.
- **Resources from AL:** this resource data is collected via the machine hardware side server. For example, machine hardware information, as well as machine state information.
- **Resources from basic service:** the data of the resource is collect when the basic services are processed. These basic services such as service for users will collect information such as new user register information and machine register information for the user during the time it is used. The information can be stored and ready for future use.
- **Resources from third party:** literally, the data provided by the third party is not belongs to the system. As shows in Figure 5.1, they are put in dot lines. However, after adaption, it can also be used in the system logic.

These data resources are all managed by its own handler as shown in Figure 5.8. The handlers adapt corresponding resources in to service function logic in CML, and then basic logic services can be constructed upon these resources. After the logic ability is realised, the basic services will be packaged into services that are available to the APIs for different basic services. Finally, as Figure 5.1 shows, these basic services are organised by Service Development Support Logic in CML and are ready for further development. The system will separate basic logics from control services, and these basic logics will operate related resources for the control process. Therefore, new services can reuse these resources by invoking the basic service APIs and there is no need to reproduce the logic for using these resources. In this way, more efforts can be invested in developing better service logic for users.

5.8. Summary and conclusions

In this Chapter, a design for a web service-based remote control service system is proposed. It is based on the many-to-many architecture. Different users can access the system and different

machines can be located on the internet to be controlled through the system. Moreover, the system offers a better service development environment compared to other designs.

According to the different functions in the system, the architecture is divided into six layers: Hardware Logic Layer, Adapter Layer, Resource Layer, Control Management Layer, Service Management Layer and Service Development Layer. The chapter provided a general description of each of the layers and an explanation of how these layers work together to build a remote control service system.

A machine model is proposed to describe the attributes and relationships between a machine, its actuators and sensors. Using the model, machine components and functions in the Hardware Logic Layer can be reasoned by the system and each machine can be considered as an independent service that can execute control command.

Using semantic messages, descriptions for machine hardware functions (MFD) and machine state (MSD) in the Adapter Layer, the requirement and feedback information from the service server will be easier for machines to understand. Combined with web service based APIs, the communication between service server and the machines is actually a process of semantic data exchange.

Using machine knowledge in Resource Layer, machine structure and functions can be retrieved and analysed according to the machine model. Machine descriptions can be automatically generated in a required way.

The system offers four basic services to supply basic control functions. Supported by different resources, these services will help the development of new services.

The idea behind the architecture is the separation of machines from the system as command execution services with supporting descriptions for reasoning. Using semantic information exchange, machines are offered as resources for service development. Compared with other designs, the proposed design is more flexible and can offer better support for other intelligent control service.

CHAPTER 6. GENERATION OF INTELLIGENT WEB USER INTERFACE FOR REMOTE CONTROL

6.1. Overview

A Web Service-based Remote Control Service System (WSRCSS) is proposed in Chapter 5. However, one of the issues that need to be tackled is dynamic Web User Interface (WUI) generation. As described in Chapter 4, user interface for conventional control system is also of interests to researchers and there is already ongoing work in this area. For a service-based remote control system, which allows both multi-user and multi-machine access via the internet, user interface design is a challenging problem.

There are two questions that need to be answered: How to generate user interface for different controlled machines? How to generate user interface for different users and different control processes and how to overcome the limitation of web user interface technologies? To solve these problems, an Intelligent Web User interface (IWUI) is proposed in this chapter. IWUI is a web application build on WSRCSS. Using the WSRCSS services, the application can generate Control Web User Interface (CWUI) automatically according to different machines and users requirements so that remote control service through the internet can be realized.

The Chapter begins with an analysis of the factors that influence the usability of CWUI. It then introduces the support that WSRCSS provides for IWUI. The architecture of IWUI and how it works is described after that. The chapter ends with a summary and conclusion section.

6.2. CWUI usability factors

As discussed in Chapter 4, the most important consideration for user interface design is usability, which includes effectiveness, efficiency and satisfaction in a specified user context. Usability is an abstract description of a general goal for user interface design. However, the detail factors that influence the design depend on what the design is supposed to achieve. Focusing on achieving a web-based interface for remote control the influencing factors for usability are:

- **MACHINE:** Understand what the controlled machines can do and how to operate them, including the control instructions and information.

- **USER:** Obtain the user's control requirements conveniently, such as which machine the user wants to control what functions need to be used.

- **PROCESS:** there are two factors that need to be considered for processes: task and state change. A task is a control action that the user wants to achieve; a state change is the machine's actual change of state when the action is carried out. Therefore, an interface must support the sending of control actions and the monitoring of state.

- **DESIGN KNOWLEDGE:** the interface layout should follow general design rules.

- **WUI LIMITATIONS:** WUI design is limited by the underlying representation language, poor navigation facilities, hard-coded task flow and lack of user model. The design needs to overcome these limitations and offers a convenient interface for the user.

6.3. Remote control service support

IWUI is a web application aims at offering user interfaces for users to achieve control goals. Among the five factors that influence IWUI, it is important to recognise that machine, user and process might be different for each control session. IWUI will provide a connection between a user and the remote control system. However, the interface is more than simple web pages representing control information. There is requirement for all the control and monitoring functions that the user required for a particular session. Therefore, IWUI is more like a web application with intelligent control functions or services, instead of a conventional website. Existing website design technologies do not support control and monitoring functions for remote control system. Also, there is lack of research on adapting WUI for the different and changing user requirements.

The proposed approach is to build IWUI based on the services provided by WSRCSS. If IWUI is a web application build up on WSRCSS then where should it be located? As illustrated in Figure 6.1, on the top layer (User Application Layer) there are web applications composed by a Service Handler and a group of web pages, such as user registration page, which are accessible by different users. Behind these web pages, the Service Handler offers support for

these web pages to achieve different functions. These functions are implemented in the Service Development Layer with some of the functions offered by WSRCSS. From the picture, there are four types of services developed for the IWUI system:

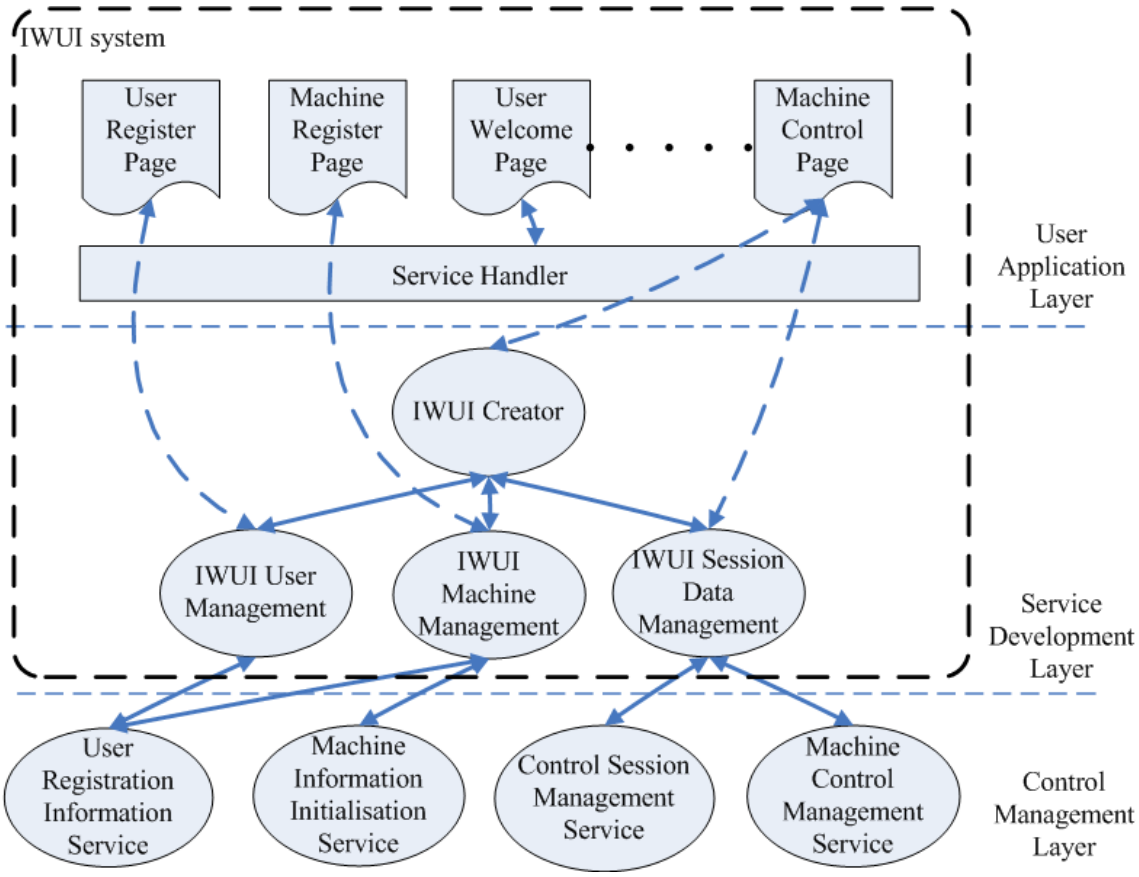


Figure 6. 1 Remote Control Service Support For IWUI

- **IWUI User management:** this service is used to support functions related to user information, such as user registration, user authorisation and user information update. It builds on functions from User Registration Information Service.

- **IWUI Machine Management:** this service is used to help user registration and update machine and control function information. User Registration Information Service offers the services to manage user registered machines. It also helps users to search for machine functions so that Machine Information Initialisation Service can be invoked.

- **IWUI Session Data Management:** this service offers functions to create and maintain control sessions. The functions will be used by the Control Session Management Service. It

also offers abilities to control machines and update machine states. Hence, Machine Control Management Service needs to be used.

- **IWUI Creator:** the kernel logic for IWUI services is located in this service. This service analyses user requirements, combines that with current control session information to generate IWUI. As shown in the figure, this service is built upon the other IWUI services, since it need to use some of the abilities from other IWUI services.

6.4. Architecture design of WUI

IWUI is a web application that offers automatic generation of CWUI that can be changed dynamically according to users, selected machines, and control processes. Since it is a web application focus based on WUI design, IWUI needs to address issues related to both web page design and web application design. However, the design of IWUI can also use knowledge from both UI design and web application model. How to adapt UI design guidelines to IWUI design has already been discussed in a previous section. Therefore, this section discusses the design of an architecture that uses the Model-View-Control (MVC) design pattern for web application [Krasner and Pope, 1988]. It also discusses the five factors that influence IWUI for automatic generation of WUI.

6.4.1. Collecting requirements that influence WUI design

Figure 6.2 shows the architecture of the IWUI service. Using the MVC design pattern, the IWUI architecture has three basic components: IWUI Control (IWUIC), IWUI View (IWUIV), and IWUI Model (IWUIM). As it is with the MVC pattern, IWUIV represents the final page that is shown to the users. IWUIM manages all the background services introduced in the section before to support dynamic WUI. According to the MVC pattern, different views (web pages) will be invoked according to the controller logic. Some data shown on a page can be obtained from background services so that pages can be shown dynamically according to a user's needs. This is very important because different pages with dynamically generated information can be offered to the user according to different requirements. Therefore, prior to start of a control session IWUI needs to collect information for the five factors that influence final the CWUI design.

The first three factors are machine, user and control process. User requirement is shown in the diagram as User Control Requirement (UCR). UCR is very important for control service as it contains information for target control tasks, such as controlled machines and functions. Machine information comes from IWUI Machine Management service. Machine control attributes and functions will also be obtained using this service. Control process is more dynamic. The task information, including commands, will be contained in IWUI Session Data Management service, while states changes have to be obtained from the machines via IWUI Machine Management service. After the requirements have been collected and analysed, the logic of the CWUI can be created. Different CWUI logic will be generated according to different users, machines and processes.

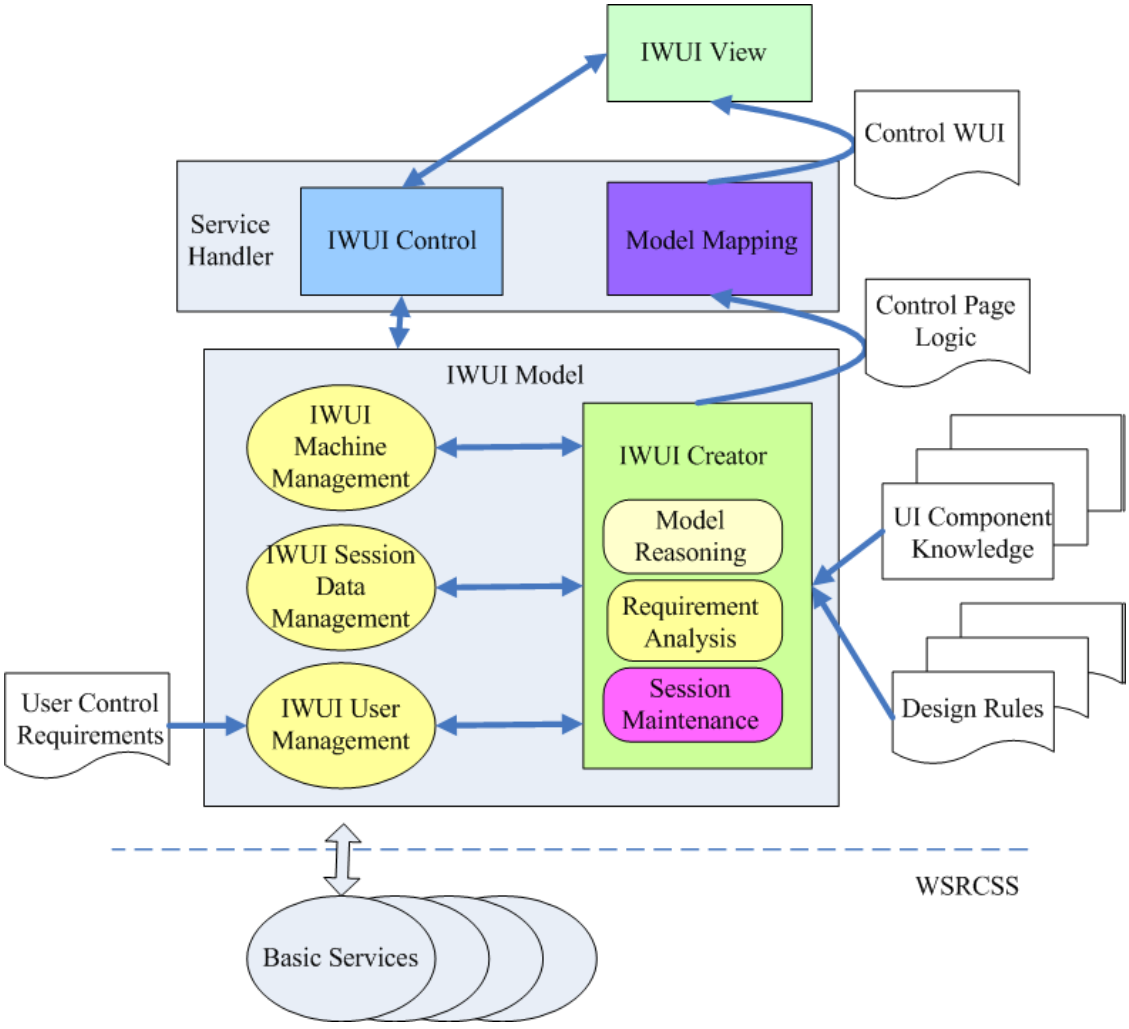


Figure 6. 2 IWUI Architecture

In this architecture, IWUIM will not directly generate a CWUI, which is the actual control page that can send commands to target machines. Instead, the Control Page Logic (CPL), which represents the organisation and attributes of the control page, is generated and later mapped to the CWUI via Model Mapping (MM). IWUIC and MM work together in Service Handler, which has been introduced before as a component that organizes services for the web pages. The separation of the CPL and the final CWUI helps to overcome largely the limitations of WUI.

Finally, additional knowledge, such as UI Component Knowledge (UICK) and Design Rules (DR), is needed to achieve IWUI. UICK is the knowledge about the framework design for the WUI. For instance, it has information about how to represent a number, or how to represent a function that a user needs to invoke from the WUI. DR contains the rules that guide the WUI design. The knowledge and rules can local resources as part of the IWUI. Third party knowledge can also be service used by IWUI. Making use of the requirements collected and the information, IWUI then generates the CWUI representation design logic and records it in CPL.

6.4.2. CWUI generation

Conventional web pages that are shown to users can be divided into two types: one is static web page that shows only use pre-set data; the other is web page that has a pre-set framework, such as pages that are used to collect user information. However, neither of these is adequate for supporting control services because the machines to be controlled and specific user requirements are not know until a user initialises a control session. It is not possible to present a general purpose control interface that suits all control scenarios. Therefore, CWUI has to be dynamically generated to provide an appropriate page structure, machine information and control functions.

IWUIM plays a crucial part in generating the appropriate CWUI and it has three functions. First, it analyse user and process requirements using the Requirement Analysis module. Second, it reasons about the CWUI model using the Model Reasoning module and decides what data should be presented to the user and how it should look. UICK and DR will be used to guide the reasoning process. Third, IWUIM updates the WUI in real-time to reflect changes in the states of the control machines by using the Session Maintenance module. Using the

three functions consequently, CPL is generated. By mapping the structured CPL to real CWUI, the WUI is generated automatically by IWUI.

In this way, the system collected the requirements for the five factors for the IWUI. Using IWUI Creator, it can generate efficient, effective and satisfactory control pages. By separating page logic from page view, it avoids having to manipulate unstructured HTML web pages so it will be much easier to improve the design and tracing the control process.

6.5. Semantic control records

In a remote control system that allows multiple users to access and control different machines simultaneously, the CWUI is more than just a browser for monitoring machine information. Among the five factors, user, machine and process are changeable between sessions and they relate closely to the data that need to be represented in the CWUI. As soon as the requirements from these factors are known for a session then the problem is to organise the corresponding information into the CWUI.

For a machine control page, there is a lot of information that needs to be represented, e.g. machine state information and machine control information. Furthermore, the WUI has to be updated as the states of the machines being monitored are changed. Therefore, the CWUI has to be linked dynamically with the actual control functions and state variables.

Using five types of semantic records the system manages to keep the control session information updated. Machine Function Description (MFD) and Machine State Description (MSD) have been introduced in Chapter 5. As shown in Figure 6.5, they are located in the local machine side; Control Page Logic, Session Context and Command Message are located in the IWUI Service Side. Each one of these descriptions/records is represented using XML. They are used by the system to update control session information and synchronize page information with all the session changes. These records, except MFD, are generated by the system automatically. MFD for a particular machine type should be provided when the machine type is first registered with the system. A MSD is generated when a machine is first initialized for a session. The others are generated and updated dynamically when the machine is controlled. The designs for Control Page Logic, Session Context and Command Message are explained in the following sections.

6.5.1. Control Page Logic

There are different kinds of information that need to be shown on the CWUI and the information need to be grouped and sorted. This requires an understanding of the data that needs to be shown on the control page. For example, as explained in Chapter 5, each machine has different actuators and sensors. A sensor value may need to be displayed and linked to a display area; an actuator may need to be controlled and linked to a functional UI component, such as a button. Therefore, the system needs to know what is being represented.

The approach used in this system is to generate semantic data that indicate the objects that needs to be displayed on a CWUI page before generating the actual interface. This kind of representation is called Control Page Logic (CPL). With the help of CPL, the system can work out what needs to be represented on a page, analyse the CWUI framework dynamically, organise the data in a logical way and link them to code that would retrieve information or send control commands.

CPL organises information into the following types:

- **Session information:** control session related information, such as session ID and user ID.
- **Machine state information:** information that indicates each machine control state, as well as commands that are available to the user for control.
- **Control context information:** data for machine control environment collected via third party context data recourses, such as local temperature for the controlled area.
- **Navigation information:** the system would recognise that there is too much information to be displayed on a single page and it would structure the information into additional pages with the appropriate links and navigation information.
- **Other information:** not every machine has the same attributes. With the help of the rules, other new information can also be put in the correct position in the structure, such as alarm information, and some session dependent information.

The machine state information, which is the most important data for controlling and monitoring, will contain control and monitor data for each machine. For a machine, the

information will be further categorised according to its function. For example, machine ID and location will be typed as “machine general information”, while sensor and actuator data will be put into “sensor information” and “actuator information” correspondingly.

However, CPL is not a form of semantic web site. It is a data structure for control page that used to describe different data types. The data in the same type in a machine will be represented on the control page in the same group, and the data for the same machine will be represent in the same block of a CWUI using the unified rule according to UICK. For example, the monitor information from sensors of a machine will be displayed in a table, while the command for its actuators can be represented with buttons. These tables and buttons for the same machine will be put in the same block. With the understanding of the data structure, it will be easier to organise the information according to its meaning. In this way, the page representation could be reorganised according to users’ requests.

6.5.2. Session Context

As the control process is changed sensor values will be changed and the actuator states will be changed when the user control them. CPL is only in charge of organising the given data in a structured way, it cannot solve the problem for data changes. To solve this problem, Session Context is created to keep a record of controlled machine states on the server side. This record is created as soon as a user enters the machine control page. It will obtain each control machine state through WSRCSS and keep synchronising the information while the control session is continuing. Later, the data will be transferred to CPL as an important composition of the page. In this research, Session Context is different from other records. Instead of directly recorded in a file, the data is stored in a database for efficiency. During the control process, Session Context keeps updating data from MSD. Session Context then is transferred in to Control Page Logic using semantic mapping rules before control page generated. In that way, the control page content will be synchronized with the control session.

The generation of Session Context from different MSDs initially aims at addressing the problem of multi-access. Figure 6.3 shows an example of how Multi-access can be solved in this design using MSD and Session Context. In the diagram, there are two Session Contexts (X and Y), which represent two user control sessions. There are two controlled machines (A and B) available. Machine A has Components a and b, while Machine B has Components c, d

and e. Components in long dashed line rectangle are controlled or monitored by Session X. While, components in the short dashed rectangle are controlled or monitored by Session Y. In that way, for different control sessions, machine state information of a machine is the same. For a monitoring process, the control sessions will just pick information they are interested in. But for a control process, IWUI will decide which Session gets the right to change machine state.

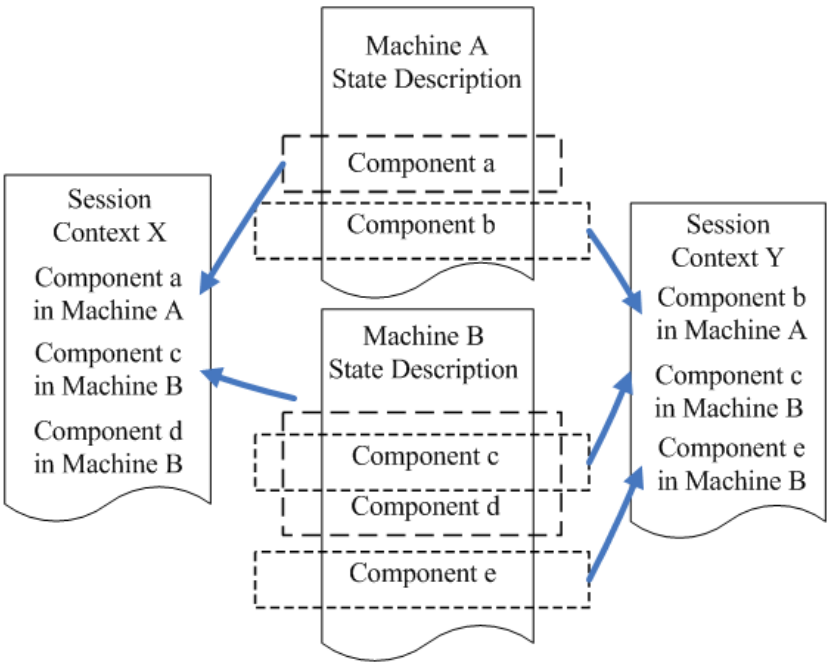


Figure 6. 3 Multi-access Adaption

During the control process, WSRCSS will synchronize the data from MSD to each Session Context. Later, the change of machine state will be reflected to CPL and then on the actual CWUI. Using the basic services in WSRCSS, machine state can be considered as special data which can be updated by actual machine hardware adapter on the client side. Using the same machine model, the data will be understood by other services, and can be separated into different segments when required. These segments can be referenced and integrated in different control sessions. Hence, as soon as the user application update the segments data from WSRCSS, and pass onto CPL, the CWUI will be synchronized with the control process.

6.5.3. Command Message

How to make the CWUI linked with the actual control actions? For example, for a light control interface, how can we create a button with a title ” turn on the light”, which can

actually turn the light on? Command Message is a kind of semantic message that helps to send such information to WSRCSS to achieve the actual control.

Chapter 5 explained how the WSRCSS make available devices as a kind of special resource. It also explained how the machine could be controlled using semantic data exchange. When a message is represented according to machine model that the system can understand then it can be understood by the system and executed on the device side. Therefore, Command Message in IWUI service is a semantic message that represents the control command using the same machine model introduced in Chapter 5.

```
<msg>
   <machineType>Super Light</machineType>
   <machineID>000010</machineID>
   <controlled_by>guoxi</controlled_by>
   <machineState>
      <componentState> Switch
         <actuatorState>
            OnOffSwitch
            <attribute type="state" value="ON" state="true">turnOn</attribute>
         </actuatorState>
      </componentState>
   </machineState>
</msg>
```

Figure 6. 4 Command Message Example

Figure 6.4 is an example for the command. The command is generated according to the light state. It is not hard to understand according to the machine model. The message indicates that the light ID is 000017 and the message is sent by the user “guoxi”. <machineState> includes the information that needs to be changed in this machine. <componentState> includes the information that needs to be changed in the actuator. The node <attribute> maintains the actual command which need to be executed, which equals to the <instruction> in machine model. The picture shows, the component need to control is Switch, and the target actuator is OnOffSwitch. There will be an attribute in the <attribute> called “state”, which shows the active state of the command. That is if it is “true”, it means the user want to activated this attribute. If it is “false” means use wants to deactivate the attribute. For example, in this command, what user wants to do is turn on the OnOffSwitch, so the state of target attribute “turnOn” has been set “true”.

For some command comes with pairs or groups, the system will analyse it influence for the other attributes. For example, if the attribute type is “state”, as it is analysed before, when it is active, the other attributes in the same group will be turned off. The hardware logic behind this group may just simply be different parameter setting for the same actuator functions, however, the corresponding result need to be represented further in the Session Context, and later in CPL so that the system will know which command is still available, and which one is currently active. Hence, the change of the process will be reflected in the CWUI.

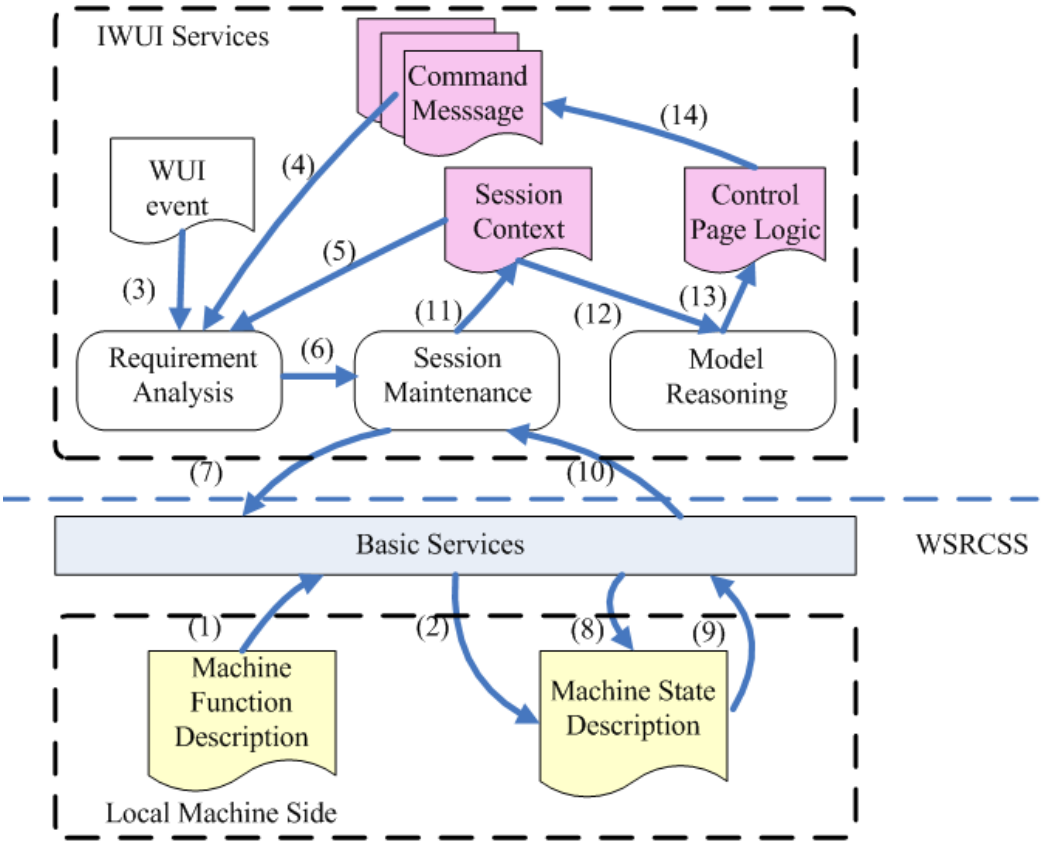


Figure 6. 5 Data Flow During CWUI Generation

6.6. Dynamic CWUI generation process

With the semantic description, the system is able to reason what data is needed to show in CWUI. Figure 6.5 shows how the data is synchronised between IWUI and a control session when a CWUI event happens. The control session starts after user selects the target machines and functions. WSRCSS will get the machine information from the MSD (1), and create or update MSD (2). Session Context will be initialized as well as CPL and available Command

Message. Once a WUI event happens, such as the user submitting a control request through IWUI (3), Requirement Analysis will select related Command Messages for the WUI (4). After checking the current Session Context (5) that the event is available, command messages will be passed to Session Maintenance (6) and it will send the request to Basic Services in WSRCSS (7). Basic Services will then update MSD on the local machine side after processing the control request (8, 9) and then Session Maintenance synchronizes the machine data (10) and update Session Context at the same time (11). When Model Reasoning synchronizes session data (12), it will find the changes and update Control Page Logic (13), meanwhile the list of Command Messages are analysed according to the Session Context. The available command list will be updated (14) and wait for the next call from WUI.

Following design rules, and using interface component knowledge, the CPL can be mapped to final CWUI. This method achieves the goal of dynamic generation of CWUI, and synchronizes it to the control process. Compared with other designs, this method better combines five factors that influence the WUI. Using semantic description, the page will be intelligently changed according to control process. Hence, it is flexibly enough to adapt to different needs.

6.7. Summary and Conclusions

In this chapter, a web application IWUI is designed. The research aims at generate better web user interface for remote control service system that allows multiple access by users and machines. Also, the control web user interface is required to adapt to remote control process, and will be able to link with real control.

The research begins with the factors that influence the usability of the WUI. It has been analysed that there are five factors that need to be considered for IWUI design – machine, user, process, design knowledge and challenge from WUI. Then, three are explanation that how IWUI uses the services resources in WSRCSS, so that most of the control related functions in IWUI will be achieved in this way. More attention could be put on dynamic WUI logic design. Then, the IWUI architecture is proposed which can combine the five factors to make IWUI design adapt solve the problem brings by the five factors. After that, a group of semantic control record is introduced. Control Page Logic is the semantic representation of CWUI data structure. It will help system understand the data that need to be represented in CWUI. Session

Context is the record that used to track the control process. Using this record, IWUI system will be able to get the machine and function state that user requires. Command Message is a semantic control command generated according to machine model introduced in Chapter 5. Command Message can be used to transmit use control request captured from operation on CWUI, and will be understood and executed via WSRCSS.

Compared with former design, IWUI offers better services for generating web user interface dynamically. The improvement can be summarized below:

IWUI is built up on basic services offered by WSRCSS. It fully makes use of the development environment, which helps to achieve control related functions so that focus on IWUI can put on user interface generation.

IWUI combines all the major factors that influence the remote control user interface. Unlike other systems, which only focus on one or some of the factors, this design is suited to generate control user interface for different users, machines, and different control process.

Compared with other designs, the control user interface can be adapted to a control process. It is not a static page that can not be changed. The generated control pages are synchronised control session using semantic records. The command and state showed on CWUI will be updated while control state changes.

Finally, this design separates interface data logic and final representation. Using semantic representation for the data structure, the system is able to reason about the page contents and further services can be added to a page.

CHAPTER 7. DYNAMIC GENERATION OF SESSION DEPENDENT NAVIGATION FOR CONTROL INTERFACE

7.1. Overview

The benefits of using web user interfaces are discussed in Chapter 1. Methods have been proposed in Chapter 6 for generating dynamically user interfaces for a remote control service system. Using the methods, specific web user interfaces can be generated according to different user requirements from the control context. However this does not cover users who have to control multiple machines and have changing requirements for different sessions. For example, a nurse may monitor different patients on different shifts. If there are many patients and the number of devices and sensors involved is large then the information can not be displayed effectively on one screen. It would be inconvenient for a user to scroll a screen up and down in order to find information. Therefore, the question is "How to find a dynamic way to organise the information and provide navigation menus so that the user can easily navigate round the system to find the information?"

To address this problem, a method to dynamically generate web user interfaces with session dependent navigation for remote control system is proposed. Using this method, a user can specify how they would like the information to be structured and navigated. Their specification will then be used to generate the web pages with the appropriate navigation structure, dynamic data updating and, control command forwarding and execution.

An example is provided in the next section showing the problem met by IWUI users. The example also shows a possible approach for solving the problem using navigation. To apply the approach, a message-driven data model and a session dependent data structure are introduced and a three-stage framework for generating dynamic navigation is proposed. The chapter ends with a summary and conclusions.

7.2. An example problem

An example is given here to indicate the need for dynamic navigation generation for a remote control service system.

Two users (A and B) log into the system. User A wants to control one machine, with machine ID 000021. Meanwhile, user B wants to control 10 machines, with machine IDs 000008, 000013, 000014, 000015, 000016, 000017, 000018, 000019, 000020 and 000021 at the same time. These machines have different attributes and are of types "Adjustable Intensity Light" and "Intelligent Light". These machines are produced by different manufacturers: Phillips, KING and IKEA. They are located in different places: "Home___Sitting Room", "Home___Kitchen" and "Home___Bath Room 1". Figure 7.1 shows the details of the machines located in the two rooms. User B wants to group all machines according to their types and group them further according to their locations. Also, user B wants to group all machines according to their locations, and then group them according to their types and then manufacturers.

Selected	Machine ID	Machine Name	Machine Type	Location	Manufacturer
☐	000020	CeilingLight1	Intelligent Light	Home___Sitting Room	Philips
☐	000019	CeilingLight0	Intelligent Light	Home___Sitting Room	Philips
☐	000018	SittingroomLight0	Adjustable Intensity Light	Home___Sitting Room	IKEA
☐	000017	WallLight0	Intelligent Light	Home___Bath Room 1	IKEA
☐	000016	BathFloorLight1	Adjustable Intensity Light	Home___Bath Room 1	IKEA
☐	000015	BathFloorLight0	Adjustable Intensity Light	Home___Bath Room 1	IKEA
☐	000014	BathCeilingLight1	Adjustable Intensity Light	Home___Bath Room 1	Philips
☐	000013	BathCeilingLight0	Adjustable Intensity Light	Home___Bath Room 1	Philips
☐	000008	KitchenLight0	Intelligent Light	Home___Kitchen	KING
☐	000021	CeilingLight0	Intelligent Light	Home___Kitchen	IKEA

Figure 7. 1 Controlled Machine Detail List

Given the user requirements, the system is able to generate dynamic control pages for user A, but because user B is accessing a large amount of information that can not be displayed within one screen and user B's requirements are changeable it cannot be achieved with a fix control page.

Therefore, the system needs to structure the information so that user B can access it easily. If the system can retrieve category keywords according to the information of selected machines and let users decide the filter conditions then a session dependent navigation interface can be built for user B for that particular control session. The whole point is to create a page with the appropriate structure and navigation links so that the user does not have to worry about indexing, coding for updating the states of devices being monitored as well as the coding for manipulating the devices. These should all be generated by the system automatically.

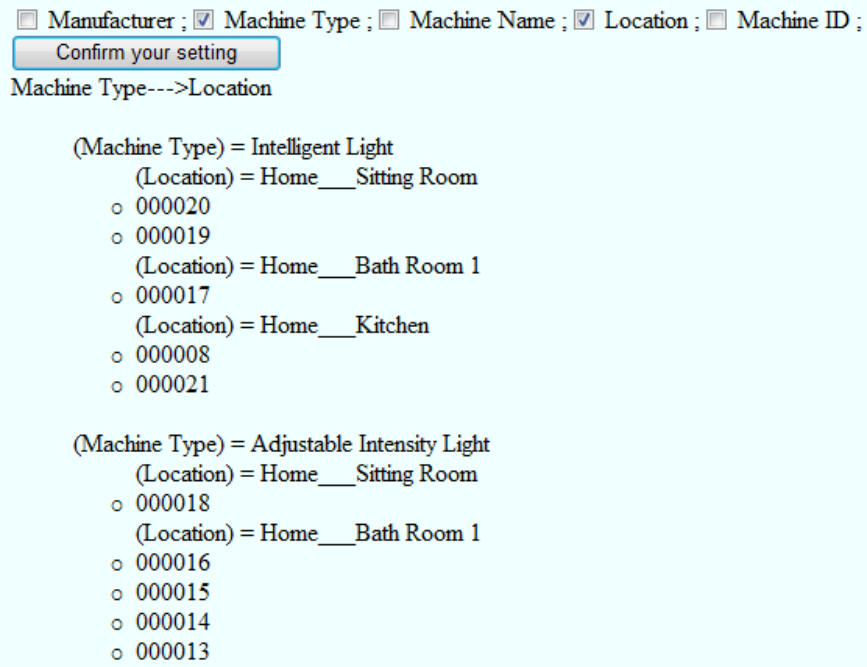


Figure 7. 2 Dynamic Navigation Setting Page Example 1

Figure 7.2 and Figure 7.3 show for a set of machines chosen by the user, the system determines from the sets of attributes what need to be shown to the user. If a user wants to group all machines according to the types then locations, then the attribute selection sequence can be set as "Machine type → Location" (Figure 7.2). However, if the user wants to group the machines according to their locations, types and then manufacturers, then the attribute selection sequence is "Location → Machine Type → Manufacturer", as shown in Figure 7.3.

When a user confirms their settings then a session dependent navigation tree is generated and added to the control page. If a node on the navigation is clicked then a different page is displayed according to the user selected preference described above. For example, when "000016" is clicked, only the information of machine ID = 000016 is shown on the control

page. If the user wants to control all the machines in "Home___Bath Room 1" then "(Location) = Home___Bath Room 1" can be clicked, and a control page containing only the machines in "Home___Bath Room 1" is shown.

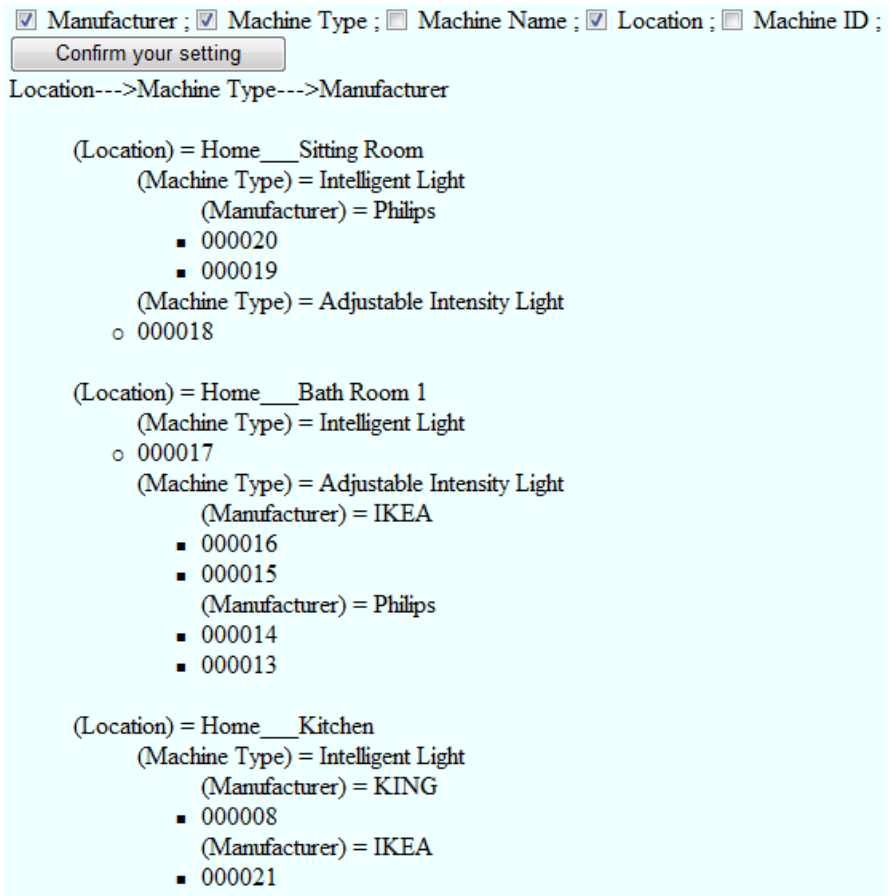


Figure 7. 3 Dynamic Navigation Setting Page Example 2

7.3. Message-driven data model

As described in the previous section, a page changes when a user request is sent. According to the MVC model, the page (view) is changed because the controller will activate different page models behind it. However, MVC only offers a design pattern to support the invoking of different models according to the requests [Kappel, et al., 2006]. The detail logic for data transmission and page model changes in the system still need to be designed. For an interface like a control page with navigation, the page content in the CWUI is changed in response to different events such as user navigation, machine state update, and transmission of control commands. It will be difficult to design a particular page or static page model to satisfy all

these requirements. It will also be hard for the server to decide how to change the data model in response to the different types of dynamic events.

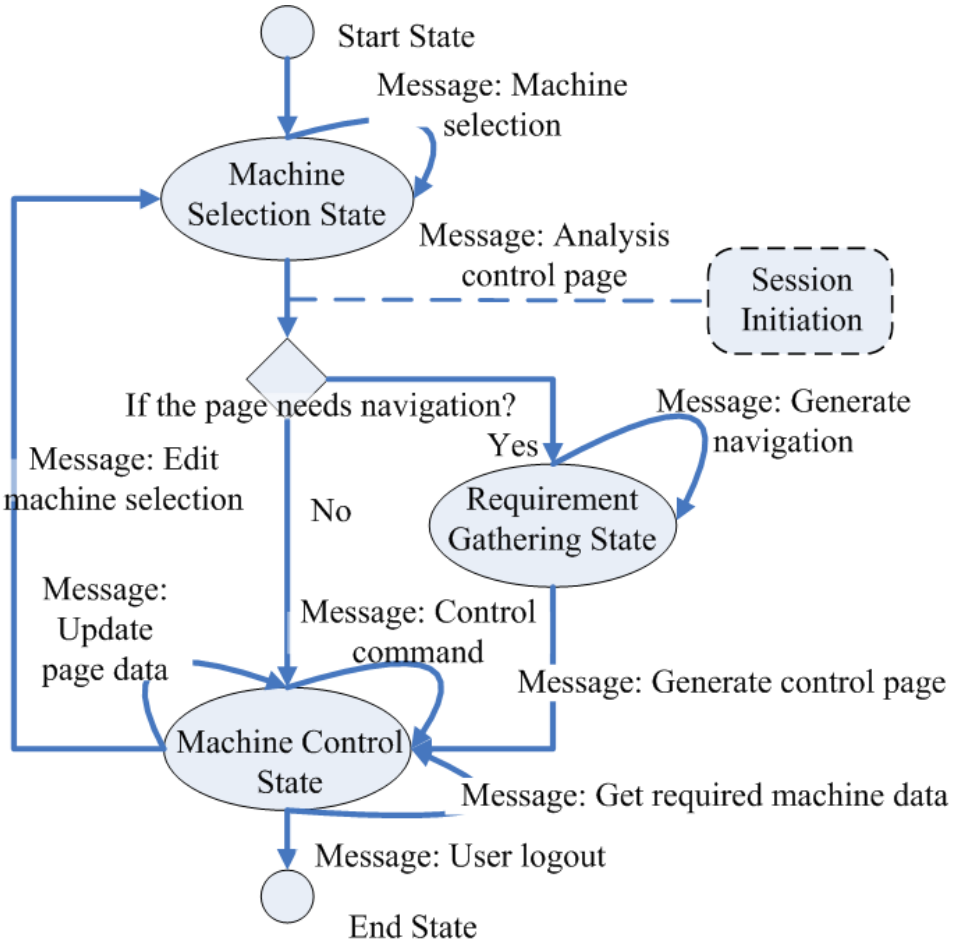


Figure 7. 4 States For Message-driven Data Model

This project proposes that the process of data model changes can be described using state changes. Message passing is central to the system design because it drives the state changes of the system. If requests from the client side are represented as messages then the server can interpret them and change the data model as well as the data logic behind it accordingly, and then switch between different states. For example, in Figure 7.4, after a user logs into the system, the system enters the “Machine Selection State”, which allows the user to select the machines that need to be controlled. When the user submits their selection then the selection is sent back to the server as machine selection messages and the server initiate an analysis control session to decide whether user navigation is needed. The required data model is move to the “Machine Control State” if navigation is not needed. Otherwise, a generate navigation

message is sent to itself to start the process of collecting the user's requirements and set the navigation data for the system in the "Requirement Gathering State". In the "Machine Control State", the data model is driven by three kinds of messages: "Update page data messages" are for synchronising machine state data; "Control command messages" are for sending user control commands to the server and updating the machine state data at the same time; "Get required machine data" messages are for collecting the user's navigation requirement and select the machine data that the user need to check. Finally, the "User logout" message clears all the session data and finishes the control session.

There are two kinds of messages are shown in Figure 7.4. One kind of messages can drive the CWUI state change, which usually means that it will drive the page model to switch from one to another. For example, generate control page message will drive the state from "Requirement Gathering State" to "Machine Control State". The other kind of messages will invoke the background application without changing the state of the page model, such as sending Control commands in the "Machine Control State". Although the page model is not changed, the data structure behind it is changed.

How are the messages generated? Usually, each state is represented using a different WUI with different message functions encoded. These functions are invoked when they are triggered by events such as mouse clicks on different elements of the page. On the other hand, Control Commands are generated automatically when the CWUI is created as explained in Chapter 6 (Command Message). Therefore, both state change messages and command are generated dynamically when the control session is processed. In that way, the messages will drive the WUI state change according to the control session dynamically.

This method uses messages to represent user requests on the client side and then updating the server side data model following the logic shown in the state diagram to achieve the required changes in the WUI view. In this way, different requests from a user can be collected and processed correctly depending on the message type.

7.4. Session dependent data

Chapter 6 introduced that CPL is generated to describe CWUI data structure. The basic idea is to categorise different types of data and represent them in a semantic way. There are some basic categories of information that are generated for every control session, such as "Session

information", "Machine state information", and "Control context information". There are also some types of information that are generated dynamically as the control session changes. "Navigation information" is one of these dynamic types as it is only generated when the user set the navigation information. The system analyses the data structure through CPL, and represents different data according to their categories.

If the navigation generation is based on the basic categories the information contained in the model is sufficient. However, if the navigation is based on attribute values that are session dependent the problems will be aroused. For example, a nurse may want to monitor all patients whose doctor is "Tom". Another nurse may want to monitor a particular patient "Sue" who requires special treatment. In both of these cases, "Tom" and "Sue" are values that are decided by the current control session according to the users' preferences. The information is not part of the CWUI structure but it is related to the control session.

Therefore, CPL should not only allow session dependent data to be added in to the existing categories dynamically but also allow some new session dependent categories to be added in to the basic CPL structure dynamically. When CWUI generation logic will analyse the CPL, it will handle the categories separately. For basic categories, basic rules and applications are used to map them to the basic component of the page. If new categories are detected, corresponding handle functions will be invoked to analyse the data in these categories and resemble the data in CWUI page. In that way, the session dependent data can be recorded used to generate more dynamic navigation.

In this thesis, two examples of session dependent data categories are introduced in the system. They will be added to the CPL dynamically depend on the control session requirement. One of the categories records critical alarm condition data to alarm users when needed. The alarm data can be added to CPL in "alarm information" category with their own structure. The alarm data will be map to the page as an independent panel with navigation function because all the data are critical conditions that need to be easy to find by users. Another session related category called "additional information" is defined by uses and is used to describe some free-style session dependent data. That is, except for the data value, the user is allowed to define each attribute value as well as attribute name for each item in this category. For example, the information the nurse needed in the example mentioned before can be represented in XML as shown in Figure 7.5. As soon the method handle the data allows user to set these two elements,

the data can be set according to the user's needs. In this way, the session dependent data is captured by the system waiting for further use.

```
< additional_information>
   <item>
      <name>doctor</name>
      <value>Tom</value>
   </item>
   <item>
      <name>special treatment</name>
      <value>yes</value>
   </item>
</ additional_information>
```

Figure 7. 5 Example Of Data From "Additional Information" Category

The idea behind it is to further extend the CWUI page structure into flexible way. Using semantic representation, new categories can be defined and processed using its own methods. New types of data collected during control process can be merged in to the CPL structure by adding new categories and corresponding handling function. Especially, some session dependent information can be collected using this method. The result is the CWUI data structure will be more flexible to adapt to different requirements. Therefore, it is possible to generate a more dynamic navigation that can adapt various requirements.

7.5. Dynamic navigation structure generation

With the support of a message-driven data model and semantic data description as discussed in Chapter 6, generation of dynamic navigation is feasible. With the session dependent data structure, it is also possible to generate a dynamic navigation for different users. However, the system faces three challenges before it can generate dynamic navigation: how to generate navigation keywords, how to create a navigation tree and how to link navigation with actual data. The following sections will explain the solutions to these problems.

7.5.1. Navigation keywords generation

Before navigation can be created, a set of keywords that can categorise these machines need to be generated. An easy method is to pick some basic attributes for the machines which are obtained by every machine so that machines can be categorised according to these keywords.

For example, machine types, location and machine name can be selected as keywords. However, if the keyword generation only focuses on using fixed keywords then the navigation will not be flexible enough to meet users' needs. For example, there maybe keywords that are specific to a control session that are helpful for navigation, such as machine component state as users may want to involve some session dependent information for navigation.

Sometimes, it may be hard to decide which attribute should be considered as keywords until the control session begins. For example, it is not known if a user would select the component "switch" for a light before the control session begins, so it is hard to decide if the state attribute of the switch should be considered as a keyword. The question is how to generate keywords that can well describe the machine, suitable for current control process while offering more session dependent options to meet users' needs? The solution proposed is to automatically generate machine keywords following some selection rules and according to CPL.

With the help for CPL, as it is explained before, machine control related information is organised in a semantic way, and can be interpreted by the system. CPL not only stored the data value, but also categorised them according to their functions. Among these categories introduced in the section before, machine state information, alarm information and additional information all contains information that can be used to describe the machine. Alarm information is organized as an independent alarm list because the data is important for a control process. The keywords will be generated according to machine state information and additional information. Some basic attributes which are common to all machines, such as machine name, type and location are categorised in a sub category called "machine general information" in machine state information. Other information in the two categories will be different according to different machines, user settings and control process. However, the semantic representation makes it possible for the system to generate keywords from these attributes automatically according to their importance for the control process.

The following rules will be used to generate the keywords automatically:

- For all machines, generate basic keywords from machine general information items in machine state information, as every machine will have these items. Usually, these

keywords generated include machine name, machine type, location, manufacturer and machine ID.

- For machines of the same type, create keywords according to the values from the same component or command. Keywords can be generated according to sub-categories of machine state information, which are used to describe machine sensors or components. This method is especially useful when a user need to monitor the component state of certain types of machines. For example, a hotel manager may want to check all the room lights that they are still on in a hotel. These keywords are automatically generated according to the controlled machines.

- Session dependent attributes can be created by users according to their control needs, as described in Section 7.4. If there are session dependent information set in the control session, keywords can be generated according to the items in the session dependent information categories.

The keywords generation logic will analyse CPL, pick the attribute and its value from given information, generate keywords according to the attribute information. For example, the keyword which is used to describe machine name can be generated according to the attribute called “machineName” in machine state information.

These generated keywords as well as their values will be stored in Navigation information in CPL for further analysis. The method provides a dynamic way of keyword selection. Instead of limiting a user to some fixed keyword options, it provides a way that allows the user to select session dependent keywords to generate a dynamic session dependent navigation.

7.5.2. Generate navigation tree

After the keywords have been presented to the user and relevant ones selected, the next step is to categorise the machines according to the keywords and generate a navigation tree (or navigation structure). The keywords selected and the selection sequence may result in different navigation structure. The system generates a “change as you choose” navigation tree as the user makes each selection. As the input is entered as a sequence the difficulty is after each value in the sequence is entered the structure has to be updated dynamically.

Table 7.1 shows the value for a group of different machines. M1 .. Mi represent machine IDs. K1 .. Kn represent keywords generated automatically, i.e. extracted from the ontology, for the user according to the selected machines, such as "machine type" and "location". V(x,y) is the actual value of the keyword Kx for corresponding machine My. For example, if K1 is "location" then V (1, 1) would be the location for machine M1 and V (1, 2) would be the location for machine M2.

ID	K1	K2	…	Kn
M1	V(1,1)	V(2,1)	…	V(n,1)
M2	V(1,2)	V(2,2)	…	V(n,2)
…	…	…	…	…
Mi	V(1,i)	V(2,i)	…	V(n,i)

Table 7. 1 Machine State Information Attribute Table

The process of generating a navigation tree is based on the user selection. The notation K1→ K2→ K3 means that the user selected K1 then K2 and finally K3. Similarly K3→ K1→ K2 means that the user selected K3 then K1, and finally K2. These are called Keyword Chains (KWC). In order to generate the structure in the right sequence, the system will keep a record of the KWC and update the list when the user makes a change to the selection through the interface, and pass the selection chain to the Navigation Generation Function (NGF).

A recursive algorithm is used in NGF to generate the navigation tree. For each keyword in the given order, the system will collect the value from each machine and put them into a list. The system will analyse the values in the list and put the machines that have the same value for that particular keyword in the same node in the navigation tree. For example, if K2 is selected, the system will loop from data in M1 to Mi. In the loop, V (2, y) is compared with the value from V (2, y+1) to V (2, i). If they are of the same kind, the machine ID will be collected and put into a list. At the end of the loop, if there are new keywords still in KWC, then each list will be checked recursively. The algorithm is described in detail using JavaScript in appendix.

After the navigation tree is generated, it is added to the control page for user access. The navigation bar will be added to CWUI, at the same time, the semantic data for the navigation tree will be added in to Navigation information in CPL as a part of the page data structure. In that case, once the machine state is updated, the navigation will be updated accordingly.

7.5.3. Linking navigation and control

Although the navigation structure has been generated and put onto the control page, the navigation will not work properly without linking it to actual control data to enable the page to indicate the machine state information in real time.

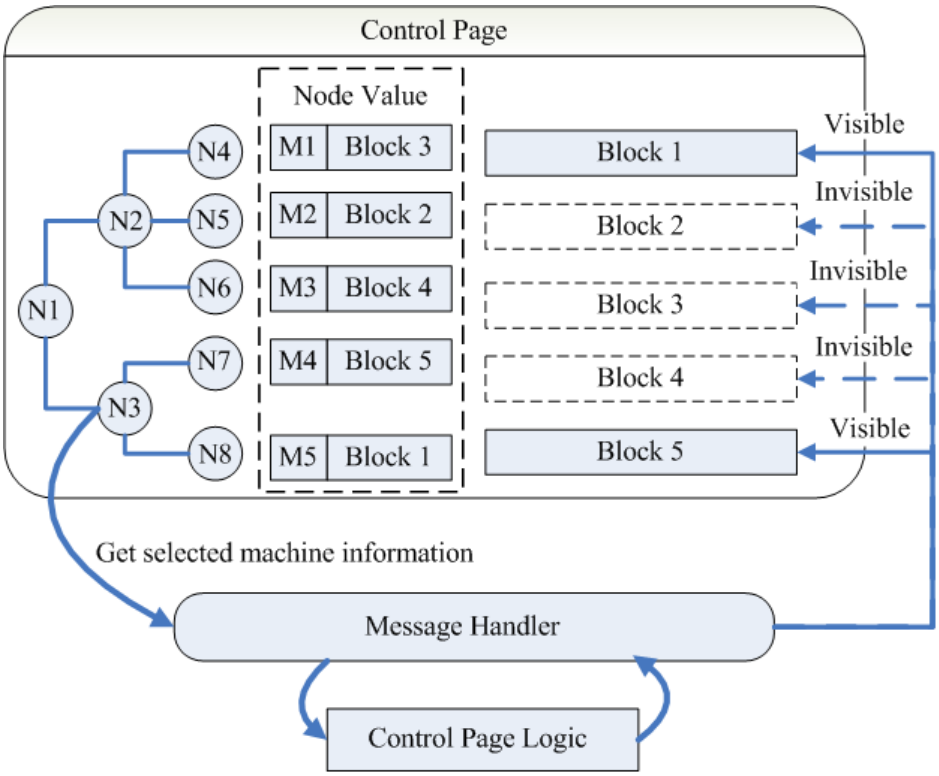


Figure 7. 6 Link From Navigation To Page Content

The core of the solution is to group the information in blocks so that they can be referenced independently and to provide the navigation node messages for referencing these blocks. Therefore when a node is activated an appropriate message is sent to the system and the system will execute a corresponding method to retrieve the blocks, and choose only the selected block to display. No matter how the blocks are displayed. In this way, users will index the required data efficiently.

As introduced in Chapter 6, when CPL was mapped to a control page, the information is grouped according to different mapping rules. However, information for the same machine will always be displayed in the same block. At the same time as the navigation tree is created the messages that are linked to the nodes are also generated accordingly. When the system

receives a message from a node, it will retrieve information about which blocks are required by the user. The system will reorganise the control page according to the user's needs and hiding information that is not of interest.

Figure 7.6 shows an example for this process. On the control page there is a navigation tree with 8 nodes (N1 to N8). M1 to M5 are the identities of the controlled machine. Block 1 to Block 5 represents the groups of data showed on the CWUI related to machines M1 to M5 respectively. As shown in the diagram, N4 to N8 are leave nodes representing machines M1 to M5 respectively. Node Value shows the message contained in each of these leave nodes. As was discussed previously, the machine information for the page is put into different blocks on the page (Block1 to Block5) according to CPL mapping rules. Each leaf node carries information for the current node name, which needs to be shown in the actual navigation tree. The node also carries navigation information, which indicates the actual block linked with this node. For example, node N4 contains node value M1 and is mapped to Block 3. Therefore, when N4 is selected M1 is the ID of the machine that needs to be shown on the page, while Block 3 is the actual required block which contains the information the user requested. For nodes like N2 and N3 that have children, there might be more then one block linked to it. When node N3 is selected, messages will be sent to the message handler, bringing the data value stored in N3. As shown in the figure, N3 includes information from both N7 and N8. N7 has node value M4 and linked to Block 5, while N8 has node value M8 and linked to Block 1. After analysing the semantic CPL, the system identifies that Blocks 1 and 5 should be visible and Blocks 2, 3, and 4 should be hidden. The control page is reorganised showing only information that should be visible to the user.

In this way, navigation is linked with dynamic page content, control commands and real-time data.

7.6. Summary and conclusions

There are two problems for multiple machine control page. The first is that different users want to control different machines and the requirement may be changeable during the control process. Therefore, different users require control pages that are tailored to their needs. Another problem is information for displaying may not fit onto a single screen. Therefore a

navigation structure for accessing the information will need to be dynamically generated for each control session according to the user's requirements.

To solve the two problems, this chapter made following contributions:

First, a message-driven model is proposed. It provides the necessary flexibility for handling different states during the page generation process. Hence different control pages models can be generated without confusion.

Second, session dependent data can be added in to the CPL basic categories which are originally generated based on domain ontology. Data in different categories can be processed separately and later map to the same CWUI. In that way, new information can be used for the generation of a more dynamic navigation.

Based on the above, a method for dynamic navigation generation is realised. By providing the user with a set of appropriate keywords based on a set of machines chosen for control and monitoring, the user can select and order a list of keywords. Based on the list a navigation tree can be created dynamically according to the user’s requirements. A page with the navigation structure can be generated and linked with the required control information.

The research on navigation for such a remote control system is novel. The idea behind it is to organise the web application information in a semantic way and offers session dependent index for the information requested. It is not only designed for a representation purpose but also for a method to structure the application output information. It is new, especially in the area of remote control applications.

CHAPTER 8. EVALUATION

8.1. Overview

The design of the WSRCSS module and IWUI application has been described in previous Chapters. To evaluate and to illustrate how the system works in practice a number of simulated application scenarios have been implemented and tested. The scenarios aim to exercise different aspects of the system.

In this Chapter, the implementation environment and scenario preparation are introduced first. Five scenarios which aim at solving different problems in remote control over the internet are then demonstrated: hoister hardware control, home facilities, hotel facilities, farm devices and hospital equipments. Finally, the result of the evaluation are summarized and concluded. As defined in Chapter 5, the word "machine" means any kind of "device" or "appliances".

8.2. Evaluation environment

8.2.1. Hardware environment

The total system is in three parts: service server, machine server and client computer. Each part is deployed on a separate computer connected via the University campus internet. The Service Server is the main server in the system that runs SDL, CML and RL. The machine side module, AL, is deployed on the Machine Server. The client computer has no special requirement except that it is connected to the network and has a web browser running. More than one client computers can run simultaneously to access the service server. The hardware specifications of these computers are listed in Table 8.1.

The server program and AL program are implemented in Java using JDK 6. HTML and JavaScript are used for the WUI module. The module is developed on NetBeans 6.9 IDE. The programs that simulate the hardware machines on the machine server are implemented using Java. Each simulation program can simulate the operation of a group of machines and the state changes when it is controlled. The control of a model hoister (Figure 8.1) has also been tested to demonstrate the adapter works with actual hardware. This hoister has a lift bar which can

load the white cartridges shown in the figure and move them in four directions – up, down, forward and backward – and position them in different slots. More details about the machine model to control this model hardware using the proposed architecture is described in a later section.

Role	CPU	RAM	Operating system
Service Server	Intel(R) Core(TM)2 Duo CPU P8400 @ 2.26GHz 2.27 GHz	3.00 GB	Windows vista home basic
Machine Server	Intel(R) Pentium(R) 4 CPU 3.00 GHz 3.00 GHz	3.00 GB	Windows XP Professional Version2002 SP3
Client Computer	Any	Any	Any

Table 8. 1 Hardware Condition For Computer In The Evaluations

Figure 8. 1 Hoister Used For The Evaluation

8.2.2. Knowledge-base and machine simulation

According to the design, there are three things that need to be prepared before the evaluation: ontology for machines; the locations of instances of machines; machine hardware needs to be

simulated. To apply the research to different kinds of machines, a simple simulation environment is built. This platform offers a virtual environment to receive control instructions, update component states and send back state information.

i. Domain ontology settings

The ontology is built to describe machines according to the machine model discussed in chapter 5. The ontology has three main parts: machine, actuator, and sensor. Using XML the information represents machine model knowledge semantically.

The machine part of the ontology records category information and basic machine component data. Part of the information is shown in Figure 8.2. The information shows a machine hierarchy. For example, there is a category called “Light” under the category of “Home Equipment”. “Normal Lamp” and “Adjustable Intensity Light” belongs to the category of “Light”. Components of machines are also specified. <type> node in each <Component> node indicates the type of the component. For example, all the “Light” will have a sensor called “Light Intensity Detector”, and all the “Adjustable Intensity Light” will have an actuator component called “Light Intensity Controller” which is operated by a motor called “Slider”. For the purpose of the evaluation, models of three main categories are specified: home equipment, hospital equipment, and farm equipment. Machines in these categories are further grouped according to machine category knowledge.

Similarly, there are specifications of actuator and sensor information. The actuator ontology includes the information and instructions that are available for different types of actuators. Six types of actuators are modelled to support the evaluations. They are “Motor”, “Valve”, “Heater”, “Slider”, “OnOffSwitch” and “OpenCloseSwitch”. The sensor ontology includes all the sensor properties used in the evaluations, such as sensor type, initial value, and units. Nine types of sensors are defined: ” Light Sensor” (sensor used to measure light intensity), “Temperature Sensor” (sensor used to measure the temperature), “Moisture Sensor” (sensor used to measure the humidity), “Rotation Speed Sensor” (sensor used to measure the rotation speed of the motor), “Water Pressure Sensor” (sensor used to measure the water pressure), “Blood Pressure Sensor” (sensor used to measure blood pressure), “Electrical Pulse Sensor” (sensor used to measure pulse) and “Thermometer” (sensor used to measure human body temperature).

The machine models enable the system to reason about machines. Hence, the user only needs to tell the system instances of these machines. The code of all ontology models can be found in Appendix A.

```
…
<machine>Home Equipment
    <machine>Intelligent Light
        <Sensor>Light Intensity Detector
            <operateBy>Light Sensor</operateBy>
        </Sensor>
        <machine>Adjustable Intensity Light
            <Component>Light Intensity Controller
                <operateBy>Slider</operateBy>
            </Component>
            <Instruction>Bright Mode</Instruction>
            <Instruction>Evening Mode</Instruction>
        </machine>
        <Component>Switch
            <operateBy>OnOffSwitch</operateBy>
        </Component>
    </machine>
    <machine>Cleaning Devices
        <machine>Washing Machine
            <Sensor>Water Temperature
                <operateBy>Temperature Sensor</operateBy>
            </Sensor>
    …
```

Figure 8. 2 Example Of Machine Ontology

ii. Location settings

In the real world, different machine instances are located in different places. In order to demonstrate this, virtual locations are needed before the machines can be initialized. According to the design in chapter 5, machine descriptions on the remote machine servers store machine information that is used for control data exchanges. Therefore, these virtual locations can be represented using different directories on the machine server side. The data exchange with different machines is achieved by obtaining and updating the machine descriptions in these directories.

To create the scenarios, some directories are created dynamically according to the virtual locations. An XML file is used to map the links from different virtual locations to the

directories. Later, machine descriptions can be stored in different directories to demonstrate the machine located in different places.

```
…
    <location>Home
       <roomType>["Kitchen", "Bed Room", "Sitting Room"]</roomType>
       <room>Kitchen</room>
       <room>Bed Room 1</room>
       <room>Bed Room 2</room>
       <room>Sitting Room</room>
       <room>Bath Room 1</room>
       <room>Bath Room 2</room>
       <room>Entrance Room</room>
    </location>
    <location>Hotel
       <roomType>["Room"]</roomType>
       <room>Room 101</room>
       <room>Room 102</room>
       <room>Room 103</room>
       <room>Room 201</room>
…
```

Figure 8. 3 Example Of Machine Location Mapping In XML Format

Figure 8.3 shows an example mapping. <location> represents a remote location connected to the server. <room> represents a sub-location within a <location>. <roomType> represents the type of sub-location. The example shows two locations: “Home” and “Hotel”. In the “Home” location there are several sub-locations such as “Kitchen”, Bed Room 1”, “Bed Room 2” and “Sitting Room”. Different simulated machines can be initialised to be in any of these different locations.

iii. Machine simulation

Before users are able to control a machine it needs to be machine simulation program which allows other software to access its embedded control APIs. There are two type of machine simulation program. One is for the real hoister hardware, another one is a simulation program.

The hoister has its own embedded control program similar to other electronic devices. This control program is written by the vendor and offers a simple API for other programs to access the hoister. The machine simulation program for the hoister is developed to invoke these API when it receives simple command strings such as “go up”, “go down”, “go forward” and “go

backward". These commands are described in MFD so that the Adapter Layer application can invoke corresponding hardware function. The hoister simulation program is programmed in java, C and C++. Using C and C++ program, the low level hardware control API is programmed in a DLL (Dynamic-Link Library) which can be invoked by the adapter program written in java. Then a simple java program invokes different hardware control commands when receiving different instructions.

Another machine simulation program simulates the machine embedded control programs that are executed on the hardware side, and cause the sensor value changes. It functions like a virtual machine. The hardware simulation program is described using an XML configuration file, and is executed dynamically according to the control message. The simulation description file has the same functions as MFD (chapter 5). The difference is that the description file is set for a group of machines as for a whole test example while MFD is set for each machine.

```
…
    <machine>BOSCH Dishwasher123
      <function name="WaterSpray_Motor_direction_Clockwise()">
        <component operatedby="Motor" value="Clockwise">
        WaterSpray</component>
        <operation>
          {"sensor_name":"Spinning speed","action_type":"=", "changevalue":"100"}
        </operation>
      </function>
      <function name="WaterSpray_Motor_direction_antiClockwise()">
        <component operatedby="Motor" value="antiClockwise">
        WaterSpray</component>
        <operation>
          {"sensor_name":"Spinning speed","action_type":"=", "changevalue":"-100"}
        </operation>
      </function>
      <function name="WaterInlet_Valve_state_ON()">
         <component operatedby="Valve" value="ON">WaterInlet</component>
         <operation>
          {"sensor_name":"Water Pressure","action_type":"+", "changevalue":"1"}
        </operation>
      </function>
…
```

Figure 8. 4 Example Of Machine Hardware Simulation Configuration File

Figure 8.4 shows an example of the simulation description file in XML format. <machine> indicates which type of machine the group of functions belongs to. A <function> node records the detail function information. The attribute "name" shows the hardware function name described in MFD; <component> shows the component information for this function; <operation> shows the action for the function in JSON. For example, when the system receives a control message from the user which is found in HDL to be "WaterSpray_Motor_direction_Clockwise()", the first function node will be invoked. <component> indicates that this action will be operated by "WaterSpray" which is a "Motor" and the value is changed to be "Clockwise". The details of <operation> specified to this function will set the value of the sensor attribute named "Spinning Speed" to "100". This is to simulate the function when the direction of the water spray of a BOSCH Dishwasher123 is turned to "Clockwise" and the rotation speed of the water spray is changed to 100 rotations per second. The rotation speed value is then sent back to the system as a sensor value.

By mapping the commands in the description in the XML configuration file, the corresponding simulation program will be executed according to the descriptions in the <operation> items. Therefore, it demonstrates that when the machine server receives the control message, the machine hardware functions are invoked and some actions cause the change of machines sensor states.

8.2.3. Demo Software development

The demo software was developed as described by the designs in Chapters 5 to 7. The demo software code is included on the DVD attached to this thesis and the class diagrams can be found in Appendix F. According to the architecture described in Chapter 5, three java web applications are designed to implement features in different layers of the architecture:

- Machine client side project
- Service server project
- IWUI service project

The programs in each project are put in different packages according to their functions

The machine client side project is developed to demonstrate AL application and HLL application. The detailed packages in this project and their function are explained in Table 8.2. The column "Layer" shows at which level the program is implemented. The Shared Tools implement commonly used software tools for the whole project.

Layer	Package Name	Description
Hardware Logic Layer	lbr.rcss.client.hardware	Implements hoister control and simulation application
Adapter Layer	lbr.rcss.client.adminapi	Implements message handle modules in adapter layer
	lbr.rcss.client.clientui	Implements user interface servlet functions
	lbr.rcss.client.data	Stores XML descriptions used in this project.
	lbr.rcss.client.adapter	Implements the function of Machine Adapter Application
	lbr.rcss.client.wmaapi	Implements functions of Web-service-based Machine Access API
Shared Tools	lbr.rcss.client.common	Implements the commonly used tools used in this project
	lbr.rcss.client.common.mfd	Implements machine function description and related methods
	lbr.rcss.client.common.msd	Implements machine state description and related methods

Table 8. 2 Package List Of Machine Client Side Project

The design is implemented as discussed in Section 5.5. The messages are passed to the application in this layer by invoking APIs in the package "lbr.rcss.client.wmaapi" and stored in a local database. Later, these messages are checked by a program from the package "lbr.rcss.client.adapter". This program will pass the control messages to the hardware APIs or simulation program in package "lbr.rcss.client.hardware". The important classes in this process are described below:

MachineAdapterApp.java (Appendix C.1) implements the Machine Adapter Application which passes hardware control messages to the hardware control program. This class also invokes the functions to update the MSD after the function has been executed.

AdapterControlHandler.java (Appendix C.2) implements the Adapter Control Handler which invokes functions from different modules in AL to process control services.

MachineWebService.java (Appendix C.3) is the web service API for the functions from AL. These APIs can be used by higher level projects such as services in service server.

Layer	Package Name	Description
Control Management Layer	lbr.cs.rcss.cml.sdsl.session	Implements the classes for machine control session management
	lbr.cs.rcss.cml.service	Implements the web services that can be used by other services or systems
Resource Layer	lbr.cs.rcss.rl	Implements the management logic for the resources used in the project
	lbr.cs.rcss.rl.device	Implements the management logic for the device resources
	lbr.cs.rcss.rl.device.state	Implements the management logic for the device state during control process
	lbr.cs.rcss.rl.device.webservice	Implements the reference of device web services used in the project
	lbr.cs.rcss.rl.user	Implements the management logic for the user data
	lbr.cs.rcss.ontology	Stores machine model ontology used in the demo
Shared Tools	lbr.cs.rcss.common	Implements the commonly used tools used in this project
	lbr.cs.rcss.common.model	Implements the machine model used in this project

Table 8. 3 Package List Of Service Server Project

A service server project is developed to demonstrate the designs in RL and CML. The project implements the designs described in Sections 5.4 - 5.7. The package description for the project is given in Table 8.3. Among the packages in RL, the programs in “lbr.cs.rcss.rl.device” and sub-packages maintain machine states and manage machine resources in RL. The ontology knowledge resource will be supplied by programs in the package “lbr.cs.rcss.ontology”. In CML, all basic web services used by services in SDL are implemented in the “lbr.cs.rcss.cml.sdsl.session” package. For session related services that involve resources from RL, will be implemented in “lbr.cs.rcss.cml.sdsl.session”:

MachineKnowledgeHandler.java (Appendix D.1) is the implementation of Machine Knowledge Resource Handler which helps the system to retrieve machine model information.

DeviceReHandler.java (Appendix D.2) is the implementation of Device Resource Handler. This class communicates with controlled machines using machine web services. It also maintains device states and records the current control machine state.

MachineControlService.ava (Appendix D.3) implements one of the basic web services described in Section 5.7. It provides the APIs for machine control management service which offers web services for machine control and machine state maintenance.

An IWUI service project is developed to demonstrate the design of the IWUI service. This service achieves the design in Chapters 6 and 7. The detailed description of project packages is listed in Table 8.4. Web user interfaces in this project are produced using JSP. "Struts2" [Apache Software Foundation, 2010] is used to achieve MVC for static web pages; Corresponding actions and Servlet for these pages are implemented in "lbr.cs.rcss.sdl.wui.action". The control "struts.xml" file is attached in the appendix (Appendix E.2). For session dependent content in the WUI, "lbr.cs.rcss.sdl.wui.sh" offers methods to generate dynamic content for the control page. "lbr.cs.rcss.sdl.wui.model" and sub-packages implements the data model behind each page. One of the important classes to achieve this process is ModelReasoner.java. (Appendix E.1) It implements the Model Reasoning module which generates CPL according to Session Context Records.

Layer	Package Name	Description
Service Development Layer	lbr.cs.rcss.sdl.wui.model.creator	Implements IWUI creator
	lbr.cs.rcss.sdl.wui.model.creator.session	Implements IWUI creator session management logic
	lbr.cs.rcss.sdl.wui.model.service	Implements IWUI model support web service that referenced from service server
	lbr.cs.rcss.sdl.wui.sh	Implements IWUI Service Handler
	lbr.cs.rcss.sdl.wui.action	Implements the Servlet and struts2 actions triggered via web user interface
Shared Tools	lbr.cs.rcss.sdl.wui.common	Implements the commonly used tools used in this project

Table 8. 4Package List Of IWUI Service Project

8.3. Evaluation process

8.3.1. Machine initialisation

Before a target machine is available for users to control, they need to be initialised. The initialisation is processed on the machine server side and then a new machine service for the target machine is published to the service server so that the machine is available for user control.

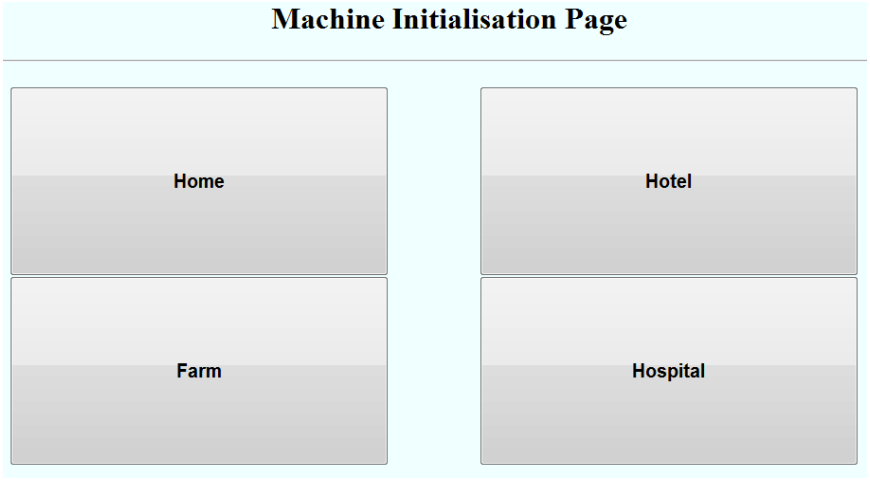


Figure 8. 5 “Machine Initialisation Page” With Main Location Options

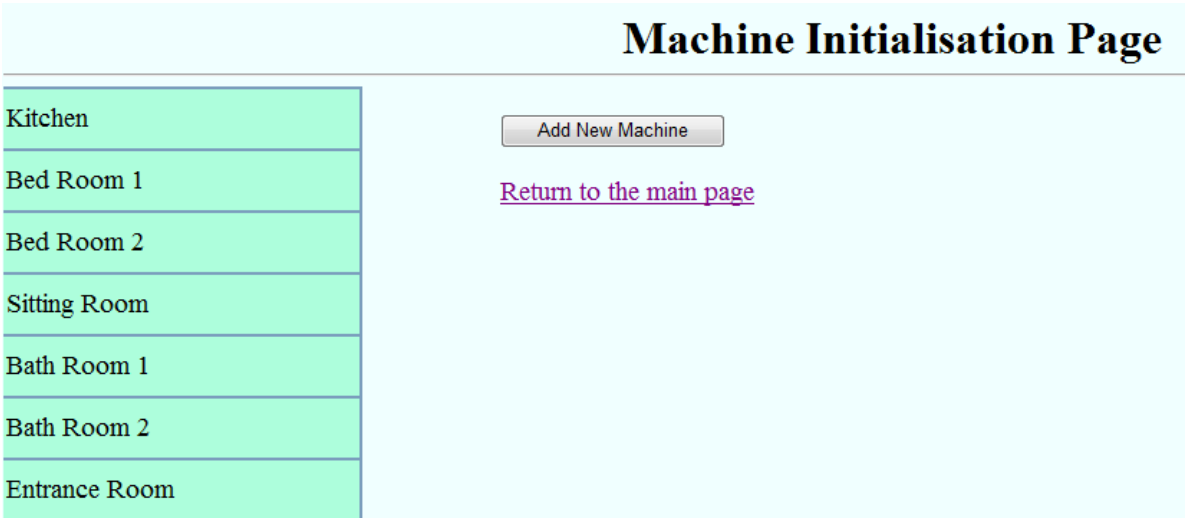


Figure 8. 6 “Machine Initialisation Page” With Sub Location

Figure 8.5 shows the “Machine Initialisation Page” generated according to the location map file. Four buttons are generated automatically according to the four available locations. After

selecting one of the locations, the page content will show the list of sub-locations for the selected location. For example, if "Home" is selected then a new page as shown in Figure 8.6 appears with a list of sub-locations for "Home". When the user selects one of the sub-location then machines already registered for that sub-location will be shown. Because there is no registered machine yet, only an "Add New Machine" button is shown.

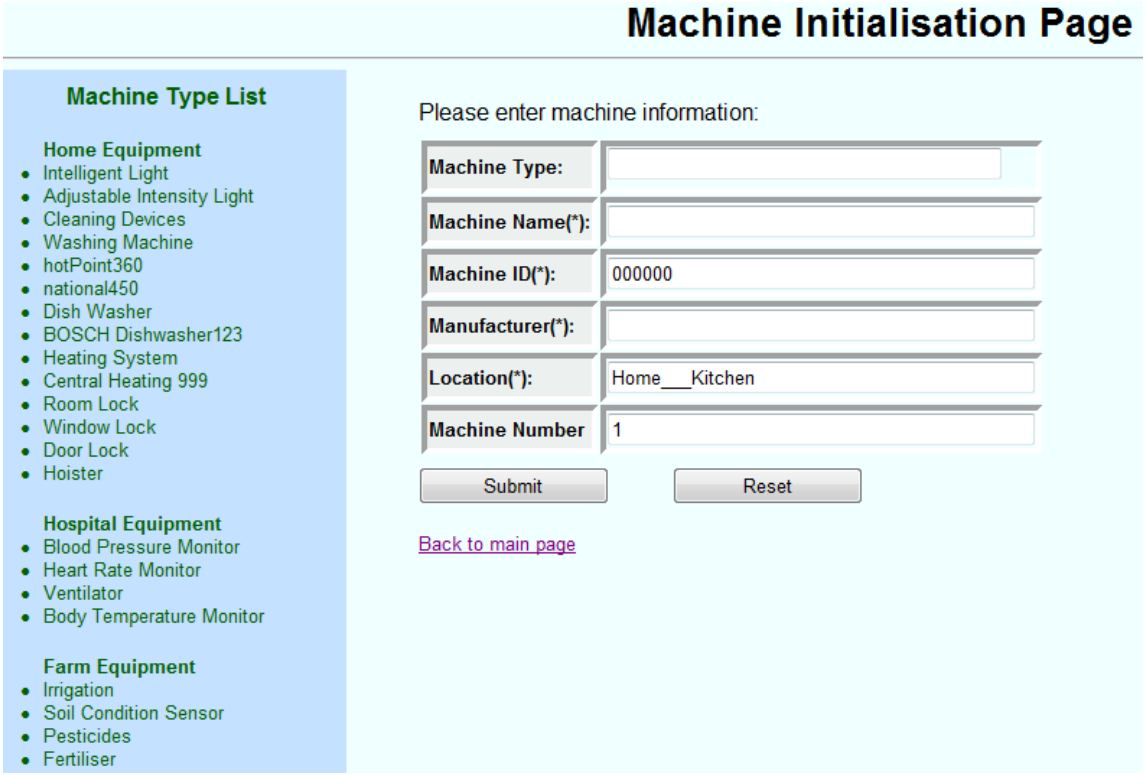


Figure 8. 7 New Machine Information Initialisation Page

When the user clicks the button "Add New Machine" an interface for adding and initialising a machine is generated as shown in Figure 8.7. On the left side of the page, a side bar is generated automatically to show a list of machine types that the system can identify. This list is generated according to the content of the ontology described earlier. According to the ontology, there are three main categories: "Home Equipment", "Hospital Equipment" and "Farm Equipment". On the right side panel, there is a form for machine initialising a machine in the selected sub-location. Some of the information supposed to be set by the factory, such as "Machine Type", "machine ID", and "Manufacture". The "machine ID" is a unique ID to identify a machine instance so it is generated automatically by the demonstration program to make sure it is unique. The user needs to set "Machine Name" and "Machine Number" to identify the machine name and how many machines need to be initialised. In order to make the

hardware simulation flexible, these items can all be set manually. When the form is submitted, the machine information will be available to the system so that the machine will be found by the server as a service ready to be controlled.

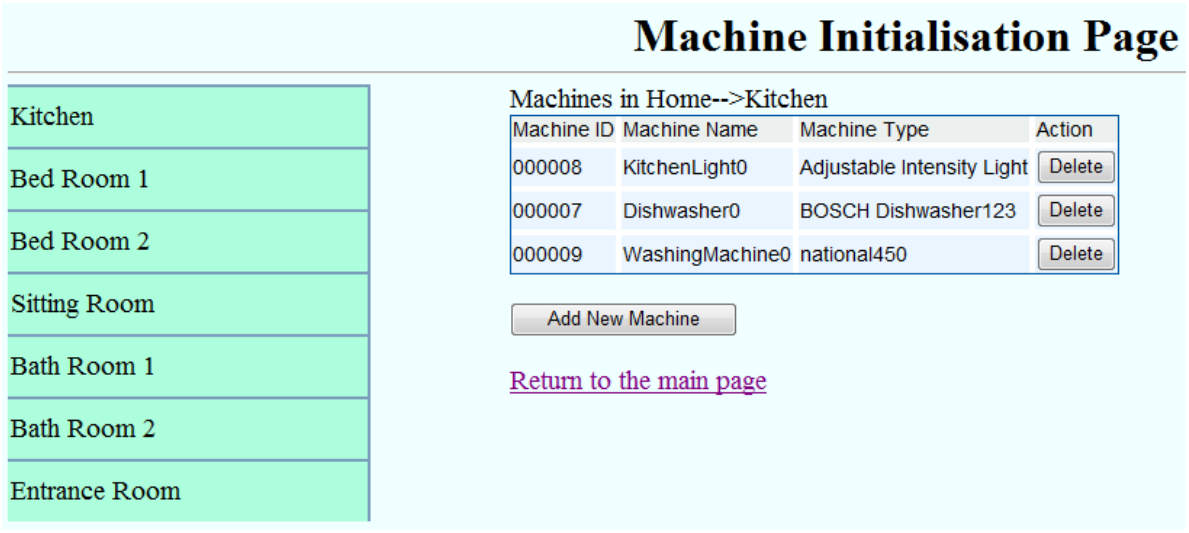


Figure 8. 8 “Machine Initialisation Page” Example With Registered Machine

The result of the machine initialisation in the sub-location “Kitchen” of “Home” is shown in Figure 8.8. It shows that there are three machine records initiated in the sub-location. At the same time, machine description files are automatically generated in the corresponding directories.

In the case of real hardware, before a machine type is registered, the initial attributes as well as hardware functions in MFD are set by the manufacturer. The machine service is published to the service server and available for user to find and control.

8.3.2. Client-side evaluation process

After all the controlled machines are registered on the machine server, a user can now login to the system and find these machines and control them from a client-side computer. In the following different evaluation scenarios is used to test different aspects of the design. However, before these evaluation scenarios are introduced, the client-side workflow will be described in this section, so that it will be easier to understand the evaluation process.

The workflow follows the site map shown in Figure 8.9. After a user has login the “User Welcome Page” will show the list of registered machines available to the user as well as some

system options. Currently, two options are available: register machine and control machine. When the user chooses register machine, the "Machine Registration Page" will show the user all available machines that have already been registered with the server. The user can also register additional machines or new functions for the machines using the "Function Registration Page". The information will be recorded by the system for future use. If the user chooses to control machine then the "Control Machine Selection Page" will show the user the list of machines that have been registered by the user. After the user has selecting the required machines the "Session Dependent Information Setting Page" will offer some forms for the user to enter information related to the control session mentioned in chapter 7. When this is finished, the system will analyse what needs to be displayed on the control page. If is the content is suitable for one screen then the "Control Panel Page" is generated for the user. Otherwise, the user needs to set navigation information using the "Navigation Analysis Page", and the corresponding "Control Panel Page" will be generated with the navigation information.

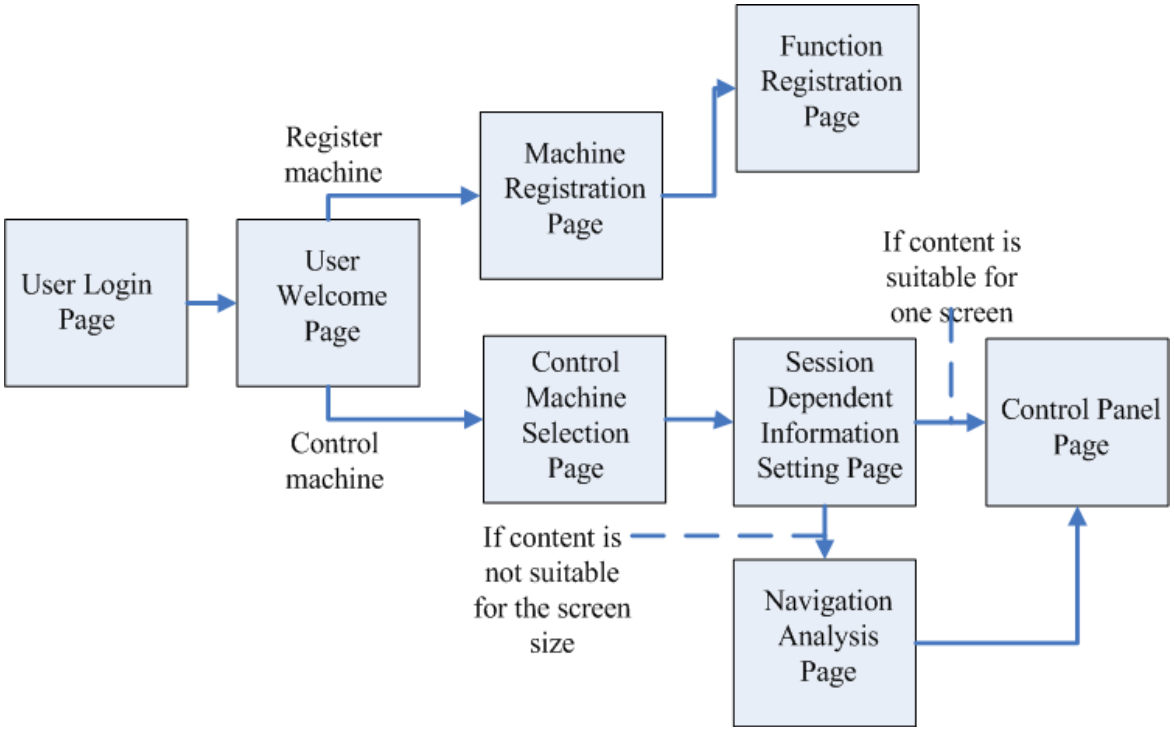


Figure 8. 9 Client Side WUI Site Map

8.3.3. Design of the evaluation scenarios

After the evaluation environment is set, different cases are created to test the feasibility and result of the design. According to the research problems and aims discussed in Chapter 1,

these evaluation scenarios are to test how the system achieves the design goals and how it functions for different users, different machines and different control processes.

The model hoister is used as the first evaluation scenario to test the connection to real hardware. If an appropriate CWUI can be generated dynamically and the user can control the hoister correctly then the architecture for control over the internet is shown to be feasible.

Second, a case for a user who wants to control their home appliances is used. This case aims to show that the CWUI can be generated automatically for different machines. In this evaluation scenario, different machines are required to be controlled and monitored. Therefore, the system needs to automatically retrieve these machines' structure and functions and generate different user interfaces for the user to control them.

Third, a simple hotel scenario is used. There are two features in this evaluation scenario: 1) there are different users who want to control the same group of machines; 2) the control information can not be easily fit onto one screen. The scenario aims to show that the system can generate different CWUI according to different user requirements and that navigation can be generated to help different users to control and monitor their selected machines easily.

In order to test the result of how the system help users to access critical control data using session dependent information such as "alarm information" the fourth evaluation scenario is given. In this scenario, different farm machines are controlled according to certain sensor values. The evaluation scenario aims to show how alarms can be set and displayed on the CWUI. This evaluation scenario also shows how the navigation help user to find the target machines according to their need.

Finally, an evaluation scenario of hospital equipment monitoring and control is used to show how the system dynamically creates a navigation page with user defined session dependent information for the user. Different from machines in other evaluation scenarios, hospital equipment usually needs to link with session dependent information, "additional information", which is set by users before control page is generated. The evaluation scenario aims at demonstrating how the system functions collect the data during the control process and merge the dynamic data into the navigation structure and how the user uses the information to navigate their way through the information.

Selected typical machines are registered for each evaluation scenario. Although each evaluation scenario has its own main focus, they can be considered as a set of evaluation scenarios that shows the system can deal with different machines, users, and processes.

8.4. Hoister hardware control

Step 1: Register machine in the service server.

The hardware hoister is linked with the computer which can access the internet. According to the process described in Section 8.3.2, after logging in to the system, the user needs to register the hoister on the Machine Registration Page. As shown on Figure 8.10, a hoister is found by the system and its information is listed on the Machine Registration Page.

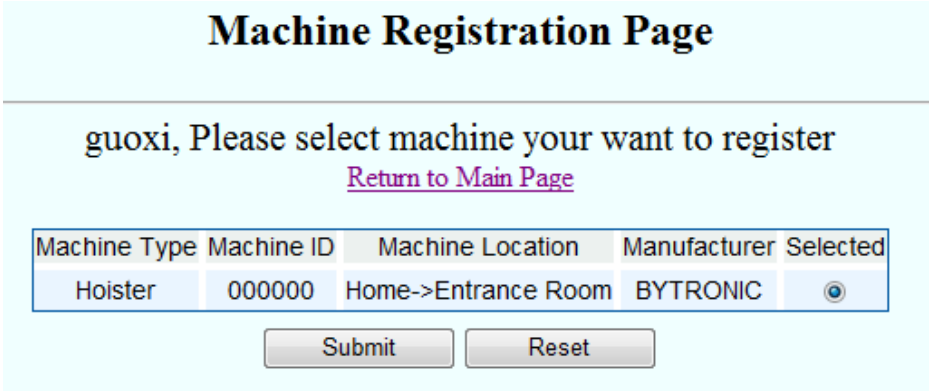


Figure 8. 10 “Machine Registration Page” In Hoister Control Evaluation Scenario

After selecting the hoister and clicking on “Submit” on the page shown in Figure 8.10, the user enters “Function Registration Page” (Figure 8.11). The “Function Registration Page” lists all available machine sensors, commands and component information. This page is generated automatically according to the ontology stored on the service server. There are two actuator options. The component called “Vertical Hoister” controls the movement of the hoister in the vertical direction; the “Horizontal Hoister” controls the movement of the hoister in the horizontal direction. Since there is no sensor or command for the hoister according to the machine model, there are note shows “There is no sensor in this machine” and “There is no command for this machine”. Here “command” means high level command described in machine model before. In this evaluation scenario, all the functions are registered.

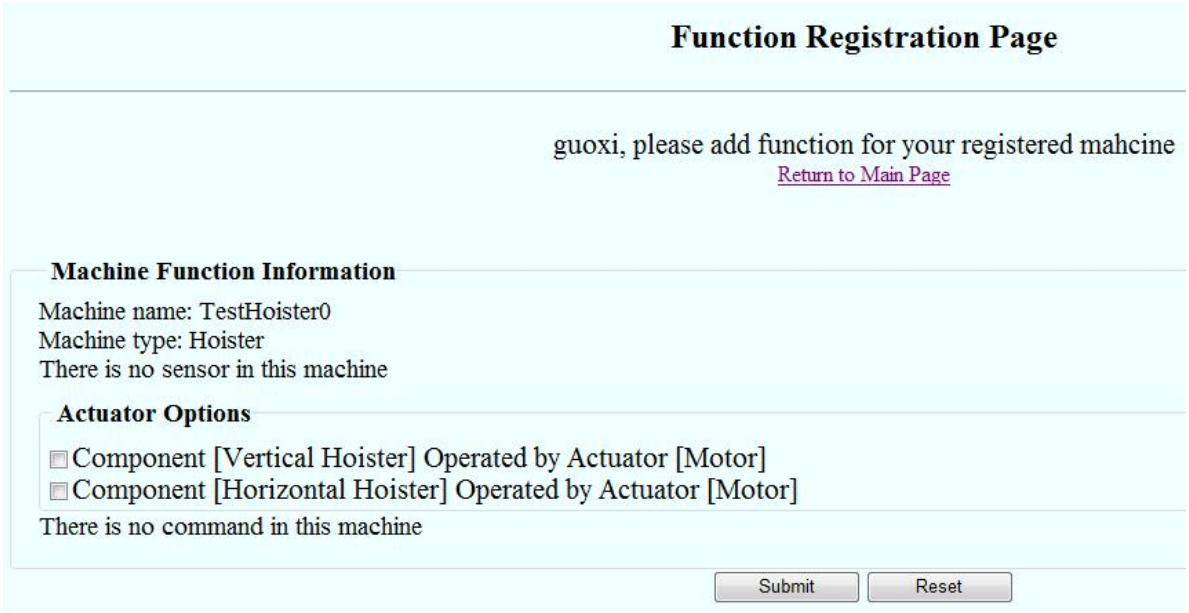


Figure 8. 11 "Function Registration Page" For The Hoister

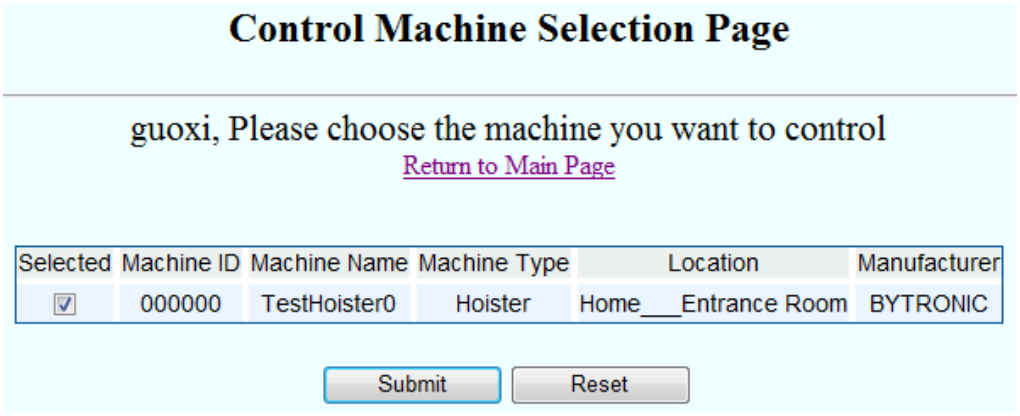


Figure 8. 12 "Control Machine Selection Page" In Hoister Control Evaluation Scenario

Step 2: Select controlled machine and generate CWUI

When the user selects controlled machines, the registered hoister will be shown on the "Control Machine Selection Page" (Figure 8.12). After selecting the hoister on the "Control Machine Selection Page", "Control Panel Page" is generated as shown in Figure 8.13. All the machine function information is grouped according to machine model. It is grouped in a "General Information Panel" at the top of the machine information and in an "Actuator Control Panel". The "General Information Panel" shows the general machine information such as "Machine Name", "Machine ID" and "Machine Type". There is no sensor or command on the hoister so there is no corresponding panel shown on the CWUI. The "Actuator Control

Panel" shows the information and state for available actuator components – "Vertical Hoister" and "Horizontal Hoister" – with their available actuators that can be controlled. The actuator instructions are represented using buttons so that they can trigger back ground control functions.

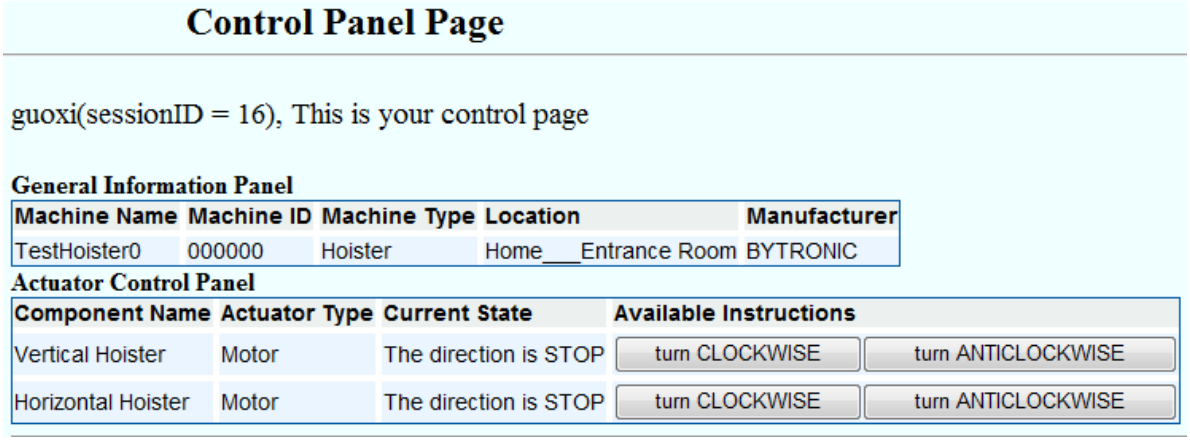


Figure 8. 13 "Control Panel Page" In Hoister Control Evaluation Scenario

Step 3: Test the function on the CWUI.

When the function buttons on the CWUI is clicked. Machine will receive control commands accordingly. For example, in Figure 8.13, when the "turn ANTICLOCKWISE" button of "Vertical Hoister" is clicked, the hoister will lift the white bar and move up. Similarly, when the "turn CLOCKWISE" button of "Vertical Hoister" is clicked, the hoister will put the white bar down. If the "turn ANTICLOCKWISE" button of "Horizontal Hoister" is clicked, the hoister will move forward while the "turn CLOCKWISE" button of "Horizontal Hoister" will make the hoister move in the opposite direction. Accordingly, after clicking the buttons, the CWUI interface is updated to current state. Figure 8.14 shows how the CWUI changes after the functions are activated. The figure shows the result when the button "turn ANTICLOCKWISE" for "Vertical Hoister". Therefore, the hoister will lift the white bar up. At the same time, it can be seen in the CWUI that "Current State" of "Vertical Hoister" will turn to be "ANTICLOCKWISE" and the available instructions changed to be "turn CLOCKWISE" and "STOP". At this time, if the user clicks "STOP" button for "Vertical Hoister", the hoister will stop the white bar movement and the CWUI will turn back as it shown in figure 8.13.

Figure 8. 14 “Control Panel Page” In Hoister Control Evaluation Scenario

The evaluation scenarios shows the hoister will process the action as required. This demonstrates that the control link, the interface as well as the message passing are all work as expected.

8.5. Home facility monitoring and control

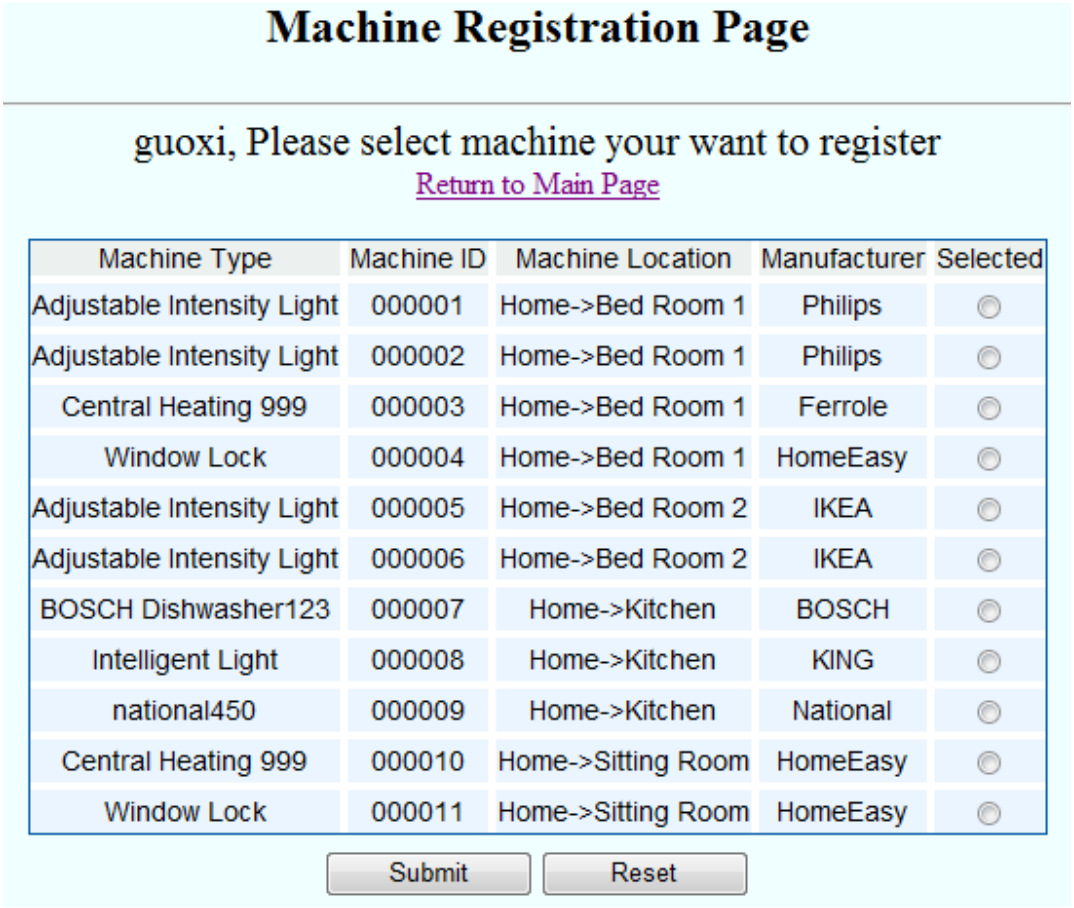


Figure 8. 15 “Machine Registration Page” In Home Facility Evaluation Scenario

In this evaluation scenario, some commonly used home appliances such as dish washer, washing machine, and lights are simulated. The system will test if different CWUI can be automatically generated for different machines. In this evaluation scenario, a user guoxi logs in the system remotely to check if everything is fine.

Step 1: Register machine in the service server.

As soon as guoxi logs in the WSRCSS system she needs to register machines she want to control and monitor from "Machine Registration Page" – intelligent lights, dish washer, washing machine, heater and window lock – in different rooms (Figure 8.15).

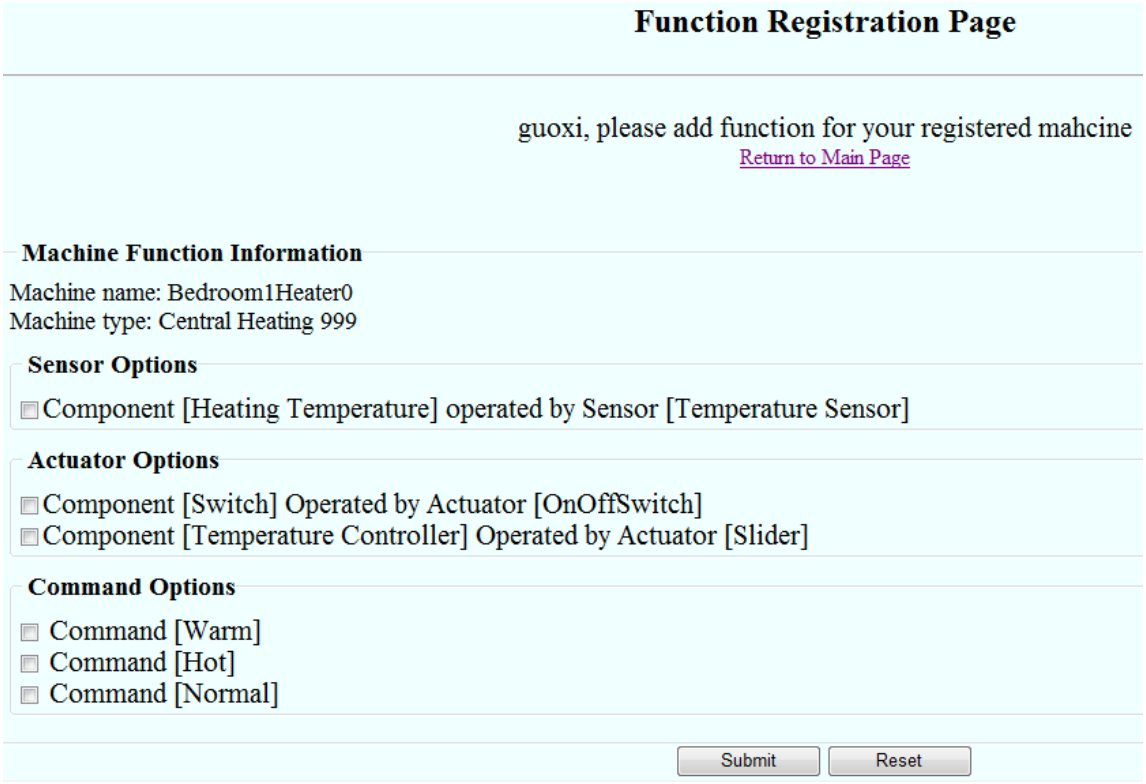


Figure 8. 16 "Function Registration Page" Example

After selecting a machine, guoxi needed to set machine detail functions she wanted to control or monitor in "Function Registration Page". Figure 8.16 shows guoxi wanted to set function for a machine with type "Central Heating999" with the name "BedroomHeater0". Without input any information other than submit the selection in Figure 8.16, the "Function Registration Page" for each machine will be generated automatically as shown in Figure 8.16. There is one sensor called "Heating Temperature" available and two components – "Switch"

and “Temperature Controller”. Three commands are available: “warm”, “hot” and “normal”. By ticking the select box before these items, guoxi is able to register the functions. In this evaluation scenario, guoxi selected all the functions available for every machine in her home.

Step 2: Check the state of each machine from its CWUI.

Guoxi began to check the machine one by one to ensure they are all in the expected states. Figure 8.17 ~ Figure 8.24 shows the CWUI generated automatically for all types of the machines with all their functions registered.

General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
BedroomLight0	000001	Adjustable Intensity Light	Home___Bed Room 1	Philips

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Light Intensity Detector	Light Sensor	0(W/m2)

Command Control Panel

Command Name	Command State	Available Command
Bright Mode	false	turn ON
Evening Mode	false	turn ON

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions
Light Intensity Controller	Slider	The state is STOP	slide DOWN slide UP
Switch	OnOffSwitch	The state is OFF	turn ON

Figure 8. 17 CWUI Example For The “Adjustable Intensity Light”

General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
KitchenLight0	000008	Intelligent Light	Home___Kitchen	KING

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Light Intensity Detector	Light Sensor	0(W/m2)

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions
Switch	OnOffSwitch	The state is OFF	turn ON

Figure 8. 18 CWUI Example For The “Intelligent Light”

The machine information is grouped according to machine model. For example, in Figure 8.17 “Adjustable Intensity Light” 000001 has three panels – “General Information Panel”, “Sensor Monitor Panel” and “Actuator Control Panel”. In “General Information Panel”, some basic machine information is known, such as the machine name “BedroomLight0”, location “Home___Bed Room1” and manufacture “Phillips”. The “Sensor Monitor Panel” shows there is a “Light Intensity Detector” on the panel of type Light Sensor, with current value 0 W/m2. The “Command Panel” shows there are two commands, “Bright Mode” and “Evening Mode”, available for this type of light. The “Actuator Control Panel” shows two components – “Light Intensity Controller” and “Switch”. “Light Intensity Controller” is controlled by actuator “Slider”. Currently, the state is “STOP”, and there are two available instructions called “slide DOWN” and “slide UP”. By clicking “slide DOWN” the light intensity will drop, and by clicking “slide UP”, the light intensity will rise. Another component “Switch” is controlled by actuator “OnOffSwitch”. Its current state is “OFF” and available command is “turn ON”. Other CWUI, such as intelligent light (Figure 8.18), dish washer (Figure 8.19) washing machine (Figure 8.20), window lock (Figure 8.21 and Figure 8.22) and heating system (Figure 8.23 and Figure 8.24) are designed in the similar style. The difference is the content of the machine control features and control command. From these examples, it can be seen that the design is also consistent.

General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
Dishwasher0	000007	BOSCH Dishwasher123	Home___Kitchen	BOSCH

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Water Temperature	Temperature Sensor	0(C)
Spinning speed	Rotation Speed Sensor	0(R/s)
Water Pressure	Water Pressure Sensor	0(mbar)

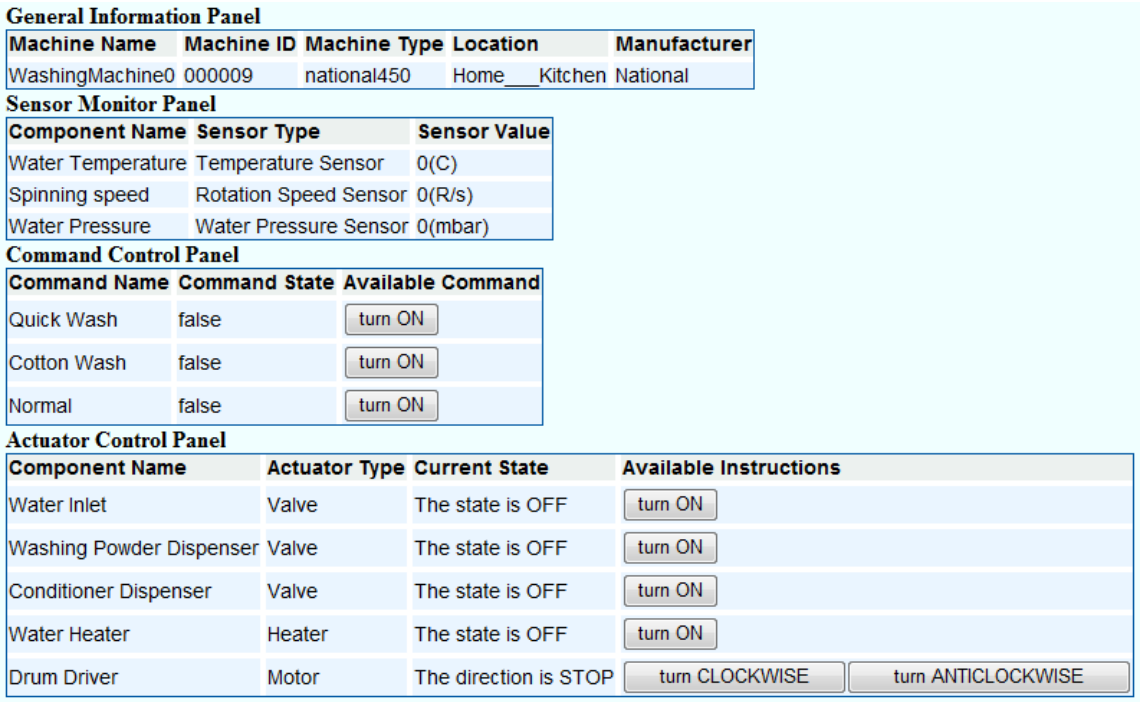
Command Control Panel

Command Name	Command State	Available Command
Quick Wash	false	turn ON
Deep Clean	false	turn ON
Normal	false	turn ON
Ecnomic	false	turn ON

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions	
Water Spray	Motor	The direction is STOP	turn ANTICLOCKWISE	turn CLOCKWISE
Water Inlet	Valve	The state is OFF	turn ON	
Washing Powder Dispenser	Valve	The state is OFF	turn ON	
Water Heater	Heater	The state is OFF	turn ON	

Figure 8. 19 CWUI Example For A Dish Washer



General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
WashingMachine0	000009	national450	Home___Kitchen	National

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Water Temperature	Temperature Sensor	0(C)
Spinning speed	Rotation Speed Sensor	0(R/s)
Water Pressure	Water Pressure Sensor	0(mbar)

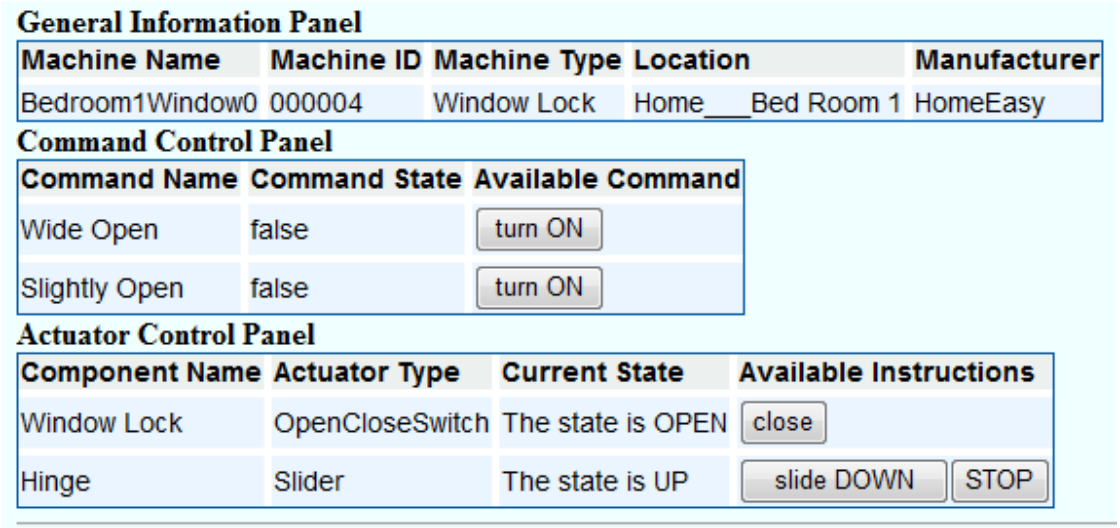
Command Control Panel

Command Name	Command State	Available Command
Quick Wash	false	turn ON
Cotton Wash	false	turn ON
Normal	false	turn ON

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions
Water Inlet	Valve	The state is OFF	turn ON
Washing Powder Dispenser	Valve	The state is OFF	turn ON
Conditioner Dispenser	Valve	The state is OFF	turn ON
Water Heater	Heater	The state is OFF	turn ON
Drum Driver	Motor	The direction is STOP	turn CLOCKWISE / turn ANTICLOCKWISE

Figure 8. 20 CWUI Example For A Washing Machine



General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
Bedroom1Window0	000004	Window Lock	Home___Bed Room 1	HomeEasy

Command Control Panel

Command Name	Command State	Available Command
Wide Open	false	turn ON
Slightly Open	false	turn ON

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions
Window Lock	OpenCloseSwitch	The state is OPEN	close
Hinge	Slider	The state is UP	slide DOWN / STOP

Figure 8. 21 CWUI Example For A Window Lock

Step 3: Change the state of the target machine from its CWUI.

By checking the information, guoxi found the “Window Lock” in “Home___Bed Room 1” is “OPEN” and the state of the “Hinge” is “UP”, which means the window is still open (Figure 8.21). After clicking ”STOP” and “close” button the “Hinge” state is changed to “STOP” and “Window Lock” state is changed to “CLOSE” (Figure 8.22) Also, guoxi also found the state

of “Switch” for the machine “Central Heater999” 000010 in the ”Home___Sitting Room” is still “ON” (Figure 8.23). She clicked “turn OFF” to turn the heater switch to “OFF”. The temperature of the heater drops from “25C” to “0C” (Figure 8.24).

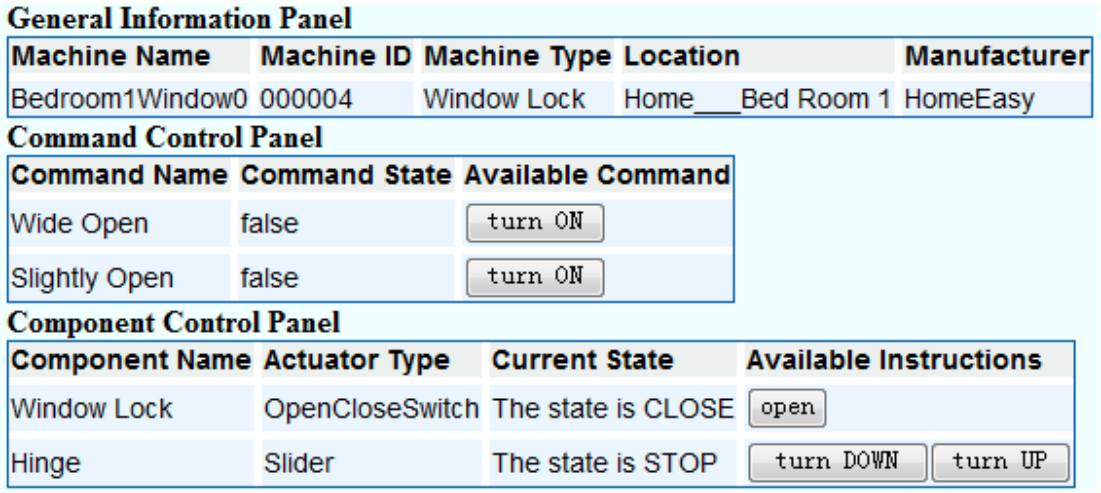


Figure 8. 22 CWUI Adaption Example For Window Lock State Change

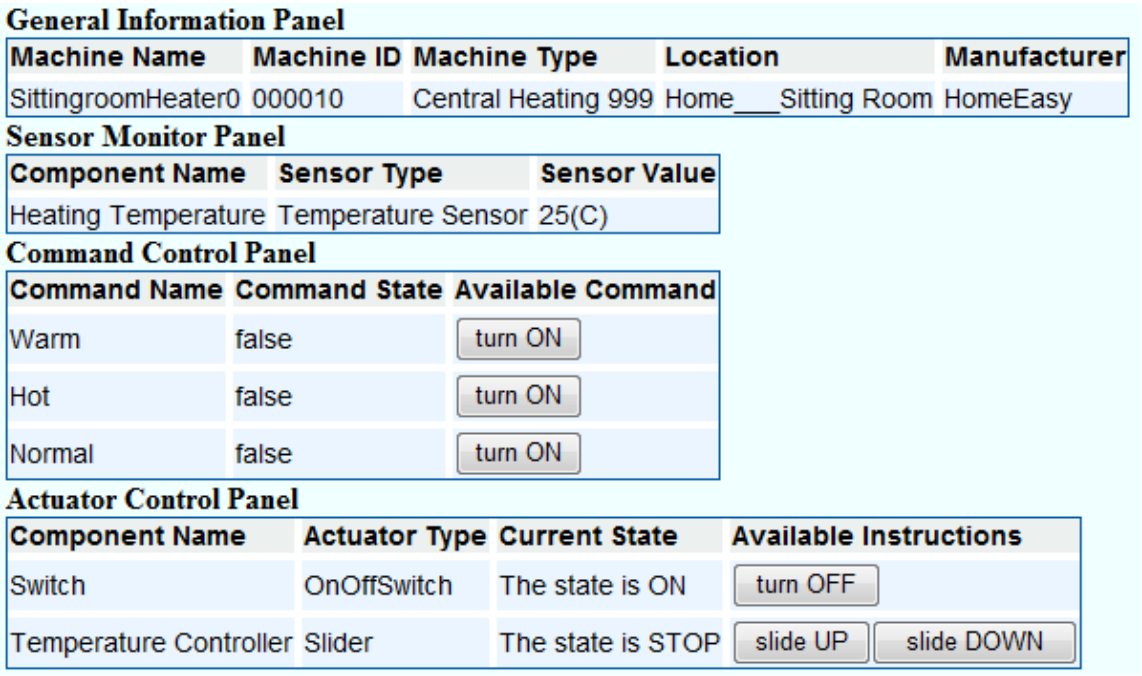


Figure 8. 23 CWUI Example For A Heating System

From the evaluation scenarios, it can be seen that when a function button is pressed, the associated control command is sent to WSRCSS, and the machine hardware simulation will be activated accordingly. The page will update when the button is pressed and will also be

updated automatically when there is no button is clicked. For example, when guoxi turned off the heater in Figure 8.23, a real time temperature drop gradually from “25C” to “0C” can be seen in “Sensor Monitor Panel”.

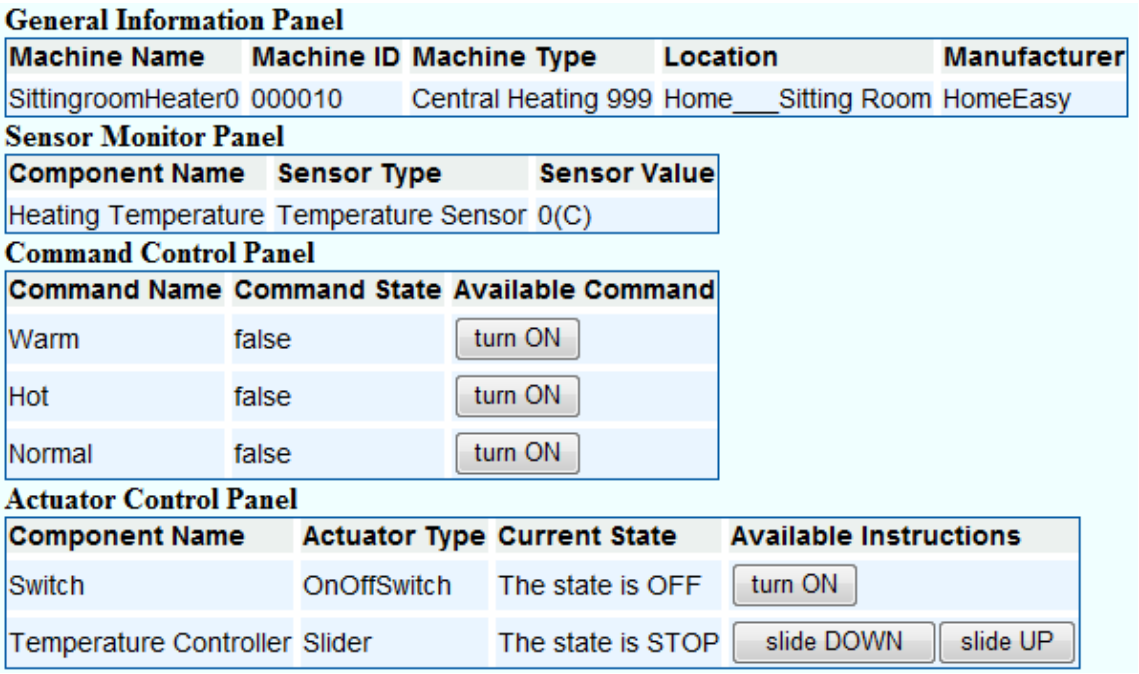


Figure 8. 24 CWUI Adaption Example For Heating System State Change

The evaluation scenario shows the design successfully generate CWUI for different machines. First, achieves automatically identify and retrieve information for different machines. After analysing using ontology, the system can get machine detail structure and functions. Second, the design achieves automatic generation of CWUI for different machines. Information for the CWUI is grouped according to its type and the user interface is designed according to consistent rules. Therefore, even when the machine is different, the CWUI is consistent. Third, the CWUI will be updated according to current control process. On the one hand, it shows only available commands in current state. On the other hand, the CWUI keeps updating real time control information. Finally, the commands available on CWUI links with the real controlled machine hardware function. Once the control command is activated, corresponding machine functions will be executed, machine state will be changed and the page content will be updated.

8.6. Hotel equipments monitoring and control

In this evaluation scenario, the machine types are similar to the home scenario. The difference is more machines need to be displayed on one CWUI. For different users, even the machines are the same, the CWUI need to be different to adapt different user requirements. The evaluation scenario shows two managers in a hotel use the system. Manager A needs to make sure the lights in every room are turned off. These rooms are in different parts of the hotel, and on different floors. Manager B needs to monitor energy consumption for the same group of lights in these rooms. If the managers need to check the room one by one, it will be a massive work. Evaluation scenario is needed to test if the design can solve these problems for the managers, and if the CWUI can be generated to meet different user requirements.

Machine Type	Machine ID	Machine Location	Manufacturer	Selected
Intelligent Light	000012	Hotel->Room 108	Philips	○
Intelligent Light	000013	Hotel->Room 1208	Philips	○
Intelligent Light	000014	Hotel->Room 201	Philips	○
Intelligent Light	000015	Hotel->Room 203	Philips	○
Intelligent Light	000016	Hotel->Room 203	Philips	○
Intelligent Light	000017	Hotel->Room 207	Philips	○
Intelligent Light	000018	Hotel->Room 802	Philips	○

Figure 8. 25 Lights Need To Be Registered In Hotel Evaluation Scenario

[Manager A] Step 1: Registers the controlled lights and functions needed

The lights needed to be controlled are listed in Figure 8.25. This figure is part of "Machine Registration Page", which listed all available machines. By ticking the radio button, a user can choose the machines they want to register. In this evaluation scenario, all machines are of the same type – "Intelligent Light", but located in different rooms. Manager A and manager B registered these machines and started control sessions at the same time, but the difference is that they registered different functions according to their needs.

Figure 8.26 shows an example "Function Registration Page" for manager A. There are three panels on the page for an "Intelligent Light". "Machine Function Information" shows general information for the machine; "Sensor Options" shows the sensor list which can be monitored by user. "Actuator Options" lists all the components can be chosen by users. There is a "Light

Intensity Detector" in Sensor Options panel but manager A only chose to register a Switch in component panel, which is related to the control task.

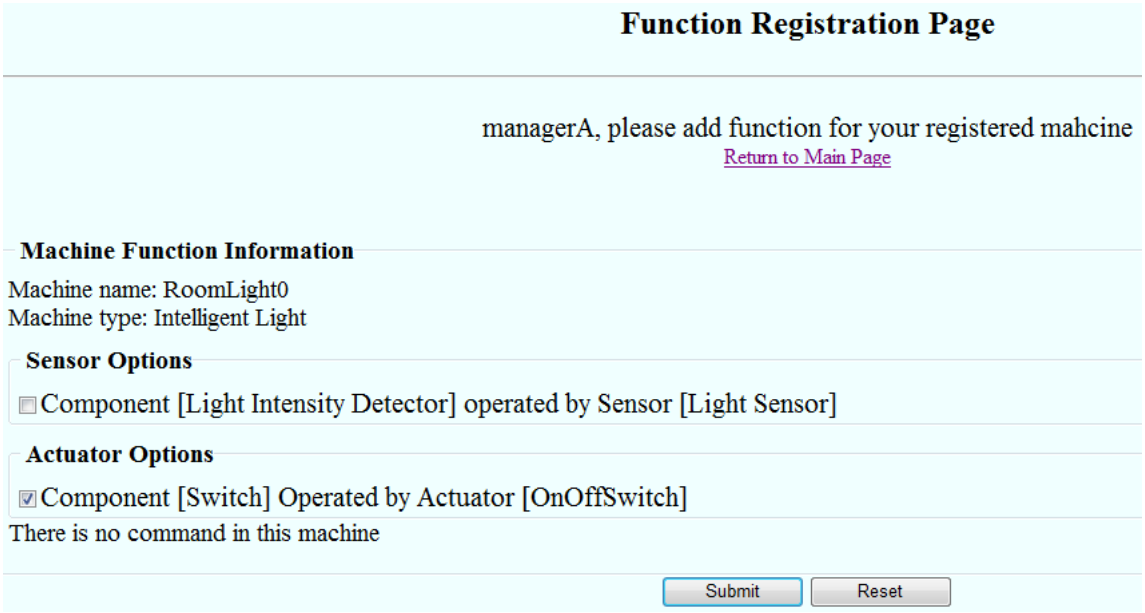


Figure 8. 26 Function Register Example For Manager A

[Manager A] Step 2: Set navigation.

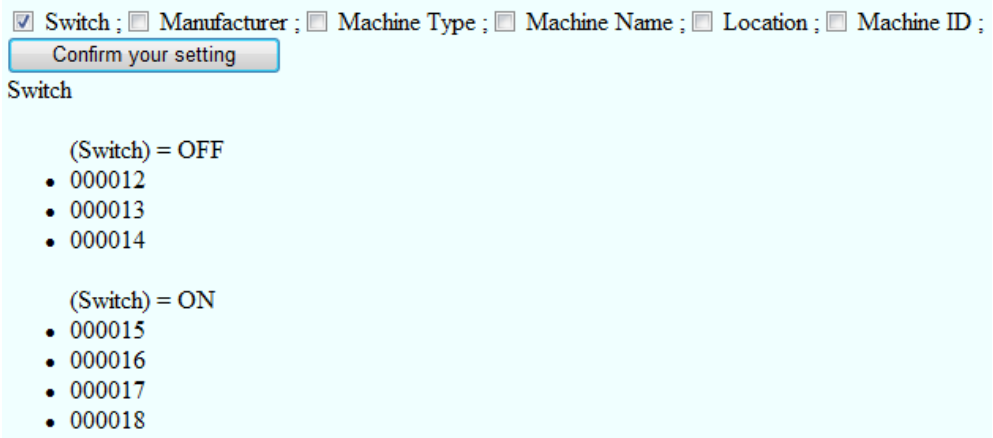


Figure 8. 27 "Navigation Analysis Page" Example For Manager A

Figure 8.27 and Figure 8.28 show the "Navigation Analysis Page" for manager A. In both figures are in three parts. On the top is the keywords list with a button "Confirm your setting". These keywords, such as "Switch", "Manufacturer" and "Machine Type" are generated automatically according to current session. Under the button, there is a line showing the KWC of the user. There is a dynamic panel under the KWC showing the navigation tree set by user.

When the user selects a keyword from the keyword list, KWC and navigation tree will change accordingly. For example, in Figure 8.27 when manager A selects “Switch”, the KWC is shown “Switch”. The navigation tree categorise the machines in to different Switch state – “ON” and “OFF”. In Figure 8.28 when manager A chose another keyword “Location”, the KWC is changed to be “Switch → Location”. The navigation tree further groups the machines according to different locations. The figure shows that KWC is generated automatically by the system.

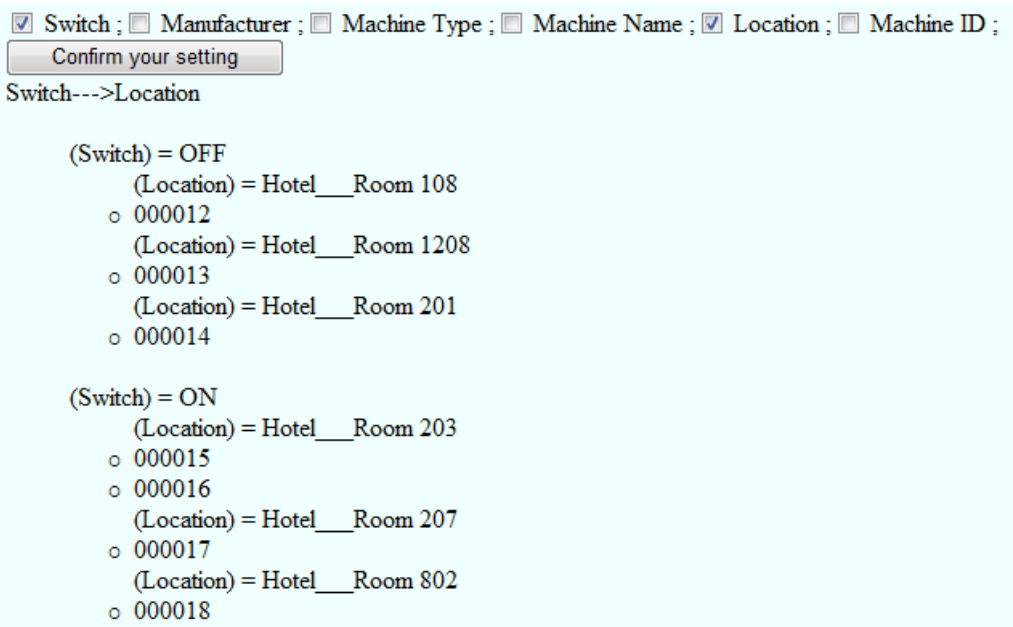


Figure 8. 28 “Navigation Analysis Page” Update Result For Manager A

The result shows that the component state information will be selected as keywords when all machines are of the same type. According to the design in Chapter 7, general information can be selected as keywords, such as “Manufacture”, “Machine Type”, and “Machine Name”. In this evaluation scenario, the component “Switch” is also chosen to be a keyword since all controlled machine types are “Intelligent Light”. In this way, it will be very easy for manager A to check which light needs to be switched off.

[Manager A] Step 3: Test the general functions for the navigation.

CWUI is automatically generated (Figure 8.29) after manager A set the navigation and press “Confirm your setting” in the navigation analysis page (Figure 8.28). Figure 8.29 shows the CWUI has a “Navigation Bar” on the left and a main control panel on the right. On the top of

the control panel, there is a label showing some general information for the current control session. For example, Figure 8.30 shows the user name is “managerA” and the current control session ID is “6”. All required machine information is listed on the control panel. Only machine functions registered by manager A is shown on the page.

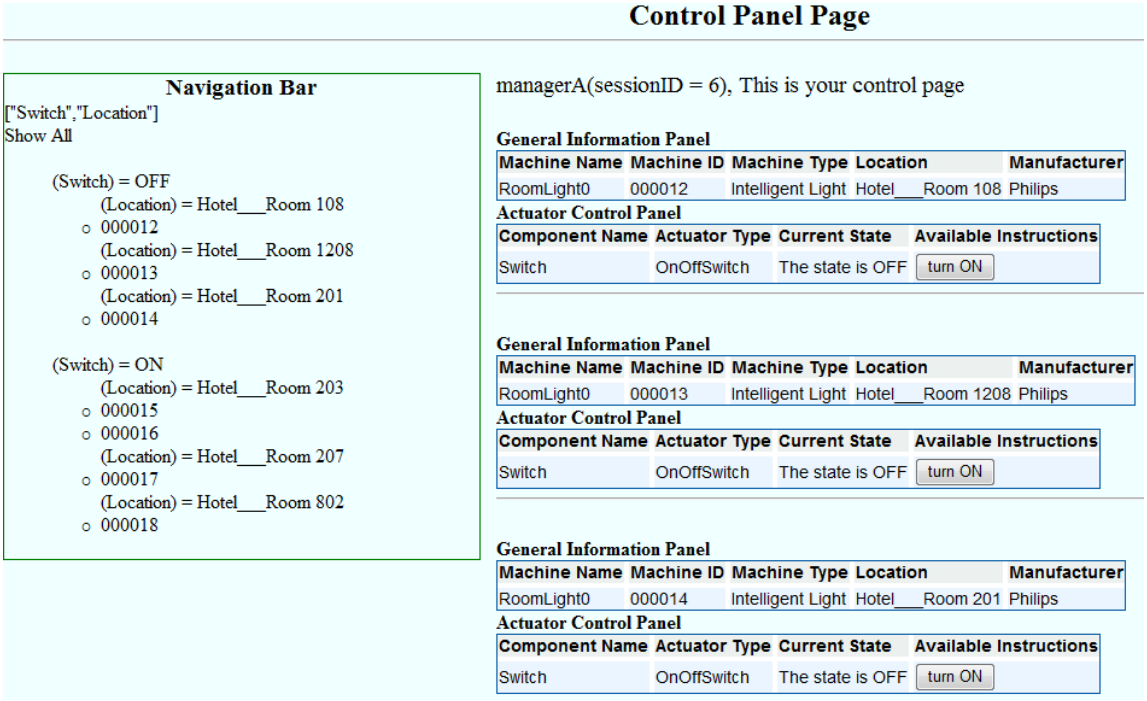


Figure 8. 29 “Control Panel Page” for manager A

[Manager A] Step 4: Test the adaption for the CWUI with navigation.

The “Control Panel Page” is updated according to user’s requirement. When manager A clicks different items on the “Navigation Bar”, different machine information will be shown as required. For example, in Figure 8.30 when manager A clicks “(Switch) = ON”, all the machines with Switch state “ON” is shown on the control panel (Figure 8.30). When manager A clicks “(Location)= Hotel___Room203” on the navigation tree (Figure 8.31) information of machine 000015 and 000016 is shown on the page. In this way, manager A can easily find the lights which are left “ON”, and turned them off.

The result also shows during the time manager A operates these machines, the “Control Panel Page” is updated dynamically according to the control process. Not only control content, but also the navigation tree can be dynamically updated according to real time changes. Figure 8.32 shows when manager A turned off the lights, all the “Switch” states for the lights are

changed to be “OFF”, and available commands are all changed to be “turn ON”. Accordingly, in the navigation tree, all the machines are grouped under the same switch state “OFF”.

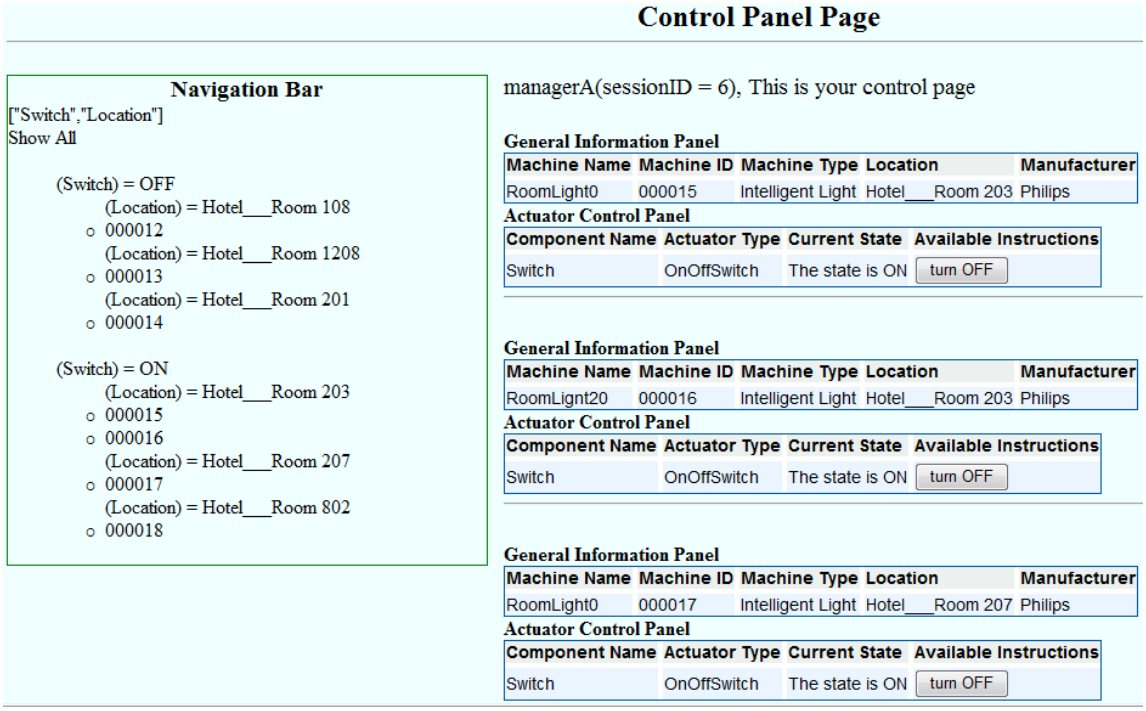


Figure 8. 30 “Control Panel Page” Adaptation Example 1 For Manager A

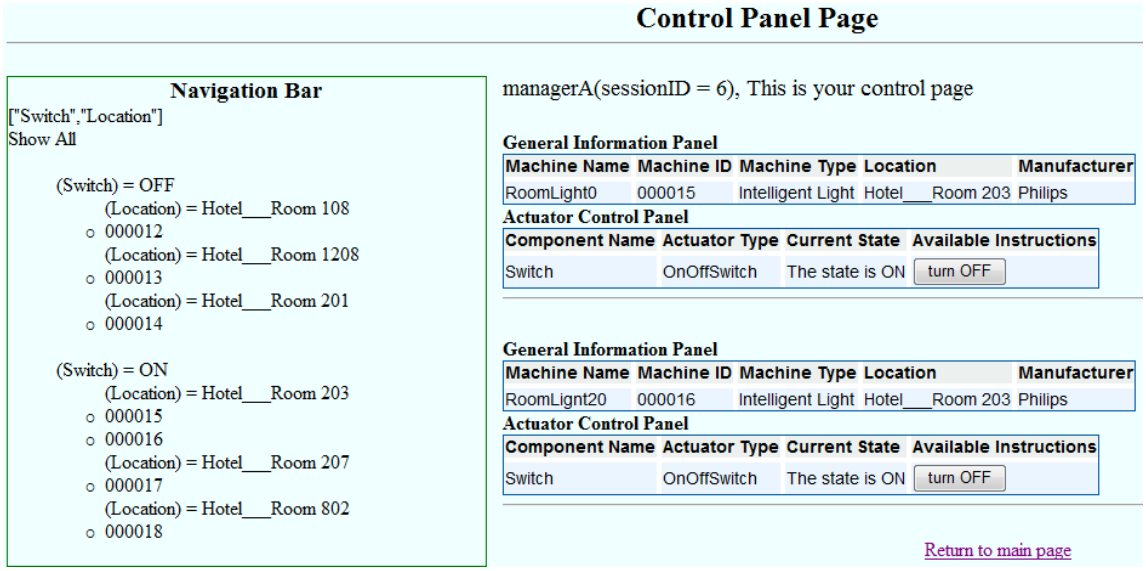


Figure 8. 31 “Control Panel Page” Adaptation Example 2 For Manager A

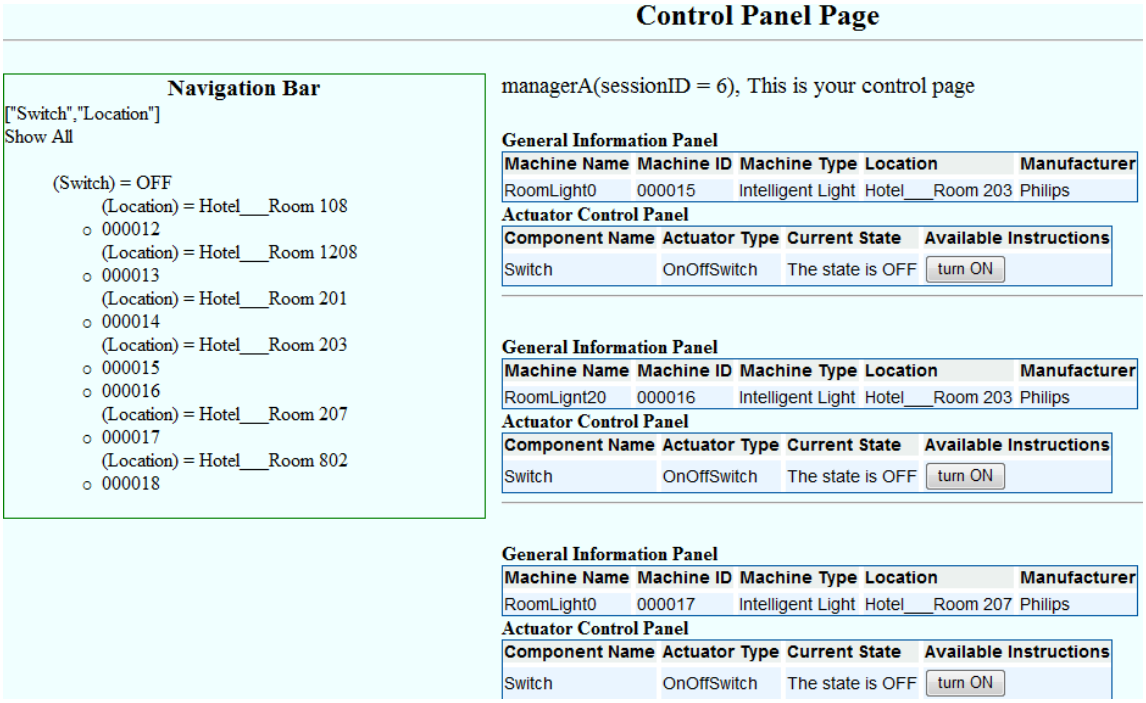


Figure 8. 32 CWUI when manager A Turned Off All The Lights

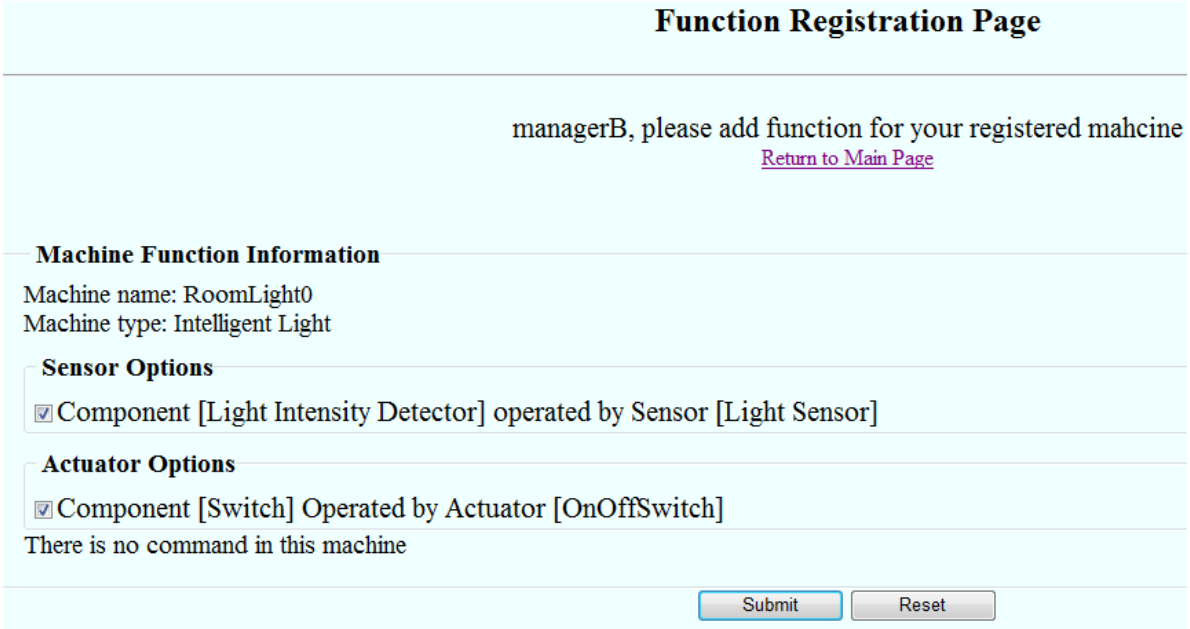


Figure 8. 33 Manager B Only Register The Sensor “Light Intensity Detector”

[Manager B] Step 1: Registers the controlled lights and functions needed

When manager A starts the control session, manager B accesses the system at the same time to monitor the same group of lights. Different from manager A, manager B only chooses both "Light Intensity Detector" and "Switch" on the "Function Registration Page" (Figure 8.33). After registering the machines and machine functions, manager B enters the session dependent information setting page. However, there is no extra information needed in this scenario, so manager B entered navigation analysis page.

[Manager B] Step 2: Set the navigation and generate the CWUI.

Same as manager A, manager B selects the navigation keywords "Switch → Location". The CWUI page is shown in Figure 8.34. The figure shows a similar CWUI to the one created by manager B. However, all the machines shown on the page for manager B have both sensor and actuator information.

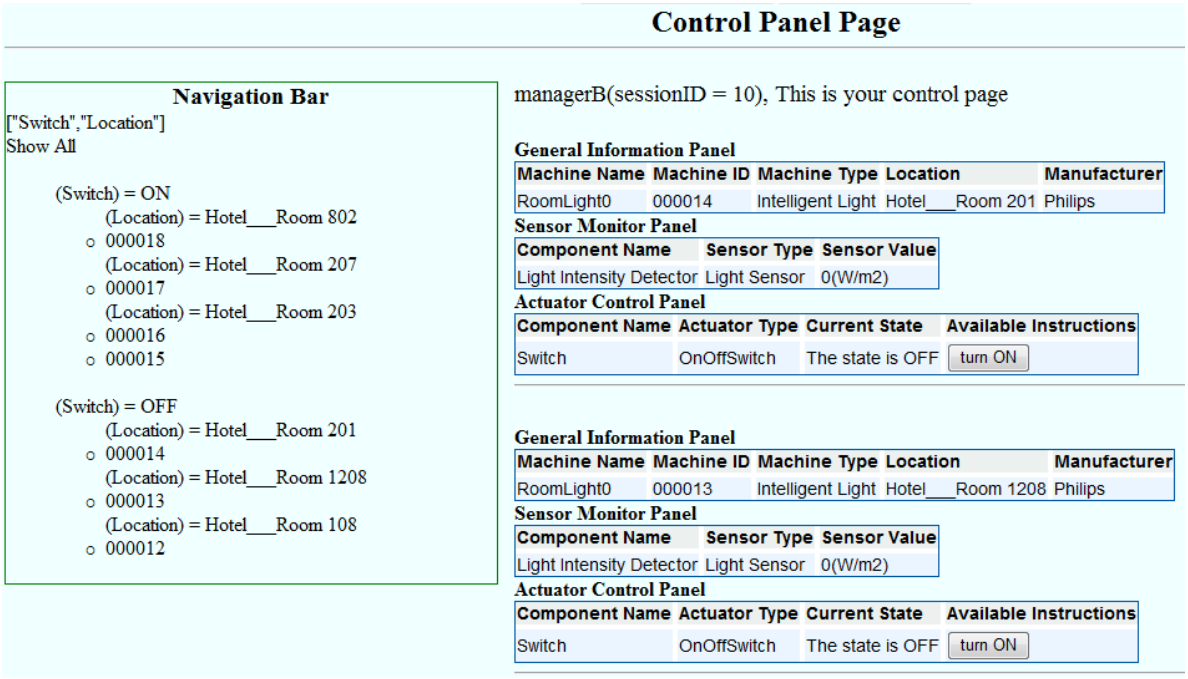


Figure 8. 34 "Control Panel Page" For Manager B

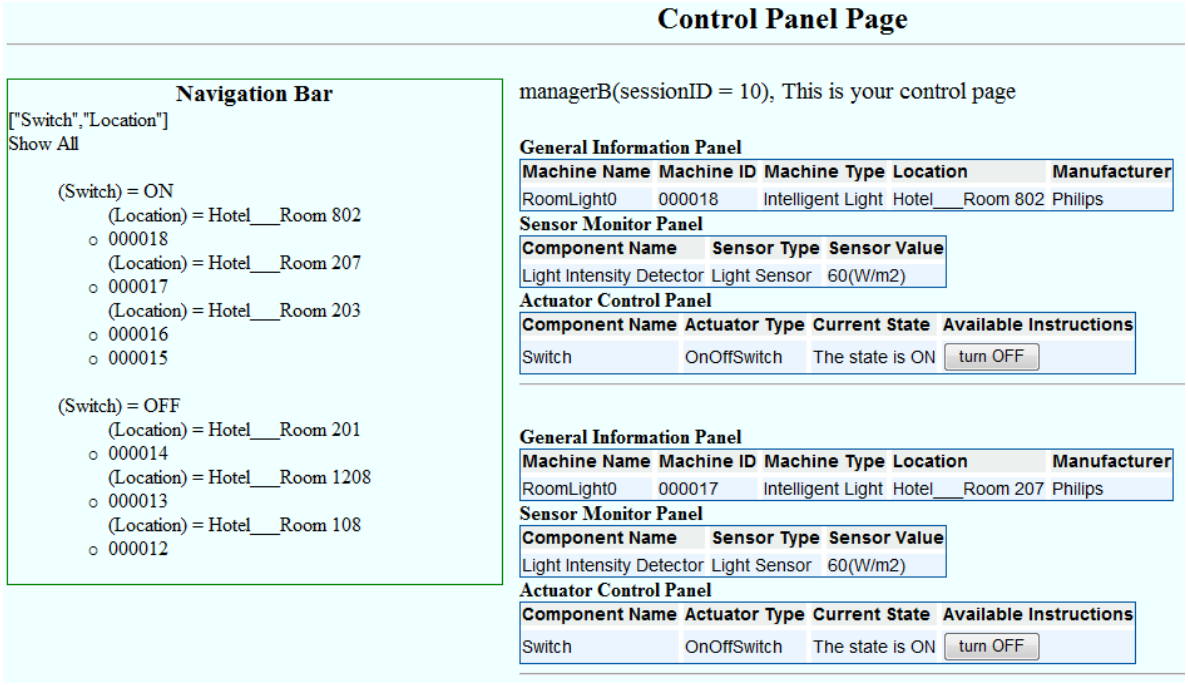


Figure 8. 35 “Control Panel Page” Adaption Example 1 For Manager B

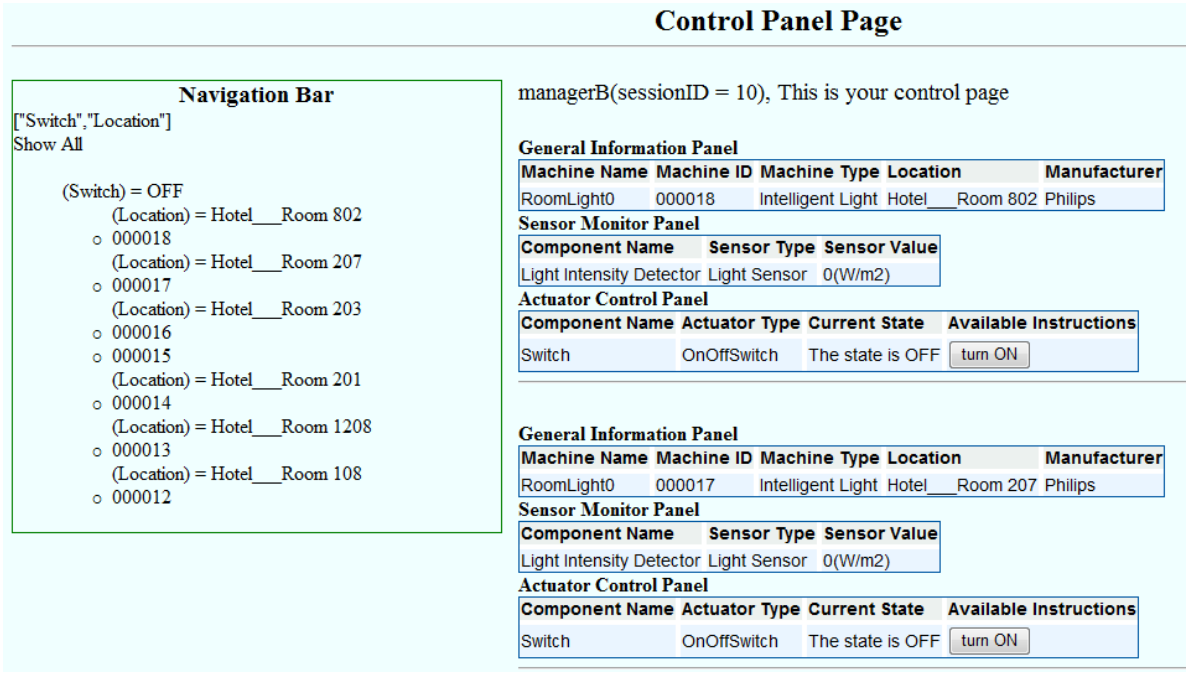


Figure 8. 36 “Control Panel Page” Adaption Example 2 For Manager B

[Manager B] Step 3: Test the adaption for the CWUI with navigation.

When manager B clicks “(Switch)=ON” on the “Navigation Bar” in CWUI shown in Figure 8.34, the “Control Panel Page” responses to the request and shows only the lights with their

switch state "On" (Figure 8.35). As shown in Figure 8.36, after manager A turned off the lights, the state and available command of the lights are updated, and the navigation tree also updated in response to the actual machine state. Also, it can be seen in the figure that when the lights are turned off, this sensor values changed from "60 W/m2" to "0 W/m2" so that manager B could monitor the light states in real time.

The evaluation scenario shows that different user interfaces for the same machines are generated to meet different user needs. Although two different managers choose the same group of machines, the system generated different CWUI after analysing their requirements. Also, the evaluation scenario shows the system allows that different users to access the same machine at the same time. It also shows that the CWUIs are updated and modified according to user actions or state changes. Once a light is turned off by a user the states is changed and CWUIs for all users monitoring that machine will be updated accordingly. Moreover, the evaluation scenario also shows examples of multiple-machine CWUI. In the multiple-machine CWUI, different controlled machines are shown in CWUI in a consistent way. The system groups the control information according to their attributes and updates the data for each machine according to the control session. Finally, the navigation is generated dynamically to help the user to access required machine or machine groups. Navigation keywords are generated automatically. When a user chooses the keywords, the navigation tree is shown in a "change as you choose" way. Using the navigation, the user can access the machine information easily.

8.7. Farm device control

In this evaluation scenario the farm machines have more sensors and the need to be operated under certain conditions. Therefore, the users need to set alarms in response to sensor values. Frank, the farmer, needs to go around his farm, which includes an orchard, a green house and a field, several times a day to finish different tasks. There are four kinds of machines need to be used – "Irrigation", "Pesticides", "Soil Condition Sensor" and "Fertiliser". A daily work is to irrigate and fertilize the plants. There are three important conditions for these tasks: irrigation needs to be started as soon as the value of "Soil Moisture Detector" falls below 50%, and need to be stopped when the value rises up to 80%. The water pressure needs to be kept below 800 mbar when using a "Fertiliser". The scenario shows how the system helps Frank to achieve this.

Selected	Machine ID	Machine Name	Machine Type	Location	Manufacturer
☐	000019	Field1Fertiliser0	Fertiliser	Farm___Field 1	Harvey
☐	000020	Field1Irrigation0	Irrigation	Farm___Field 1	Harvey
☐	000021	Field1Irrigation1	Irrigation	Farm___Field 1	Harvey
☐	000022	Field1Irrigation2	Irrigation	Farm___Field 1	Harvey
☐	000023	Field1Pesticides0	Pesticides	Farm___Field 1	FieldGuard
☐	000024	Field1SoilSensor0	Soil Condition Sensor	Farm___Field 1	FieldGuard
☐	000025	GH1Fertiliser0	Fertiliser	Farm___GreenHouse 1	Harvey
☐	000026	GH1Irrigation0	Irrigation	Farm___GreenHouse 1	SmartFarm
☐	000027	GH1Irrigation1	Irrigation	Farm___GreenHouse 1	SmartFarm
☐	000028	GH1Pesticides0	Pesticides	Farm___GreenHouse 1	FieldGuard
☐	000029	GH1SoilSensor0	Soil Condition Sensor	Farm___GreenHouse 1	Harvey
☐	000030	GH1SoilSensor1	Soil Condition Sensor	Farm___GreenHouse 1	Harvey
☐	000031	OrchardFertiliser0	Fertiliser	Farm___Orchard	SmartFarm
☐	000032	OrchardIrrigation0	Irrigation	Farm___Orchard	Harvey
☐	000033	OrchardIrrigation1	Irrigation	Farm___Orchard	Harvey
☐	000034	OrchardIrrigation2	Irrigation	Farm___Orchard	Harvey
☐	000035	OrchardPesticides0	Pesticides	Farm___Orchard	SmartFarm
☐	000036	OrchardSoilSensor0	Soil Condition Sensor	Farm___Orchard	Harvey

Figure 8. 37 Machines Need To Be Controlled In The Farm

Step 1: Register the controlled machine.

After logging in the system and registering machines needed, Frank begins to set his control panel. Figure 8.37 shows the farm machines that need to be controlled. The figure shows that there are four types of machines (“Fertiliser”, “Irrigation”, “Pesticides” and “Soil Condition Sensor”) located in three different places (“Farm___Field 1”, “Farm___GreenHouse 1” and “Farm____Orchard”). These machines have different manufacturers (“Harvey”, “FieldGuard” and “SmartFarm”).

General Information Panel

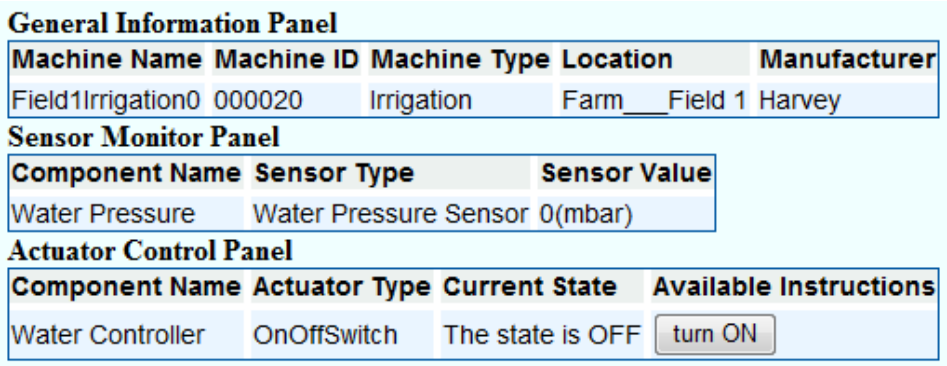
Machine Name	Machine ID	Machine Type	Location	Manufacturer
Field1SoilSensor0	000024	Soil Condition Sensor	Farm___Field 1	FieldGuard

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Soil Moisture Detector	Moisture Sensor	0(Percent)
Soil Temperature Detector	Temperature Sensor	0(C)

Figure 8. 38 CWUI Example For “Soil Condition Sensor”

An example of CWUI for a "Soil Condition Sensor" is given in Figure 8.38. This type of machine has two sensor components. "Soil Moisture Detector" is a sensor that measures the percentage of water the soil contains. Soil Temperature Detector is a sensor that measures the soil temperature.



General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
Field1Irrigation0	000020	Irrigation	Farm___Field 1	Harvey

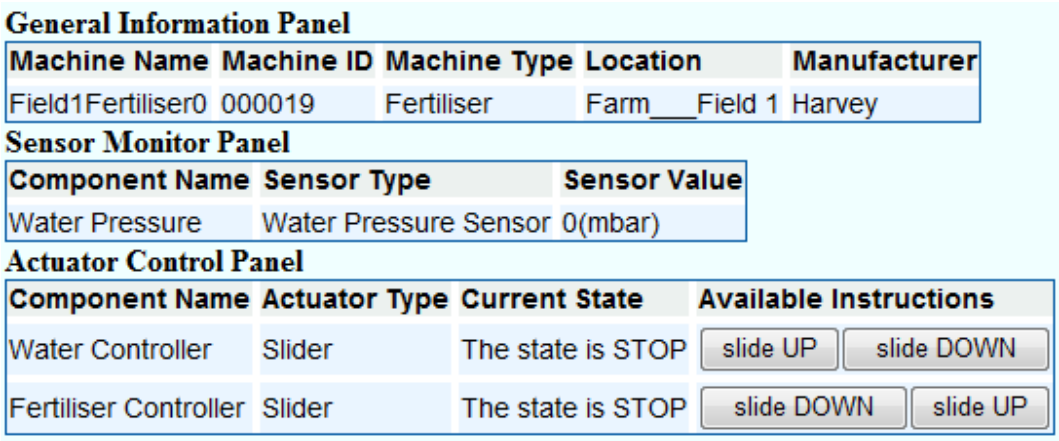
Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Water Pressure	Water Pressure Sensor	0(mbar)

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions
Water Controller	OnOffSwitch	The state is OFF	turn ON

Figure 8. 39 CWUI Example For "Irrigation"



General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
Field1Fertiliser0	000019	Fertiliser	Farm___Field 1	Harvey

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Water Pressure	Water Pressure Sensor	0(mbar)

Actuator Control Panel

Component Name	Actuator Type	Current State	Available Instructions	
Water Controller	Slider	The state is STOP	slide UP	slide DOWN
Fertiliser Controller	Slider	The state is STOP	slide DOWN	slide UP

Figure 8. 40 CWUI Example For "Fertiliser"

An example of CWUI for a machine "Irrigation" is shown in figure 8.39. The figure shows that the "Irrigation" has a "Water Pressure Sensor", which measures the water pressure in the "Irrigation" pipes. There is also an "OnOffSwitch" switch which can be used to turn on and turn off the machine. Currently in this example, the state of the Water Controller for "Irrigation" is "ON" and the value of "Soil Moisture Detector" rises up to 80 percent correspondingly.

Figure 8.40 is an example of a "Fertiliser". The figure shows there is a sensor component and an actuator component contained in this machine. "Water Pressure" is used to check the water

pressure in the pipe transferring the fertilizer liquid. “Water Controller” controls the fertiliser liquid pressure. The “Water Controller” can be “slide UP”, “slide DOWN” and “STOP”. In this example, Frank needs to click “slide UP” to raise the water pressure to a required value. If the Water Pressure is higher than 800 mbar, he needs to turn the Water Controller to “STOP” or “slide DOWN”.

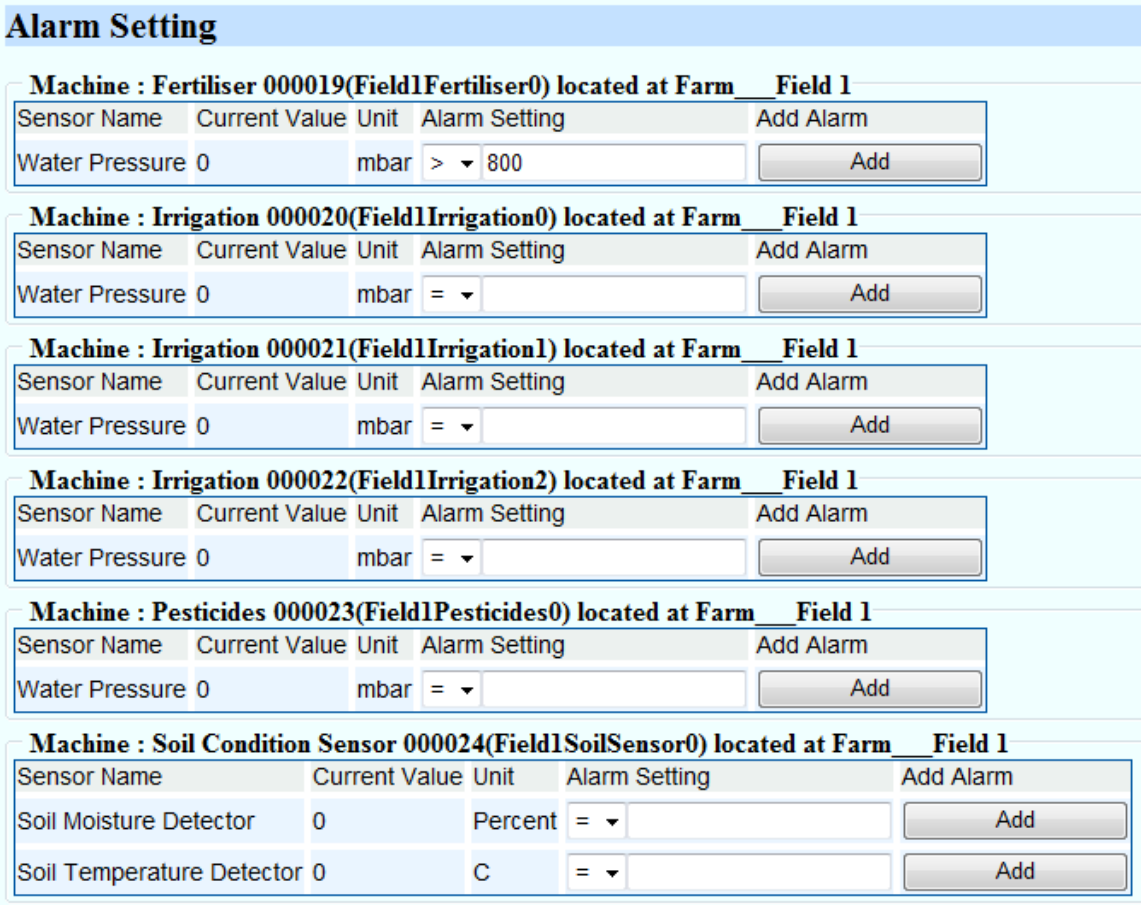


Figure 8. 41 “Alarm Setting Panel” In “Session Dependent Information Setting Page”

Step 2: Set alarm condition for target machines.

To observe the conditions mentioned above during the irrigation and fertilization processes, Frank needed some reminds to alert him of important sensor values. After Frank registered machine functions, the system offers a page which helps him to set the alarms so that he can operate the system according to his needs. Figure 8.41 shows the “Alarm Setting Panel” in the “Session Dependent Information Setting Page” generated automatically by the system. In this panel, there is a list for controlled machines that have sensor values. There is a form for each sensor in every machine. Alarm conditions can be set through these forms. After setting

required alarm condition for target sensor, the user just needs to click the “Add” button at the end of each form to add the information in to CPL for further analysis.

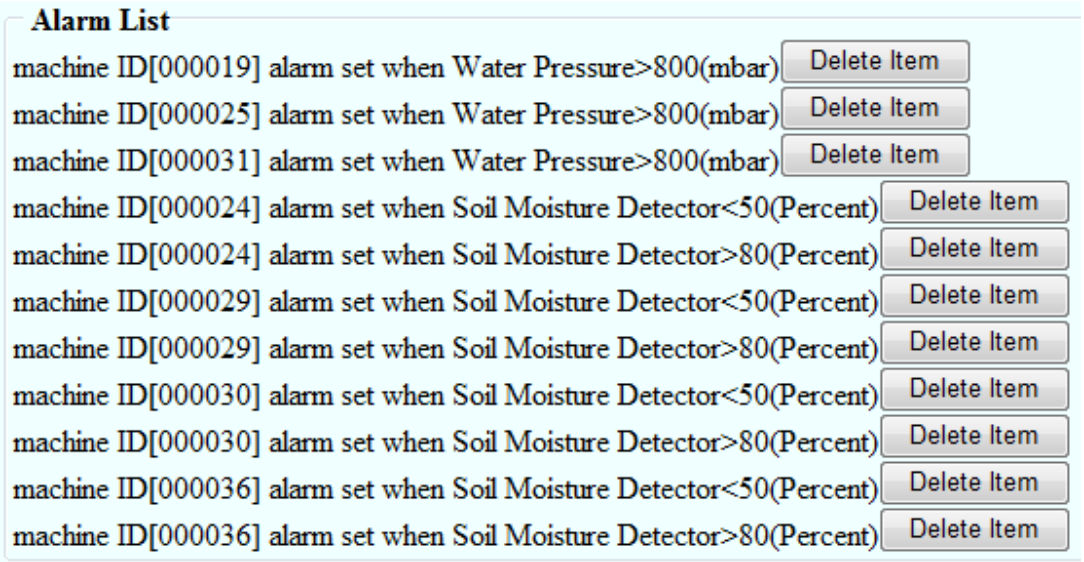


Figure 8. 42 Alarm Set List

The result of the alarm setting is shown dynamically as soon as the condition is added. As shown in Figure 8.42, an “Alarm List” is generated under the “Alarm Setting” forms. In Figure 8.43 an alarm is set for “Water Pressure” of “Fertilizer” 000019, 000025 and 000031. Once the sensor value has risen to 800 mbar, the system will alert the user. In this system, it allows users to set several sensor alarms for the same sensor. For example, Frank wanted to set two alarms for “Soil Moisture Detector’s in the “Soil Condition Sensor” 000024, 000029, 000030 and 000036. One alarm activates when the sensor value is lower than 50 percent, which means irrigation is needed in the field. Another alert is when the sensor value is higher then 80 percent, which means there is enough water in the soil and irrigation should be stopped. There is a button called “Delete Item” at the end of each alarm to delete the alarm.

For this multiple-machine CWUI, navigation is needed. Result of navigation setting is shown in Figure 8.43. Some general information: “Manufacture”, “Machine Type”, “Machine Name”, “Location” and “Machine ID”. In this example, Frank wanted to check the machines in one area by type. Therefore, he selected “Location” first and then “Machine Type”. “Machine Name” is also selected to identify controlled machines by name. The figure shows that the machine list is grouped by Location, and then machine list in the same group is separated

further by “Machine Type”, and then by “Machine Name”. In this way, although there are a lot of machines, Frank can access them efficiently.

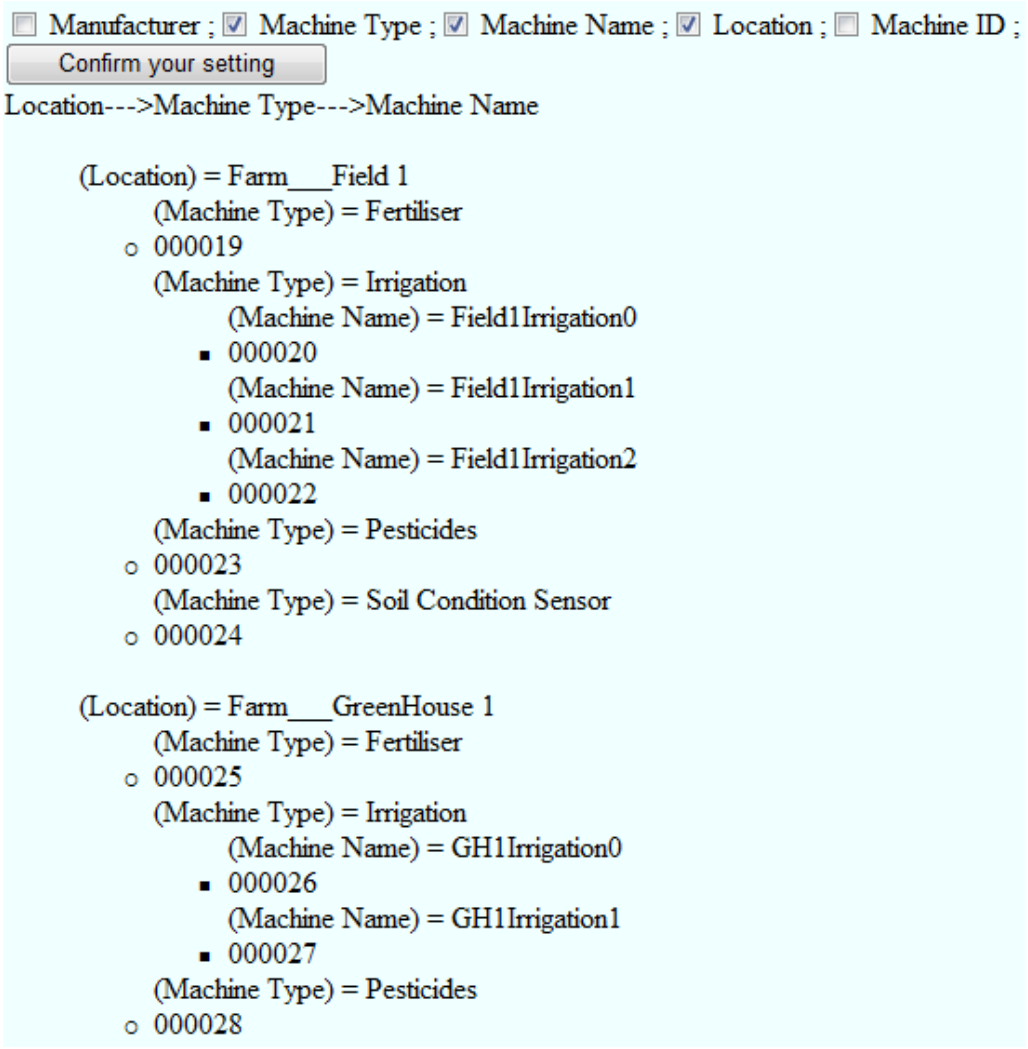


Figure 8. 43 Navigation Analysis Page Result For Farm Evaluation Scenario

Step 3: Generate CWUI with alarm information.

The CWUI page is generated as shown in Figure 8.44. The CWUI has three components: “Navigation Bar”, “Alarm Information” and the control panel for controlled machines. The “Navigation Bar” shows the navigation set by Frank in Figure 8.43. The control panel lists machines that Frank wants to control. “Alarm Information” shows the alarms set by Frank. Frank can access the target machines from “Navigation Bar” easily by just clicking a machine or the group of machines he needs. Also, by clicking the alarm items in “Alarm Information”, corresponding machine information can be shown in the control panel.

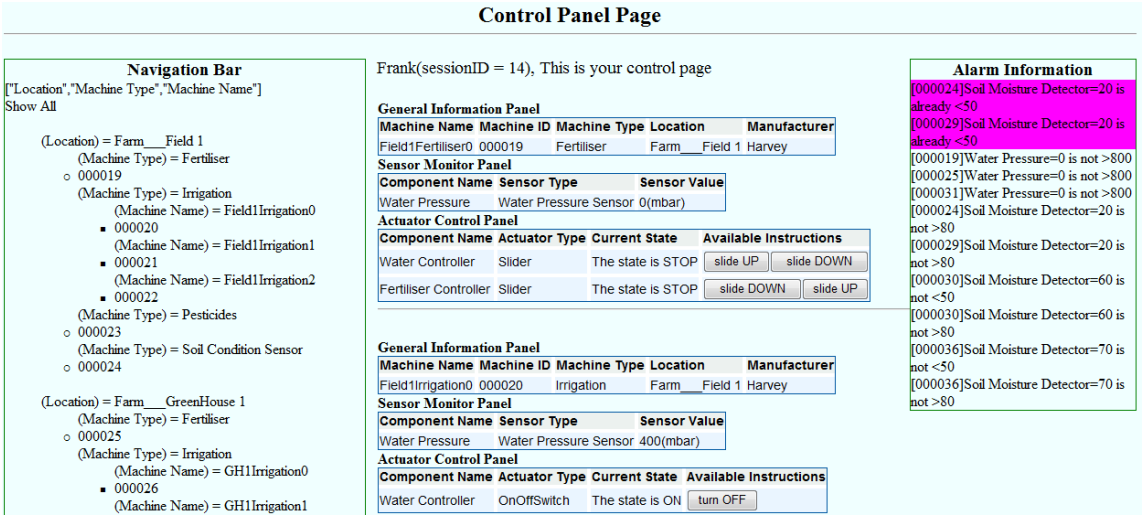


Figure 8. 44 CWUI with "Navigation Bar" And "Alarm Information" For Farm Devices

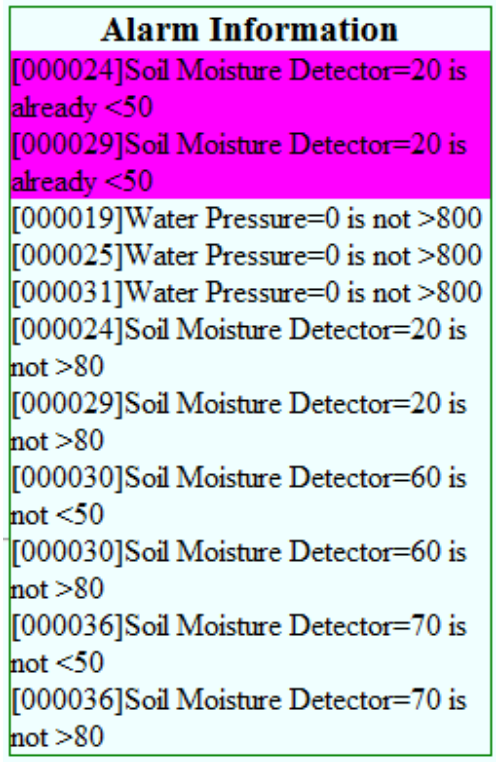


Figure 8. 45 "Alarm Information" Example For Farm Devices

An example of the "Alarm Information" display is given in Figure 8.45. In the figures, some items that have reached the alarm condition will trigger the alarm and be highlighted using different colours. The ones that are not highlighted are still within the safe range. The "Alarm Information" can be used as a special navigation tree. When an item is clicked, the

corresponding machine will be shown on the control panel. In the example, Frank can see two alarms have been triggered. After clicking on the item, Frank can find the information for the target machine. For example, in Figure 8.46, Frank finds it is the soil moisture value of the "Soil Condition Sensor" in "Farm___Field1" has reached 20 percent, which is lower than the alarm of 50 percent. The figure shows the content change when Frank clicks the first item "[000024] Soil Moisture Detector=20 is already<50" in "Alarm Information".

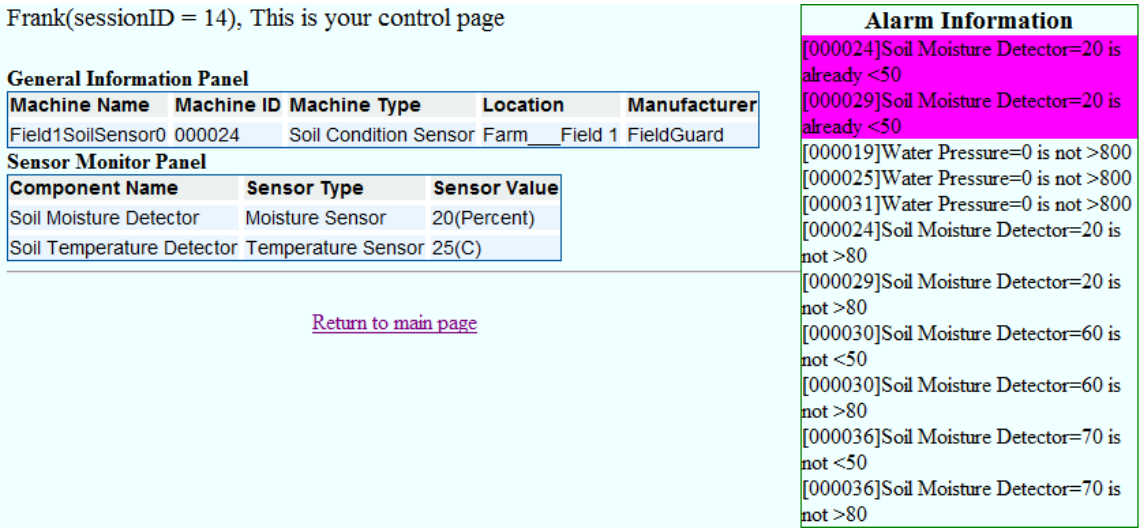


Figure 8. 46 Example Of "Alarm Information" As Navigation For Farm Devices

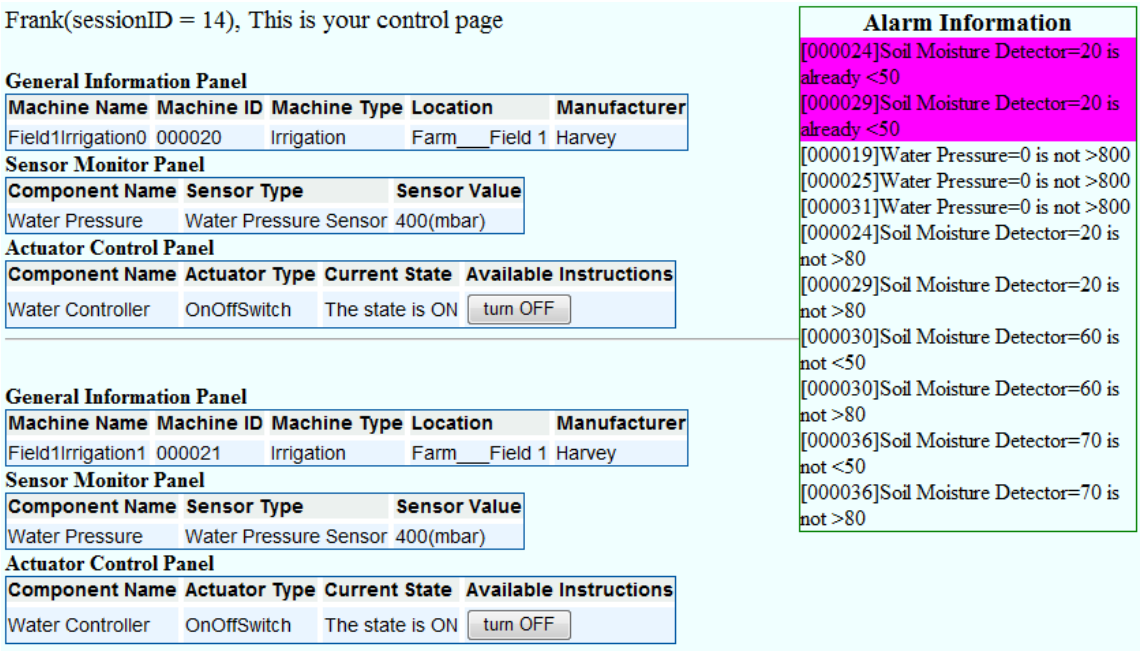


Figure 8. 47 "Alarm Information" Update Example For Farm Devices

Step 4: Test the adaptation of CWUI with alarm information.

The "Alarm Information" will be updated automatically according to the control process. Figure 8.47 shows the result of irrigations in "Farm___Field1" when Frank turned on all the "Irrigation" machines (000020, 000021, 000022) located in "Farm___Field1". The soil moisture value in "Farm___Field1" rises up gradually to 82 percent (Figure 8.48). The alarm, which specifies the value should be lower than 80 percent, is trigged. As shown in Figure 8.47, Frank now needs to click "turn OFF" in the control panel to stop irrigation.

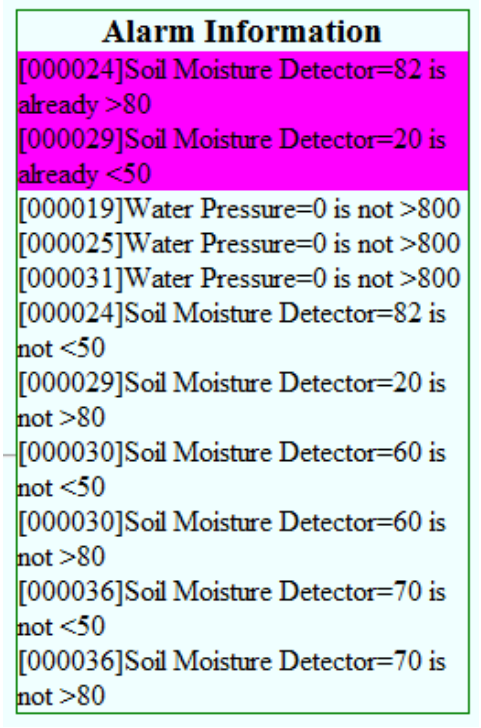


Figure 8. 48 "Alarm Information" Update Result For "Soil Moisture Detector"

This evaluation scenario demonstrates how users set alarms for the important changeable sensor values and how to use the alarm to access target machine. Alarm settings can be generated dynamically according to the user's requirement. Also, the content of the alarm will be shown on the CWUI and be dynamically updated according to the machine state. Moreover, when the alarm condition is triggered, the system will remind the user. This evaluation scenario actually shows some formatted session dependent information can be added in to the CWUI page. Using background handle function, this data can be merged in to automate generated CPL as described in Chapter 6. This method successfully solves the problem for users to navigate important information and make the CWUI more personalised. Also, the

evaluation scenario shows another example of dynamic navigation for multiple-machine CWUI.

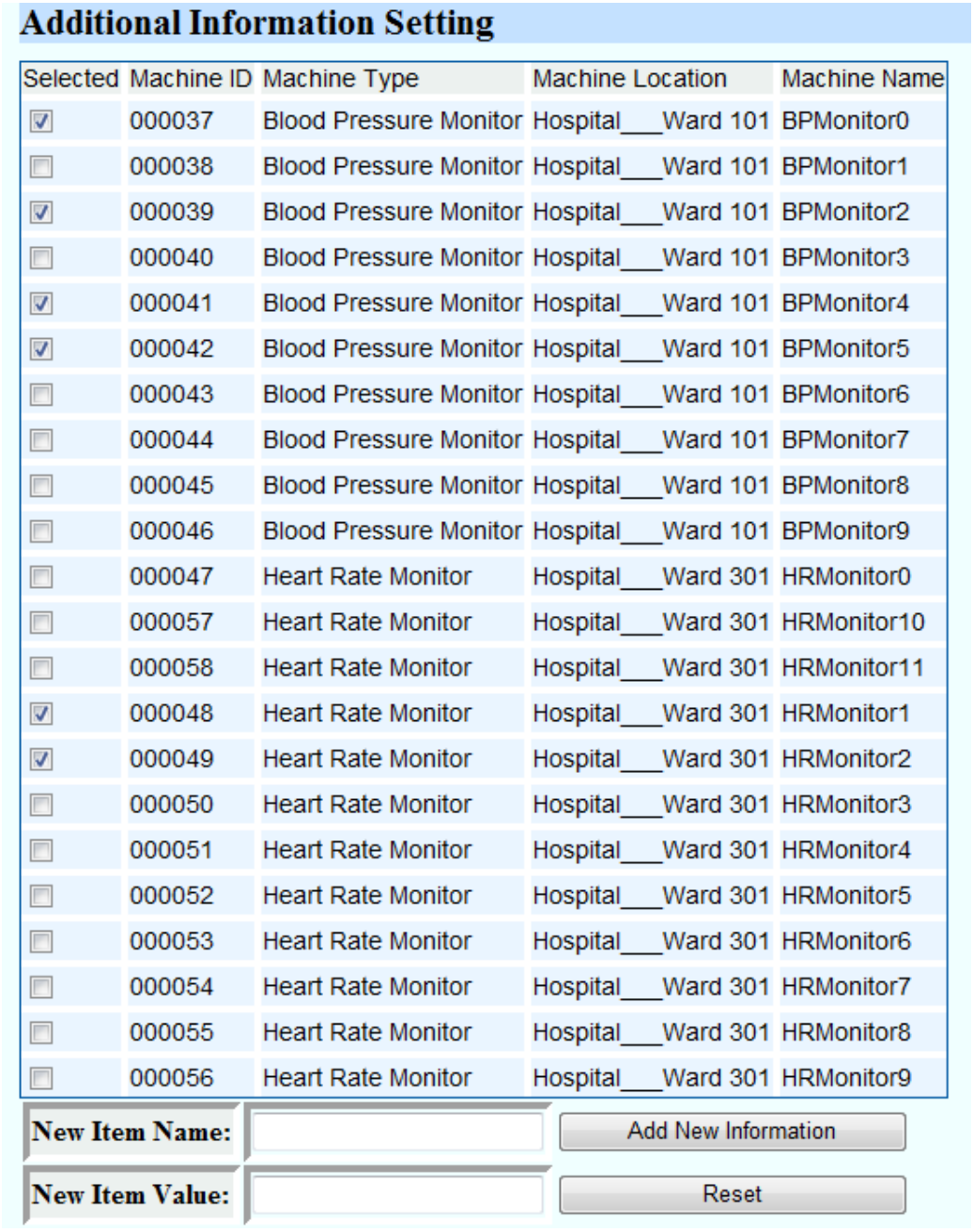
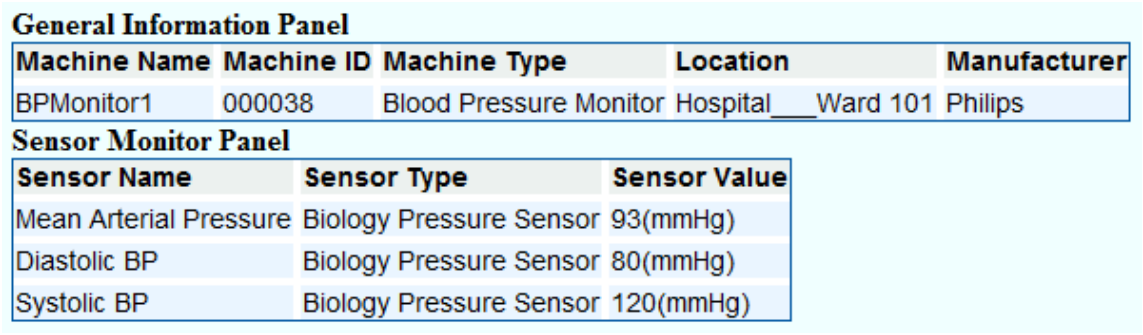


Figure 8. 49 Machines Needing To Be Monitored In The Hospital

8.8. Hospital equipment monitoring and control

Equipment in a hospital is difficult for a remote control system to organise because the facilities are usually attached to some dynamic information such as a certain patient, or a

doctor who is using the equipment. In this evaluation scenario, a nurse named Kate uses the system to monitor blood pressure for 10 different patients in one ward and the heartbeat for 12 patients in another ward at the same time. The nurse is required to report the information to corresponding doctors as soon as she finds any abnormal situation. It would be easier to group the machines using the name of the doctor who is in charge. Individual patient's name is needed for each machine so that it can be more convenient for the nurse to check which patient needs help from the machine information. However, doctors' names and patients' names are not values related to the control session but they are session dependent information that can help to index the machines. This evaluation scenario shows the nurse user the system to organise and access the controlled machine using the session dependent information.



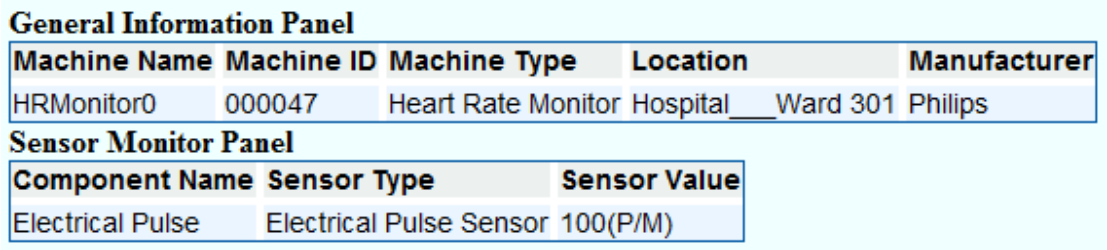
General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
BPMonitor1	000038	Blood Pressure Monitor	Hospital___Ward 101	Philips

Sensor Monitor Panel

Sensor Name	Sensor Type	Sensor Value
Mean Arterial Pressure	Biology Pressure Sensor	93(mmHg)
Diastolic BP	Biology Pressure Sensor	80(mmHg)
Systolic BP	Biology Pressure Sensor	120(mmHg)

Figure 8. 50 CWUI Example For The "Blood Pressure Monitor"



General Information Panel

Machine Name	Machine ID	Machine Type	Location	Manufacturer
HRMonitor0	000047	Heart Rate Monitor	Hospital___Ward 301	Philips

Sensor Monitor Panel

Component Name	Sensor Type	Sensor Value
Electrical Pulse	Electrical Pulse Sensor	100(P/M)

Figure 8. 51 CWUI Example For The "Heart Rate Monitor"

Step 1: Register the controlled machine information.

Figure 8.49 shows the machines that need to be monitored. In this figure, there are two types of machines ("Blood Pressure Monitor" and "Heart Rate Monitor"). These machines are located in two wards ("Hospital___Ward 101" and "Hospital___Ward 301") and all of them have the same manufacturer "Phillips". Using only this limited information, it will be hard for Kate to find the relevant machines easily. The examples of single CWUIs for the two types of

machines are shown in Figure 8.50 and Figure 8.51 correspondingly. Figure 8.50 shows the “Blood Pressure Monitor" that has three sensors: “Mean Arterial Pressure”, “Diastolic BP” and “Systolic BP”, while in Figure 8.51, the “Heart Rate Monitor” has only one sensor – “Electrical Pulse”.

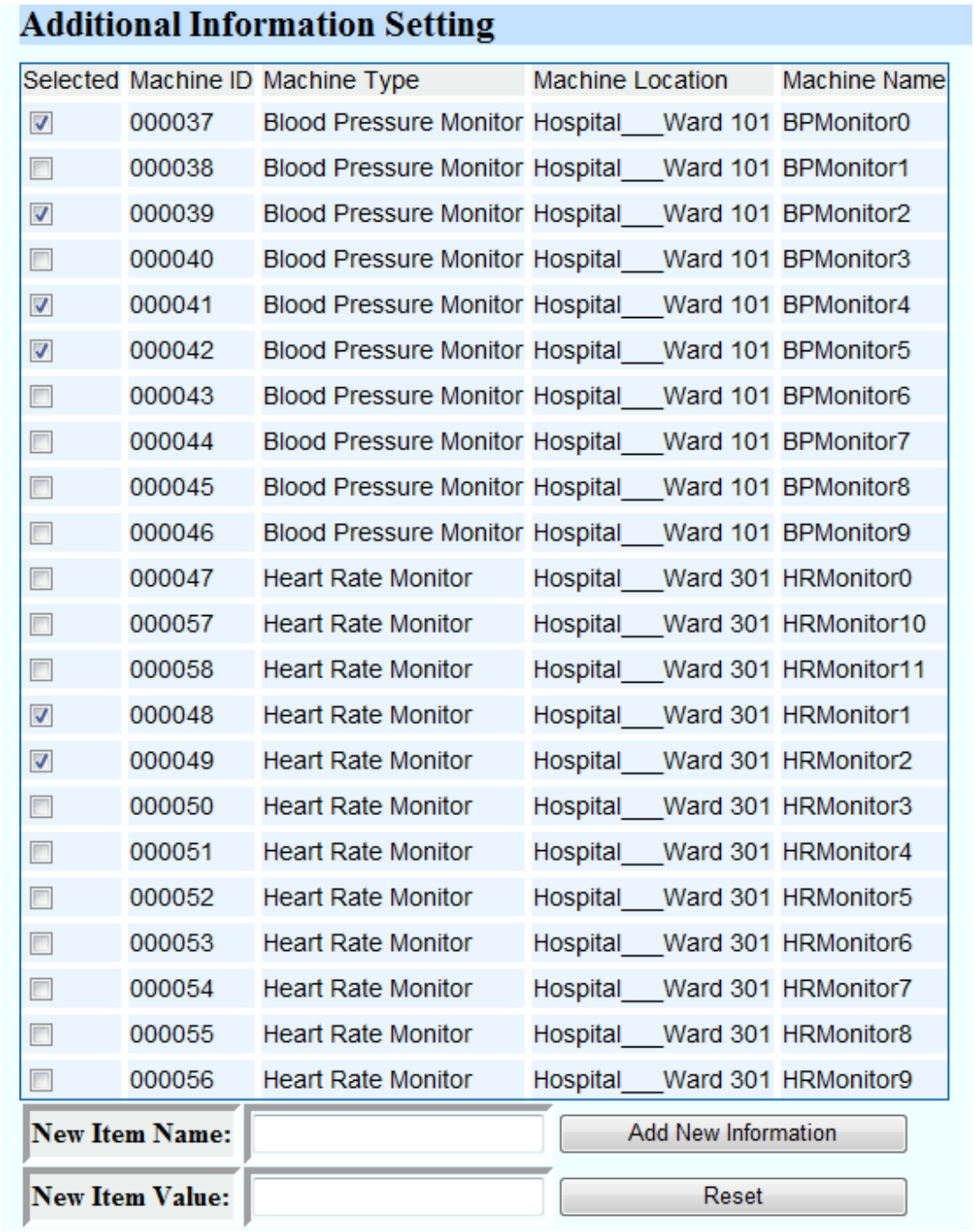


Figure 8. 52 “Additional Information Setting” Panel Example

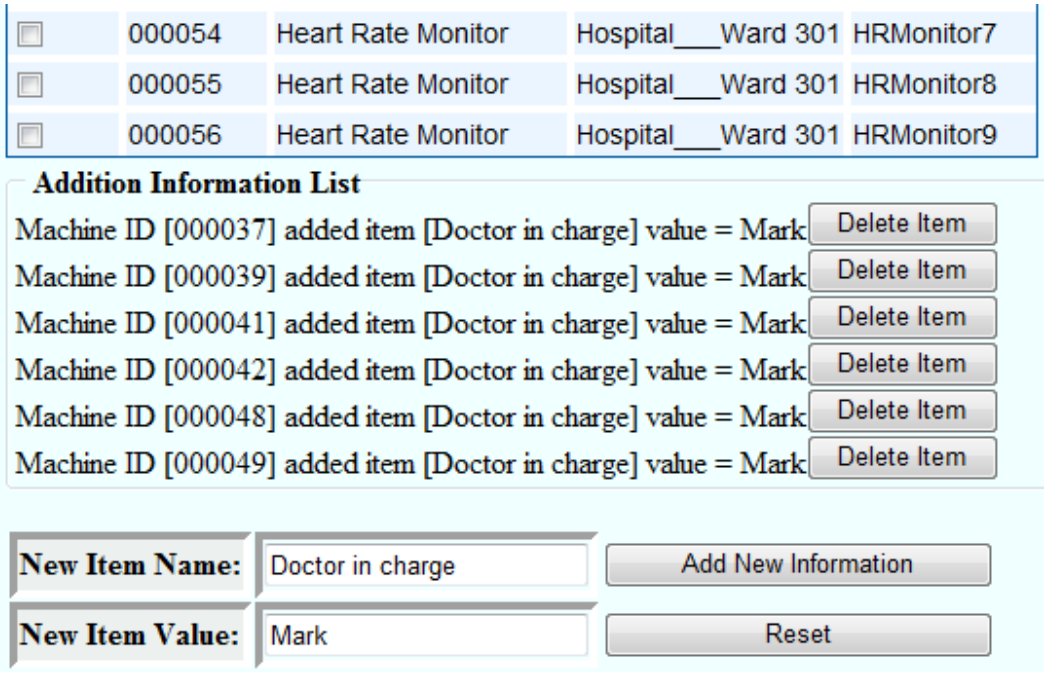


Figure 8. 53 User Set Session Dependent Information Example

Step 2: Set user defined session dependent information.

After selecting the machines, the user enters the "Session Dependent Information Setting Page" where the system allows the users to set session dependent information for each machine in the "Additional Information Setting" panel. Figure 8.52 shows Kate selected six machines: 000037, 000039, 000041, 000042, 000048 and 000049, and she tried to add a new feature called "Doctor in charge" with value "Mark" for these machines. This new information "Doctor in charge" records the name of the doctor who is in charge for the patient using the machine. This new information can be considered as a new Item which will be added into the information for each machine to help generate further navigation and CWUI. As shown in the Figure 8.52, "New Item Name" is the item name to identify the new feature, while "New Item Value" is the actual value of this Item. When Kate wants to add new items for the controlled machines, she just needs to select the machine and set the name and value of the item. By pressing "Add New Information", the information can be added in to the CPL logic and later be analysed by the system logic.

After pressing the button "Add New Information" in Figure 8.52, the result will be shown in Figure 8.53. This additional information set by Kate will be added dynamically between machine list and the new item setting form. By clicking the "Delete Item" button, the items

can be deleted. Finally “Addition Information List” for “Doctor in charge” item is shown in Figure 8.54. These machine states need to be reported to five doctors (“Mark”, “Lee”, “Luke”, and “Paul”). Using the same method, the list of the new items with a feature name “Name of the patient” are set and the result is shown in Figure 8.55. An item called “Name of the patient” is added for each machine, and the value of the item will be the name of the patient using the machine. When the machine is used by other patient or needs to be reported to another doctor, Kate can simply change the value of the item.

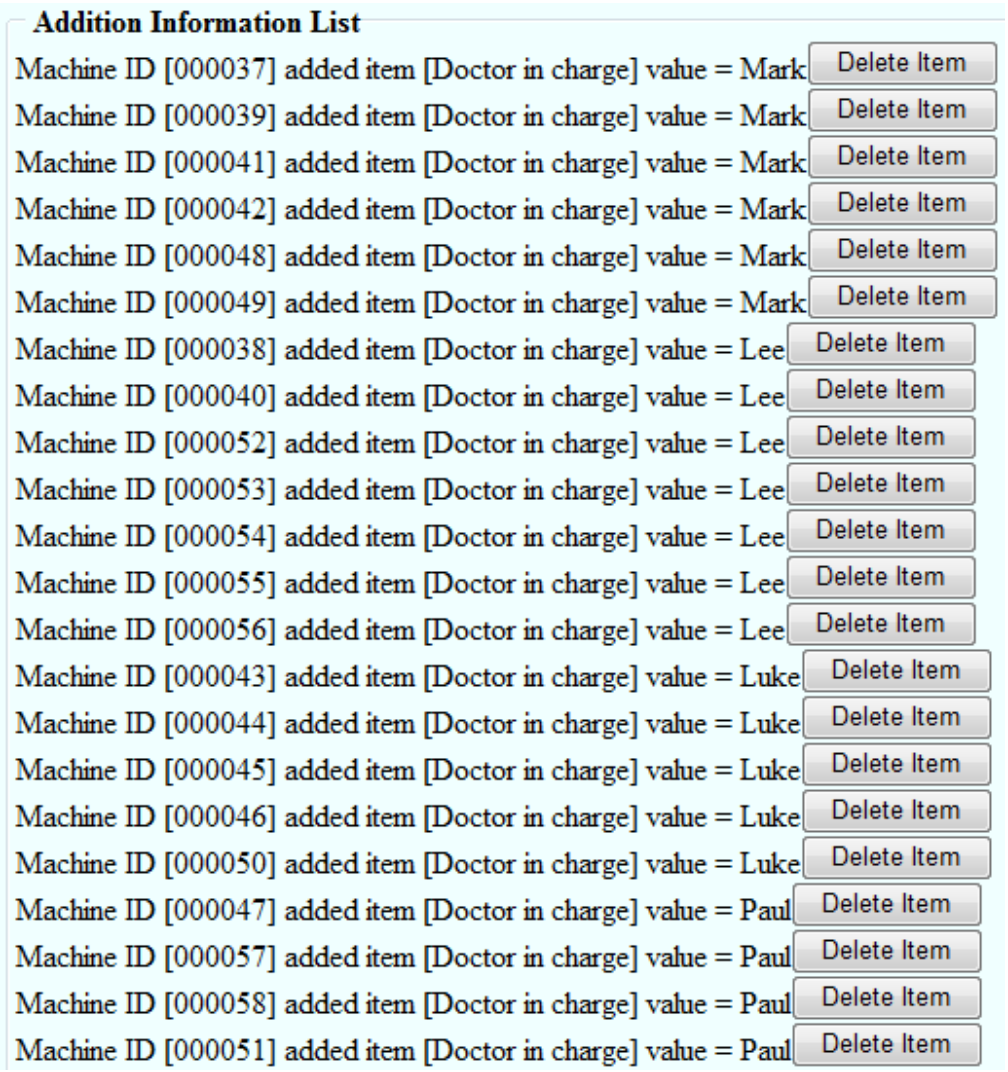


Figure 8. 54 “Addition Information List” For New Item “Doctor in charge”

The result of the user set session dependent information setting for “Doctor in charge” and “Name of the patient” is shown in Figure 8.54 and Figure 8.55 correspondingly.

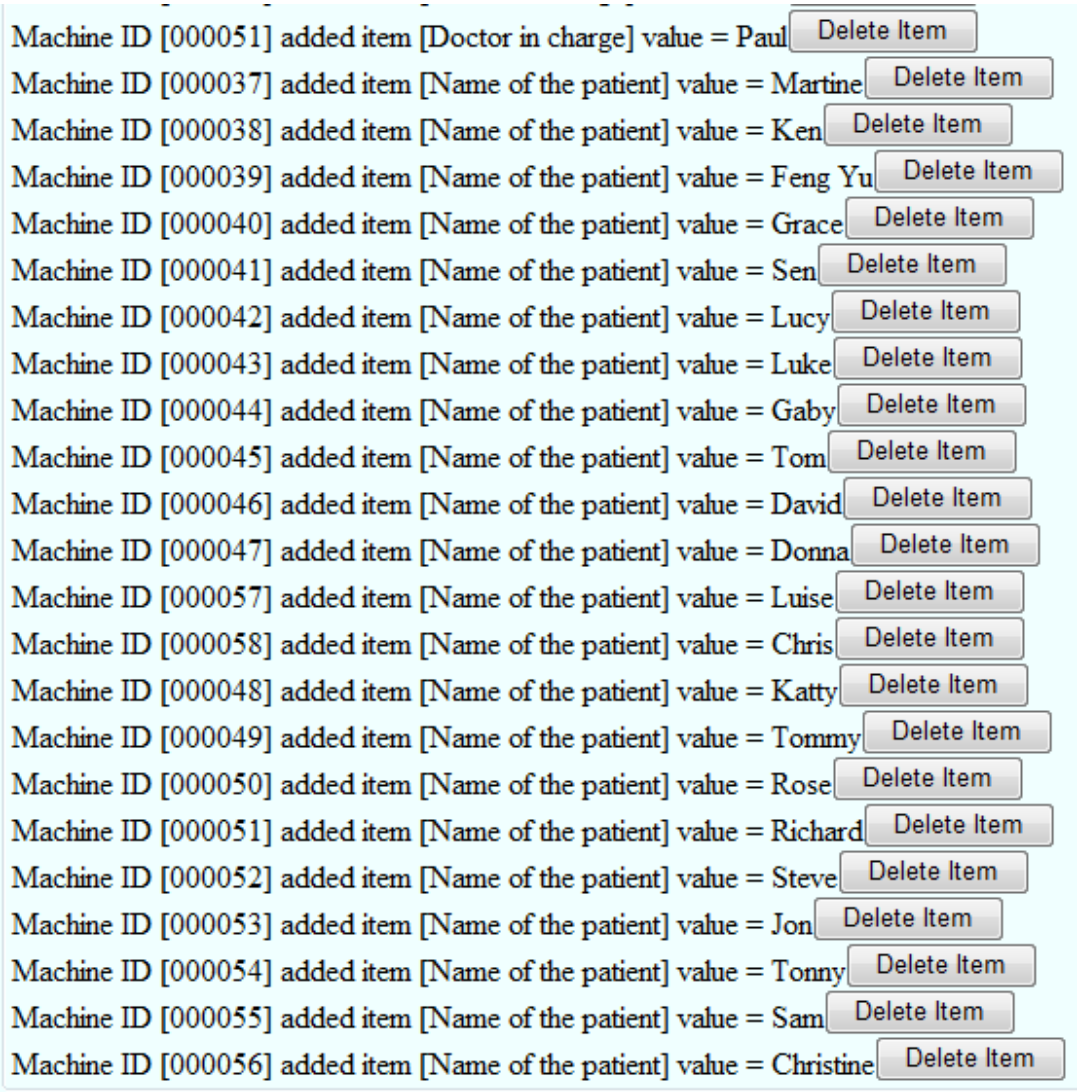


Figure 8. 55 “Addition Information List” For New Item “Name of the patient”

Step 3: Test navigation adaptation result and set navigation with user defined session dependent information.

The navigation analysis page is shown in Figure 8.56. As it is demonstrated in the figure, the session dependent information “Doctor in charge” and “Name of the patient” are added in to the keyword list. In this figure, Kate chose “Location” first to separate the group of machines in to “Hospital___Ward 101” and “Hospital___Ward 301”. Then she chose “Doctor in charge” and “Name of the patient” to group the patients in different rooms according to corresponding doctor and labelled with the doctor’s name.

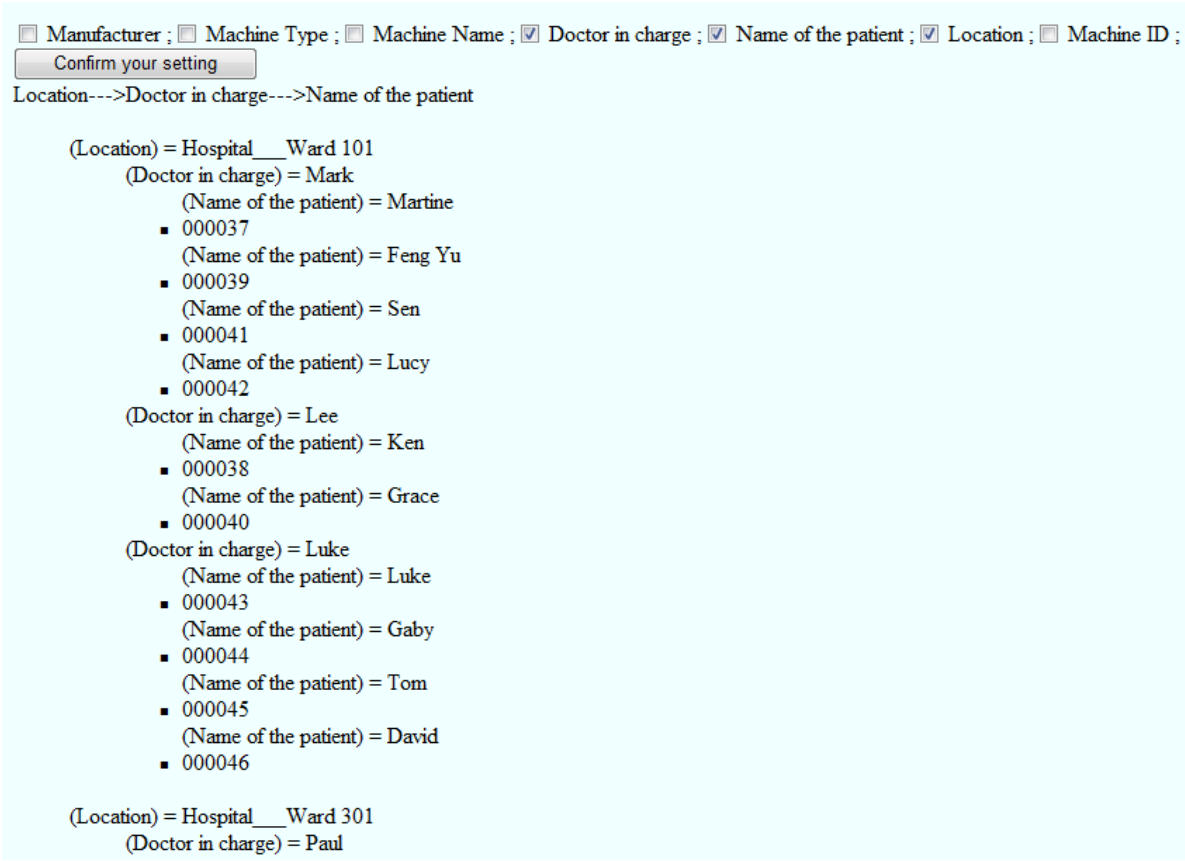


Figure 8. 56 Navigation Analysis Result For Hospital Equipments

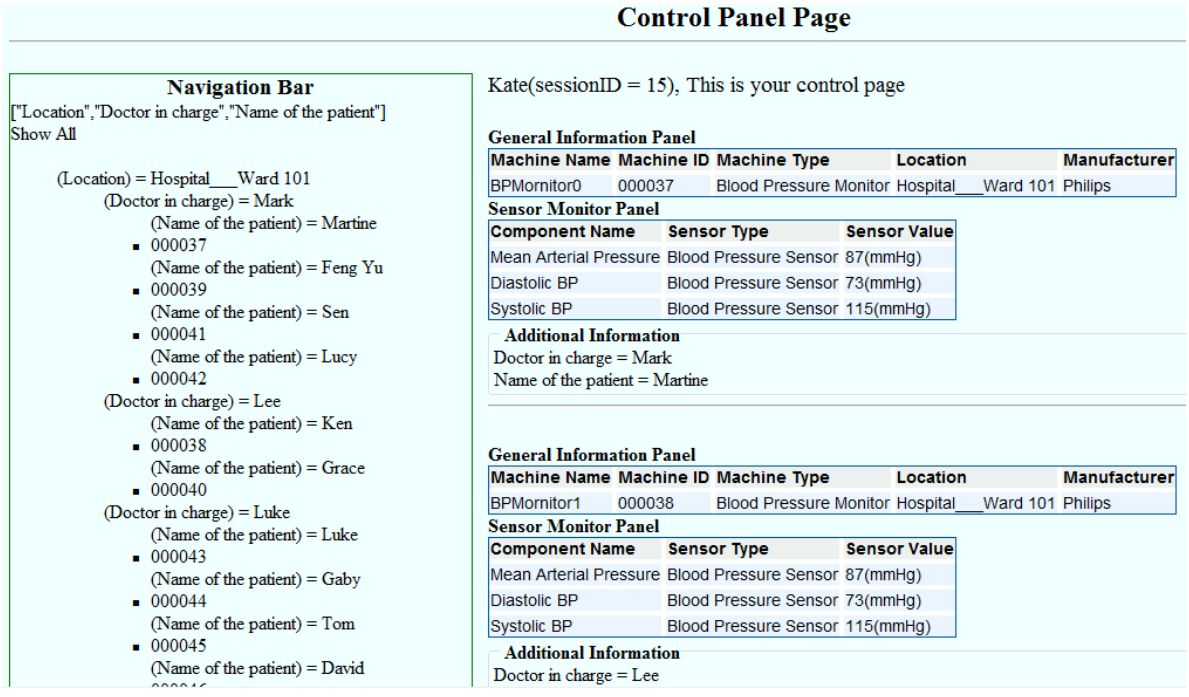


Figure 8. 57 “Control Panel Page” Example For Hospital Equipments

Step 4: Test CWUI adaptation with user defined session dependent information.

Figure 8.57 shows the final result of the "Control Panel Page". The figure shows that if the machine has session dependent information that not related directly to control an "Additional Information" panel will be added behind machine control information. For example, in this figure, the "Additional Information" panel of 000037 added by Kate shows the patient uses the machine is named "Martine" and the doctor responsible for that is "Mark". By clicking the item on the navigation tree in "Navigation Bar", required machine will be selected and shown on the main control page. Figure 8.58 shows when "(Name of the patient) = Ken" under "Hospital___Ward 101" is clicked, machine information for 000038 will be displayed on the main control panel.

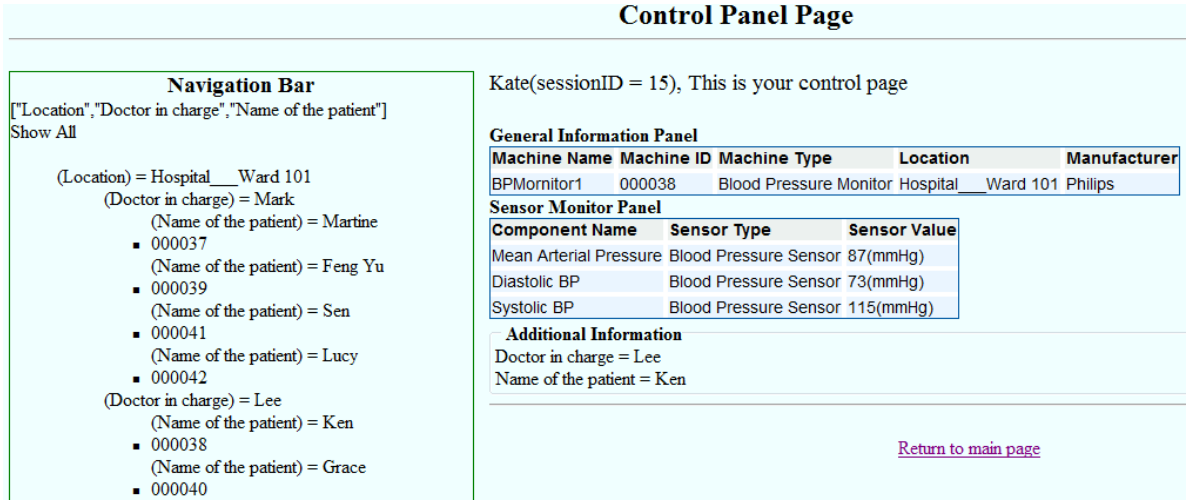


Figure 8. 58 "Control Panel Page" Adaption Example For Hospital Equipments

This evaluation scenario mainly demonstrates how the system help uses to set user defined session dependent information in to the CWUI. As shown in the evaluation scenario, the design not only allows user defined session dependent information to be added in to the control page dynamically but also involves the information in to CWUI navigation structure. Therefore, this evaluation scenario proofs the flexibility of the CWUI design, and this design benefit users by offering a better personalised WUI service.

8.9. Summary and Conclusions

In this chapter, five evaluation scenarios are tested and explained with the purpose of demonstrating different aspects of the design.

The hoister hardware control is given as the first evaluation scenario. It aims to test the connection of the hardware. The result shows CWUI can be generated dynamically and the command on the CWUI can be sent to the real hardware and can be executed correctly.

The second scenario aims at generating CWUI for different home facilities. The result shows the system can identify and automatically retrieve the components for different machines, organise machine functions and generate CWUI for different machines. It also shows that the system can group the control information in a CWUI in a consistent way, and link control function on the CWUI to the hardware functions.

The third scenario demonstrates multi-use access. The result shows that different users can control and monitor the same group of machine at the same time. Also, multiple-machines CWUI can be generated according to different users' needs and adapt to the control session. Moreover, flexible navigation can be set to help to index in the multiple-machine CWUI. The navigation keywords are generated automatically and the navigation tree content are updated automatically according to the control session.

The fourth scenario demonstrates control and monitoring of farm devices. The result shows that the design allows some formatted session dependent information such as alarm information be added in to the CWUI and be linked with the control process. The alarm information can be handled by background functions, and later represented in the control panel and functions as a special navigation to access some important information on CWUI.

The fifth scenario shows how the system helps a nurse to monitor patient states in a hospital. Navigations with user defined session dependent information is generated to help the nurse to access the target machines among large amount of information. Some session dependent information which allows users to define both the feature name and feature value can be added in to the control page dynamically. The information can be selected as keywords for navigation and attached in the CWUI dynamically.

These evaluation scenarios demonstrate that the system achieves the research goals successfully:

First of all, it provided a proof of concept for the architecture. It shows the system achieves the "many-to-many" architecture, which allows multiple-access for both users and machines. In all of the evaluation scenarios, machine model for different machines can be automatically retrieved according to the ontology based machine knowledge. Moreover, these evaluation scenarios shows services such as the IWUI service can be developed on top of the basic services offered by WSRCSS. With the support of basic services, developing remote control services will be much easier. The architecture is flexible so that it can be extended easily.

The evaluation also shows the feasibility of IWUI design. The scenarios demonstrate that different CWUI can be generated automatically according to different users, different machines, and different control processes. Machine information is groups in a consistent way. The commands on the CWUI can be sent to the target machines and executed correctly.

Moreover, the evaluation scenarios show navigation with session dependent information can be generated to help users to index target machines in multiple-machine CWUI. Keywords can be analysed and automatically generated by the system. Some session dependent information such as alarm information can be obtained by the system and added to the control page dynamically. Also, some other session dependent information defined by users can be collected by the system and later selected as navigation keywords. Hence, the problem of changeable user requirements for CWUI can be addressed.

These evaluation scenarios are all designed according to typical scenarios in daily life. Therefore, the design has proven to be feasible and the goals of the research have been achieved. The method can be further applied in other cases in the real world to help users to achieve remote control over the internet.

CHAPTER 9. CONCLUSIONS AND FUTURE WORK

9.1. Conclusions

This research aims at investigating how a remote control service system may be provided over the internet, and generating dynamic web user interface for the system. The following conclusions can be draw from this work.

Conclusion 1: remote control over the internet is not new, however, current works still have limitations on architecture and automatically generation of web user interface.

The limitation of current remote control architecture is analysed in Chapter 3. The literature review analyses some typical work on the most flexible architecture (many-to-many architecture). But these work lack of standardised SOA mechanism to communicate and understand device level applications, lack of machine model description, machine structure and function knowledge support and control service development support.

The limitation of current work on WUI generation for remote control systems has been explained in Chapter 4. The research finds web user interface automate generation is a problem for multi-access remote control system. Also, there are seldom design can generate web user interface to meet all the dynamic requirements from different users, machines and control process of the remote control systems.

Conclusion 2: to overcome the problem of machine integration over the internet, web service can be used as a standard communication method while machine model based semantic data exchange can be used to make different machine functions understandable by the system.

The benefit of using SOA as well as web services are discussed in Chapter 2. Since web service can make web applications cross language, platform and independent from execution environment, it will benefit the system by reduce the limitation of machine application compared with other middleware.

A machine model and a semantic data exchange method are introduced in Chapter 5. The research finds a high level machine model can be used to describe the relationships for the actuators and sensors in the same machine, so that their relationships can be understood by the system. Also, a machine model based semantic data exchange method is explained in Chapter 5. This method helps to describe different machine components and functions using a machine model, and semantically represents the machine model information and processing the information in a semantic way that the system can understand. Therefore it is possible for the system to process further intelligent reasoning.

Conclusion 3: ontology-based knowledge can be applied to retrieve machine structure and functions.

Ontology is discussed in Chapter 2 as a way of describing both data value and the relationship between the data. In Chapter 5, ontology is used as description for machine, sensor and actuator knowledge, which helps the system to retrieve and understand machine structure and functions. After analysis according to the knowledge, different machine descriptions can be retrieved and generated automatically. Therefore, ontology helps the system to reason machine structure and function and represent them in a semantic way for further reasoning.

Conclusion 4: the separation of basic services can reduce the complexity of remote control service development.

Although remote control services are various from each other, but there are some basic services can be reused. A clear separation of these basic services from other services will increase the usability of existing resources, and the work for new service development can be reduced. Chapter 5 introduced four basic services that are separated from the high level remote control services to provide some commonly used functions for remote control process. The beginning of Chapter 6 also introduced how the IWUI service can reuse these basic services efficiently.

Conclusion 5: there are five factors that influence the usability of the CWUI. These important factors need to be addressed in the design.

In Chapter 4, there are explanations about the importance of usability of UI design. In Chapter 6, there are explanations about five factors that influence the WUI design for remote control

system: user, machine, process, design knowledge and WUI limitations as well as the reason they influence the design. Therefore, there are explanations that how the IWUI architecture combines these five factors in to CWUI automatically generation.

Conclusion 6: although different machines have different attributes, these attributes can be organised to be a data structure. When the data structure is created, these attributes, as well as their values, can be used to help generate a dynamic navigation to index the contents displayed on CWUI.

In Chapter 4, there are some introductions about WUI design models and some method for represent WUI in a semantic way. There is also description for the problems of dynamic navigation. In Chapter 6, the thesis further developed this idea on the representation for CWUI data structure. Using semantic representation, the content of web user interface can be categorised and represented in a way computer can understand. In this way, the content of CWUI data can be dynamically organised and linked with a dynamic navigation that is generated according to a group of attributes selected from the data structure.

Conclusion 7: with semantic representation, it is possible for session dependent preference to be considered by the system and generate suitable user interface.

In Chapter 6, there are explanations about the session dependent data structure which allows new data to be added in flexible. The idea behind it is some new categories can be added to semantic data structure representation and later be processed by independent functions. In this way, the new data can be absorbed and mapped to the CWUI accordingly.

9.2. Contributions

The architecture proposed in this research offers a flexible and extensible remote control services system over the internet. An IWUI service is designed to overcome the difficulties of WUI design for such a multi-access remote control system. Navigation generation is designed to adapt to the CWUI to changeable user requirements. This thesis provides contributions in the field of IoT research. The following contributions are made based on the new ideas.

i. A better multi-access remote control architecture over the internet

This research proposed a multi-access remote control service according to many-to-many architecture, so that the system supports not only multiple-access for both users and machines. Compared with other designs, this system uses a semantic representation for machine model to specify and describe isolated atomic actuators and sensors. This architecture uses web service based APIs to support the machine model based semantic data exchange, which can be independent from development language, platform as well as executed environment.

ii. **Knowledge-based machine model automatic retrieve**

Using ontology, the architecture offers a knowledge-based method to support the reasoning about machine structure and functions, which is actually machine model. Compared with other research, this method increases the usability of machine knowledge and makes it possible for the system to reason about machine components, instructions. It also reduced the work for manually generation of machine description files and makes the architecture adaptable to the changes brought by new machines.

iii. Web service based control service development support

The research makes a clear separation between basic services and high level remote control services. Compared with other system, this method increases the reuse of the remote control abilities in the system and makes further control service development easier and efficient. Therefore, it benefits the system by offering better support for the development of new remote control services.

iv. Automatically generate adaptive web user interface

According to the review, currently there is seldom research on automatically generation of WUI for remote control systems. The thesis contributed this topic by proposing an IWUI service which combines all five factors that influence the WUI usability to solve the problem. This service is built upon the remote control service system and can generate CWUI automatically automatically according to different machines, users and control process.

v. Flexible information navigation structure

The research on index method for a multiple machine web user interface is new. In this thesis, a method to dynamically generate appropriate navigation for remote control system is proposed. Using this method, a user can specify how they would like the information to be structured and navigated. Their specification will then be used to generate the multiple machines WUI with the appropriate navigation structure, dynamic data updating and forwarding of control commands.

vi. Dynamic data structure for CWUI

Instead of just semantically represent the CWUI content, the thesis proposed a new idea to separate CWUI data structure from the content need to represent on the CWUI. The data structure represents the CWUI data in a semantic way that allows some session dependent data to be added in the structure flexibly. Therefore, it is possible for the system to organise the WUI data dynamically according to different requirements. In this way, more personalised CWUI and dynamic navigation can be generated for the users.

9.3. Limitations and future work

Remote control over the internet has many research issues covering a broad range of areas. Although the research in this thesis is finished, and the result shows the design achieves the research goals, there are still some limitations that need to be improved in the future.

Some work is needed to improve the functionalities of the system. For example, current architecture cannot detect machine hardware and services automatically. Manual preregistration is needed to initialise the machine information. When the machines are moved to other places the information need to be registered again. The current system does not capture machine control history, so it is not possible to maintain a control session if there is a break in the communication link. Also, in the machine server side, the hardware should have ability to communicate with the adapter. Therefore, usually, a hardware control tool is needed to read write and execute the control messages. Another limitation is current system only supports instructions with type “state”. More types of instructions including the instructions with parameters need to be extended in the future. The design only works well for machines with simple commands and only supports some simple data types. For example, it does not

work very well on the machines with complicated instruction set like “Switch the heater on for 3 hours at 75 degrees.

At the same time, there are some limitations in the design of IWUI. Firstly, the rules for generation of WUI is very basic as well as the UI component represented on the page. Secondly, the rules for web user interface generation as well as navigation selection are still very simple. More work is needed to generate more personalised interface.

According to the analysis above, future work needs to be done to improve the current design are listed below:

- Research is needed for automatic machine detection when they access the system in order to achieve automatic machine registration.
- Research is needed to improve the hardware connection method, and introduce a more efficient way to access hardware functions.
- Future research is needed to support more types of instructions, including the instructions with parameters.
- A more complicated rules need to be developed to support more personalised style for control user interface. There could be more choices for users to set page preferences according to their needs.

REFERENCE

1-800-Remotes, 2010, “History of the Television Remote Control” , http://remotes.com/store/main_history.html (Last updated at 18/04/2011).

Abrams, M. Phanourious, C. Batongbacal, A.L. Williams, S.M. and Shuster, J.E., 1999, “UIML: An Appliance-Independent Xml User Interface Language” , 8th international conference on World Wide Web.

Allen, B. Civit, A. Fellbaum, K. Kemppainen, E. Bitterman, N. Freitas, D. and Kristiansson, K., 2009, “Smart Home Environment” , http://www.tiresias.org/cost219ter/inclusive_future/inclusive_future_ch3.htm (Last updated at 18/04/2011).

Anupam, V. Freire, J. Kumar, B. and Lieuwen, D., 2000, “Automating Web Navigation With The WebVCR” , Computer Networks, vol. 33, pp. 503–517.

Apache Software Foundation, Aug. 2010, “Struts 2”, http://struts.apache.org/2.2.1/index.html (Last updated at 18/04/2011).

Baluja, S., 2006, “Browsing On Small Screens: Recasting Web-Page Segmentation Into An Efficient Machine Learning Framework” , Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland, May 23–26.

Barry, D. K. and Gannon, P. J., 2003, “Web Services And Service-Oriented Architecture: The Savvy Manager's Guide” , Morgan Kaufmann Publishers Inc, pp 17-26.

Beier, B. and Vaughan, M.W., Apr. 2003, “The Bull’s-Eye: A Framework For Web Application User Interface Design Guidelines” , Proceedings of the SIGCHI conference on Human factors in computing systems, Ft. Lauderdale, Florida, USA.

Booch, G., 1994, “Object-Oriented Analysis and Design” , 2nd edition, Addison-Wesley: Reading, MA.

Booth, D. Haas, H. McCabe, F. Newcomer, E. Champion, M. Ferris, C. and Orchard, D., 2004, “Web Services Architecture” , http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ (Last updated at 18/04/2011).

Brinck, T. Gergle, D. and Wood, S. D., 2002, “Usability For The Web: Designing Web Sites That Work Usability For The Web” , Academic Press, pp.2–3.

Bruninx, C. Raymaekers, C. Luyten, K. and Coninx, K., 2007, “Runtime Personalization Of Multi-Device User Interfaces: Enhanced Accessibility For Media Consumption In Heterogeneous Environments By User Interface Adaptation” , Second International Workshop on Semantic Media Adaptation and Personalization.

Burger, E. W. and Frieder, O., May 2006, “A Novel System For Remote Control Of Household Devices Using Digital IP Phones” , Consumer Electronics, IEEE Transactions, pp. 575 – 582.

Cai, T. Ju, S.G. Qian, Z.J. Guo, D.C. Cai, S.C. and Qian, H.H., May 2004, “Implement Of A Web-Based And Serial Port Agent Remote Control Demonstrating Systems” , In Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design, vol.2 pp. 567–572.

Castro, M. Sebastian, R. Yeves, F. Peire, J. Urrutia, J. and Quesada, J., Nov. 2002, “Well-Known Serial Buses For Distributed Control Of Backup Power Plants. RS-485 Versus Controller Area Network (Can) Solutions” , 28th Annual Conference of the IEEE Industrial Electronics Society (IECON02), vol. 3, 5-8 pp. 2381 -2386.

Cato, J., 2001, “User-Centered Web Design” , Addison-Wesley, pp.5–6.

Ceri, S. Fraternali, P. and Bonio, A., 2000, “Web Modeling Language (WebML): A Modeling Language For Designing Web Sites” , Computer Networks Journal, vol. 33, p137–156.

Chen, T.M. and Luo, R.C., Jul 1997, “Remote Supervisory Control Of An Autonomous Mobile Robot Via World Wide Web” , In Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '97), vol. 1, pp. 7-11.

Cho, K. and Kawamura, T., 2007, “BlogAlpha: Home Automation Robot Using Ontology In Home Environment” , Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications, pp. 197–203.

Chrysoulas, C. Koumoutsos, G. Denazis, S. Thramboulidis, K. and Koufopavlou, O., June 2007, “Dynamic Service Deployment Using An Ontology Based Description of Devices And Services” , Networking and Services, ICNS. Third International Conference, pp. 80.

Dai, C.W. Yang, S.H. and Knott, R.P., 2006, “Data Transfer Over The Internet For Real Time Applications” , International journal of Automation and computing, vol. 4, pp. 414–424.

Dai, C.W. Yang, S.H. and Tan, L., 2004, “An Approach For Controller Fault Detection” , in Proceeding of the 5th World Congress on Intelligent Control and Automation, pp. 1637–1641.

David, A.S. and Jeffrey, S., 1998, “Hand Held Remote Control Device With Trigger Button” , US Patent 5,724,106.

de, Souza L. M. S. Spiess, P. Guinard, D. Koehler, M. Karnouskos, S. and Savio, D., 2008, “SOCRADES: A Web Service Based Shop Floor Integration Infrastructure” , in Proc. of the Internet of Things Conference (IOT 2008), Springer.

Deutch, D. Milo, T. and Yam, T., 2009, “Goal-Oriented Web-Site Navigation For On-Line Shoppers” , Proceedings of the VLDB Endowment, vol. 2, pp.1642–1645.

Endrei, M. Ang, J. Arsanjani, A. Chua, S. Comte, P. Krogdahl, P. Luo, M. and Newling, T., 2004, “Patterns: Service-Oriented Architecture And Web Services Red Books” , http://www.raimcomputing.com/books/Patterns-SOAandWebServices.pdf (Last updated at 18/04/2011).

Fons, J. Valderas, P. Ruiz, M. Rojas, G. and Pastor, O., 2003, “OOWS: A Method To Develop Web Applications From Web-Oriented Conceptual Models” , International Workshop on Web Oriented Software Technology (IWWOST), pp.65–70.

Fowler, S. and Stanwick, V., 2004, “Web Application Design Handbook: Best Practices For Web-Based Software” , Elsevier Publishing, pp. 220–222.

Gajos, K. and Weld, D.S., 2004, “Supple: Automatically Generating User Interface, 9th International Conference On Intelligent User Interfaces” , Proceedings of the 9th international conference on Intelligent user interfaces, pp. 93–100.

Galitz, W.O., 2007, “The Essential Guide To User Interface Design: An Introduction To GUI Design Principles And Techniques” , Wiley Publishing, pp.24–58.

Gershenfeld, N. Krikorian, R. and Cohen, D., 2004, “The Internet Of Things” , Scientific American, vol. 291, no. 4, pp. 76–78.

Gisolfi, D., Jun. 2001, “Web Services Architect, Part 3: Is Web Services The Reincarnation of CORBA?” , http://www.ibm.com/developerworks/webservices/library/ws-arc3/ (Last updated at 18/04/2011).

Gokhale, A. Kumar, B. and Sahuguet, A., 2002, “Reinventing the Wheel? CORBA vs. Web Services” , In Proceedings of International World Wide Web Conference, Honolulu, Hawaii.

Grooters, B.A., 2006, “Method And Apparatus For Automatically Generating A Device User Interface” , United States Patent, US 7 111 242 B1.

Gruber, T. R., 1993, “A Translation Approach to Portable Ontology Specifications” , Knowledge Acquisition, vol. 5, Issue 2, pp. 199-220.

Gruber, T., 1992, “What is an Ontology?” , http://www-ksl.stanford.edu/kst/what-is-an-ontology.html (Last updated at 18/04/2011).

Gruber, T., 2009, “Ontology” , http://tomgruber.org/writing/ontology-definition-2007.htm, (Last updated at 18/04/2011).

Guinard, D. and Trifa, V., 2009, “Towards The Web Of Things: Web Mashups For Embedded Devices” , In Proceedings of WWW (International World Wide Web Conferences), Madrid, Spain.

Guo, X. and Chung, P.W.H., 2008, “The Architecture Of A Web Service-Based Remote Control Service” , 10th International Conference on Information Integration and Web-based Applications & Services.

Ha, Y. G. Sohn, J. C. and Cho, Y. J., 2005, “Service-Oriented Integration of Networked Robots with Ubiquitous Sensors and Devices Using the Semantic Web Services Technology” , Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference, pp. 3947– 3952.

Ha, Y. G. Sohn, J. C. Cho, Y.J. and Yoon, H., 2007, “A Robotic Service Framework Supporting Automated Integration Of Ubiquitous Sensors And Devices” , Elsevier Science Inc. New York, NY, USA, vol. 177, Issue 3, pp. 657–679.

Ha, Y. G. Sohn, J.C. and Cho, Y.J., June 2007, “ubiHome: An Infrastructure For Ubiquitous Home Network Services” , Consumer Electronics, 2007. ISCE 2007. IEEE International Symposium, pp. 1– 6.

Haartsen, J. Naghshineh, M. Inouye, J. Joeressen, O. J. and Warren Allen, Oct. 1998, “Bluetooth: vision, goals, and architecture” , Mobile Computing and Communications Review, vol. 2, Issue 4, pp. 38–45.

Haas, H. and Brown, A., 2004, “Web Services Glossary” , http://www.w3.org/TR/ws-gloss/ (Last updated at 18/04/2011).

Haller, S. and Karnouskos, S., 2008, “The Internet Of Things In An Enterprise Context” , Future Internet Symposium (FIS 2008) Vienna, pp. 14-28.

Han, R. Perret, V. and Naghshineh, M., 2000, “Websplitter: A Unified Xml Framework For Multi-Device Collaborative Web Browsing” , Proceedings of the 2000 ACM conference on Computer supported cooperative work, Philadelphia, Pennsylvania, United States, pp.221–230.

Harth, A., 2009, “VisiNav: Visual Web Data Search And Navigation” , Proceedings of the 20th International Conference on Database and Expert Systems Applications.

He, H., 2003, “What Is Service-Oriented Architecture” , http://www.xml.com/pub/a/ws/2003/09/30/soa.html (Last updated at 18/04/2011).

Helal, A. Yang, H. and Bose, J. King and R., Apr. 2007, “Atlas-Architecture for Sensor Network Based Intelligent Environments” , ACM Transactions on Sensor Networks.

Helal, S. Mann, W. Zabadani, H. E. King, J. Kaddoura, Y. and Jansen, E., 2005, “The Gator Tech Smart House: A Programmable Pervasive Space” , IEEE Computer Society Press Los Alamitos, CA, USA, vol. 38, pp. 50 – 60.

Hung, M.H. Chen, K.Y. and Lin, S.S., 2004, “Development Of A Web-Services-Based Remote Monitoring And Control Architecture” , In Proceedings of ICRA '04, vol. 2, pp. 1444–1449, April 26-May 1.

IEEEGHN, Mar. 2007, “Milestones: Early Developments In Remote-Control, 1901” , http://www.ieeeghn.org/wiki/index.php/Milestones:Early_Developments_in_Remote-Control,_1901 (Last updated at 18/04/2011).

Ito, K. and Murai, R., Jun. 2007, “Abstraction And Compression Of Information Utilizing Real World For Controlling Remote Controlled Robot - Application To Snake-Like Robot” , 2007 IEEE International Conference on Networking, Sensing and Control, London, UK, pp. 674 – 679.

Jammes, F. Mensch, A. and Smit, H., 2005, “Service-oriented Device Communications Using The Devices Profile For Web Services” , Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, pp.1– 8.

Jammes, F. and Smit, H., pp. 62–70. Feb. 2005, “Service-oriented Paradigms In Industrial Automation” , IEEE Transactions on Industrial Informatics, vol. 1, no. 1.

Joo, I. Park, J. and Paik, E., 2007, “Developing Ontolo Or Intelligent Home Service Framework” , Consumer Electronics, ISCE 2007. IEEE International Symposium, pp. 1–6.

JSON, 2011, “Introducing JSON” , http://www.json.org/ (Last updated at 18/04/2011).

Kappel, G. Pröll, B. Reich, S. and Retschitzegger, W., 2006, “Web Engineering: The Discipline Of Systematic Development Of Web Applications” , Wiley Publishing, pp.85–110.

Karnouskos, S. Baecker, O. Souza, L. M. S. D. and Spiess, P., 2007, “Integration Of SOA-Ready Networked Embedded Devices In Enterprise Systems Via A Cross-Layered Web Service Infrastructure” , in Proceedings of the 11th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 293–300.

Kim, K.S. Park, C. Seo, K.S. Chung, I.Y. and Lee, J., Feb. 2007, “ZigBee And The UPnP Expansion For Home Network Electrical Appliance Control on the internet” , Advanced Communication Technology, The 9th International Conference, pp. 1857 – 1860.

Krasner, G.E. and Pope, S.T., Aug./Sep. 1988, “A Cookbook For Using The Model-View Controller User Interface Paradigm In Smalltalk-80” , Journal of Object-Oriented Programming, vol.1, n.3, pp. 26–49.

Krikorian, R. and Gershenfeld, N., 2004, “Internet 0 – Inter-Device Internetworking” , BT Technical Journal, vol. 22, no. 4 (October 2004), pp. 278–284.

Lee, T. B. Hendler, J. and Lassila, O., May 2001, “The Semantic Web” , Scientific America, pp. 34–43.

Li, Z. and Nakagawa, H., 2002, “OPC (OLE for Process Control) Specification And Its Developments” , Proceedings of the 41st SICE Annual Conference, vol. 2, pp. 917 – 920.

Lieberman, H. and Espinosa, J., 2006, “A Goal-Oriented Interface To Consumer Electronics Using Planning And Commonsense Reasoning” , Proceedings of the 11th international conference on Intelligent user interfaces, Sydney, Australia.

Liu, P.X. Meng, M.Q.H. and Yang, P.R. Liu and S.X., Oct. 2005, “An End-To-End Transmission Architecture For The Remote Control Of Robots Over IP Networks” , IEEE Trans. Mechatronics, vol. 10(5).

Ljungkrantz, O. Åkesson, K. Richardsson, J. and Andersson, K., Apr. 2007, “Implementing A Control System Framework For Automatic Generation Of Manufacturing Cell Controllers,” , presented at the IEEE Int. Conf. Robot. Autom., Rome, Italy.

Mahnke, W. Leitner, S.-H. and Damm, M., 2009, “OPC United Architecture” , Springer-Verlag Berlin Heidelberg.

Maier, M. Emery, D. and Hilliard, R., 2009, “Recommended Practice For Architectural Description Of Software-Intensive Systems” , http://www.iso-architecture.org/ieee-1471/ieee-1471-faq.html (Last updated at 18/04/2011).

Marti, P. Yepez, J. Velasco, M. Villa, R. and Fuertes, J.M., Dec. 2004, “Managing Quality-Of-Control In Network-Based Control Systems By Controller And Message Scheduling Co-Design” , IEEE Trans. Ind. Electron., vol. 51(6), pp. 1159–1167.

Mathes, M. Stoidner, C. Heinzl, S. and Freisleben, B., 2009, “SOAP4PLC: Web Services For Programmable Logic Controllers” , Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing table of contents, Washington DC, USA.

Namgoong, H. Sohn, J.C. Cho, Y.J. and Chung, Y.K., 2006, “An Adaptive User Interface In Smart Environment Exploiting Semantic Descriptions” , IEEE 10th International Symposium on Consumer Electronics.

Nichols, J. and Myers, B. A., 2009, “Creating A Lightweight User Interface Description Language: An Overview And Analysis Of The Personal Universal Controller Project” , ACM Transactions on Computer-Human Interaction (TOCHI) archive, vol. 16, Issue 4.

Nichols, J. Myers, B. A. Higgins, M. Hughes, J. Harris, T. K. Rosenfeld, R. and Litwack, K., Apr. 2003, “Personal Universal Controllers: Controlling Complex Appliances With Guis And Speech” , CHI '03 extended abstracts on Human factors in computing systems, Ft. Lauderdale, Florida, USA.

Nichols, J. Myers, B. A. Higgins, M. Hughes, J. Harris, T. K. Rosenfeld, R. and Pignol, M., 2002, “Generating Remote Control Interfaces For Complex Appliances” , Proceedings of the 15th annual ACM symposium on User interface software and technology, Paris, France, pp.161–170, October 27–30.

Nichols, J. Rothrock, B. Chau, D. H. and Myers, B. A., 2006, “Huddle: Automatically Generating Interfaces For Systems Of Multiple Connected Appliances” , Proceedings of the 19th annual ACM symposium on User interface software and technology, Montreux, Switzerland, Oct. 15-18.

Noy, N. F. and McGuinness, D. L., 2001, “Ontology Development 101: A guide to creating your first Ontology” , SMI technical report SMI-2001-0880, Stanford University.

OASIS, May 2009, “Devices Profile for Web Services Version 1.1” , http://docs.oasis-open.org/ws-dd/dpws/1.1/cs-01/wsdd-dpws-1.1-spec-cs-01.html (Last updated at 18/04/2011).

Oltean, S. E. Abrudean, M. and Dulau, M., May 2006, “Remote Monitor And Control Application For Thermal Processes Using TCP/IP” , Automation, Quality and Testing, Robotics, 2006 IEEE International Conference, vol. 1, pp. 209–213.

OPC Foundation, 2011b, “What Is OPC?” , http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC (Last updated at 18/04/2011).

OPC Foundation, 2011a, “OPC Unified Architecture” , http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC (Last updated at 18/04/2011).

OSGi Alliance, 2011, “The OSGi Architecture” , http://www.osgi.org/About/WhatIsOSGi (Last updated at 18/04/2011).

Panne, M.V.D. and Fiume, E., 1993, “Sensor-Actuator Networks” , Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp.335-342, Anaheim, August 02-06.

Ray, K., 2010, “Introduction to Service Oriented Architecutre” , http://anengineersperspective.com/wp-content/uploads/2010/03/Introduction-to-SOA.pdf (Last updated at 18/04/2011).

Rossi, G. Pastor, O. Schwabe, D. and Olsina, L., 2008, “Web Engineering: Modelling And Implementing Web Applications” , Springer publishing, pp. 7–45.

Rowlett, R. and the University of North Carolina at Chapel Hill, 2008, “How Many? A Dictionary of Units of Measurement” , http://www.unc.edu/~rowlett/units/dictL.html (Last updated at 18/04/2011).

Rucker, J., 1997, “Siteseer: Personalized Navigation For The Web” , Communications of the ACM, vol. 40 n.3, pp. 73–75.

Schwabe, D. and Rossi, G., Aug. 1995, “The Object-Oriented Hypermedia Design Model” , Communications of the ACM, vol.38 n.8, pp.45–46.

Sheridan, T.B., 1992, “Telerobotics, Automation And Human Supervisory Control” , MIT Press, Cambridge, MA, pp. 1 – 3.

Sprott, D. and Wilkes, L., 2004, “Understanding Service-Oriented Architecture” , http://msdn.microsoft.com/en-us/library/aa480021.aspx (Last updated at 18/04/2011).

Stone, D. L. Stone, D. and Jarrett, C., 2005, “User interface design and evaluation” , Open University, Morgan Kaufmann Publishing, pp.6–7.

Suh, W., 2004, “Web Engineering Principles And Techniques” , pp.17.

Tillman, J., Nov. 2003, “User Interface Design for Web Applications” , http://www.digital-web.com/articles/user_interface_design_for_web_applications/ (Last updated at 18/04/2011).

Tvarožek, M. and Bieliková, M., 2007, “Personalized Faceted Navigation In The Semantic Web” , Web Engineering, pp. 511–515.

TVhistory, 2001, “Television History - The First 75 Years” , http://www.tvhistory.tv/Remote%20Controls.htm (Last updated at 18/04/2011).

University Of Florida, 2004, “Gator Tech Smart House” , http://www.icta.ufl.edu/gt.htm (Last updated at 18/04/2011).

UPnP Forum, 2011, http://www.upnp.org (Last updated at 18/04/2011).

Vazquez, J.I. Ruiz-de-Garibay, J. Eguiluz, X. Doamo, I. Renteria, S. and Ayerbe, A., Mar. 2010, “Communication Architectures And Experiences For Web-Connected Physical Smart Objects” , 8th IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 684 – 689.

W3C, 2011, “Extensible Markup Language (XML)” , http://www.w3.org/XML (Last updated at 18/04/2011).

Walsh, G. C. and Hong, Y., Feb. 2001, “Scheduling Of Networked Control Systems” , IEEE Control Syst., vol. 21, no. 1, pp. 57–65.

Wang, C. Teng, F. Wang, Y. and Ma, G., 2003, “Web-Based Remote Control Service System” , Industrial Electronics, 2003. ISIE '03. 2003 IEEE International Symposium, pp. 337–341 vol. 1.

Wang, Q. Liu, S. and Wang, Z., Oct. 2006, “A New The internet Architecture for Robot Remote Control” , Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference, pp. 4989 – 4993.

Xie, H. and Shen, W., May 2006, “Ontology As A Mechanism for Application Integration And Knowledge Sharing In Collaborative Design: A Review” , Computer Supported Cooperative Work in Design, CSCWD '06. 10th International Conference, pp. 1– 7.

Yang, S.H. Chen, X. Tan, L.S. and Yang, L., 2005, “Time Delay And Data Loss Compensation For The Internet-based Process Control Systems” , Transactions of the Institute of Measurement and Control, vol. 27, no. 3, pp. 103–118.

Yang, S.H. Dai, C.W. and Knott, R.P., 2007, “Remote Maintenance Of Control System Performance Over The Internet” , Control Engineering Practice, vol. 15, pp. 103–118.

Yang, S.H. Tan, L.S. and Chen, X., 2002, “Requirements Specification And Architecture Design For The Internet-Based Control Systems” , Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, pp. 75–80.

Yang, Y.C., Nov. 2002, “Autonomous And Universal Remote Control Scheme” , IECON 02 (Industrial Electronics Society, IEEE 2002 28th Annual Conference of the), pp. 2266 – 2271.

ZigBee Alliance, 2011, “ZigBee Specification Overview” , http://www.zigbee.org/Specifications/ZigBee/Overview.aspx (Last updated at 18/04/2011).

APPENDIX

A. **Top-down navigation generation code using javascript**

```
/**
 * json: input json code that need to be analyze
 * item_list: the list of keywords
 */
 function topdownmethod(json, item_list){

   //current search key
   var item = item_list[0];
   //declear the return code
   var codeall = "";
   var single = 0; //the number of single type item

   //list of the value already compared
   var itemvaluelist =  new Array();
   //loop input data
   for(var i=0; i<json.length; i++){
      var code = "";
      var newjson="";// new json code for next round
      //get json general_information
       var cginfor1 = json[i];
      var cginfor = cginfor1[item];
      //get machineID
      var cmaID = json[i]["Machine ID"];
      var idlist = new Array()//machine ID list that in current group
      idlist.push(cmaID);
      //printed if there is no key in this code other than just type as other types
      if(cginfor == null){
         var tcode  = "<ul value=\"" + cmaID + "\">Type("+ item + ") not exists for
" + cmaID + "</ul>";
         code = code + tcode;
         continue;
      }

      //check if the value ia already been compared
      var alreadydone =false;
      for(var j=0; j<itemvaluelist.length; j++){
         if(itemvaluelist[j] == cginfor){
            //already been compared then ignore and move to next
            alreadydone = true;
```

```
            //move the data into new input data
             break;
                    }

        }
        //if already been compared, then move to the next
        if(alreadydone == true){
           continue;
        }
        itemvaluelist.push(cginfor);

        for(var j=i+1; j<json.length; j++){
           //get caompaed item value
           var value2 = json[j];
           //if is same then collected
           var value3 = value2[item];
           if(cginfor == value3){
              newjson = newjson + "," + JSON.stringify(json[j]);
              idlist.push(json[j]["Machine ID"]);
           }

        }

        //if can not find a similar one then output
        if(newjson == ""){
           if(cginfor==""){
              cginfor = "NONE";
           }

           if(cginfor == null){
              cginfor = "NONE";

           }

           var icode = "<ul value=\"" + cmaID + "\">(" + item + ") = " + cginfor
              + "<li value=\"" + cmaID + "\">" + cmaID +"</li></ul>";
           code = code + icode;

           codeall = codeall + code;
           single++;
           continue;
        }
        //else if there are a group of data, then continue analysis
        if((item_list.length>1)&&(idlist.length>1 )){
           var njson = "[" + JSON.stringify(json[i]) + newjson +"]";
           var newjsonco = JSON.parse(njson);
```

```
                //get the id list for this node
                var nodeidlist = """ + idlist[0] + """;
                for(var x=1; x<idlist.length; x++){
                 nodeidlist =  nodeidlist + ","+ idlist[x] + """;

                }

                var newlist = new Array();
                for(var x=1 ; x<item_list.length; x++){
                   newlist.push(item_list[x]);

                }

                var newcode = topdownmethod(newjsonco, newlist);
                code = code + "<ul value=\"" + nodeidlist + "\">(" + item + ") = "
                          + cginfor
                          + newcode + "</ul>";

             }else{
                //if item_list.length = 1; then loop ended and print the result
                var leave = "<li value=\""+ idlist[0] +"\">" + idlist[0] +"</li>";
                var ids = """ + idlist[0] + """;
                for(var x=1; x<idlist.length; x++){
                   ids = ids + "," + idlist[x] + """;
                   leave = leave + "<li value=\""+ idlist[x] +"\">" + idlist[x] +"</li>";
                }

                code = "<ul value=\"" + ids + "\">(" + item + ") = " + cginfor
                code = code + leave;
                code = code + "</ul>";
             }

             codeall = codeall + code;
          }

          //return the navigation tree
          return codeall;

       }
```

B. Ontology used in the demo

B.1. Machine ontology (machine.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xml-body>
  <machine>Home Equipment
    <machine>Intelligent Light
      <Sensor>Light Intensity Detector
        <operateBy>Light Sensor</operateBy>
      </Sensor>
      <machine>Adjustable Intensity Light
        <Component>Light Intensity Controller
          <operateBy>Slider</operateBy>
        </Component>
        <Instruction>Bright Mode</Instruction>
        <Instruction>Evening Mode</Instruction>
      </machine>
      <Component>Switch
        <operateBy>OnOffSwitch</operateBy>
      </Component>
    </machine>
    <machine>Cleaning Devices
      <machine>Washing Machine
        <Sensor>Water Temperature
          <operateBy>Temperature Sensor</operateBy>
        </Sensor>
        <Sensor>Spinning speed
          <operateBy>Rotation Speed Sensor</operateBy>
        </Sensor>
        <Sensor>Water Pressure
          <operateBy>Water Pressure Sensor</operateBy>
        </Sensor>
        <Component>Water Inlet
          <operateBy>Valve</operateBy>
        </Component>
        <Component>Washing Powder Dispenser
          <operateBy>Valve</operateBy>
        </Component>
        <Component>Conditioner Dispenser
          <operateBy>Valve</operateBy>
        </Component>
        <Component>Water Heater
          <operateBy>Heater</operateBy>
        </Component>
```

```
        <Component>Drum Driver
          <operateBy>Motor</operateBy>
        </Component>
        <machine>hotPoint360
          <Component>Drum Driver 365
            <operateBy>Motor</operateBy>
          </Component>
          <Instruction>Wool Wash</Instruction>
          <Instruction>Mix Load</Instruction>
        </machine>
        <machine>national450
          <Instruction>Quick Wash</Instruction>
          <Instruction>Cotton Wash</Instruction>
          <Instruction>Normal</Instruction>
        </machine>
      </machine>
      <machine>Dish Washer
        <Sensor>Water Temperature
          <operateBy>Temperature Sensor</operateBy>
        </Sensor>
        <Sensor>Spinning speed
          <operateBy>Rotation Speed Sensor</operateBy>
        </Sensor>
        <Sensor>Water Pressure
          <operateBy>Water Pressure Sensor</operateBy>
        </Sensor>
        <Component>Water Spray
          <operateBy>Motor</operateBy>
        </Component>
        <Component>Water Inlet
          <operateBy>Valve</operateBy>
        </Component>
        <Component>Washing Powder Dispenser
          <operateBy>Valve</operateBy>
        </Component>
        <Component>Water Heater
          <operateBy>Heater</operateBy>
        </Component>
        <machine>BOSCH Dishwasher123
          <Instruction>Quick Wash</Instruction>
          <Instruction>Deep Clean</Instruction>
          <Instruction>Normal</Instruction>
          <Instruction>Ecnomic</Instruction>
        </machine>
      </machine>
    </machine>
```

```
<machine>Heating System
  <Sensor>Heating Temperature
    <operateBy>Temperature Sensor</operateBy>
  </Sensor>
  <machine>Central Heating 999
    <Instruction>Warm</Instruction>
    <Instruction>Hot</Instruction>
    <Instruction>Normal</Instruction>
    <Component>Switch
      <operateBy>OnOffSwitch</operateBy>
    </Component>
    <Component>Temperature Controller
      <operateBy>Slider</operateBy>
    </Component>
  </machine>
</machine>
<machine>Room Lock
  <machine>Window Lock
    <Instruction>Wide Open</Instruction>
    <Instruction>Slightly Open</Instruction>
    <Component>Window Lock
      <operateBy>OpenCloseSwitch</operateBy>
    </Component>
    <Component>Hinge
      <operateBy>Slider</operateBy>
    </Component>
  </machine>
</machine>
<machine>Hoister
  <Component>Vertical Hoister
    <operateBy>Motor</operateBy>
  </Component>
  <Component>Horizontal Hoister
    <operateBy>Motor</operateBy>
  </Component>
  <Component>High Speed Controller
    <operateBy>OnOffSwitch</operateBy>
  </Component>
</machine>
</machine>
<machine>Hospital Equipment
  <machine>Blood Pressure Monitor
    <Sensor>Mean Arterial Pressure
      <operateBy>Blood Pressure Sensor</operateBy>
    </Sensor>
    <Sensor>Diastolic BP
      <operateBy>Blood Pressure Sensor</operateBy>
    </Sensor>
```

```
        <Sensor>Systolic BP
          <operateBy>Blood Pressure Sensor</operateBy>
        </Sensor>
    </machine>
    <machine>Heart Rate Monitor
      <Sensor>Electrical Pulse
        <operateBy>Electrical Pulse Sensor</operateBy>
      </Sensor>

    </machine>
    <machine>Ventilator
      <Sensor>Peak Inspiratory Pressure
        <operateBy>Blood Pressure Sensor</operateBy>
      </Sensor>
    </machine>
    <machine>Body Temperature Monitor
      <Sensor>Body Temperature
        <operateBy>Body Thermometer</operateBy>
      </Sensor>
    </machine>
  </machine>
  <machine>Farm Equipment
    <machine>Irrigation
      <Component>Water Controller
        <operateBy>OnOffSwitch</operateBy>
      </Component>
      <Sensor>Water Pressure
        <operateBy>Water Pressure Sensor</operateBy>
      </Sensor>
    </machine>
    <machine>Soil Condition Sensor
      <Sensor>Soil Moisture Detector
        <operateBy>Moisture Sensor</operateBy>
      </Sensor>
      <Sensor>Soil Temperature Detector
        <operateBy>Temperature Sensor</operateBy>
      </Sensor>
    </machine>
    <machine>Pesticides
      <Component>Water Controller
        <operateBy>Slider</operateBy>
      </Component>
      <Component>Pesticides Controller
        <operateBy>Slider</operateBy>
      </Component>
      <Sensor>Water Pressure
        <operateBy>Water Pressure Sensor</operateBy>
      </Sensor>
```

```
      </machine>
       <machine>Fertiliser
          <Component>Water Controller
             <operateBy>Slider</operateBy>
          </Component>
          <Component>Fertiliser Controller
             <operateBy>Slider</operateBy>
          </Component>
          <Sensor>Water Pressure
             <operateBy>Water Pressure Sensor</operateBy>
          </Sensor>
       </machine>

   </machine>
   <machine>Experiment equipment
   <machine>Sensor Equipment
      <Sensor>Light Sensor
         <operateBy>Light Sensor</operateBy>
      </Sensor>
      <Sensor>Temperature Sensor
         <operateBy>Temperature Sensor</operateBy>
      </Sensor>
      <Component>LED1
         <operateBy>OnOffSwitch</operateBy>
      </Component>
      <Component>LED2
         <operateBy>OnOffSwitch</operateBy>
      </Component>
   </machine>
   </machine>
</xml-body>
```

B.2. Actuator ontology (actuator.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <Actuator>Motor
      <instruction attributes="direction" state="CLOCKWISE">turn
CLOCKWISE</instruction>
      <instruction attributes="direction" state="ANTICLOCKWISE">turn
ANTICLOCKWISE</instruction>
      <instruction attributes="direction" state="STOP">STOP</instruction>
   </Actuator>
   <Actuator>Valve
      <instruction attributes="state" state="ON">turn ON</instruction>
      <instruction attributes="state" state="OFF">turn OFF</instruction>
   </Actuator>
   <Actuator>Heater
      <instruction attributes="state" state="ON">turn ON</instruction>
      <instruction attributes="state" state="OFF">turn OFF</instruction>
   </Actuator>
   <Actuator>Slider
      <instruction attributes="state" state="UP">slide UP</instruction>
      <instruction attributes="state" state="DOWN">slide DOWN</instruction>
      <instruction attributes="state" state="STOP">STOP</instruction>
   </Actuator>
   <Actuator>OnOffSwitch
      <instruction attributes="state" state="ON">turn ON</instruction>
      <instruction attributes="state" state="OFF">turn OFF</instruction>
   </Actuator>
   <Actuator>OpenCloseSwitch
      <instruction attributes="state" state="OPEN">open</instruction>
      <instruction attributes="state" state="CLOSE">close</instruction>
   </Actuator>
</root>
```

B.3. Sensor ontology (sensor.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <Sensor>Light Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>W/m2</units>
      <range>
         <min>0</min>
         <max>1280</max>
      </range>
   </Sensor>
   <Sensor>Temperature Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>C</units>
      <range>
         <min>-5</min>
         <max>800</max>
      </range>
   </Sensor>
   <Sensor>Moisture Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>Percent</units>
      <range>
         <min>0</min>
         <max>100</max>
      </range>
   </Sensor>
   <Sensor>Rotation Speed Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>R/s</units>
      <range>
         <min>0</min>
         <max>800</max>
      </range>
   </Sensor>
   <Sensor>Water Pressure Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>mbar</units>
      <range>
         <min>0</min>
         <max>8000</max>
```

```
      </range>
   </Sensor>
   <Sensor>Blood Pressure Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>mmHg</units>
      <range>
         <min>0</min>
         <max>300</max>
      </range>
   </Sensor>
   <Sensor>Electrical Pulse Sensor
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>P/M</units>
      <range>
         <min>0</min>
         <max>500</max>
      </range>
   </Sensor>
   <Sensor>Body Thermometer
      <type>INT_NUMBER</type>
      <initial_value>0</initial_value>
      <units>C</units>
      <range>
         <min>0</min>
         <max>60</max>
      </range>
   </Sensor>
</root>
```

C. Machine client project implementation

C.1. MachineAdapterApp.Java

```
package lbr.rcss.client.adapter;

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import lbr.rcss.client.hardware.HardwareController;
import lbr.rcss.client.hardware.MachineSimulation;
import lbr.rcss.client.adminapi.AdapterControlHandler;
import lbr.rcss.client.wmaapi.ServerWSHandler;
import lbr.rcss.client.common.DOMTools;
import lbr.rcss.client.common.FileTools;
import lbr.rcss.client.common.mfd.MachineFunctionDescription;
import lbr.rcss.client.common.msd.MachineStateDescription;
import lbr.rcss.client.common.rcssClientConstant;
import org.dom4j.Document;

/**
 * This class is designed to achieve Machine Adapter Application
 *  for communicating with the hardware application
 * @author guoxi
 */
public class MachineAdapterApp extends Thread {
   // private int k;

   private Vector<String> machineEvent;

   public Vector<String> getMachineEvent() {
      return machineEvent;
   }

   public void setMachineEvent(Vector<String> machineEvent) {
      this.machineEvent = machineEvent;
   }
```

```
    /**
     * Delete executed control message from the local database
     * @param msgID: message ID
     * @return: if the function is executed successfully
     */
    public boolean deleteMsg(String msgID) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(MachineAdapterApp.class.getName()).log(Level.SEVERE,
null, ex);
        }
        java.sql.Connection conn;
        try {
            conn =
java.sql.DriverManager.getConnection(rcssClientConstant.CLIENT_DATABASE_NAM
E, "root", null);
            java.sql.Statement stmt = conn.createStatement();
            String sql = "delete from messagequeue where MsgID=" + msgID + "";
            stmt.executeUpdate(sql);
            stmt.close();
            conn.close();
            return true;
        } catch (SQLException ex) {
            Logger.getLogger(MachineAdapterApp.class.getName()).log(Level.SEVERE,
null, ex);
            return false;
        }
    }

    /**
     * This function gets message from local data base and execute the function
     * @return true if execution is successful
     *  return false if execution is failed
     */
    public boolean executeMsg() {
        //load messages from the data base
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(MachineAdapterApp.class.getName()).log(Level.SEVERE,
null, ex);
        }
        java.sql.Connection conn;
        try {
            //execute the message one by one
```

```
   conn =
java.sql.DriverManager.getConnection(rcssClientConstant.CLIENT_DATABASE_NAM
E, "root", null);
          java.sql.Statement stmt = conn.createStatement();
          java.sql.Statement stmt2 = conn.createStatement();
          Vector<String> changelist = new Vector<String>();
          String sql = "select * from messagequeue order by 'Time'";
          java.sql.ResultSet rs = stmt.executeQuery(sql);
          while (rs.next()) {
             String msg = rs.getString("Message");
             String msgID = rs.getString("MsgID");
             //read message
             if ((msg.equals("")) || (msg == null)) {
                System.out.println("The input message do not have content--from
MachineAdapterApp.executeMsg()");
             }
             MachineStateDescription msgRecord = new MachineStateDescription();
             Vector<String> funclist = msgRecord.GetExecuteFunctionList(msg);//put msg
into a machine state record structure also get the function list
             String filename =
AdapterControlHandler.getMFDbyMID(msgRecord.getMachineID());
             //get the function name and analysis
             HardwareController control = new HardwareController();
             for (int i = 0; i < funclist.size(); i++) {
                String funcName = funclist.elementAt(i);
                String hardwarefunc =
AdapterControlHandler.getHardwareFunction(filename, funcName);
                //begin to execute the functions
                System.out.println("hardware function invoked: " + hardwarefunc);
                //execute hardware function here (by invoke hardware function)
                if (control.MessageHandler(hardwarefunc)) {
                   //if successful update the machine state
                   MSWriter writer = new MSWriter();
                   writer.updateMachineState(msg);
                   //notify the change
                   if (this.getMachineEvent() == null) {
                      this.setMachineEvent(new Vector<String>());
                   }
                   this.getMachineEvent().addElement(msg);
                } else {
                   System.out.println("The function invoke is failed: " + msg);
                }
                //update mahcine sensor state for the machine
                //message has been executed
                this.deleteMsg(msgID);
             }
          }
          rs.close();
```

```
            stmt.close();
            stmt2.close();
            conn.close();
            return true;
        } catch (SQLException ex) {
            Logger.getLogger(MachineAdapterApp.class.getName()).log(Level.SEVERE,
null, ex);
            return false;
        }
    }

    /**
     * update corresponding state
     * @param filename
     * @return
     */
    public String updateMachineStateDescription(String MFDfileName, String
functionName) {
        String result = "";
        //get function name from the Machine Function Description
        String function = AdapterControlHandler.getHardwareFunction(MFDfileName,
functionName);

        HardwareController hardware = new HardwareController();
        result = hardware.getState(function);
        return result;
    }

    /**
     * Update the sensor information and write it in to the Machine State Description
     * @param filepath. Machine state Description path. the path should be ended by \\
     * @return
     */
    public boolean updateMachineState(String filepath) {
        //read Machine State Description
        //added all the machine state description to the list
        List<String> filenamelist = new ArrayList<String>();
        FileTools.getFileFullPathList(filepath, filenamelist);
        Iterator iter = filenamelist.listIterator();
        int i = 0;
        while (iter.hasNext()) {
            String text = (String) iter.next();
            //read MFD and get sensor information updated
            MachineStateDescription srecord = new MachineStateDescription();
            String filename = text;
            srecord.ReadRecord(filename);
            if (srecord.getMachineState() != null) {
                if (srecord.getMachineState().getSensorState() != null) {
```

```
                //begin to analysis and update sensor state
                int begin = 0;
                int end = text.indexOf("__");
                String machineID = text.substring(end + "__".length(), text.indexOf(".xml"));
                end = filepath.lastIndexOf("\\");
                String dir = filepath.substring(begin, end) + "\\";
                String MFDdir = dir.replace(rcssClientConstant.STATE_DIR_CODE,
rcssClientConstant.FUNCTION_DIR_CODE);
                //get Machine Function Description Name
                MachineFunctionDescription localRecord = new
MachineFunctionDescription();
                localRecord.GetLocalFileName(MFDdir, machineID);
                String path = MFDdir.substring(0, MFDdir.lastIndexOf("\\"));
                localRecord.setMachineID(machineID);
                String MFDfileName = localRecord.loadRecordByID(path);
                boolean changed = false;
                for (int j = 0; j < srecord.getMachineState().getSensorState().size(); j++) {
                   String compName =
srecord.getMachineState().getSensorState().elementAt(j).getSensorname();
                   String senName =
srecord.getMachineState().getSensorState().elementAt(j).getSensortype();
                   String stateType =
srecord.getMachineState().getSensorState().elementAt(j).getStatetype();
                   String funcName = compName.replaceAll(" ", "") + "_" +
senName.replaceAll(" ", "") + "_" + stateType.replaceAll(" ", "");
                   String senValue =
srecord.getMachineState().getSensorState().elementAt(j).getSensorvalue();
                   //update machine function
                   String stateValue = this.updateMachineStateDescription(MFDfileName,
funcName);
                   if (!senValue.equals(stateValue)) {

srecord.getMachineState().getSensorState().elementAt(j).setSensorvalue(stateValue);
                      changed = true;
                   }
                }
                //write it back to the Machine State Description
                if (changed) {
                   srecord.updateRecord(filename);
                }
             }
          }
          i++;
       }
       return true;
    }
```

```
   public static void main(String args[]) {
      MachineAdapterApp mApp = new MachineAdapterApp();
      int i = 0;
      int timeControl = 5;//when the loop reach this number, it will trigger simulate
program once
      String code = "";
      while (true) {
         //execute the command
         mApp.executeMsg();

         //send the event message to server
         if (mApp.getMachineEvent() != null) {
            for (int j = 0; j < mApp.getMachineEvent().size(); j++) {
               code = code + mApp.getMachineEvent().elementAt(j);
            }
            mApp.getMachineEvent().clear();
         }
         //trigger report event
         if ((code != null) && (!code.equals(""))) {
            code = "<root>" + code + "</root>";
            Document doc = DOMTools.string2Document(code);
            String sendcode = DOMTools.doc2String(doc);
            String resultcode = ServerWSHandler.handleClientMsg(sendcode);
            System.out.println("send event");
            code = "";
         }
         if (i == timeControl) {
            MachineSimulation simulation = new MachineSimulation();
            code = simulation.simulateAllMStateChange();
            i = 0;
            //trigger report event
            if ((code != null) && (!code.equals(""))) {
               code = "<root>" + code + "</root>";
               Document doc = DOMTools.string2Document(code);
               String sendcode = DOMTools.doc2String(doc);
               ServerWSHandler.handleClientMsg(sendcode);
               code = "";
            }
         }
         System.out.println("ended update");
         i++;
         try {
            sleep(500);
         } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
         }
      }
```

```
        public static void main(String args[]) {
        MachineAdapterApp mApp = new MachineAdapterApp();
        int i = 0;
        int timeControl = 5;//when the loop reach this number, it will trigger simulate
program once
        String code = "";
        while (true) {
           mApp.executeMsg();
           //send the event message to server
           if (mApp.getMachineEvent() != null) {
              for (int j = 0; j < mApp.getMachineEvent().size(); j++) {
                 code = code + mApp.getMachineEvent().elementAt(j);
              }
              mApp.getMachineEvent().clear();
           }
           //trigger report event
           if ((code != null) && (!code.equals(""))) {
              code = "<root>" + code + "</root>";
              Document doc = DOMTools.string2Document(code);
              String sendcode = DOMTools.doc2String(doc);
              String resultcode = ServerWSHandler.handleClientMsg(sendcode);
              System.out.println("send event");
              code = "";
           }
           if (i == timeControl) {
              MachineSimulation simulation = new MachineSimulation();
              code = simulation.simulateAllMStateChange();
              i = 0;
              //trigger report event
              if ((code != null) && (!code.equals(""))) {
                 code = "<root>" + code + "</root>";
                 Document doc = DOMTools.string2Document(code);
                 String sendcode = DOMTools.doc2String(doc);
                 ServerWSHandler.handleClientMsg(sendcode);
                 code = "";
              }
           }
           System.out.println("ended update");
           i++;
           try {
              sleep(500);
           } catch (InterruptedException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
           }
        }
     }
}
```

C.2. AdapterControlHandler.Java

```
 package lbr.rcss.client.adminapi;

import lbr.rcss.client.adapter.MSWriter;
import lbr.rcss.client.common.DOMTools;
import lbr.rcss.client.common.mfd.MachineFunctionDescription;
import lbr.rcss.client.common.rcssClientConstant;
import org.dom4j.Document;
import org.dom4j.Element;
import lbr.rcss.client.common.msd.MachineStateDescription;

/**
 * This class is used for handle message from server side.
 * The messages are mainly state change message requests.
 * It also implement API for server get machine client side information.
 * @author guoxi
 */
public class AdapterControlHandler {

   /**
    * Function that handle the message come from the server side
    * If it is included by <msg></msg> then it will be a command that change machine
states
    * This function will aims to change the Machine State Record directly.
    * @param msg : message String
    * @return: if the function is executed successfully
    */
   public boolean handlerMessage(String msg) {
      if ((msg == null) || (msg.equals(""))) {
         return false;
      }
      String code = "";
      code = msg;
      Document doc = DOMTools.string2Document(code);
      if (doc == null) {
         System.out.println("ERROR: input message is not correct xml -- from
AdapterControlHandler");
         return false;
      }
      Element iter = DOMTools.GetRoot(doc);
      String type = iter.getName().trim();
      if (type.equals("msg")) {
         MSWriter writer = new MSWriter();
         boolean re = writer.updateMachineState(msg);
         return re;
      }
```

```
      System.out.println("ERROR: Undefended message type -- from
AdapterControlHandler.handlerMessage()");
      return false;
   }

   /**
    * Forward message for message writer so that it will write the message to the message
queue data base
    * @param msg : message code that need to be write in
    * @return: if the function is executed successfully
    */
   public boolean WriteMsg(String msg) {
      if ((msg == null) || (msg.equals(""))) {
         return false;
      }
      MSWriter writer = new MSWriter();
      return writer.writeMessage(msg);
   }

   /**
    * This function will return machine list that available for control in local machine
    * @return: the list of machine information
    */
   public String getAvailableMachineList() {
      MLRManager manager = new MLRManager();
      String dir = rcssClientConstant.MSRECORD_DIR +
rcssClientConstant.FUNCTION_DIR_CODE;
      manager.setMLRdir(dir);
      return manager.getAvailableMachine();
   }

   /**
    * Get machine State Record file content
    * input machineID
    * after checking the file list. find the file and return the code
    * @param macID: machineID
    * @return: the state record for this machineID
    */
   public String readStateFile(String macID) {
      MSRManager manager = new MSRManager();
      String dir = rcssClientConstant.STATE_DESCRIPTION_DIR;
      manager.setMSRDir(dir);
      return manager.readStateFile(macID);
   }
```

```
    /**
     * Return the content of Machine State Record for a machine.
     * But remember to set the value of machineId and machineName
     * @param mName: Machine Name
     * @param mID: Machine ID
     * @return Machine State Description file content
     */
    public String getMachineRecordString(String mName, String mID) {
        MachineStateDescription record = new MachineStateDescription();
        record.setMachineName(mName);
        record.setMachineID(mID);
        String k = record.getMachineRecordString();
        return k;
    }

    /**
     * Analyse and return the location of current machine
     * @param macID: Machine ID
     * @param macType: Machine Type
     * @return: Location of current machine
     */
    public String analyzeLocation(String macID, String macType) {
        MachineStateDescription record = new MachineStateDescription();
        record.setMachineID(macID);
        record.setMachineType(macType);
        return (record.analyzeLocation());
    }

    /**
     * Get hardware function name from the Machine Function Description (MFD)
     * @param MFDfileName: MFD file name
     * @param functionName: file path + name of the MFD
     * @return: hardware function node <hardware></hadware>
     */
    public static String getHardwareFunction(String MFDfileName, String functionName)
{
        //get function name from the Machine Function Description
        MachineFunctionDescription lrecord = new MachineFunctionDescription();
        String function = lrecord.getHardwareFunction(functionName, MFDfileName);
        return function;
    }
```

```
    /**
     * Get Machine Function Description according to input mahcine ID
     * @param mID: machine ID
     * @return: Machine Function Description file content
     */
    public static String getMFDbyMID(String mID) {
        MachineFunctionDescription localRecord = new MachineFunctionDescription();

        localRecord.setMachineID(mID);
        String dir = rcssClientConstant.MSRECORD_DIR +
rcssClientConstant.FUNCTION_DIR_CODE;
        //System.out.append(dir);
        String filename = localRecord.loadRecordByID(dir);
        return filename;
    }

}
```

C.3. MachineWebService.java

```
package lbr.rcss.client.wmaapi;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import lbr.rcss.client.adapter.MSWriter;
import lbr.rcss.client.adminapi.AdapterControlHandler;

/**
 * This class is used to expose web service APIs to other projects or services
 * @author guoxi
 */
@WebService()
public class MachineWebService {

    /**
     * Return the content of Machine State Description for a machine.
     * Before using the value of machine ID and machine name need to be set
     * @param mName: Machine Name
     * @param mID: Machine ID
     * @return: The file content of Machine State Description
     */
    @WebMethod(operationName = "getMachineRecordString")
    public String getMachineRecordString(@WebParam(name = "mName") String
mName, @WebParam(name = "mID") String mID) {
        AdapterControlHandler handler = new AdapterControlHandler();
        String k = handler.getMachineRecordString(mName, mID);
        return k;
    }

    /**
     * Update Machine State Description according to input message
     * @param msg: command message
     * @return: if the function is successful
     */
    @WebMethod(operationName = "updateMSRecord")
    public boolean updateMSRecord(@WebParam(name = "msg") String msg) {
        MSWriter writer = new MSWriter();
        return writer.updateMachineState(msg);
        //java.lang.String code = msg;
        //return code;
    }
```

```
/**
 * Get available machine list on the local machine client server side
 * @return the available machine list
 */
@WebMethod(operationName = "getAvailableMachineList")
public String getAvailableMachineList() {

    AdapterControlHandler handler = new AdapterControlHandler();
    return handler.getAvailableMachineList();

}

/**
 * This function is used to get the location information from Machine State Description
 * @param macID: machine ID
 * @param macType: machine type
 * @return: the address in machine state record
 */
@WebMethod(operationName = "analyzeLocation")
public String analyzeLocation(@WebParam(name = "mID") String macID,
        @WebParam(name = "mType") String macType) {
    AdapterControlHandler handler = new AdapterControlHandler();
    return handler.analyzeLocation(macID, macType);
}

/**
 * Get machine Machine State Description file content.
 * This function will check the documents stores MSD.
 * If it find the file it will return the content of the file.
 * @param macID: machineID
 * @return: the required Machine State Description file content
 */
@WebMethod(operationName = "readStateFile")
public String readStateFile(@WebParam(name = "mID") String macID) {
    AdapterControlHandler handler = new AdapterControlHandler();

    return handler.readStateFile(macID);
}
```

```
    /**
     * Function that handle the message come from the service server side.
     * If it is included by node <msg></msg> then it will be a command that change
machine states
     * @param msg: message String
     * @return: the result of the function
     */
    @WebMethod(operationName = "handlerMessage")
    public boolean handlerMessage(@WebParam(name = "message") String msg) {
        if ((msg == null) || (msg.equals(""))) {
            return false;
        }
        AdapterControlHandler handler = new AdapterControlHandler();
        return handler.WriteMsg(msg);
    }
}
```

D. Service Server project implementation

D.1. MachineKnowledgeHandler.java

```
  package lbr.cs.rcss.rl;

import lbr.cs.rcss.common.model.ActuatorComp;
import lbr.cs.rcss.common.model.SensorComp;
import lbr.cs.rcss.common.model.Actuator;
import lbr.cs.rcss.common.model.Machine;
import org.dom4j.Attribute;
import org.dom4j.Document;
import org.dom4j.Element;
import lbr.cs.rcss.common.*;
import java.util.Iterator;
import java.util.Vector;
import java.util.List;
import lbr.cs.rcss.common.model.Command;

/*
 * This class is used to manage machine ontology resource.
 * It can analysis the machine functions according to machine, actuator and sensor
ontology.
 */
public class MachineKnowledgeHandler {

   private boolean findMachine;
   //ontology position and file name
   private String mOntologyFile = rcssConstants.ONTOLOGY_DIR + "machine.xml";

   /**
    * Get the attributes of a given machine xml element
    * @param element: xml element that need to be analysed
    */
   public void getAttributes(Element element) {
      for (Iterator i = element.attributeIterator(); i.hasNext();) {
         Attribute attribute = (Attribute) i.next();
         String attrName = attribute.getText();
         String attr = attribute.getName();
      }

   }
```

```
/**
 * This function is used to get a instruction of a certain actuator node
 * @param element: xml element that need to be analysed
 * @param aInstruction: the instruction that need to be searched
 */
public void getAttributes(Element element, Command aInstruction) {
   for (Iterator i = element.attributeIterator(); i.hasNext();) {
      Attribute attribute = (Attribute) i.next();
      String attrName = attribute.getText();
      String attr = attribute.getName();
      //System.out.println("PUT " + attr + " TOBE " + attrName);
      aInstruction.setName(element.getTextTrim());
      if (attr.equals("attributes")) {
         aInstruction.setAttribute(attrName);
      }
      if (attr.equals("state")) {
         aInstruction.setState(attrName);
      }
   }
}

/**
 * Get the attributes of a certain actuator,
 * This function only in charge of the function belong to the same element
 * @param element: xml element that need to be analysed
 */
public void getInstruction(Element element) {
   List elements = element.elements();
   String curMotor = new String();
   for (Iterator it = elements.iterator(); it.hasNext();) {
      Element elem = (Element) it.next();
      String nodeName = elem.getName();
      if (nodeName.equals("instruction")) {
         System.out.println("The instruction is " + elem.getTextTrim());
         getAttributes(elem);
      }
   }
}

/**
 * Get the instruction of a certain actuator
 * @param element: input as the element that needed to be analyzed
 * @param resultMotor: result actuator object
 */
public void getInstruction(Element element, Actuator resultMotor) {

   List elements = element.elements();
   String curMotor = new String();
```

```
        for (Iterator it = elements.iterator(); it.hasNext();) {
          Element elem = (Element) it.next();
          String nodeName = elem.getName();
          if (nodeName.equals("instruction")) {
             //System.out.println("The instruction is " + elem.getTextTrim());
             Command instru = new Command();
             getAttributes(elem, instru);
             resultMotor.insertInstruction(instru);
          }
        }
    }

    /**
     * Search the element in actuator ontology and find the actuator information needed
     * @param element: the element that needed to be analysed
     * @param motor: the motor that needed to be analysed
     */
    public void ActuatorTreeAnalysis(Element element, String motor) {
        List elements = element.elements();
        String curMotor = new String();
        for (Iterator it = elements.iterator(); it.hasNext();) {
          Element elem = (Element) it.next();
          String nodeName = elem.getName();
          if (nodeName.equals("Actuator")) {
             String motorName = elem.getTextTrim();
             if (motorName.equals(motor)) {
                getInstruction(elem);
                break;
             }
          }
          ActuatorTreeAnalysis(elem, motor);
        }
    }

    /**
     * Search the motor ontology and get the motor according to motor name
     * @param element: the element that needed to be analysed
     * @param motor: the actuator name
     * @param resultMotor: the result actuator object
     */
    public void ActuatorTreeAnalysis(Element element, String motor, Actuator
resultMotor) {
        List elements = element.elements();
        String curMotor = new String();
        for (Iterator it = elements.iterator(); it.hasNext();) {
          Element elem = (Element) it.next();
          String nodeName = elem.getName();
          if (nodeName.equals("Actuator")) {
```

```
            String motorName = elem.getTextTrim();
            if (motorName.equals(motor)) {
               getInstruction(elem, resultMotor);
               break;
            }
         }
         ActuatorTreeAnalysis(elem, motor);
      }
   }

   /**
    * Get the Actuator data according to its name
    * @param motor: actuator name
    * @param resultMotor: the result actuator object of the search
    */
   public void SearchActuator(String motor, Actuator resultMotor) {
      String filename = rcssConstants.ONTOLOGY_DIR +
rcssConstants.ACUTUATOR_ONTOLOGY;
      Document doc = DOMTools.loadXML(filename);
      Element titleElement = DOMTools.GetRoot(doc);
      resultMotor.setName(motor);
      ActuatorTreeAnalysis(titleElement, motor, resultMotor);
   }

   /**
    * Get the motor information for a certain actuator component
    * This function will return the motor which is needed by system
    * @param element: the element that needed to be analysed
    * @param motor: actuator list that need to be returned.
    * @return: if the function executed successfully
    */
   public boolean getMotor(Element element, Vector<Actuator> motor) {
      List elements = element.elements();
      String curMotor = new String();

      for (Iterator it = elements.iterator(); it.hasNext();) {
         Element elem = (Element) it.next();
         String nodeName = elem.getName();
         if (nodeName.equals("operateBy")) {
            Actuator tempMotor = new Actuator();
            tempMotor.setName(elem.getTextTrim());
            //get the motor detail
            SearchActuator(tempMotor.getName(), tempMotor);
            //added new motor
            motor.add(tempMotor);
            return true;
         }
      }
```

```
    return false;
}

/**
 * Get instruction list according to given xml element
 * @param macElem: machine element that need to be analysed
 * @param instruList: instruction list that need to be get
 * @return: if the function executed successfully
 */
public boolean getInstruction(Element macElem, Vector<Command> instruList) {
    List elements = macElem.elements();

    if (elements.iterator().hasNext() == false) {
        System.out.println("There is no Instruction");
        return false;
    }
    for (Iterator it = elements.iterator(); it.hasNext();) {
        Element elem = (Element) it.next();
        String nodeName = elem.getName();
        if (nodeName.equals("Instruction")) {
            //set component
            Command tempInstru = new Command();
            tempInstru.setName(elem.getTextTrim());
            instruList.add(tempInstru);
        }
    }
    return true;
}

/**
 * This function analyses the actuator component and function for a certain machine
 * @param macElem: machine element that need to be analysed
 * @param comp: component list that need to be get
 * @return: if the function executed successfully
 */
public boolean getComponent(Element macElem, Vector<ActuatorComp> comp) {
    List elements = macElem.elements();
    boolean hasComponent = false;
    if (elements.iterator().hasNext() == false) {
        System.out.println("There is no subclass(Component or subtype machines)");
        return false;
    }
    for (Iterator it = elements.iterator(); it.hasNext();) {
        Element elem = (Element) it.next();
        String nodeName = elem.getName();
        if (nodeName.equals("Component")) {
            //set component
            ActuatorComp tempComp = new ActuatorComp();
```

```
            tempComp.setName(elem.getTextTrim());
            //set the motor
            Vector<Actuator> temMotor = new Vector<Actuator>();
            if (getMotor(elem, temMotor)) {
               Actuator tempM = temMotor.elementAt(0);
               tempComp.setActuator(tempM);//get the first one of the motor
               //added new component into the list
               comp.add(tempComp);
            } else {
               System.out.println("There is no motor for the component : " +
elem.getTextTrim());
            }
         }
      }
      return true;
   }

   /**
    * Analyse the sensor data and put the data into a Sensor Component object
    * @param macElem: the node <SensorComp> and its contents
    * @param sens: the sensor list needed to be set
    * @return: if the function executed successfully
    */
   public boolean getSensor(Element macElem, Vector<SensorComp> sens) {
      List elements = macElem.elements();
      if (elements.iterator().hasNext() == false) {
         System.out.println("There is no subclass(sensors)--from
FunctionAnalysiser.getSensor()");
         return false;
      }
      //set the sensor list
      for (Iterator it = elements.iterator(); it.hasNext();) {
         Element elem = (Element) it.next();
         String nodeName = elem.getName();
         if (nodeName.equals("Sensor")) {
            SensorComp sensor = new SensorComp();
            String senname = elem.getTextTrim();
            sensor.setSensorname(senname);
            //set component
            List selements = elem.elements();
            for (Iterator sit = selements.iterator(); sit.hasNext();) {
               Element sele = (Element) sit.next();
               String node = sele.getName();
               if ("operateBy".equals(node)) {
                  String stype = sele.getTextTrim();
                  sensor.setSensortype(stype);
                  String result = new String();
                  if (!sensor.getSensor(result)) {
```

```
                    System.out.println("ERROR:There is no sensor type =" + stype);
                } else {
                    sens.add(sensor);
                }
                break;
            }//end of <operateBy>
        }
    }//end of <SensorComp>
  }
  return true;
}

/**
* Find the needed element for a machine
 * Record the path of the node and the father node of current node.
 * If find the required node the method will analyses its function
 * @param root: root element of the xml ontology.
 * @param macName: required Machine type name
 * @param funcMachine: required Function Name
 * @return: if the function executed successfully
 */
public boolean getMachineElement(Element root, String macName, Machine
funcMachine) {
    List elements = root.elements();
    String curMachine = new String();
    this.findMachine = false;
    Element fatherElemnt = root;
    //begin to search the tree
    for (Iterator it = elements.iterator(); it.hasNext();) {
      Element elem = (Element) it.next();
      String nodeName = elem.getName();
      fatherElemnt = elem;
      if (nodeName.equals("machine")) {
        if (this.findMachine) {
          continue;
        }
        curMachine = elem.getTextTrim();
        //if find the machine, record the element and begin to search function
        if (curMachine.equals(macName)) {
          //put flag into true;
          this.findMachine = true;
          //find the sensor for the machine
          Vector<SensorComp> Senlist = new Vector<SensorComp>();
          if (this.getSensor(elem, Senlist)) {
            for (int i = 0; i < Senlist.size(); i++) {
              SensorComp sens = Senlist.elementAt(i);
              funcMachine.insertSensor(sens);
            }
```

```
            } else {
               System.out.println("ERROR: Sensor initialize failed-- from
FuncAnalysiser.getMachineElement()");
            }
            //find component for the machine
            Vector<ActuatorComp> tempComp = new Vector<ActuatorComp>();
            // if there is component then collect it
            if (getComponent(elem, tempComp)) {
               for (int i = 0; i < tempComp.size(); i++) {
                  ActuatorComp newComp = tempComp.elementAt(i);
                  //get all the motors
                  funcMachine.insertComponent(newComp);
               }
               //insert instructions
               Vector<Command> tempInstr = new Vector<Command>();
               getInstruction(elem, tempInstr);
               for (int i = 0; i < tempInstr.size(); i++) {
                  funcMachine.insertInstruction(tempInstr.elementAt(i));
               }
            } //if there is not then break and go back to the father component
            else {
               System.out.println("the machine :" + macName + " has no components");
            }
            break;//jump out of the loop and begin to collect the information from fathers
         }// end of find machine
         else {
            //if it is not the machine then continue search
            //if the machine is already be found. then just record the path
            if (findMachine == true) {
            } else {
               getMachineElement(elem, macName, funcMachine);
               //after find the nodes search all the parents and get their attributes
               if (findMachine) {
                  Vector<SensorComp> Senlist = new Vector<SensorComp>();
                  if (this.getSensor(elem, Senlist)) {
                     for (int i = 0; i < Senlist.size(); i++) {
                        SensorComp sens = Senlist.elementAt(i);
                        funcMachine.insertSensor(sens);
                     }
                  } else {
                     System.out.println("ERROR: Sensor initialize failed-- from
FuncAnalysiser.getMachineElement()");
                  }
                  Vector<ActuatorComp> tempCompo = new Vector<ActuatorComp>();
                  if (getComponent(elem, tempCompo)) {
                     for (int i = 0; i < tempCompo.size(); i++) {
                        ActuatorComp newComp = tempCompo.elementAt(i);
                        funcMachine.insertComponent(newComp);
```

```
                    }
                    //funcMachine.printComponentList();
                }//end of if(getComponent(elem, tempCompo)
                //Get the instruction from father
                Vector<Command> tempInstr = new Vector<Command>();
                getInstruction(elem, tempInstr);
                for (int i = 0; i < tempInstr.size(); i++) {
                    funcMachine.insertInstruction(tempInstr.elementAt(i));
                }
            }//end of if(findMachine)
        }
      }
    }//end of if (nodeName == "machine")
    //if it is a component
    else if ((nodeName.equals("Component")) && (findMachine == true)) {
      //give a record of these components
      Vector<ActuatorComp> newCompos = new Vector<ActuatorComp>();
      getComponent(fatherElemnt, newCompos);
      for (int i = 0; i < newCompos.size(); i++) {
        funcMachine.insertComponent(newCompos.elementAt(i));
      }
    }//end of else if(nodeName == "Component")&&(findMachine == true)){
    else if ((nodeName.equals("Sensor")) && (findMachine == true)) {
      Vector<SensorComp> Senlist = new Vector<SensorComp>();
      if (this.getSensor(elem, Senlist)) {
        for (int i = 0; i < Senlist.size(); i++) {
          SensorComp sens = Senlist.elementAt(i);
          funcMachine.insertSensor(sens);
        }
      } else {
        System.out.println("ERROR: Sensor initialize failed-- from
FuncAnalysiser.getMachineElement()");
        }
      }
    }//end of for(Iterator it = elements.iterator(); it.hasNext();){
    return findMachine;
  }

  /**
   * Search for component.xml to find if there is duplicated component
   * @param machine: machine that need to be analysed
   * @return: if the function executed successfully
   */
  public boolean TrimComponent(Machine machine) {
    String filename = rcssConstants.ONTOLOGY_DIR + "component.xml";
    Document doc = DOMTools.loadXML(filename);
    Element root = DOMTools.GetRoot(doc);
    List elements = root.elements();
```

```
        for (Iterator it = elements.iterator(); it.hasNext();) {
            Element elem = (Element) it.next();
            if (elem.getName().equals("component")) {
                System.out.println(elem.getText());
            }
        }
        return true;
    }

    /**
     * The function aims at find the element attributes of a certain machine
     * @param macName: machine type name
     * @param machine: machine object that need to be set
     */
    public void getMachineFunction(String macName, Machine machine) {
        //record the 'path' for the machine and find the
        String filename = rcssConstants.ONTOLOGY_DIR + "machine.xml";
        Document doc = DOMTools.loadXML(filename);
Iterator iterList = DOMTools.GetElementIterator(doc);
        while (iterList.hasNext()) {
            Element titleElement = (Element) iterList.next();
            this.getMachineElement(titleElement, macName, machine);
           }
    }

    /**
     * Get the type list of the machine from machine ontology
     * @return: machine type list
     */
    public String getMList() {
        StringBuffer buf = new StringBuffer();
        //get ontology data
        Document doc = DOMTools.loadXML(this.mOntologyFile);
        Iterator iter = DOMTools.GetElementIterator(doc);
        while (iter.hasNext()) {
            Element element = (Element) iter.next();
            String mName = element.getName();
            String mType = element.getTextTrim();
            if ("".equals(buf.toString())) {
                if ("machine".equals(mName)) {
                    buf.append("\"" + mType + "\"");
                    buf.append(getMList(element));
                }
            } else {
                if ("machine".equals(mName)) {
                    buf.append(",\"" + mType + "\"");
                    buf.append(getMList(element));
                }
```

```
            }
        }
        String result = new String();
        result = buf.toString();
        return result;
    }

    /**
     * Get the machine list for the machine initial page
     * @return: Machine list required
     */
    public String getAllMachineList(){
          StringBuffer buf = new StringBuffer();
        //get ontology data
        Document doc = DOMTools.loadXML(this.mOntologyFile);
        //analysis

        Iterator iter = DOMTools.GetElementIterator(doc);


        while (iter.hasNext()) {
            Element element = (Element) iter.next();
            String mName = element.getName();
            String mType = element.getTextTrim();
            //System.out.println("Node type = " + mName);
            //System.out.println("Node Name = " + mType);
            if ("".equals(buf.toString())) {
                if ("machine".equals(mName)) {
                    buf.append("<ul>" + mType);
                    buf.append(getSubMachineList(element));
                    buf.append("</ul>");
                }
            }else{
                if ("machine".equals(mName)) {
                    buf.append("<ul>" + mType);
                    buf.append(getSubMachineList(element));
                    buf.append("</ul>");
                }
            }
        }
        String result = new String();
        result = buf.toString();
        //generate data list
        return result;
    }
```

```
    /**
     * Get the type list of the machine from machine ontology
     * element: the root element of the analysed ontology document
     * @return: the machine list
     */
    public String getMList(Element element) {
        StringBuffer buf = new StringBuffer();
        String result = new String();
        Iterator iter = DOMTools.GetElementIterator(element);
        while (iter.hasNext()) {
            Element elem = (Element) iter.next();
            String nType = elem.getName();
            if ("machine".equals(nType)) {
                String nName = elem.getTextTrim();
                System.out.println(nName);
                buf.append(",\"" + nName + "\"");
                buf.append(getMList(elem));
            }
        }
        result = buf.toString();
        return result;
    }

    /**
     * This function is used to generate machine list in<li> format
     * for the initial page in client side
     * @param element
     * @return the machine list code for the page
     */
        public String getSubMachineList(Element element) {
        StringBuffer buf = new StringBuffer();
        String result = new String();
        Iterator iter = DOMTools.GetElementIterator(element);
        while (iter.hasNext()) {
            Element elem = (Element) iter.next();
            String nType = elem.getName();
            if ("machine".equals(nType)) {
                String nName = elem.getTextTrim();
                System.out.println(nName);
                buf.append("<li>" + nName + "</li>");
                buf.append(getSubMachineList(elem));
            }
        }
        result = buf.toString();
        return result;
    }
}
```

D.2. DeviceReHandler.java

```
 package lbr.cs.rcss.rl.device;

import lbr.cs.rcss.rl.StateRecord;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
import lbr.cs.rcss.rl.device.webservice.MachineWSHandler;
import lbr.cs.rcss.common.DOMTools;
import lbr.cs.rcss.rl.SessionRecord;
import lbr.cs.rcss.common.SessionTools;
import lbr.cs.rcss.common.rcssConstants;
import org.dom4j.Attribute;
import org.dom4j.Document;
import org.dom4j.Element;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * This class helps to communicate with controlled machines via machine web services.
 * It also maintain device states and record current control machine state.
 */
public class DeviceReHandler {

    private String msg; // control command

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    /**
     * This class is used to initialise machine state so that it can be watched and updated in
the server side
     * @param userName: user's name that can identify a user
     * @return: the result of the action. if it is sessionID, then the function is successfully
executed
     * Otherwise, the result will be "".
     */
    public String InitialSelMachineState(String userName) {
        String result = "";
        //analysis the msg and
        if ("".equals(msg)) {
```

```
            String erro = "ERROR: there is no data in msg-- reprot from Service server
DeviceReHandler.createSession()";
            System.out.println(erro);
            return result;

        }
        if (msg == null) {
            String erro = "ERROR: message is null--reprot from Service server
DeviceReHandler.createSession()";
            System.out.println(erro);
            return result;
        }
        if ("".equals(userName)) {
            String erro = "ERROR: there is no data in userName-- reprot from Service server
DeviceReHandler.createSession()";
            System.out.println(erro);
            return result;

        }
        if (userName == null) {
            String erro = "ERROR: userName is null--reprot from Service server
DeviceReHandler.createSession()";
            System.out.println(erro);
            return result;
        }
        if (msg.indexOf(",") == 0) {
            String erro = "ERROR: the input message is invalidated--reprot from Service
server DeviceReHandler.createSession()";
            System.out.println(erro);
            return result;
        }
        //initial session ID
        int SID = SessionTools.getMaxIDNum("sessionID");
        String sessionID = String.valueOf(SID);
        //break msg to get machineID
        int begin = 0;
        while (msg.indexOf(",") > 0) {
            String maID = msg.substring(begin, msg.indexOf(","));
            SessionRecord srecord = new SessionRecord();
            srecord.setControlType("1"); //0: monitor only 1: control
            srecord.setSessionID(sessionID);
            srecord.setTimeStamp(rcssConstants.TIME_STAMP);
            srecord.setUserName(userName);
            srecord.setMachineID(maID);
            boolean bol = srecord.insertData(result);
            if (!bol) {
                System.out.println("ERROR: session record for machine " + maID + "
initialisation failed from DeviceReHandler.InitialSelMachineState() ");
```

```
            return result;
         }
         boolean inistate = this.initialStateRecord(result, maID);

         msg = msg.substring(msg.indexOf(",") + ",".length());

         if (!inistate) {
            System.out.println("ERROR: state record for machine " + maID + "
initialisation failed from DeviceReHandler.InitialSelMachineState() ");
            return result;
         }
      }
      if (msg.indexOf(",") < 0) {
         SessionRecord srecord = new SessionRecord();
         srecord.setMachineID(msg);
         srecord.setControlType("1"); //0: monitor only 1: control
         srecord.setSessionID(sessionID);
         srecord.setTimeStamp(rcssConstants.TIME_STAMP);
         srecord.setUserName(userName);
         boolean bol = srecord.insertData(result);
         if (!bol) {
            System.out.println("ERROR: session record for machine " + msg + "
initialisation failed from DeviceReHandler.InitialSelMachineState() ");
            return result;
         }
         boolean inistate = this.initialStateRecord(result, msg);
         if (!inistate) {
            System.out.println("ERROR: state record for machine " + msg + " initialisation
failed from DeviceReHandler.InitialSelMachineState() ");
            return result;
         }
      }
      return sessionID;
   }

   /**
    * Initial machine State Record from input Machine State Record data
    * @param result : Error information during this process
    * @param sRecord: Session Record
    * @return: if the function is executed successfully
    */
   public boolean initialStateRecord(String result, String mID) {

      if ((mID.equals("")) || (mID == null)) {
         result = "ERROR: There is no machine ID input! machineID = " + mID;
         System.out.println("ERROR: There is no machine ID input! machineID = " +
mID);
         return false;
```

```
        //search in database if the record already exists
        StateRecord arecord = new StateRecord();
        arecord.setMachineID(mID);
        boolean isreg = arecord.isMachineRegistered(result);
        if (!isreg) {
            //get state record from the client side
            String code = MachineWSHandler.readStateFile_client(mID); //get machine state
record
            Document doc = DOMTools.string2Document(code);
            Iterator iter = DOMTools.GetElementIterator(doc);
            while (iter.hasNext()) {
                Element elem = (Element) iter.next();
                String item = elem.getName();
                //begin to analyze
                if (item.equals("machineState")) {
                    this.initialMachineState(elem, mID);
                }
            }
            return true;
        }
        //if it comes to here it means the machine already be initiated
        return true;
    }

    /**
     * Input all the information from machine state description into the data base
     * The input information including machine component record and machine command
record
     * @param elem: <machineState> element from Machine State Record
     * @param mID: machineID
     * @return: if the function is executed successfully
     */
    public boolean initialMachineState(Element elem, String mID) {
        Iterator iter = elem.elementIterator();
        if (!iter.hasNext()) {
            System.out.println("ERROR: There is no item state in the machine, Please check
state record-- from service server DeviceReHandler. initialMachineState()");
            return false;
        }
        while (iter.hasNext()) {
            Element ele = (Element) iter.next();
            //if it is a command
            if (ele.getName().equals("command")) {
                //get command name
                String comName = ele.getTextTrim();
                //get command value
                Iterator it = ele.attributeIterator();
                while (it.hasNext()) {
```

```
                Attribute attribute = (Attribute) it.next();
                if ("state".equals(attribute.getName())) {
                    String comValue = attribute.getText();
                    StateRecord arecord = new StateRecord();
                    arecord.setMachineID(mID);
                    arecord.setFunction_type("1");
                    arecord.setFunction_name(comName);
                    arecord.setState(comValue);
                    arecord.setStartTime(rcssConstants.TIME_STAMP);
                    String code = "";
                    arecord.insertData(code);
                }
            }
        }//if(ele.getName().equals("command")){
        //if it is component State
        else if (ele.getName().equals("componentState")) {
            String compName = ele.getText();
            this.initialCompState(ele, compName, mID);
        }
    }
    return true;
}

/**
 * initial component State record
 * insert related record into state_record Database
 * @param elem: component element from Machine State Description
 * @param compName: component name
 * @param macID: MachineID
 * @return: if the function is executed successfully
 */
public boolean initialCompState(Element elem, String compName, String macID) {
    Iterator m_motor = elem.elementIterator();
    if (!m_motor.hasNext()) {
        System.out.println("ERROR: no actuator record in this component. Please check
the initial record!");
        return false;
    }
    while (m_motor.hasNext()) {
        Element mElem = (Element) m_motor.next();
        if (mElem.getName().equals("actuatorState")) {
            String mName = mElem.getTextTrim();
            Iterator attribute = mElem.elementIterator();
            if (!attribute.hasNext()) {
                System.out.println("ERROR: There is no attribute in the actuator. please
check the record!");
                return false;
            }
```

```
            while (attribute.hasNext()) {
               Element aEle = (Element) attribute.next();
               String aName = aEle.getTextTrim();
               Iterator iter = aEle.attributeIterator();
               if (!iter.hasNext()) {
                  System.out.println("ERROR: There is no attribute for actuator's Atribute
item");
                  return false;
               }
               StateRecord arecord = new StateRecord();
               arecord.setMachineID(macID);
               arecord.setComponent(compName);
               arecord.setMotor(mName);
               arecord.setFunction_type("3");
               arecord.setFunction_name(aName);
               while (iter.hasNext()) {
                    Attribute aAttribute = (Attribute) iter.next();
                  String t_name = aAttribute.getName();
                  if ("type".equals(t_name)) {
                     arecord.setMotor_attributes(aAttribute.getText());
                  } else if ("value".equals(t_name)) {
                     arecord.setCommand_value(aAttribute.getText());
                  } else if ("state".equals(t_name)) {
                     arecord.setState(aAttribute.getText());
                  }
               }
               String erro = "";
               arecord.insertData(arecord.getMachineID());
            }
         } else if (mElem.getName().equals("sensor")) {
            String sencode = mElem.getTextTrim();
            this.initialSensorState(sencode, macID);
         }
      }
      return true;
   }

   /**
    * Initial sensor state, put all the data from State record to state_record Data Base
    * @param senJson : input JSON from machine state record
    * @param macID: machine ID
    * @return:if the function is executed successfully
    */
   public boolean initialSensorState(String senJson, String macID) {
      try {
         JSONObject obj = new JSONObject(senJson);
         String sensorname = obj.getString("sensorName");
         String sensorvalue = obj.getString("sensorValue");
```

```
            String sensortype = obj.getString("sensorType");
            String range = obj.getString("sensorRange");
            String statetype = obj.getString("stateType");
            String unit = obj.getString("unit");
            StateRecord arecord = new StateRecord();
            arecord.setMachineID(macID);
            arecord.setFunction_type("4");
            arecord.setComponent(sensorname);
            arecord.setCommand_value(sensorvalue);
            arecord.setSensortype(sensortype);
            arecord.setRange(range);
            arecord.setUnit(unit);
            arecord.setMotor_attributes(statetype);
            String result = "";
            boolean erro = arecord.insertData(result);
            return erro;
        } catch (JSONException ex) {
            Logger.getLogger(DeviceReHandler.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return false;
    }

    /**
     * Handle the event message reported from the machine client side.
     * Update the change in machine state database.
     * @param msg: reported message
     * @param erroinfo: error information
     */
    public void handleMachineEvent(String msg, String erroinfo) {
        StateRecord record = new StateRecord();
        if (!record.updateMachineState(msg, erroinfo)) {
            System.out.println("ERROR happened when update the state + " + msg);
        }
        record.updateMachineState(msg, erroinfo);
        return;
    }
}
```

D.3. MachineControlService.java

```
 package lbr.cs.rcss.cml.service;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import lbr.cs.rcss.rl.device.webservice.MachineWSHandler;

/**
 * This class will offer web service API for Machine Control Service
 * It includes services to control the machine
 * and collect machine client side information
 */
@WebService()
public class MachineControlService {

    /**
     * Get client side available machine list and information
     * @return: the list of the available machine list
     */
    @WebMethod(operationName = "getAvailableMachineList")
    public String getAvailableMachineList() {
        return MachineWSHandler.getAvailableMachineList_client();
    }

    /**
     * Get Machine State Description code from the client machine server side
     * @param mName: machine type name
     * @param mID: machine ID
     * @return: Machine State Description code
     */
    @WebMethod(operationName = "getMachineRecordString")
    public String getMachineRecordString(@WebParam(name = "mName") String
mName, @WebParam(name = "mID") String mID) {
        try {
            return MachineWSHandler.getMachineRecordString_client(mName, mID);
        } catch (Exception ex) {
            String error = "ERROR: failed to connect remote device web service - from
MachineControlService";
            return error;
        }
    }
```

```
    /**
     * This function is used to get the location information from state record
     * in the future, the state supposed to get according to the IP address or more
complicated parameters.
     * @return the address in machine state description
     */
    @WebMethod(operationName = "analyzeLocation")
    public String analyzeLocation(@WebParam(name = "mID") String macID,
            @WebParam(name = "mType") String macType) {
        return (MachineWSHandler.analyzeLocation_client(macID, macType));
    }

    /**
     * Get machine State Record file content
     * after checking the file list. It will find the corresponding file and return the code
     * @param macID: machineID
     * @return: the Machine State Description for this machineID
     */
    @WebMethod(operationName = "readStateFile")
    public String readStateFile(@WebParam(name = "mID") String macID) {
        return MachineWSHandler.readStateFile_client(macID);
    }

    /**
     * Function that handle the message come from the server side
     * If it is included by <msg></msg> then it will be a commmand that change machine
states
     * @param msg : message String
     * @return: the result of the function
     */
    @WebMethod(operationName = "handlerMessage")
    public boolean handlerMessage(@WebParam(name = "message") String msg) {
        if ((msg == null) || (msg.equals(""))) {
            return false;
        }
        return MachineWSHandler.handlerMessage_client(msg);
    }
}
```

E. IWUI service project implementation

E.1. ModelReasoner.java

```
package lbr.cs.rcss.sdl.wui.model.creator;

import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import lbr.cs.rcss.sdl.wui.common.PageData;
import lbr.cs.rcss.sdl.wui.common.rcssConstants;
import lbr.cs.rcss.sdl.wui.model.creator.session.ActuatorCompSession;
import lbr.cs.rcss.sdl.wui.model.creator.session.AttributeSession;
import lbr.cs.rcss.sdl.wui.model.creator.session.CommandSession;
import lbr.cs.rcss.sdl.wui.model.creator.session.SensorCompSession;
import lbr.cs.rcss.sdl.wui.model.creator.session.SessionContextRecord;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * This class is used to reason CPL according to Session Context Record
 * @author guoxi
 */
public class ModelReasoner {

    private String sessID;
    private SessionContextRecord sessionContext;
    private String CPLcode;
    private String rulefile;

    public String getCPLcode() {
        return CPLcode;
    }

    public void setCPLcode(String CPLcode) {
        this.CPLcode = CPLcode;
    }

    public SessionContextRecord getSessionContext() {
        return sessionContext;
    }

    public void setSessionContext(SessionContextRecord sessionContext) {
        this.sessionContext = sessionContext;
    }
```

```
    public String getSessionID() {
        return sessID;
    }

    public void setSessionID(String sessionID) {
        this.sessID = sessionID;
    }

    public String getRulefile() {
        return rulefile;
    }

    public void setRulefile(String rulefile) {
        this.rulefile = rulefile;
    }

    /**
     * Analysis session context and generate CPL for session context
     * @param erroInform: error information if the function is failed
     * @return: if the function is successfully executed
     * NOTE: session context record need to be set before using
     */
    public boolean analysisCPL(String erroInform) {
        if (this.getSessionContext() == null) {
            this.setCPLcode("");
            erroInform = "ERROR: there is no input session context record -- from
ModelReasoner.initiateCPL()";
            return false;
        }
        //initial json object
        JSONObject finalobj = new JSONObject();//final object
        JSONObject base = new JSONObject(); // the base element -- session_information
        JSONArray machineList = new JSONArray();
        try {
            base.put("sessionID", this.getSessionContext().getSessionID());
            base.put("username", this.getSessionContext().getUserName());
        } catch (JSONException ex) {
            erroInform = "ERROR: JSON format wrong when generate session_information
item -- from ModelReasoner.initiateCPL()";
            Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE, null, ex);
            return false;
        }
        //get control machine information (machineID)
        ///////////////Get Sessioniformation and Keys/////////////////
        JSONArray controlinf = new JSONArray();
        for (int i = 0; i < this.getSessionContext().getMSession().size(); i++) {
            JSONObject masession = new JSONObject();
            try {
```

```
            JSONObject generalinf = new JSONObject();
            generalinf.put("machineName",
this.getSessionContext().getMSession().elementAt(i).getMachineName());
            generalinf.put("machineType",
this.getSessionContext().getMSession().elementAt(i).getMachineType());
            generalinf.put("location",
this.getSessionContext().getMSession().elementAt(i).getLocation());
            generalinf.put("productor",
this.getSessionContext().getMSession().elementAt(i).getManufacturer());
            generalinf.put("machineID",
this.getSessionContext().getMSession().elementAt(i).getMachineID());

machineList.put(this.getSessionContext().getMSession().elementAt(i).getMachineID());
            masession.put("machineID",
this.getSessionContext().getMSession().elementAt(i).getMachineID());
            masession.put("general_information", generalinf);

         } catch (JSONException ex) {
            erroInform = "ERROR: JSON format wrong when generate
general_informaiont item -- from ModelReasoner.initiateCPL()";

            Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE, null,
ex);
            return false;
         }
         //begin to analysis command list
         Vector<CommandSession> cmmlist =
this.getSessionContext().getMSession().elementAt(i).getCommSession();
         JSONArray comminfo = new JSONArray();
         if (cmmlist != null) {
            for (int j = 0; j < cmmlist.size(); j++) {
               JSONObject commobj = new JSONObject();
               try {
                  commobj.put("function_name",
cmmlist.elementAt(j).getCommandName());
                  commobj.put("state", cmmlist.elementAt(j).getCommandState());
               } catch (JSONException ex) {
                  erroInform = "ERROR: JSON format wrong when generate
command_information item -- from ModelReasoner.initiateCPL()";
                  Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE,
null, ex);
                  return false;
               }
               if (commobj != null) {
                  comminfo.put(commobj);
               }
            }
         }
```

```
        //added command information into JSON code
        if (comminfo != null) {
            try {
                masession.put("command_information", comminfo);
            } catch (JSONException ex) {
                erroInform = "ERROR: JSON format wrong when generate
command_information item -- from ModelReasoner.initiateCPL()";
                Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE,
null, ex);
                return false;
            }
        }
        //begin to analysis sensor list
        Vector<SensorCompSession> senlist =
this.getSessionContext().getMSession().elementAt(i).getSenSession();
        JSONArray seninfo = new JSONArray();
        if (senlist != null) {
            for (int j = 0; j < senlist.size(); j++) {
                JSONObject senobj = new JSONObject();
                try {
                    senobj.put("component", senlist.elementAt(j).getSensorname());
                    senobj.put("sensortype", senlist.elementAt(j).getSensortype());
                    senobj.put("motor_attributes", senlist.elementAt(j).getStatetype());
                    senobj.put("command_value", senlist.elementAt(j).getSensorvalue());
                    senobj.put("unit", senlist.elementAt(j).getUnit());
                    JSONObject range = new
JSONObject(senlist.elementAt(j).getSensorrange());
                    senobj.put("range", range);
                } catch (JSONException ex) {
                    erroInform = "ERROR: JSON format wrong when generate
sensor_information item -- from ModelReasoner.initiateCPL()";
                    Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE,
null, ex);
                    return false;
                }
                if (senobj != null) {
                    seninfo.put(senobj);
                }
            }
        }
        if (seninfo != null) {
            try {
                masession.put("sensor_information", seninfo);
            } catch (JSONException ex) {
                erroInform = "ERROR: JSON format wrong when added Sensor information
item to machine item -- from ModelReasoner.initiateCPL()";
                Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE,
null, ex);
```

```
                return false;
            }
        }
        //begin to analysis actuator list
        Vector<ActuatorCompSession> actlist =
this.getSessionContext().getMSession().elementAt(i).getActuatorSession();
        JSONArray actinfo = new JSONArray();
        if (actlist != null) {
            for (int j = 0; j < actlist.size(); j++) {
                try {
                    Vector<AttributeSession> attlist =
actlist.elementAt(j).getActSession().getAttrState();
                    for (int m = 0; m < attlist.size(); m++) {
                        JSONObject actobj = new JSONObject();
                        actobj.put("component", actlist.elementAt(j).getComponentName());
                        actobj.put("motor",
actlist.elementAt(j).getActSession().getMotorName());
                        actobj.put("function_name", attlist.elementAt(m).getAttrName());
                        actobj.put("motor_attributes", attlist.elementAt(m).getAttrType());
                        actobj.put("command_value", attlist.elementAt(m).getAttrValue());
                        actobj.put("state", attlist.elementAt(m).getAttrState());
                        if (actobj != null) {
                            actinfo.put(actobj);
                        }
                    }
                } catch (JSONException ex) {
                    erroInform = "ERROR: JSON format wrong when generate actuator
information item -- from ModelReasoner.initiateCPL()";
                    Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE,
null, ex);
                    return false;
                }
            }
        }
        if (actinfo != null) {
            try {
                masession.put("component_information", actinfo);
            } catch (JSONException ex) {
                erroInform = "ERROR: JSON format wrong when put actuator item to
machine item -- from ModelReasoner.initiateCPL()";
                Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE,
null, ex);
                return false;
            }
        }
        if (masession != null) {
            controlinf.put(masession);
        }
```

```
        }// end of for loop for machine session
        // add all the information in to the return object
        try {
            base.put("machineID", machineList);
            finalobj.put("session_information", base);
            finalobj.put("control_information", controlinf);
        } catch (JSONException ex) {
            erroInform = "ERROR: JSON format wrong when generate the final item-- from
ModelReasoner.initiateCPL()";
            Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE, null, ex);
            return false;
        }
        //get general information for the machine
        if (finalobj != null) {
            this.setCPLcode(finalobj.toString());
        } else {
            erroInform = "ERROR: final JSON code is null-- from
ModelReasoner.initiateCPL()";
            return false;
        }
        //analysis alarm information
        this.checkAlarm(erroInform);
        //analysis navigation information
        //make sure to get other PageData information in to the CPL
        PageData page = new PageData();
        page.setSessionID(this.getSessionContext().getSessionID());
        page.setDatacode(this.getCPLcode());
        //should update navigation
        if (!page.updateData(erroInform)) {
            erroInform = "ERROR: Can not update the CPL data -- from
ModelReasoner.analysisCPL() :" + erroInform;
            return false;
        }
        String currentCPL = new String();
        currentCPL = page.getDataCodeRecord(erroInform);
        if ((currentCPL == null) || ("".equals(currentCPL))) {
            erroInform = "ERROR: Can not load CPL from database-- from
ModelReasoner.analysisCPL() : " + erroInform;
            return false;
        }
        this.setCPLcode(currentCPL);

        if (this.getSessionContext().getMSession().size() > 1) {
            String code = this.analysisNavContent(erroInform);
            if ((code == null) || (code.equals(""))) {
                System.out.println("ERROR in navigation analysis. Reason : " + erroInform);
            }
        }
```

```
        return true;
    }

    /**
     * Get the navigation content and put it into data base as "navigation_content" item
     * @param result: error information if the function if failed
     * @return: the navigation object code (the value of "navigation_content")
     */
    public String analysisNavContent(String result) {

        if ((this.getCPLcode() == null)) {
            result = "ERROR: the session context information is not complete---from
ModelReasoner.getNavContent() ";
            return "";
        }
        if ("".equals(this.getCPLcode())) {
            result = "ERROR: the session context information is invalid--from
ModelReasoner.getNavContent()";
            return "";
        }
        String code = "";
        //read the page code from the data base
        //get addtional_information and general information
        //mapping all general information into the new code
        JSONObject obj;
        try {
            obj = new JSONObject(this.getCPLcode());
            JSONArray conarr = obj.getJSONArray("control_information");
            JSONArray naviInfor = new JSONArray();//final list that put into the view
            //judge if the machine belongs to the same type
            boolean sametype = true;
            String comparedtype = "";
            for (int i = 0; i < conarr.length(); i++) { //loop control information array
                JSONObject tempobj = conarr.getJSONObject(i);
                if (tempobj.has("general_information")) { //get general information
                    JSONObject genobj =
tempobj.getJSONObject("general_information");//general information object
                    if (genobj.has("machineType")) {
                        //get the first type value
                        if (i == 0) {
                            comparedtype = genobj.getString("machineType");
                        }
                        String type = genobj.getString("machineType");
                        if (!comparedtype.equals(type)) {
                            sametype = false;
                            break;
                        }
                    }
```

```
            }//end of get general information
        }
        for (int i = 0; i < conarr.length(); i++) {
            JSONObject navlist = new JSONObject();// item name list for each view item
            //for each machine
            JSONObject navobj = new JSONObject(); //each item that need to be put in to
the view
            String maID = conarr.getJSONObject(i).getString("machineID");
            JSONObject tempobj = conarr.getJSONObject(i);
            if (tempobj.has("general_information")) {
                JSONObject genobj =
tempobj.getJSONObject("general_information");//general information object
                 //get all the key format {machineID: machine ID , machineName: Machine
Name... key: showed name}
                for (int j = 0; j < JSONObject.getNames(genobj).length; j++) {
                    String key = JSONObject.getNames(genobj)[j]; // such as "machineID"
                    String value = genobj.getString(key);
                    if (!navlist.has(key)) {
                        //check the map to find the name of the item
                        String itemvalue = new String();
                        String map = new String();
                        map = rcssConstants.NAME_MAP;  //mapping if the name has the same
meaning
                        JSONObject mapobj = new JSONObject(map);
                        if (mapobj.has(key)) {
                            itemvalue = mapobj.getString(key);
                            navlist.put(key, itemvalue);
                            navobj.put(itemvalue, value);//if find the maping. put the item as
itemvalue in tot the view
                        } else {
                            navlist.put(key, key);
                            navobj.put(key, value);//if it is not in the map ,put the original key in
to the value
                        }
                    }
                }
                //if they are of the same type, there are more key words
                if (sametype) {
                    //get component list
                    if (tempobj.has("component_information")) {
                        JSONArray compobj =
tempobj.getJSONArray("component_information");
                        //put all component information in to the key
                        for (int k = 0; k < compobj.length(); k++) {
                            String comkey = compobj.getJSONObject(k).getString("component");
                            String statevalue =
compobj.getJSONObject(k).getString("command_value");
                            String validate = compobj.getJSONObject(k).getString("state");
```

```
                        if (validate.equals("true")) {
                            navobj.put(comkey, statevalue);
                        }
                    }
                }
            }
            naviInfor.put(navobj); //put the item into the navigation content
             //////////end of get key
         } else {
            System.out.println("ERROR: There is no generate information in the page
data. Error machine ID = " + maID);
            return "";
        }
     }//end of loop
    //now begin to add aditional_information into naviInfor
     if (obj.has("additional_information")) {
        JSONArray aditemlist = new JSONArray();
        JSONArray addarr = obj.getJSONArray("additional_information");
        //initiallze the view
        for (int j = 0; j < addarr.length(); j++) {
            JSONObject key = addarr.getJSONObject(j).getJSONObject("key");
            String keyname = key.getString("itemName");
            boolean find = false;
            for (int k = 0; k < aditemlist.length(); k++) {
                if (aditemlist.getString(k).equals(keyname)) {
                    find = true;
                    break;
                }
            }
            //if it is new item then added it into the list
            if (!find) {
                aditemlist.put(keyname);
                //initiate the "NONE" value to new items in the informatino view
                for (int n = 0; n < naviInfor.length(); n++) {
                    naviInfor.getJSONObject(n).put(keyname,
rcssConstants.NAVIGATION_NONE_VALUE);
                }
            }
        }
        //begin to set the value into the view
        for (int j = 0; j < addarr.length(); j++) {
            JSONObject key = addarr.getJSONObject(j).getJSONObject("key");
            String keyname = key.getString("itemName");
            String keymaID = key.getString("machineID");
            String keyvalue = addarr.getJSONObject(j).getString("itemValue");
            boolean find = false;
            for (int n = 0; n < naviInfor.length(); n++) {
                if (naviInfor.getJSONObject(n).getString("Machine
```

```
{
                    find = true;
                    naviInfor.getJSONObject(n).put(keyname, keyvalue);
                    break;
                }
            }
        }
    }
    //finally, put the information into the data base
    //System.out.println(naviInfor.toString());
    obj.put("navigation_content", naviInfor);
    this.setCPLcode(obj.toString());
    code = naviInfor.toString();
    return code;
  } catch (JSONException ex) {
    Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE, null, ex);
    return "";
  }
}

/**
 * This function is used to check the alarm set by user.
 * Get the "alarm_information " JSON code from the CPL
 * Compare the data with sensor data in CPL and analyze if it is a dangerous
 * Give a rank to the current data
 * @return whether the alarm check is successful
 */
public boolean checkAlarm(String result) {
  if ((this.getCPLcode() == null)) {
    result = "ERROR: the session context information is not complete---from
ModelReasoner.getNavContent() ";
    return false;
  }
  if ("".equals(this.getCPLcode())) {
    result = "ERROR: the session context information is invalid--from
ModelReasoner.getNavContent()";
    return false;
  }
  try {
    PageData pData = new PageData();
    pData.setSessionID(this.getSessionContext().getSessionID());
    String code = pData.getDataCodeRecord(result);
    //get the alarm information from DB
    if ((code == null) || ("".equals(code))) {
      result = "ERROR: Page code is not initiated -- from
ModelReasoner.CheckAlarm()";
      return false;
    }
```

```
        JSONObject pagecode = new JSONObject(code);
        //newpagecode contains new session information
        JSONObject newpagecode = new JSONObject(this.getCPLcode());
        if (!pagecode.has("alarm_information")) {
           return false;
        }
        JSONArray alarmarr = pagecode.getJSONArray("alarm_information");
        JSONArray newalarm = new JSONArray();
        for (int i = 0; i < alarmarr.length(); i++) {
           JSONObject alitem = alarmarr.getJSONObject(i);
           //check the alarm information from database
           JSONObject alitemkey = alitem.getJSONObject("key");
           String mID = alitemkey.getString("machineID");
           String sName = alitemkey.getString("sensorName");
           String temp = this.getSensorValue(mID, sName, result); //get current result
from the database
           if ((temp == null) || ("".equals(temp))) {
              System.out.println("ERROR: Can not get sensor value. Error Information:" +
result);
              return false;
           }
           JSONObject resultobj = new JSONObject(temp);
           String value = resultobj.getString("value");
           String type = resultobj.getString("type");
           String oldvalue = alitem.getString("sensorValue");// value from the record
           String compvalue = alitem.getString("compareValue"); //value should be
compared
           String comptype = alitem.getString("compareType"); //compared type
           //analysis of the information
           String astate = new String();//alarm state : normal, risky, alarm
           String distance = new String();
           if (type.equals(rcssConstants.INT_NUMBER)) {
              //if value is not set yet
              if (value.trim().equalsIgnoreCase("null")) {
                 astate = rcssConstants.ALARM_ERROR;
                 distance = "0";
                 alitem.put("sensorValue", value);
                 alitem.put("distance", distance);
                 alitem.put("alarmType", astate);
                 newalarm.put(alitem);
                 continue;
              }
              //if there is null in the old value, replace it with new value
              if (oldvalue.trim().equalsIgnoreCase("null")) {
                 oldvalue = value;
              }
              int nvalue = Integer.parseInt(value);
              int ovalue = Integer.parseInt(oldvalue);
```

```
                int cvalue = Integer.parseInt(compvalue);

                /**
                 * There are four ranks:
                 * alarm-- already arrived the alarm.
                 * risky-- very dangerous situation;
                 * normal-- normal situation
                 * error-- there is an error in the sensor value
                 * There are also ranks -- depends on how many steps it took to the alarm
                 * Rules 1: if the data meet the alarm condition -- alarm
                 * Rules 2: if the data next change (2 step) will meet the condition( the gap
change speed is fast) -- risky
                 * Rules 3: if the data still get a while before meet the condition. -- normal
                 * Rules 4: if the sensor data is null or other condition which is not belongs to
any of the state described before-- error
                 */
                if (comptype.equals(">")) {
                   if (nvalue > cvalue) {
                      astate = rcssConstants.ALARM_ALARM;
                      distance = "0";
                   } else {
                      int gap = Math.abs(cvalue - nvalue);
                      distance = String.valueOf(gap);
                      if ((nvalue > ovalue) && (Math.abs(nvalue - ovalue) > gap)) {
                         astate = rcssConstants.ALARM_RISKY;
                      } else {
                         astate = rcssConstants.ALARM_NORMAL;
                      }
                   }
                } else if (comptype.equals("<")) {
                   if (nvalue < cvalue) {
                      astate = rcssConstants.ALARM_ALARM;
                      distance = "0";
                   } else {
                      int gap = Math.abs(cvalue - nvalue);
                      distance = String.valueOf(gap);
                      if ((nvalue < ovalue) && (Math.abs(nvalue - ovalue) > gap)) {
                         astate = rcssConstants.ALARM_RISKY;
                      } else {
                         astate = rcssConstants.ALARM_NORMAL;
                      }
                   }
                } else if (comptype.equals("=")) {
                   if (nvalue == cvalue) {
                      astate = rcssConstants.ALARM_ALARM;
                      distance = "0";
                   } else {
                      int gap = Math.abs(cvalue - nvalue);
```

```
                    distance = String.valueOf(gap);
                    //if nvalue and ovalue are saparated on each side of cvalue
                    if (Math.abs(nvalue - ovalue) > gap) {

                        astate = rcssConstants.ALARM_RISKY;
                    } else {
                        astate = rcssConstants.ALARM_NORMAL;
                    }
                }
            } else {
                System.out.println("ERROR: Can not find this type when analyze sensor
alarm data "
                        + "in machineID=" + mID + ";  sensor name = " + sName);
            }
          }
          ////end of analysis
          alitem.put("sensorValue", value);
          alitem.put("distance", distance);
          alitem.put("alarmType", astate);
          newalarm.put(alitem);
        }
        //update the alarm data -- update the "alarm_information" with the updated data
        newpagecode.put("alarm_information", newalarm);
        this.setCPLcode(newpagecode.toString());
        return true;
      } catch (JSONException ex) {
        Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE, null, ex);
      }
      return false;
    }

    /**
     * search for sensor value from CPL according to given machineID and component
name
     * @param maID: machine ID
     * @param compName: component name
     * @param errorinf: error information
     * @return the value of the sensor
     */
    public String getSensorValue(String maID, String compName, String errorinf) {
      if ((this.getCPLcode() == null)) {
        errorinf = "ERROR: the CPL is not complete---from
ModelReasoner.getNavContent() ";
        return "";
          }
      if ("".equals(this.getCPLcode())) {
        errorinf = "ERROR: the CPL is invalid--from ModelReasoner.getSensorValue()";
        return "";
```

```
        if ((maID == null)) {
            errorinf = "ERROR: the machineID is not complete---from
ModelReasoner.getSensorValue() ";
            return "";
        }
        if ("".equals(maID)) {
            errorinf = "ERROR: the machineID is invalid--from
ModelReasoner.getSensorValue()";
            return "";
        }
        if ((compName == null)) {
            errorinf = "ERROR: the component Name  is not complete---from
ModelReasoner.getSensorValue() ";
            return "";
        }
        if ("".equals(compName)) {
            errorinf = "ERROR: the component Name is invalid--from
ModelReasoner.getSensorValue()";
            return "";
        }

        try {
            JSONObject cpl = new JSONObject(this.getCPLcode());
            JSONArray coninf = cpl.getJSONArray("control_information");
            for (int i = 0; i < coninf.length(); i++) {
                JSONObject mainf = coninf.getJSONObject(i);
                if (mainf.get("machineID").equals(maID)) {
                    JSONArray seninf = mainf.getJSONArray("sensor_information");
                    for (int j = 0; j < seninf.length(); j++) {
                        JSONObject sensor = seninf.getJSONObject(j);
                        if ((sensor.has("component")) &&
(sensor.getString("component").equals(compName)) &&
(sensor.has("command_value"))) {
                            JSONObject obj = new JSONObject();
                            try {
                                String value = sensor.getString("command_value");
                                String type = sensor.getString("motor_attributes");
                                obj.put("value", value);
                                obj.put("type", type);
                                String code = obj.toString();
                                return code;
                            } catch (JSONException ex) {
                                Logger.getLogger(PageData.class.getName()).log(Level.SEVERE,
null, ex);
                            }
                        }
                    }
                }
```

```
            }
            errorinf = "ERROR: Can not find the data";
            return "";
        } catch (JSONException ex) {
            Logger.getLogger(ModelReasoner.class.getName()).log(Level.SEVERE, null, ex);
            return "";
        }
    }
}
```

E.2. Struts.xml

```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
   <package name="struts2" extends="struts-default">
      <action name="Login" class="lbr.cs.rcss.sdl.wui.action.LoginAction">
         <result name="success">User_welcome.jsp</result>
         <result name="fail">Login_fail.jsp</result>
      </action>
      <action name="UserRegister" class="lbr.cs.rcss.sdl.wui.action.UserRegAction">
         <result name="success">User_login.jsp</result>
         <result name="error">User_register_error.jsp</result>
         <result name="fail">User_register_fail.jsp</result>
      </action>
      <action name="MachineRegister"
class="lbr.cs.rcss.sdl.wui.action.MachineRegister">
         <result name="success">Func_register.jsp</result>
         <result name="fail">Machine_register_fail.jsp</result>
         <result name="error">User_login.jsp</result>
         <result name="main">User_welcome.jsp</result>
      </action>
      <action name="Welcome" class="lbr.cs.rcss.sdl.wui.action.WelcomeAction">
         <result name="m_register">Machine_register.jsp</result>
         <result name="m_control">Machine_control.jsp</result>
         <result name="m_manage">Machine_manage.jsp</result>
         <result name="f_manage">Func_register.jsp</result>
      </action>
      <action name="FuncRegister" class="lbr.cs.rcss.sdl.wui.action.FuncRegAction">
         <result name="success">Machine_register.jsp</result>
         <result name="fail">Func_register_fail.jsp</result>
         <result name="error">index.jsp</result>
         <result name="main">User_welcome.jsp</result>
      </action>
      <action name="SelectMachine" class="lbr.cs.rcss.sdl.wui.action.SelectMachine">
         <result name="error">Machine_control.jsp</result>
         <result name="generatepage">MaControl_Penal.jsp</result>
         <result name="navigation">Navigation_setting.jsp</result>
      </action>
      <action name="SetNavigation" class="lbr.cs.rcss.sdl.wui.action.SetNavigation">
         <result name="navigationset">MaControl_Panel.jsp</result>
         <result name="error">Navigation_setting.jsp</result>
      </action>
```

```
        <action name="AddMachineInfor"
class="lbr.cs.rcss.sdl.wui.action.AddMachInfor">
            <result name="oversized">Navigation_setting.jsp</result>
            <result name="fit">NonNavControl_Panel.jsp</result>
            <result name="error">Machine_control.jsp</result>
        </action>
        <action name="ControlPageUpdate"
class="lbr.cs.rcss.sdl.wui.action.ControlPageUpdate">
            <result></result>
        </action>
        <action name="CommandUpdate"
class="lbr.cs.rcss.sdl.wui.action.CommandUpdate">
            <result></result>
        </action>
        <action name="AddInformation" class="lbr.cs.rcss.sdl.wui.action.AddInformation">
            <result></result>
        </action>
        <action name="MachineDelete" class="lbr.cs.rcss.sdl.wui.action.MachineDelete">
            <result></result>
        </action>
    </package>
</struts>
```

F. Class diagram for the demonstration software

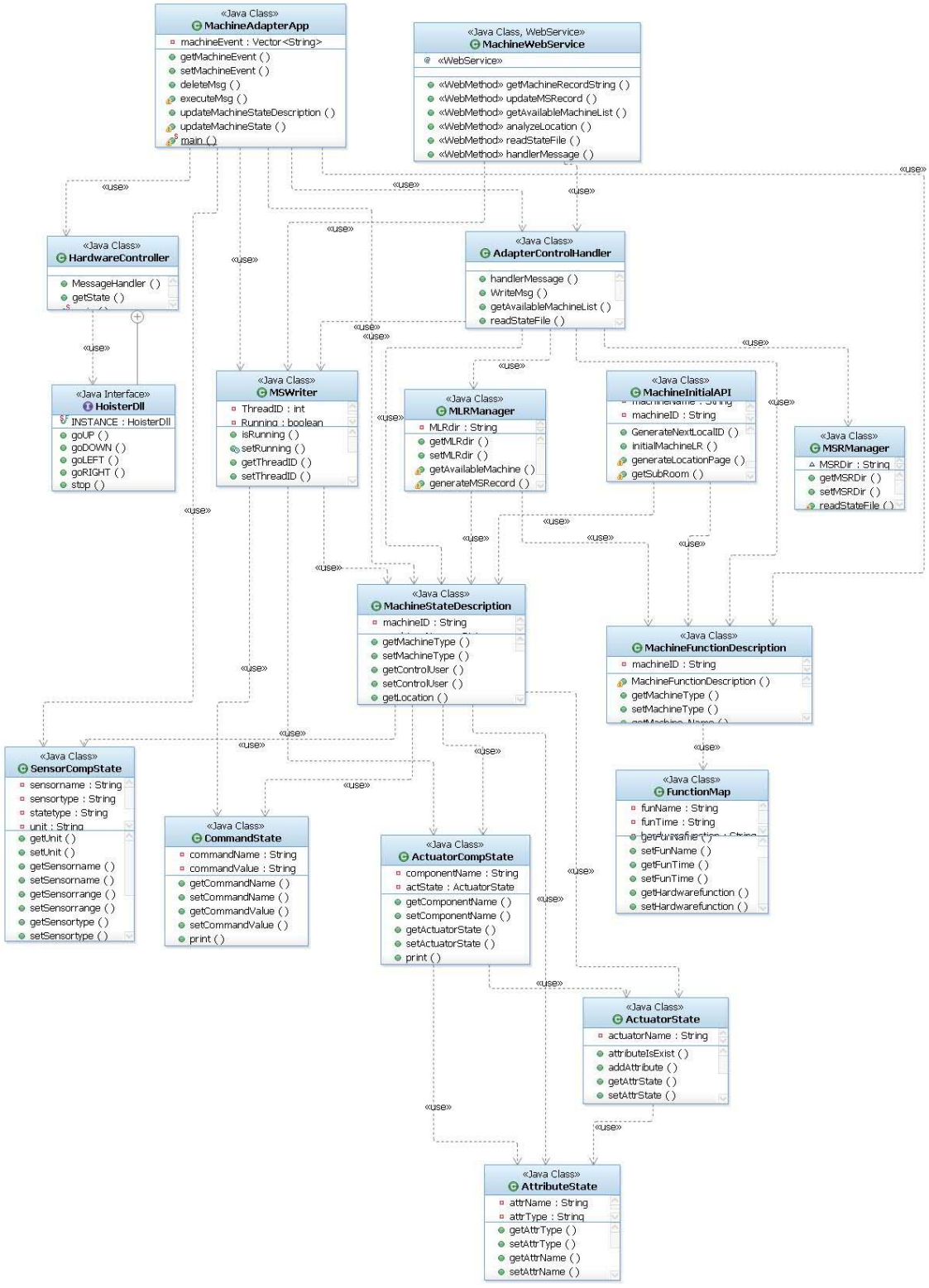


Figure F. 1 Class Diagram For Machine Client Side Project

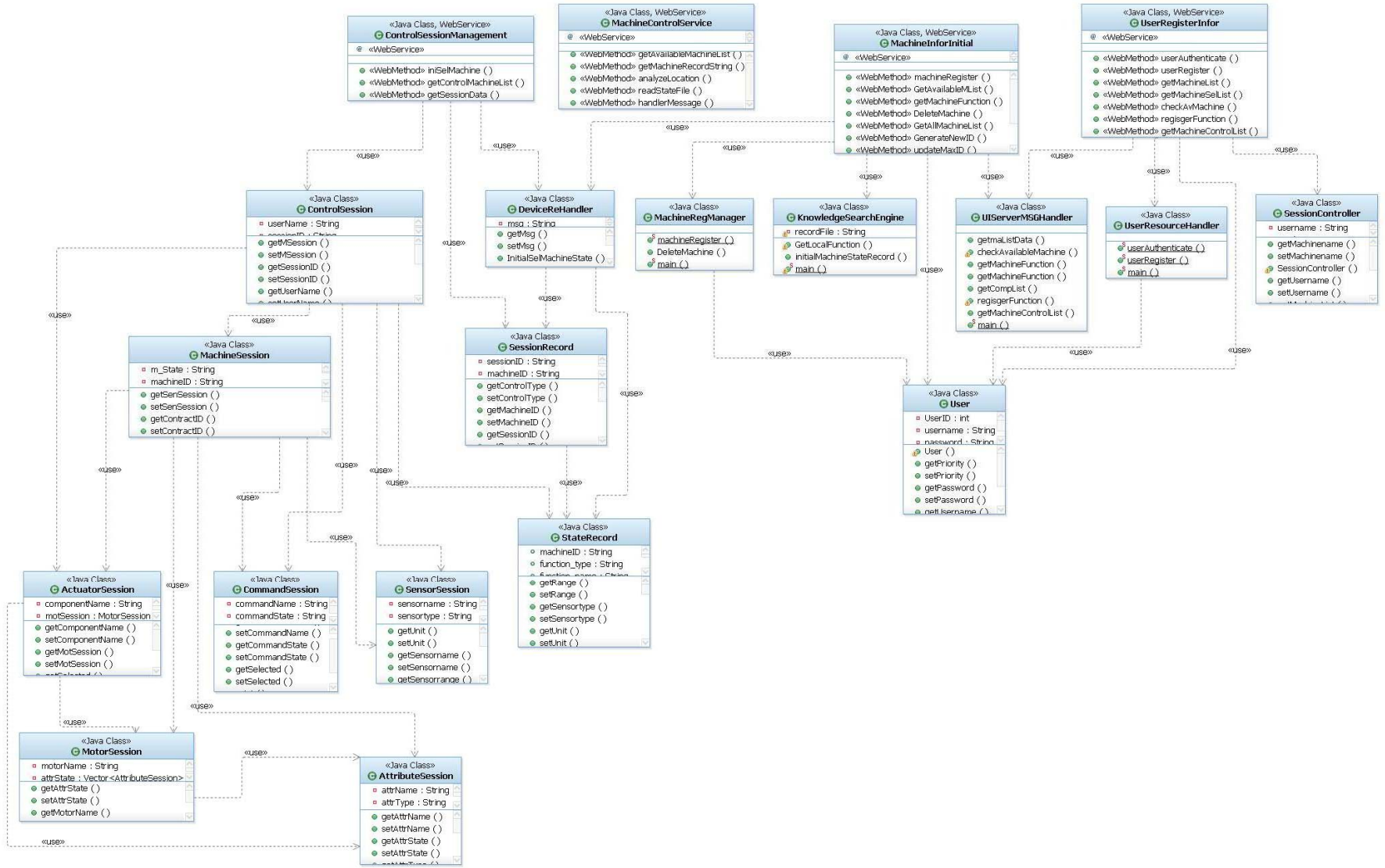


Figure F. 2 Class Diagram For Service Server Project

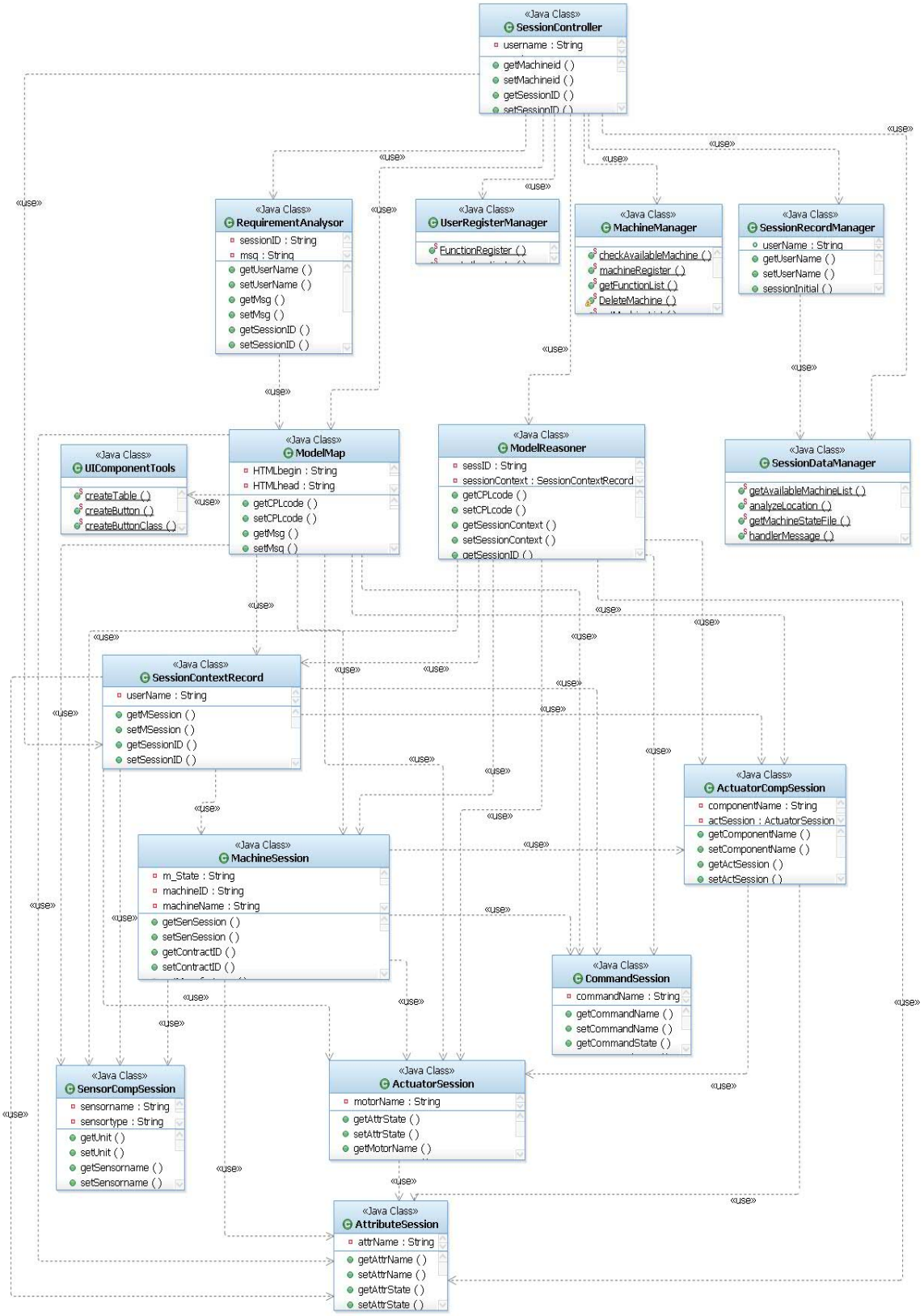


Figure F. 3 Class Diagram For Service Server Project

List of publications

Guo, X. Chung, P.W.H., 2010, Dynamic Generation of Flexible User Interface for Remote Control, 7th IEEE, IET international symposium on communication systems, networks and digital signal processing, pp. 268~271.

Guo, X. Chung, P.W.H., 2009, “Intelligent web user interface service for remote control”, 2nd IEEE International Conference on Broadband Network& Multimedia Technology, pp. 837~84.

Guo, X. Chung, P.W.H., 2008, “The Architecture Of A Web Service-Based Remote Control Service”, 10th International Conference on Information Integration and Web-based Applications & Services, pp. 555~558.

Liu, W. Guo, X. Chung, P.W.H., 2008, “The development of remote control for processing plants Proceedings”, 35th International Conference of Slovak Society of Chemical Engineering, Slovak Society of Chemical Engineering, ISBN 978-80-227-2903-1, paper 221.