THE DEVELOPMENT AND IMPLEMENTATION OF AN INTELLIGENT, SEMANTIC MACHINE CONTROL SYSTEM WITH SPECIFIC REFERENCE TO HUMAN-MACHINE INTERFACE DESIGN

JIACHUN WU

2005

THE DEVELOPMENT AND IMPLEMENTATION OF AN INTELLIGENT, SEMANTIC MACHINE CONTROL SYSTEM WITH SPECIFIC REFERENCE TO HUMAN-MACHINE INTERFACE DESIGN

BY

JIACHUN WU

A DISSERTATION PRESENTED TO THE HIGHER DEGREES COMMITTEE OF THE CAPE PENINSULA UNIVERSITY OF TECHNOLOGY IN FULFILLMENT OF THE REQUIKEMENTS FOR THE MASTER OF TECHNOLOGY: INFORMATION TECHNOLOGY DEGREE.

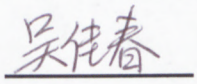
CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

2005

DECLARATION

The contents of this dissertation represent my own work, and the opinions contained herein are my own and not necessarily those of the university. All references have been accurately reported.

Name: **Jiachun Wu**

Signature: 

Date: **March 2006**

Acknowledgements

I am deeply indebted to many people who have provided help, support and encouragement. I would like to thank my supervisor Mr. Bennett Alexander for his invaluable help and unselfish support throughout the preparation of this thesis. I thank him for teaching me to be a researcher.

I would also like to thank my colleagues at the department, in particular Jiang Wu for his cooperation and fruitful discussions.

Finally I would like to express my warmest gratitude to my family and all my friends for making my life rich and joyful. I express my heartfelt thanks to my parents for their support, care, and love.

Abstract

This thesis explores the design and implementation of an intelligent semantic machine control system with specific reference to human-machine interface design. The term "intelligent" refers to machines that can execute some level of decision taking in context. The term "semantic" refers to a structured language that allows user and machine to communicate.

This study will explore all the key concepts about an intelligent semantic machine control system with human-machine interface. The key concepts to be investigated will include Artificial Intelligence, Intelligent Control, Semantics, Intelligent Machine Architecture, Human-Machine Interaction, Information systems and Graphical User Interface. The primary purpose of this study is to develop a methodology for designing a machine control system and its related human-machine interface.

The emerging discipline of "Usability Engineering" is at the core of the user-centred design and human-machine interface aspects of this project. The Usability Engineering approach to the design of complex machines focuses on developing machines that are efficient and error-free. Usability Engineering provides a methodological framework for the optimum design of human-machine interfaces by recognising - user needs, design restrictions, and other environmental constraints. The Usability Engineering also provides guidelines for integration with Object-Oriented Software Engineering (OOSE) and Unified Modelling Language (UML). The integration is based on linking OOSE models and UML with Usability Engineering tasks. OOSE is a new technology based on objects and classes. By providing first class support for the objects and classes of objects of an application domain, the object-oriented paradigm precepts offer better modelling and implementation of

Key words

Artificial Intelligence

Intelligent Control

Semantic

Usability Engineering

Object-oriented Software Engineering

Unified Modelling Language

Intelligent Machine Architecture

Machine control systems

Human-Machine Interaction

Information systems

Graphical User Interface

systems. The UML is an open method used to specify, visualize, construct, and document the artifacts of an object-oriented software system under development.

This study illustrates the design and implementation of a human-machine interface of an intelligent exercise machine as a specific practical Information Technology (IT) application. We will follow the integration of the Usability Engineering and OOSE to develop the specific application. The application of machines is composed by three major systems as an intelligent control system, an information system, and an interactive system. The intelligent control system will automatically respond and execute a task or a function of the machines immediately in terms of decision taking of machines. The important aspect of the information system is to record all users' data for customizing their future plans and retrieving the data. Human-machine interface provides an interactive dialogue style and a semantic machine between users and machines in terms of user needs on the interactive system design.

Table of Contents

List of Figures

(Cacciabue and Hollnagel, 1998). According to they claimed, the increasing use of automation has improved efficiency, safety and easiness of operations, but it has complicated the operator's situation awareness, because of the opaqueness of intelligent control of autonomous automatic system.

Cacciabue and Hollnagel also pointed out, as machines become pervasive, numerous, and even autonomous fixtures in the home and workplace, people's interactions with them will become more sophisticated and inevitable. The machine's interaction with us and other machines will develop as explicit interactions in order to manage the increasing number of entities sharing the environment to accomplish their tasks. Machines and related devices have to be designed with an understanding that people with specific tasks in mind will want to use them in a way that is seamless with respect to their everyday activities. To do this, those who design these systems need to know how to think in terms of the eventual users' tasks and how to translate that knowledge into an executable system.

Dix et al (1998) indicated: "The study of Human-computer Interaction (HCI) tends to come late. It is concerned with the development and implementation of interactive systems for human use and with the multi-disciplinary study of various issues affecting this interaction. It is concerned with the joint performance of tasks by humans and machines; the structure of human-machine communication; human capabilities to use machines; the process of specification, design and implementation of interfaces. The aim of HCI is to ensure the safety, utility, effectiveness, efficiency, accessibility and usability of such systems.

In recent years, HCI has attracted considerable attention by the academic and research communities, as well as by the Information Technology and Telecommunications (IT&T) industry. An early argument in favour of HCI studies, in many cases, exceeds 50% of the total programming effort required for the development of the entire system. Though this is generally true, today there are other more compelling reasons why HCI

is becoming critical in the emerging Information Society.

Diagrammatically, a general interaction framework is depicted as follows:

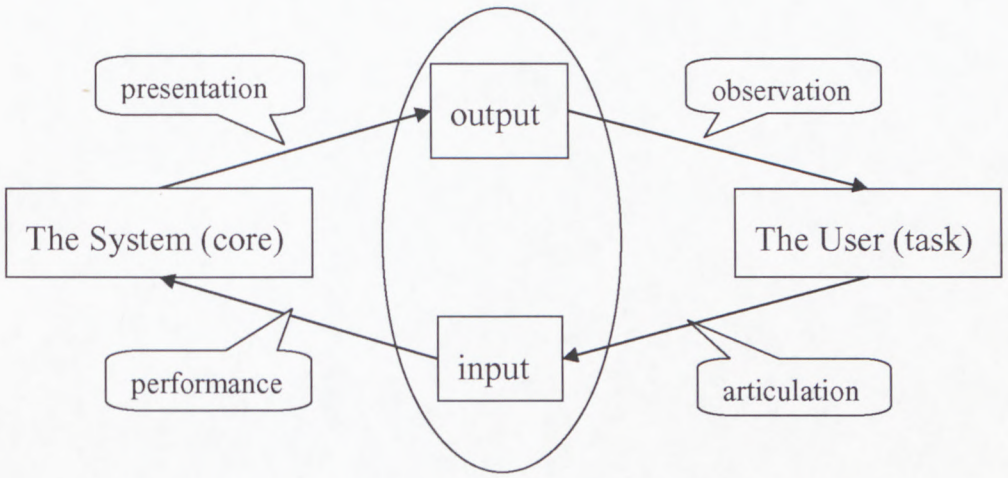


Figure 1.1: The General Interaction Framework (Dix et al., 1998)

The interaction framework attempts a more realistic description of interaction by including the system explicitly, and breaks it into four main components, as shown in Figure 1.1. The nodes represent the four major components in an interactive system – the System, the User, the Input and the Output. Each component has its own language. In addition to the User's task language and the System's core language, there are languages for both the Input and Output components to represent those separate, though possibly overlapping, components. Input and Output together form the Interface."

1.3 The Logic Model towards solution in a research environment

The logic model is an effective tool that helps to lay out the groundwork for the

evaluation of the outcome of an intervention, which in development and in action research is an important factor in determining future success.

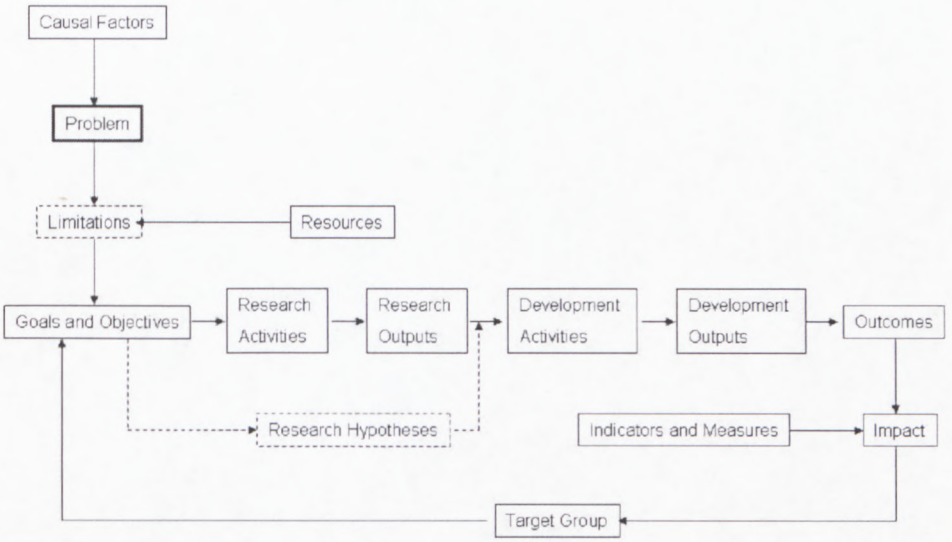


Figure 1.2: Logic Model (Bytheway, n.d.)

Figure 1.2 shows the main features of the simple logic model. There are six components of particular interest that are largely self-evident. The detailed components of the model are discussed in the section as follows:

- **Goals and objectives** are aimed or desired results; what one hopes to achieve.
- **Outputs** are what the activities produce.
- **Outcomes** are what will be expected to happen with the available outputs.
- **Activities** are what are to be done in order to achieve the goals and objectives
- **The target group** is the community or population of people (or organisations) that are the intended beneficiaries.
- **Resources** are what are needed to carry out the activities.

1.3.1 Problem Statement

The development of Graphical User Interface (GUI) is faced today with the problem of adapting conventional methods, techniques and technologies to work in evolutionary, rapid, extreme and other non-classical styles of software development. There is a need for Information Technology practitioner to investigate a time saving and cost reduced method for developing a suitable GUI integrated with a complex machine control system. As a development of a practical Information Technology application, this work is to design and implement an intelligent semantic machine control system with specific reference to human-machine interface design. So, the core problem of this work is:

How to develop a best optimal GUI for supporting and linking to an intelligent semantic machine control system?

The following aims will be investigated, in answering to the main problem of this thesis:

First of all, we need to investigate all the important concepts in this work, such as Artificial Intelligence, Intelligent Control, Semantics, Intelligent Machine Architecture, Human-Machine Interaction, Information Systems and Graphical User Interface (GUI).

Second of all, we must work out a suitable methodology and framework for the design the GUI of an intelligent semantic machine.

Last but not lest, we will develop a GUI of an intelligent exercise machine as a practical project against the methodology and framework we used.

intelligent exercise machine;

Objective 1.3: The intelligent exercise machine supplies a series of exercise modes;

Objective 1.4: The exercise machine offers several different exercise styles to satisfy users' needs;

Objective 1.5: The exercise machine affords a variety of trajectories and intensities in a motion of exercise;

Objective 1.6: Fail-Safe measures must be designed in the intelligent exercise machine;

Objective 1.7: Design a semantic machine which is a driver of information processing in order to interact between user and exercise machine easily (related to Artificial Intelligence).

Goal 2: Design a human-machine interface.

Objective 2.1: Analyze user requirements for designing a LCD screen using user-centred principles and usability engineering;

Objective 2.2: Adapt a dialog-based LCD screen;

Objective 2.3: Design a keypad for user input;

Objective 2.4: Design suitable functions depending on users' needs in LCD screen;

Objective 2.5: Collect and Analyze effective human-centred data for interface;

Objective 2.6: Apply the interaction requirements to provide a firm foundation for the actual user-centred interface design and implementation;

1.3.5 Research Activities

Activity 1.1: Design a control module for controlling a Pneumatic Actuator;

Activity 1.2: Design a monitoring module for capturing user data from the intelligent exercise machine;

Activity 1.3: The intelligent exercise machine supplies a series of exercise modes;

Activity 1.4: The exercise machine affords several different exercise styles to satisfy users' needs;

Activity 1.5: Fail-Safe measures must be designed in the intelligent exercise machine;

Activity 1.6: Design a semantic machine which is a driver of information processing in order to interact between user and exercise machine easily (related to Artificial Intelligence);

Activity 2.1: Analyze user requirements for designing a LCD screen using user-centred principles and usability engineering;

Activity 2.2: Design a dialog-based LCD screen;

Activity 2.3: Design a keypad for user input;

Activity 2.4: Design suitable functions depending on users' needs in LCD screen;

1.3.6 Research Outputs

Output 1.1: A control module for controlling a Pneumatic Actuator;

Output 1.2: A monitoring module for capturing user data from the intelligent exercise machine;

Output 1.3: The intelligent exercise machine to supply a series of exercise modes;

Output 1.4: The exercise machine to afford several different exercise styles to satisfy users' need;

Output 1.5: Fail-Safe measures properly designed in the intelligent exercise machine;

Output 1.6: A semantic machine is successfully designed in the interactive system (related to Artificial Intelligence);

Output 2.1: User interface design meets the user requirements to apply user-centred principles and usability engineering;

Output 2.2: A dialog-based LCD screen;

Output 2.3: A keypad for user input;

Output 2.4: Some suitable functions are set up in LCD screen;

1.3.7 Outcomes

Outcome 1: A user interface of an intelligent machine has been designed properly.

Outcome 2: The user interface meets architectural constraints of an interactive system.

Outcome 3: The user interface meets target user requirements.

Outcome 4: The user interface has been flexibly designed (interoperability) for future adaptability in a machine control system environment.

Outcome 5: The user interface provides adequate user support provisions.

1.3.8 Target Groups

Fitness industry

Medical insurance industry

All professional disciplines

High-achieving recent master graduates

Business companies interested in the field of user interface design

1.4 Organisation of the Thesis

This dissertation consists of five chapters. The first chapter introduces the study, establishes the field of research, and prepares basic concepts for intelligent semantic machine control system and human-machine interface design, drawn from the human-computer interaction and software engineering journals and books.

Chapter two, Literature Review, describes the recent developments in the field of intelligent machine control system and user interface design. The chapter starts with a brief historical perspective and carries on to a discussion of the useful knowledge in intelligent control and human-machine interface design. From the review we present artificial intelligence plays a very important role in field of intelligent control system, and describe the architecture of intelligent machine to clarify the design of intelligent

Chapter 2: Literature Review

In this chapter, we will investigate the key concepts for intelligent machines, such as Artificial Intelligence, Intelligent Control, Semantics, Intelligent Machine Architecture, Human-Machine Interaction, Information Systems and Graphical User Interface. Intelligent machines get information from a set of sensors and control a set of actuators. Intelligent machines incorporate the ability to react to changes in their environment, incorporate a priori knowledge and work to complete tasks. An intelligent system design is an interdisciplinary field, requiring coordination with, and assistance from, related fields, such as Artificial Intelligence, machine control system, human-machine interaction, and information systems (Meystel and Messina, 2000).

2.1 Artificial Intelligence

Russell and Norvig (1995) claimed: "humankind has given itself the scientific name homo sapiens - man the wise - because our mental capacities are so important to our everyday lives and our sense of self. The field of artificial intelligence, or AI, attempts to understand intelligent entities. Thus, one reason to study it is to learn more about ourselves. But unlike philosophy and psychology, which are also concerned with intelligence, AI strives to build intelligent entities as well as understand them. Another reason to study AI is that these constructed intelligent entities are interesting and useful in their own right. AI has produced many significant and impressive products even at an early stage in its development. Although no one can predict the future in detail, it is clear that computers with human-level intelligence would have a huge impact on our everyday lives and on the future course of civilization."

machine control system. We will indicate the reason why human-machine interface can easily communicate between user and the system via semantic information. When we mention the word "information", we must describe the information systems' role in the whole project.

Chapter three, Framework and Methodology, presents the methods that apply usability engineering and object-oriented principles for user centred design and user interface design. The chapter starts with introduction of usability and points out the important rules for designing a valuable user interface. The chapter continues presenting the Object-Oriented Methodology offering a new powerful model for designing a interactive system, and relying on the lifecycle of usability engineering and principle of user centred design.

Chapter four, Case Study, describes practical experiences of applying a proposed design process. We discuss the intelligent exercise machine as the case. Firstly, we should present the project environment for designing electro-pneumatic control component which is set up inside an intelligent exercise machine. secondly, we rely the object-oriented method and Unified Modelling Language (UML) to analysis the system requirements and task requirements of user interface design, and we design the software features and functionality, and flow logic of graphical user interface for a stand-alone exercise machine. Finally, we will test and evaluate the GUI application using a technique dialog modelling.

Chapter five, Conclusion and Future works, encapsulates the conclusions for all aspects of the research project.

2.1.1 Intelligence

A good definition for intelligence has eluded researchers in psychology, philosophy, and biology for a long time now (Buss and Hashimoto, 1996). Nevertheless there are large numbers of articles discussing "intelligent" systems of all kinds. It is considered to be too complex a concept for a neat and precise definition (Rzevski, 2003). His view is that if we call a class of systems "intelligent", we should define in what way these systems differ from the rest. He suggests that the following definition of intelligence is quite adequate for our purpose: Intelligence is the capability of a system to achieve its goals under conditions of uncertainty.

To exhibit intelligent behaviour a system must have access to the knowledge on the domain in which it operates, and to act upon this knowledge in response to, or in anticipation of, external inputs (rather than to passively react to input data in a pre-programmed manner). In most cases, to "act upon knowledge" means selecting a pattern of behaviours, which takes advantage, or neutralises undesirable consequences, of unpredictable events. It is important to note that when an intelligent system meets a new problem it must find a solution by the trial-and-error method, just like human beings do (Holland, 1998).

The study of intelligence is one of the oldest disciplines (Russell and Norvig, 1995). They say: "For over 2000 years, philosophers have tried to understand how seeing, learning, remembering, and reasoning could, or should, be done. The advent of usable computers in the early 1950s turned the learned but armchair speculation concerning these mental faculties into a real experimental and theoretical discipline. Many felt that the new "Electronic Super-Brains" had unlimited potential for intelligence. "Faster than Einstein" was a typical headline. But as well as providing a vehicle for creating artificially intelligent entities, the computer provides a tool for testing theories of intelligence, and many theories failed to withstand the test -- a case of "out of the armchair, into the fire." AI has turned out to be more difficult than many at first

imagined and modern ideas are much richer, more subtle, and more interesting as a result."

2.1.2 An Attempt Definition for AI

Luger (2002) points out: "Artificial intelligence (AI) may be defined as the branch of computer science that is concerned with the automation of intelligent behaviour. This definition is particularly appropriate to this work in that it emphasizes our conviction that AI is a part of computer science and, as such, must be based on sound theoretical and applied principles of that field. These principles include the data structures used in knowledge representation, the algorithms needed to apply that knowledge, and the languages and programming techniques used in their implementation".

Russell and Norvig (1995) presented some definitions of artificial intelligence according to eight textbooks (shown in Figure 2.1). These definitions vary along two main dimensions. The ones on top are concerned with thought processes and reasoning, whereas the ones at the bottom address behaviour. Also, the definitions on the left measure success in terms of human performance, whereas the ones on the right measure against an ideal concept of intelligence, which we will call rationality. A system is rational if it does the right thing. This gives us four possible goals to pursue in artificial intelligence.

Historically, all four approaches have been followed. As one might expect, a tension exists between approaches centred around humans and approaches centred around rationality (Kahneman et al., 1982). A human-centred approach must be an empirical science, involving hypothesis and experimental confirmation. A rationalist approach involves a combination of mathematics and engineering (Russell and Norvig, 1995).

"The exciting new effort to make computers think . . . *machines with minds,* in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making,problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Some definitions of AI. They are organized into four categories:

Systems that think like humans.	Systems that think rationally.
Systems that act like humans.	Systems that act rationally.

Figure 2.1: definitions of AI (Russell and Norvig, 1995)

Luger (2002) also said: "Artificial intelligence has always been more concerned with expanding the capabilities of computer science than with defining its limits. Keeping this exploration grounded in sound theoretical principles is one of the challenges facing AI researchers in general.

Because of its scope and ambition, artificial intelligence defies simple definition. For the time being, we will simply define it as the collection of problems and methodologies studied by artificial intelligence researchers. This definition may seem silly and meaningless, but it makes an important point: artificial intelligence, like every science, is a human endeavour, and perhaps, is best understood in that context.

There are reasons that any science, AI included, concerns itself with a certain set of problems and develops a particular body of techniques for approaching these problems. A short history of artificial intelligence and the people and assumptions that have shaped it will explain why a certain set of questions has come to dominate the field."

2.1.3 Overview of AI application Areas

In this section, we will fully draw on Luger's knowledge (2002) to explain how we should use subdisciplines of artificial intelligence in this project. We appreciated Luger's contribution in this field. Like most sciences, AI is decomposed into a number of subdisciplines that, while sharing an essential approach to problem solving, have concerned themselves with different applications. Luger outlines several of these major application areas and their contributions to help us to design an intelligent semantic machine as a whole.

2.1.3.1 Natural language understanding and Semantic Modelling (Luger, 2002)

One of the long-standing goals of artificial intelligence is the creation of programs that are capable of understanding and generating human language. Not only does the ability to use and understand natural language seem to be a fundamental aspect of human intelligence, but also its successful automation would have an incredible impact on the usability and effectiveness of computers themselves. Much effort has been put into writing programs that understand natural language. Although these programs have achieved success within restricted contexts, systems that can use natural language with the flexibility and generality that characterize human speech are beyond current methodologies.

Understanding natural language involves much more than parsing sentences into their individual parts of speech and looking those words up in a dictionary. Real understanding depends on extensive background knowledge about the domain of discourse and the idioms used in that domain as well as an ability to apply general contextual knowledge to resolve the omissions and ambiguities that are a normal part of human speech.

The task of collecting and organizing this background knowledge in such a way that it

2.1.3.2 Modelling Human Performance (Luger, 2002)

Although much of the above discussion uses human intelligence as a reference point in considering artificial intelligence, it does not follow that programs should pattern themselves after the organization of the human mind. Indeed, many AI programs are engineered to solve some useful problem without regard for their knowledge from human experts, and do not really attempt to simulate human internal problem solving processes.

Human performance modelling, in addition to providing AI with much of its basic methodology, has proved to be a powerful tool for formulating and testing theories of human cognition. The problem-solving methodologies developed by computer scientists have given psychologists a new metaphor for exploring the human mind. Rather than casting theories of cognition in the vague language used in early research or abandoning the problem of describing the inner workings of the human mind entirely, many psychologists have adopted the language and theory of computer science to formulate models of human intelligence, but also computer implementations of these theories offer psychologists an opportunity to empirically test, critique, and refine their ideas (Luger, 1994).

2.1.3.3 Planning and Robotics (Luger, 2002)

Research in planning began as an effort to design robots that could perform their tasks with some degree of flexibility and responsiveness to the outside world. Briefly, planning assumes a robot that is capable of performing certain automatic actions. It attempts to find a sequence of those actions that will accomplish some higher-level task, such as moving across an obstacle-filled room.

Planning is a difficult problem for a number of reasons, not the least of which is the size of the space of possible sequences of moves. Even an extremely simple robot is capable of generating a vast number of potential moves sequences. Imagine, for

example, a robot that can move forward, backward, right, or left, and consider how many different ways that robot can possibly move around a room. Assume also that there are obstacles in the room and that the robot must select a path that moves around them in some efficient fashion. Writing a program that can intelligently discover the best path under these circumstances, without being overwhelmed by the huge number of possibilities, requires sophisticated techniques for representing spatial knowledge and controlling search through possible environments.

One method that human beings use in planning is hierarchical problem decomposition. If you are planning a trip to London, you will generally treat the problems of arranging a flight, getting to the airport, making airline connections, and finding ground transportation in London separately, even though they are all part of a bigger overall plan. Each of these may be further decomposed into smaller subproblems such as finding a map of the city, negotiating the subway system, and finding a decent pub. Not only does this approach effectively restrict the size of the space that must be searched, but also allows saving of frequently used subplans for future use.

While humans plan effortlessly, creating a computer program that can do the same is a difficult challenge. A seemingly simple task such as breaking a problem into independent subproblems actually requires sophisticated heuristics and extensive knowledge about the planning domain. Determining what subplans should be saved and how they may generalized for future use is an equally difficult problem.

A robot that blindly performs a sequence of actions without responding to changes in its environment or being able to detect and correct errors in its own plan could hardly be considered intelligent. Often, a robot will have to formulate a plan based on incomplete information and correct its behaviour as it executes the plan. A robot may not have adequate sensors to locate all obstacles in the way of a projected path. Such a robot must begin moving through the room based on what it has "perceived" and correct its path as other obstacles are detected. Organizing plans in a fashion that

allows response to changing environmental conditions is a major problem for planning (Lewis and Luger, 2000).

Finally, robotics was one of the research areas in AI that produced many of the insights supporting agent-oriented problem solving. Frustrated by both the complexities of maintaining the large representational space as well as the design of adequate search algorithms for traditional planning, researchers, including Agre and Chapman (1987) and Brooks (1991), restated the larger problem in terms of the interaction of multiple semi-autonomous agents. Each agent was responsible for its own portion of the problem task and through their coordination the larger solution would emerge.

Planning research now extends well beyond the domains of robotics, to include the coordination of any complex set of tasks and goals. Modern planners are applied to agents (Nilsson, 1994) as well as to the control of particle beam accelerators (Klein et al., 1999, 2000).

2.1.3.4 Machine Learning (Luger, 2002)

Learning has remained a challenging area for AI. The importance of learning, however, is beyond question, particularly as this ability is one of the most important components of intelligent behaviour. An expert system may perform extensive and costly computations to solve a problem. Unlike a human being, however, if it is given the same or a similar problem a second time, it usually does not remember the solution. It performs the same sequence of computations again. This is true the second, third, fourth, and every time it solves that problem – hardly the behaviour of an intelligent problem solver.

Most expert systems are hindered by the inflexibility of their problem-solving strategies and the difficulty of modifying large amounts of code. The obvious solution

may be applied to language comprehension forms the major problem in automating natural language understanding. Responding to this need, researchers have developed many of the techniques for structuring semantic meaning used throughout artificial intelligence.

Because of the tremendous amounts of knowledge required for understanding natural language, most work is done in well-understood, specialized problem areas. One of the earliest programs to exploit this "micro world" methodology was Winograd's SHRDLU, a natural language system that could "converse" about a simple configuration of blocks of different shapes and colours (Winograd, 1973). SHRDLU could answer queries such as "What colour block is on the blue cube?" as well as plan actions such as "move the red pyramid onto the green brick." Problems of this sort, involving the description and manipulation of simple arrangements of blocks, have appeared with surprising frequency in AI research and are known as "block world" problems.

In spite of SHRDLU's success in conversing about arrangements of blocks, its methods did not generalize from the blocks world. The representational techniques used in the program were too simple to capture the semantic organization of richer and more complex domains in a useful way. Much of the current work in natural language understanding is devoted to finding representational formalisms that are general enough to be used in a wide range of applications yet adapt themselves well to the specific structure of a given domain. A number of different techniques (most of which are extensions or modifications of semantic networks) are explored for this purpose and used in the development of programs that can understand natural language in constrained but interesting knowledge domains. Finally, in current research (Marcus 1980, Manning and Schutze 1999, Jurafsky and Martin 2000) stochastic models, describing how sets of words "co-occur" in language use, are employed to characterize both syntax and semantics. Full computational understanding of language, however, remains beyond the current state of the art.

to these problems is for programs to learn on their own, either from experience, analogy, and examples or by being "told" what to do.

Although learning is a difficult area, there are several programs that suggest that it is not impossible. One striking program is AM, the Automated Mathematician, designed to discover mathematical laws (Lenat, 1977, 1982). Initially given the concepts and axioms of set theory, AM was able to induce such important mathematical concepts as cardinality, integer arithmetic, and many of the results of number theory. AM conjectured new theorems by modifying its current knowledge base and used heuristics to pursue the "best" of a number of possible alternative theorems. More recently, Cotton et al. (2000) designed a program that automatically invents "interesting" integer sequences.

Early influential work includes Winston's research on the induction of structural concepts such as "arch" from a set of examples in the blocks world (Winston, 1975). The ID3 algorithm has proved successful in learning general patterns from examples (Quinlan, 1986). Meta-DENDRAL learns rules for interpreting mass spectrographic data in organic chemistry from examples of data on compounds of known structure. Teiresias, an intelligent "front end" for expert systems, converts high-level advice into new rules for its knowledge base (Davis and Lenat, 1982). Hacker devised plans for performing blocks world manipulations through an iterative process of devising a plan, testing it, and correcting any flaws discovered in the candidate plan (Sussman, 1975). Work in explanation-based learning has shown the effectiveness of prior knowledge in learning (Mitchell et al. 1986, De Jong and Mooney 1986).

The success of machine learning programs suggests the existence of a set of general learning principles that will allow the construction of programs with the ability to learn in realistic domains.

Luger (2002) has used these resources to attempt to define artificial intelligence

through its major areas of research and application. This survey reveals a young and promising field of study whose primary concern is finding an effective way to understand and apply intelligent problem solving, planning, and communication skill to a wide range of practical problems, such as intelligent machines.

2.2 The Challenge of Intelligent Systems

This section we will cite Meystel and Messina's (2000) work. We use their work with my respect and appreciate them offering up their knowledge. They analyze the meaning of intelligence within the domain of intelligent control. Because of historical reasons, there are varying and diverse definitions of what constitutes an intelligent control system. Due to the multidisciplinary nature of intelligent control, there are no good formal measures for what is an intelligent controller or for its performance. They argue that there are characteristics which can be used to categorize a controller as being intelligent. Furthermore, the measure of how intelligent a system is should take into account a holistic view of the system's operations and design requirements. Beginning with the definition of intelligence characteristics and performance measures, we can move towards a science of intelligent systems for control.

"Intelligent Control" is a commonly used term, applied to an amorphous set of tools, techniques, and approaches to implementing control systems that have capabilities beyond those attainable through classical control. There is a well-developed and deep body of work entailing a theory of control. This is not the case for intelligent control, which is more of an ad hoc approach to constructing systems that generally contain some internal kernel of the more classical control, for which there is a theory.

2.2.1 What is Intelligent Control?

In the late seventies, Fu and Saridis introduced the combination of the words

Intelligent Control. In the eighties, it was brought to widespread practice and better understanding, as shown in papers by Saridis (1985), Meystel (1985), and Albus et al. (1985), who jointly spearheaded this movement. The National Science Foundation (NSF) funded the first IEEE Workshop on Intelligent Control (1986, Troy, NY), and this event continued as the annual IEEE International Symposia on Intelligent Control and multiple workshops on Autonomous Intelligent Control Systems. The nascent science of Intelligent Control was actively supported by participants from multiple research groups, including those belonging to the school of Soft Computing, established by Zadeh and focusing on fuzzy systems, neural networks, computing with words, and other techniques.

As far as methodology is concerned, the area was then and is now far from being stable. The main obstacle to stability is due to the multidisciplinarity of the intelligent control field. At the first Symposium on Intelligent Control in 1985, it was proclaimed a theoretical domain, in which control theory, AI, and operations research intersected (Figure 2.2 from Saridis 1985).

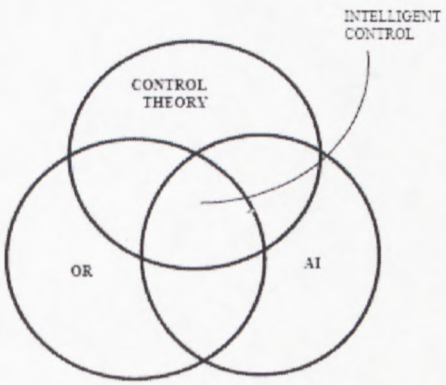


Figure 2.2: Intelligent Control as of 1985 (Saridis, 1985)

Problems arose due to the reluctance of members from the different constituent disciplines to venture too far into the others' domains. For instance, specialists in control theory and automatic control tried to avoid getting involved in issues of cognitive science, psychology, biology, and ecology associated with the notion of

intelligence. Specialists in the Artificial Intelligence domain (itself comprised of various, rather diverse, sub-specialities), traditionally abstained from getting involved in the mathematical control issues related to system dynamics and felt uncomfortable with domains of research that were beyond their discipline. The essence of the conflict lies in the term "intelligent," which has myriad implications. It is still impossible today to distinguish an intelligent controller from one that is not intelligent. It is also unclear whether a specialist in control theory can be considered a specialist in intelligent control, or vice versa.

2.2.2 The Term "Intelligent Control" and Its Usage

As discussed above, the problem of Intelligent Control has turned out to be an intrinsically interdisciplinary one. This is why, starting in 1995, the IEEE and the National Institute of Standards and Technology (NIST), in collaboration with other agencies, organized a series of conferences intended to expand the topic of Intelligent Control towards the more consistent, yet more difficult theme of Intelligent Systems.

Fu linked a concept of intelligent control with the following features that were traditionally out of the scope of specialists in conventional control theory: decision making, image recognition, adaptation to the uncertain media, self organization, planning, and more (Fu, 1971). Saridis (1977) gave the definition of Intelligent Control as a statement of expected functions: "Intelligent Control would replace a human mind in making decisions, planning control strategies, and learning new functions by training and performing other intelligence functions whenever the environment doesn't allow or doesn't justify the presence of a human operator. Such systems can solve problems, identify objects, or plan a strategy for a complicated function of a system with intelligent functions such as simultaneous utilization of a memory, learning, or multilevel decision making in response to fuzzy or qualitative comments" (Saridis, 1977).

In these excerpts, the concept of goal is not mentioned, because goal is a part of a more general statement, which includes intelligent control by necessity: "control of a process employs driving the process to effectively attain a prespecified goal" (Saridis, 1977). A popular and all-encompassing definition is by Albus (1991): "Intelligence will be defined as an ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioural subgoals that support the system's ultimate goal."

Pang describes an intelligent controller as a controller that is utilized for shaping the behaviour of an intelligent system (Pang, 1991). The distinct properties of this controller are to provide the following features:

- It should "know" what actions to take and when to perform them
- It should reconcile the desirable and feasible actions
- It should vary the high resolution details of control heuristics
- The acquired control heuristics should be the most suitable ones and they should change dynamically
- It should be capable of integrating multiple control heuristics
- It should dynamically plan the strategic sequence of actions
- It should be able to reason between domain and control actions. In other words, it should be able to use at least two levels of resolution simultaneously: the level of the domain actions and the domain of the control actions.

In their foreword White and Sofge (1992) wrote, "To us, 'intelligent control' should involve both intelligence and control theory. It should be based on a serious attempt to understand and replicate the phenomena that we have always called 'intelligence' – i.e., the generalized, flexible and adaptive kinds of capability that we see in the human brain."

2.2.3 Characteristic of Intelligent Control

Some definitions of intelligent control would limit it to those systems that rely on soft computing techniques, such as fuzzy logic, neural networks, and genetic algorithms. Another perspective is to analyze intelligent control systems with respect to their characteristics, which were inherent in the quotes of the previous section.

We begin by looking at control laws, which are the heart of a control system, whether it is intelligent or not. An analysis of control laws indicates that they are subservient to the control system's goal of reducing the deviation from a pre-specified trajectory and/or the final state (Maybeck, 1979) and (Kuipers and Astrom, 1994). It is more typical in the present paradigm of automatic control to consider control laws to be part of a broader "control strategy" for the overall system (Song and Mitchell, 1993) and (Chiacchio et al., 1993). The assignment for the control law – to keep a certain variable of interest within some bounds around a reference trajectory – comes as the result of some external intelligence operating beyond the limits of the intelligence of the particular control law. Intelligent controllers tend to incorporate these assignments (and their genesis) into the function of the controller. This difference is a fundamental one.

In the fuzzy logic controllers, such as those described in (Liu and Lewis, 1993) and (Ma et al., 1998), the fundamental transformations move the set of input information from the high resolution domain into the low resolution domain by using the tool of "fuzzification." Fuzzification plays the role of a generalization procedure, because it searches for adjacency among information units, focuses attention, and groups. Fuzzy logic controllers contain control mappings only for generalized information at the lower resolution. When the required control action is found, this lower resolution recommendation is instantiated or moved to the domain of high resolution by the process of "defuzzification." Hence, fuzzy control systems are multi-resolutional, and they move between resolutions for the purposes of achieving the required control

performance.

Neural networks are a computation device for generalizing in a vicinity (spatial or temporal). They are a natural tool for moving information from higher to lower levels of resolution. Therefore, neural networks, such as described in (Hesselroth et al., 1994), are multiresolutional systems.

Expert System based controllers make a judgement concerning generalizations and rules to be applied at the lower resolution (Ling, 1993). They too presume a multi-level, multi-resolution framework within which they operate.

Hybrid logic control systems are multi-level control systems in which lower levels of resolution are formulated in terms of logic based controllers, while the higher resolution levels are analytical controllers. Again, there is at minimum a dual-level architecture of control, with differing resolutions.

Behaviour-based controllers employ a concept of superposition of the activities of multiple controllers working simultaneously, each providing for a separate type of behaviour. A hierarchical structure of some sort generally underlies implementation. A low resolution level may select which behaviour controller is invoked at which time, or it may arbitrate between the different proposed behaviours. Each individual behaviour generation controller generates sub-goals for itself.

Although the previous examples are not an exhaustive analysis of techniques and approaches for intelligent controllers, they serve to illustrate that certain generalizations can be made about the essence of what distinguishes a controller that is intelligent from one that isn't. A meta level of control, which functions at a lower level of resolution is necessary to guide the underlying control system and expand its envelope of functioning.

2.2.4 Beyond Intelligent Control to Intelligent Systems

Saridis has defined the intelligent control problem as "intelligent control is postulated as the mathematical problem of finding the right sequence of internal decisions and controls for a system structured according to the principle of increasing intelligence with decreasing precision such that it minimizes its total entropy" (Musto and Saridis, 1998).

If intelligent control is responsible for finding the right decisions and controlling the system to enact those decisions, then there must be complementary supporting functions which provide models of the world that the system is functioning in. There must be perception and world modelling processes that occur alongside with the control function (Meystel and Messina, 2000). "Machine intelligence is the process of analyzing, organizing, and converting data into knowledge, where (machine) knowledge is defined to be the structured information acquired and applied to remove ignorance and uncertainty about a specific task pertaining to the intelligent machine" (Antasklis, 1994). A procedural characterization of an intelligent system is given by "intelligence is the property of the system that emerges when the procedures of focusing attention, combinatorial search, and generalization are applied to the input information in order to produce the output" (Antasklis, 1994).

2.3 Software Architecture for Robotics/Machine Control

This section describes the software architecture for a robotics/machine control system. It includes the knowledge of Electronic Engineering which plays an important role in the design of control system. We will adequately quote Pack's (1998) work to present the knowledge of robotics/machine control, and we thank his work to give me an instruction how we can design a software system for machine control as a whole.

The Webster dictionary defines a robot as: An automatic apparatus or device that performs functions ordinarily ascribed to humans or operates with what appears to be almost human intelligence. The Robotics Institute of America defines a robot as follows: A robot is a reprogrammable multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks.

Although it is practical that intelligent machines ultimately be measured by their behaviour (McFarland and Bosser, 1993), such machines are not black boxes. It is of primary importance to the development and implementation of intelligent machines, like service robots, that there exists a carefully designed architecture to guide development of the software system that utilizes the robot's computational resources to connect sensors and actuators.

There is currently no accepted software architecture for general robot control or even agreement on what is desirable for such an architecture (Pack, 1998). However, many disparate approaches are available. This presents the opportunity to take the best features of several design methods and philosophies and combine them into a new and potentially powerful approach to the design of robotic control systems. The architecture design problem then becomes how to decompose the system and to provide mechanisms for software integration that support intelligent behaviour, help to manage the complexity of the design process, and support software reuse. One of Brooks's observations about robotics guides the selection of architectural features: "The complex behaviour of a system emerges from the intensive interactions of its internal structure with the physics of the outside world" (Brooks, 1986). The software that support such complex behaviour on a robot quickly becomes complex as tasks are added, requirements for robustness are increased, and constraint on resources are imposed by the physical design of the robot (Pack, 1998).

The following sections will describe the role of software architecture in the design of software systems for robots and then a review of software systems and architectures from the robotics literature will follow, ending in a summary of architectural issues and features from the literature.

2.3.1 Software Architecture

The type of computer hardware and the name of the operating system do not constitute an architecture for the software of an intelligent machine. Arkin (1995) points out that an architecture is a set of organizing principles and basic components that form the basis for the system. By describing the fundamental components and their interactions, an intelligent control system is decomposed into more primitive abstractions that serve as models for the physical resources, skills, behaviours, and tasks that are developed in the control system. This abstraction is the set of building blocks and interactions with which to construct the software system for intelligent robots. These building blocks, their interactions, and the principles behind them constitute a system architecture (Pack, 1998).

Thus, a software architecture for intelligent robot control is a set of organizing principles and fundamental components that help designers manage the complexity of building robot control software that supports intelligent action. A good architecture also serves as a model of the robot's capabilities tasks and resources and provides an abstraction of the intelligent machine for developers of the system. Given the above discussion, the primary functions of software architecture for intelligent control are as follow (drawn from ideas in Garlan and Shaw 1994):

- The architecture must model the behaviours, skills, resources, and tasks of the robot with more fundamental elements that represent the building blocks used to synthesize an intelligent control system. This decomposition helps manage the complexity of system development.

- The architecture must provide a set of mechanisms and connectors for integrating these fundamental elements into an overall system so that the desired intelligent behaviour can be realized.

As Shaw et al. (1995) put it, an architecture is "components and connectors." The specification of an architecture is a description of components and the connections between components. A good architecture can provide a set of abstractions that are easy to use and understand.

It is also desirable to reuse the components of a software system to build larger systems, adding new or different components as needed. We would also like to incorporate new types of connection between elements/connectors as the system grows, so that the overall system is extensible both with new components and new connectors. This requires that the architecture support a strong concept of subsystem and connection so that the design of reusable subsystems is encouraged.

2.3.2 System Complexity and Decomposition

One major aspect of developing a software system for an intelligent machine, such as a service robot, is that such systems immediately present a monumental software engineering project for any group attempting to develop a system of appreciable size. This problem is compounded by the fact that most task and problem descriptions are in a proscriptive format, while software provides only a prescriptive interface to the system as described by Stewart (1995). This is sometimes called "impedance mismatch" between specification and design in software system (Booch, 1994). The software system can become quite complex, and this complexity grows with the level of complexity of the tasks that are desired. Several authors have commented on system complexity; below is a collection of comments that reflect directly on system architecture problems (collected by Booch 1994):

- "Frequently complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached. (Courtois, 1985)"
- "The choice of what components in a system are primitive is relatively arbitrary and is largely up to the discretion of the observer of the system. (Simon, 1982)"
- "Intra-component linkages are generally stronger than inter-component linkages. This fact has the effect of separating the high-frequency dynamics of the components- involving the internal structure of the components from the low-frequency dynamics-involving interaction among components. (Simon, 1982)"
- "Hierarchic systems are usually composed of only a few different kinds of subsystems in various combination and arrangements."
- "A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a working simple system. (Gall, 1986)"

To address the problem of system complexity, an architecture must provide models for system decomposition and integration. A broad range of philosophies guides which methods are in use. Knowledge-based approaches decompose the system into pieces of knowledge and reasoning mechanisms. Behaviour-based approaches decompose the system into directly interacting behaviours. Hybrid systems may have both knowledge-based and behaviour-based elements. The types of connections that are permitted in the software define how the system grows when new elements are added. Some architectures are essentially fixed and only "data" or "knowledge" is added, while other architectures change structure to accommodate new elements and functions (Pack, 1998).

2.3.3 Integration and Arbitration

Integration refers to the definition of connections between components, and although there are many types of connections, the most complex types of connection employ some kind of arbitration mechanism to control aspects of the interaction (Pack, 1998). Typical software integration approaches employ an interface connection strategy, but more advanced architectures use a strategy to insure the interactions are correct and protocols are followed. Intelligent machines need architectures that support a further type of integration that includes arbitration as well as an interconnection strategy. Some robot control architectures are designed to explore a single arbitration mechanism. The use of an arbitration mechanism helps achieve high-level integration of system components. The most useful architecture would combine an advanced interface connection strategy with explicit allowances for arbitration. This implies that the "connectors" of the system can become a complex part of the architecture as described by Pack.

Viewed as a whole, the intelligent control software operates by selecting actions for the robot actuators to achieve tasks and goals. Maes describes this as the action selection problem (Maes, 1993) for intelligent agents. Identifying the action selection problem places the focus on building software that provides resources and mechanisms for action selection. The connector may be used in any one approach and may vary in its properties, but action selection, or action arbitration, is the bottom line for intelligent activity. Given a set of inputs, arbitration is the bottom line for intelligent activity. Given a set of inputs, arbitration uses some properties of those inputs to produce an effective output. Selection of an input is one class of mechanism for arbitration, while combination of inputs is a different class. Some mechanism is also needed to provide a set of alternative actions to feed into this action selection process. This is where other types of connectors are used. Arbitration is not limited to computing physical outputs of a system. It can be applied at various levels within the software to manage resources for action selection (sensor action arbitration), allocate

resources to goals (task/goal arbitration), and control actuators (motor action arbitration).

Figure 2.3 shows a generalized picture of a software system for an intelligent machine. The boxes in the diagram are intended to describe elements of the system at a conceptual level and do not indicate a particular software structure per se (Pack, 1998). The idea of the figure is to show that some elements of the software (Sensor Resources and Actuator Resources) represent software access to the transducers of the robot. Other elements (Context Nodes) are involved in building varying levels of abstraction of context or sensing for use by closed loop processes (Skill/Behaviour Nodes), while other elements of the software are involved in sequencing or goal directed control (Task Nodes). These high-level, goal-directed elements of the software rely on the abstractions provided by lower level context components. It is also possible that some elements of the software are involved in modelling the environment and capturing the current context in a form that is useful for action. Many reviewed architectures provide some form of these elements, but few architectures investigate building the functionality of each of these elements from an object-oriented or agent-based perspective.

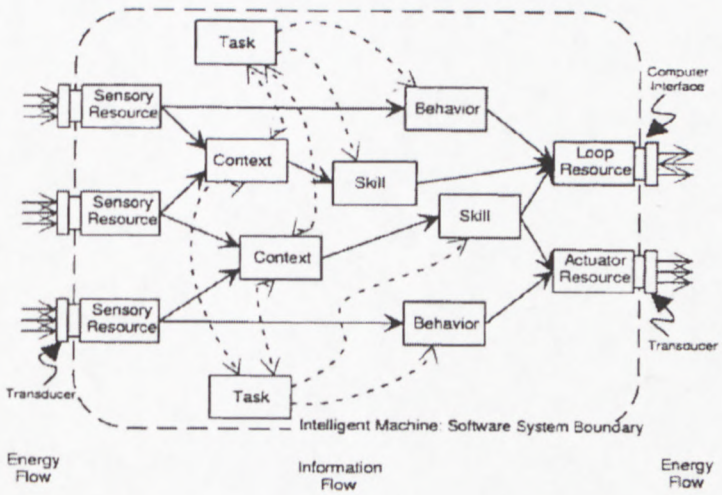


Figure 2.3: General Concept of Software System (Pack, 1998)

Arbitration mechanisms enable a software system to handle multiple connections between components. It is natural for multiple paths from sensors to actuators to come together and combine various sources of influence as seen in Figure 2.3. These sources then spread out again and influence various actuator systems of the intelligent machine. Knowledge and context encoded in the system will affect operation of these influences. At each junction there is some arbitration mechanism that is responsible for combining input flow and distributing output flow for the component. In the sensory processing for an intelligent service robot, action selection relates to sensor fusion. For example, sensor fusion combines sensor data into logical sensors, much like those developed by Luo and Kay (1989), to be used by other modules. Intelligent motion control can use arbitration to combine influences on motion (goal points, obstacles) to yield an overall path that simultaneously meets several goals or constraints for the robot, similar to motor schema used by Arkin (1989). For planning and sequencing, an arbitration mechanism may be used to predict or activate a sequence of operations as shown by Bagchi et al. (1996).

The action selection or arbitration mechanism is pervasive in the design of intelligent control software, and many reviewed architectures focus on the development of a single arbitration mechanism that serves as the connector for combining components in the software system. Much of the flexibility and extensibility of an architecture comes from the types of connections between components that it supports. Rigid architectures typically allow a single kind of connection, while more flexible architectures combine several arbitration mechanisms at different points in the system. There is a design tradeoff between the complexity of connectivity, data, and components of a system. Some systems have trivial components and data and encode behaviour purely in the connectivity between components (like neural networks). Other systems have simple connectivity, but complex data and components (like traditional knowledge-based systems). Neither of these extremes adequately matches the granularity available in modern, Parallel, distributed computer systems. The architecture developed in this dissertation supports an open set of connections that can

grow as the rest of the software system is developed. The conceptual modules shown in Figure 2.3 are implemented by a wide variety of software structures.

2.3.4 Robot Software Architecture

Pack (1998) indicated: "Intelligent robots need an explicit software architecture to manage the complexity of developing the many modules and components required for intelligent behaviour. Robotics researchers need an approach that allows them to apply new techniques, algorithms, and mechanisms to build integrated systems. Providing the resources and mechanisms that generate intelligent behaviour on a robot can be viewed as a complex system integration problem. One way to manage the complexity of development is to adopt a software architecture that specifies possible components and connections. The architecture specifies a system decomposition that divides the system into simpler elements and provides integration mechanisms for combining the simpler elements into the whole control system. Inherent in most architectures is a set of design decisions about the location of complexity in the overall system. This complexity may arise in the knowledge stored in an intelligent machine or may arise in interconnections between simpler components that comprise the intelligent machine. What follow is a review of architectures, each of which provides its own solutions to the architecture problem, and a discussion of various architectural features. The architectures are loosely classified based on what type of component was selected as the fundamental element of the system."

2.3.5 Knowledge-Based Architecture

Both the State Operator and Result architecture by Laird and Newell (1987 and 1991) and the Adaptive Control of Thought approach of Anderson (1983) are examples of

knowledge-based approaches to the intelligent system problem. Some authors call them deliberative because the declarative representation used requires application or "deliberation" in order to generate actions. This process is usually based on the evaluation of a knowledge based express in logic or a search process (Bender, 1996) with heuristic evaluation of alternative situations (situation calculus) (Russell and Norvig, 1995). The resources, tasks, and goals in the system are all represented as knowledge, usually defined in the form of a logical language or production rules. The focus of these approaches is the pursuit of explicitly represented goals and the acquisition, manipulation, and application of "knowledge". When a sufficiently restricted environment is used, these architectures excel by applying their knowledge of that environment. The essential limitation of such architectures is that they are prescriptive in nature. Every element and action must be prescribed and defined by a formal set of rules. The robot must maintain a formal model of its world in order to act within that world.

Most approaches in knowledge-based architectures are based on the physical symbol system hypothesis described by Rich and Knight (1991). Given an appropriate world model, these systems can solve intricate problems. The success of these architectures in reasoning problems like games has given them the reputation as the best path toward general intelligence. To be useful for robots the techniques of Artificial Intelligence must be embedded in an architecture for the agent, and due to the monolithic nature of the representation used in traditional approaches, a centralized "intelligent agent structure" was devised using the sense-model-plan-act (SMPA) (Brooks, 1986) loop shown in Figure 2.4.

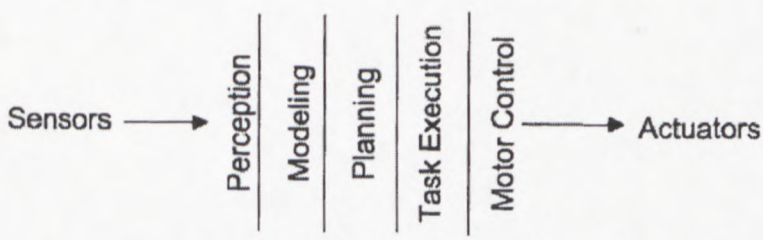


Figure 2.4: SMPA Approach (Brooks, 1986)

This approach has been used to build the control system for robots and has been extended to include notions from utility theory (Bender, 1996) and (Russell and Norvig, 1995). These extensions resulted in a new architecture in which to apply AI techniques called the utility-based agent approach (Bender, 1996), shown in Figure 2.5. Although it incorporates a kind of software value judgment (utility), this new approach is that the AI reasoning methods are a single mechanism for selection of action, and the architecture is the structure that supports the use of this mechanism as a robot (or other) system controller. The AI mechanism is very complex, but the architecture of the software is essentially a single, serial data flow diagram.

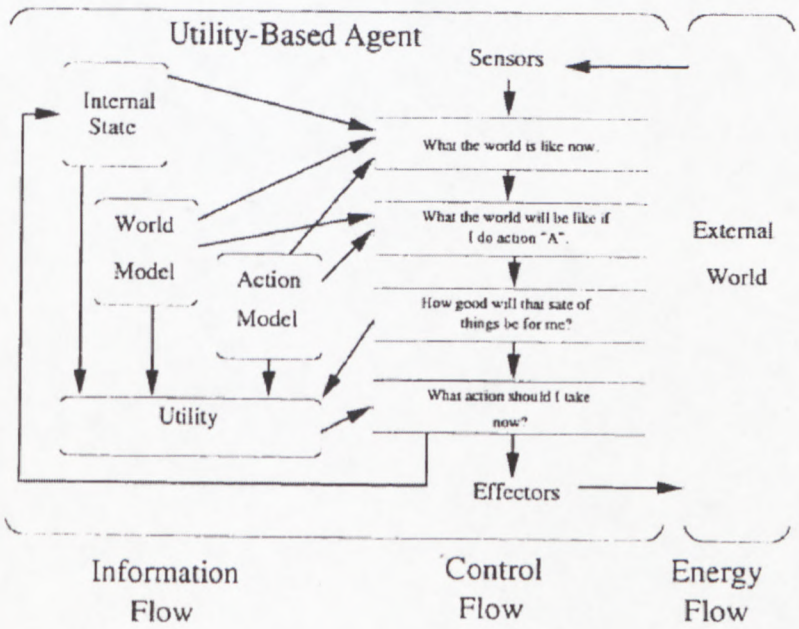


Figure 2.5: Utility-Based Agent (Bender, 1996)

There have been recent developments of a new type of model for knowledge-based agents. This model was influenced by philosophical contributions of Dennett (1991) and his psychology of the intentional stance. The concept is that some systems are most easily described in terms of beliefs, desires, and intentions (BDI), which are separate parts of the agent. Researchers on multi-agent systems have picked up this

concept, and several BDI architectures have been implemented with some success. Belief is typically implemented as a traditional knowledge base of the agent. Desires are explicit representations of goals or goal-states that need not be realizable or consistent with the rest of the knowledge base. Some desires are selected to be goals for the agent, and plans are developed to achieve these goals. Intentions are usually a set of partially instantiated plans associated with a set of goals that the agent has developed to achieve its desires. The BDI architecture is primarily a conceptual framework in which various traditional AI techniques, such as search, heuristics and production systems, are employed to build an agent. In this respect it resembles the utility-based agent used as a model for more recent AI systems. The "desires" of a utility-based agent are implicitly represented by the utility function, but BDI agents explicitly represent their desires using the same language as the agent knowledge base. Extensive work has been invested in developing a formal model for this type of agent. Most models in this category rely on the use of modal branching-time logic (Wooldridge, 1994) to describe the interaction between agents and between belief, desire, and intentions, while others use more easily managed state machines to model the interaction between agents (Correa and Coelho, 1993).

This line of research is interesting because of its relationship to the philosophical concepts about cognitive science that are borrowed from Dennett. The intentional stance was co-opted for the design of the control system, but the multiple-drafts concept was completely ignored, although it was introduced in the same body of work, possibly because of the desire to make the BDI concept fit neatly into the rigid and serial structure of more traditional AI approaches.

Rao (1996) introduces a variation of the BDI architecture by showing the development of AgentSpeak, an agent communication language that supports communication between agents that are based on reasoning mechanisms. Mayfield et al. (1996) evaluates another agent communication language called the knowledge Manipulation and Query Language, which is part of a suite of languages developed to

support knowledge-based, multi-agent software systems. Both of these architectures base communication on speech-act theory.

The InteRRap architecture, developed by Muller (1996), borrows from the BDI line of development and build up a three-layered architecture with behaviours on the bottom, a local knowledge-based system in the middle, and a social knowledge-based system at the top to control coordination between multiple agents as shown in Figure 2.6. The InteRRap architecture was simulated and then tested on a small group of miniature robotic fork-lifts. Each robotic forklift was represented by an InteRRap agent, and they all cooperated to perform tasks in a simulated loading dock environment.

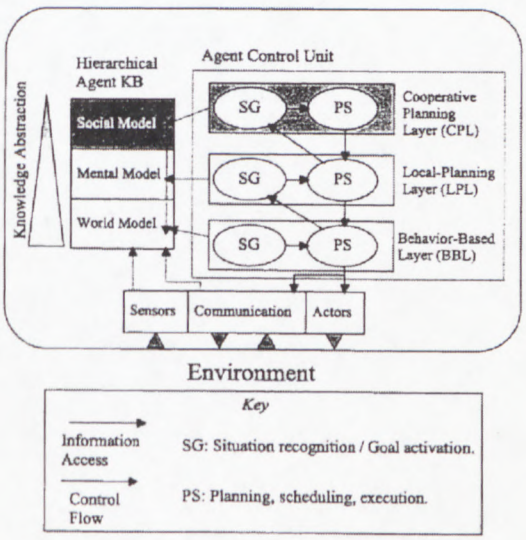


Figure 2.6: InteRRap Architecture (Muller, 1996)

Another development in multi-agent systems is the emerging research area of Agent-Oriented programming. The premise of this work is that the intentional stance is naturally used by developers when speaking about software, so software development tools and environments should support specification of programs in terms of the intentional stance. This typically involves specifying a software system as a set of knowledge based agents that communicate using an agent communication language. The first such system was AGENT0 (Shoham, 1993), where the agents

were simple rule systems, but AGENT0 has been extended by the development of PLACA (Thomas, 1993), where each agent is endowed with a symbolic planning mechanism as well.

2.4 Semantics in a Machine Control System

In this section, the key word of the thesis, semantics, will be explained using Sloman (1994a) knowledge. We appreciated his insight to describe the effect of semantics in a machine control system. We fully cite his work with my respect. This section also introduces some of the key concepts, sketches a subset of the links between philosophy and engineering relating to the "information level" description of behaving systems, and relates this to the aims of AI.

Much research on intelligent systems has concentrated on low level mechanisms or limited subsystems. We need to understand how to assemble the components in an architecture for a complete agent with its own mind, driven by its own desires. A mind is a self-modifying control system, with a hierarchy of levels of control, and a different hierarchy of levels of implementation. AI needs to explore alternative control architectures and their implications for human, animal, and artificial minds. Only when we have a good theory of actual and possible architectures can we solve old problems about the concept of mind and causal roles of desires, beliefs, intentions, etc. The global information level "virtual machine" architecture is more relevant to this than detailed mechanisms. E.g. differences between connectionist and symbolic implementations may be of minor importance. An architecture provides a framework for systematically generating concepts of possible states and processes. Lacking this, philosophers cannot provide good analyses of concepts, psychologists and biologists cannot specify what they are trying to explain, and psychotherapists and educationalists are left groping with ill-understood problems. The section shows the

importance of an idea shared between engineers and philosophers: the concept of "semantic information" (Sloman, 1994a).

2.4.1 Syntactic Information

A notation, language, or representing structure has different aspects, sometimes referred to as syntax, semantics, pragmatics and inference. Sloman (1994b) has shown elsewhere how these notions can be applied to different kinds of control substates, some not usually thought of as having a syntax or semantics, e.g. the "representation" of ambient temperature in a thermostat.

Syntax is concerned with the structure and forms of variation of a class of objects (sentences, pictures, signals or control states). Semantics involves reference to other things, in the environment, within the system, in the future, in the past, and possibly also things that have never existed and never will. Sloman takes pragmatics to cover the roles or functions of representing structures within an integrated system. Inference is concerned with the transformation of structures in a manner that usefully preserves or changes semantic properties or pragmatic roles.

We can discern two uses of the word "information" by engineers, one primarily syntactic and one semantic.

When dealing with issues of signal transmission, signal compression and detection and correction of transmission errors, engineers often use syntactic concepts of information, concerned mainly with the patterns in structures independently of whether those structures refer to anything outside themselves, as sentences and pictures typically do. One syntactic measure of "information" in a string is the length of the shortest program which, given any N, returns the Nth symbol in the string. This defines "algorithmic complexity", an inverse measure of the amount of pattern or

regularity in the string: the more regularity the less information. Logicians, linguists and computer scientists use concepts of syntax that are not concerned with measures but with descriptions of structures and their transformations (Sloman, 1994).

2.4.2 Semantic Information

Sloman pointed out: "The semantic concept of information, used informally by engineers, is closer to the familiar, though extremely ill-defined, notion of the "meaning", or "information" that may be conveyed by a message, or stored in a book or videotape. It is applicable to internal states as well as external communications. This includes ordinary concepts of mental states and processes such as "understanding", "believing", "desiring", "deliberating", "perceiving", "regretting", and many more. Philosophers call these states and processes "intentional" because they refer to something outside themselves, including, in some cases, nonexistent entities, e.g. hallucinating daggers, intending to perform actions which turn out to be impossible, or praying. Doing, preventing and trying are also semantic states, for they involve reference to future results.

All semantic information-processing depends ultimately on (internal) syntactic capabilities, for semantic information-processing depends on syntax-processing (i.e. Structure-manipulating) engines, whether in computers, neural nets or other mechanisms. Agents need representational forms whose syntactic manipulation is semantically useful, unlike "languages" invented merely to illustrate the theory of syntax or parsing. A notation some of those syntactically well formed formulae are semantically uninterpretable (like "Happiness eats democratic numbers intelligibly") is wasteful, though some meaning gaps may be important seeds of semantic development. Internal syntax is useful for processing information if it provides syntactic manipulations that map onto useful semantic relationships (e.g. syntactic derivability preserves truth). Much of the development of science and mathematics

has been the creation of new formalisms that usefully compile semantic relationships into syntactic ones.

Computer programs have several kinds of semantics. One sort (called the "information level" below) refers to the application domain, usually outside the computer, e.g. information about employees. Another sort refers to the abstract datastructures manipulated at run time in the machine, e.g. numbers, strings, lists. Another refers to the low level machine instructions and memory locations manipulated when a program runs. Some languages also include compile-time semantics e.g. for macros, compiler directives or type definitions, which control compilation rather than subsequent running. Semantic information can vary both in content and pragmatic role. For instance it may be about what is the case, about what to do, about how to do things, or it may express a question or test."

2.4.3 Semantic in Control Systems

According to Sloman summarized, engineers designing plant control systems, office support systems and the like, often start at the global information level, analysing semantic requirements for the whole system. For example a system may need information about the environment, rules and procedures to be followed, legal constraints, company objectives and which risks to avoid. Metain-formation (information about information) is also needed, for example: where to get certain information, what to do when it arrives, how to cope with contradictory reports, and so on. Internal monitoring might yield metain-formation about how quickly information is dealt with, which kinds turn out to be unreliable, and so on.

Designing information systems also raises implementation issues at different levels, such as:

- how information will be stored in the system,

- how to access it,
- selecting forms of representation,
- selecting syntactic transformations,
- selecting programming language(s) and operating systems,
- selecting computers, interfaces, network links, etc.
- functional decomposition of the system.

The semantic, or information level, specification, e.g. that the system must include information about employees and their roles and use it to perform certain tasks, says little about such implementational details. The specification can be given in an implementation independent way: including requirements for and the behaviour of a certain kind of information-processing virtual machine, leaving the computational or electronic details concerning "lower level" virtual or physical machines for later.

Moreover, implementation details may be revised as technology advances. Processors used, the memory technology, and even the programming languages and some of the low level algorithms may all change without implying any change in what information is processed, as far as the users and designers are concerned: i.e. the global information level description is not affected.

However, information level descriptions may imply a certain sort of architecture: a top level functional decomposition, defining which sorts of major subsystems coexist, and which information they handle. For instance, being undecided about whether to go to A or to B, presupposes mechanisms for manipulating goals, for evaluating and selecting between alternatives, and for acting on a selection.

For subsystems we can also define an information level: what they "know" about the rest of the system, i.e. their environments and their tasks. Typically, users will not be concerned with that, though designers and maintainers will.

2.5 Intelligent Control for Human-Machine Interface

The design of control systems and human-machine interfaces in the field of technologically complex and safety-critical environments is today an open issue. The increasing use of automation has improved efficiency, safety and easiness of operations but, at the same time, it has complicated the operator's situation awareness, because of the opaqueness of the autonomous automatic system (Cacciabue and Hollnagel, 1998). This section we will mainly cite Buss and Hashimoto's (1996) work, and we thank them for supplying a good reference for my project. Their work effectively links intelligent control with human-machine interface under a complex circumstance called Mechatronics.

2.5.1 Mechatronics

In recent years mechanical, electrical, and computer engineering have fused to the field of mechatronics. Intelligent devices including sensor, actuator, and software are built into intelligent mechatronic components integrating all these aspects naturally in minimum space. The complexity of a mechatronic module is mainly in the control software often referred to as the intelligence of the system (Buss and Hashimoto, 1996).

In terms of they indicated, toward Intelligent Control (IC) may be the following:

- Integrating sensor, actuator, and software into a mechatronic module is one step toward but short of realization of IC systems;
- Hierarchical control systems with increasing levels of abstractness in information processing;
- Structure changing adaptive systems including parameter adaptation on low-level

control and more sophisticated adaptation mechanisms on higher levels in the control hierarchy;

- Intelligent "reference generating algorithms" for hierarchical control systems such as cars, mobile robots, robotic mechanisms, numerically controlled manufacturing equipment, etc. The term "reference generating" means algorithms which flexibly generate appropriate reference trajectories depending on the measured environment state;
- Flexible intersystem communication among distributed semiautonomous systems and system components with defined and guaranteed global behaviour; networked mechatronics through a computer network information infrastructure;
- Optimization schemes for automatic and optimal IC system design;
- Intelligent human interaction to make the system be perceivable as an IC system;
- Discrete event dynamical systems-DEDS, hybrid systems; as such gain scheduling and structure adaptation redefined from a DEDS point of view;
- Increased flexibility, robustness, and fault-tolerance;
- Ability for careful interactive improvements, learning, and self-organization.

System and technology integration is an important issue of crucial importance to IC. Particular problem areas may only be recognized when designing large scale control systems.

To summarize the proposition for an approach of IC: System and technology integration including mechatronics, computer science, system design optimization, communication, and human interaction may yield what the authors expect from IC (Buss and Hashimoto, 1996). This view of IC is illustrated in Figure 2.7 from Buss and Hashimoto.

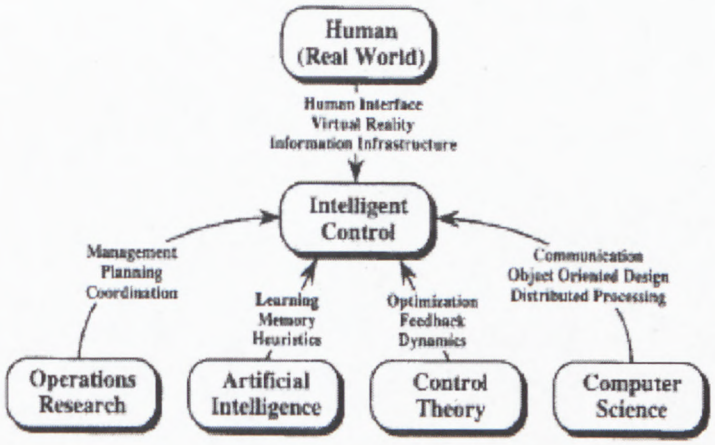


Figure 2.7: Key technologies for Intelligent Control Systems

Some approaches have been reported, which are based on watching human performance closely and then imitating or acquiring the behaviour (Delson and West, 1993), (Yang and Asada, 1990) and (Yang, Xu and Chen, 1993). Human-oriented machines are also being proposed (Schweitzer, 1995).

2.5.2 Intelligent System Behaviour Levels

One of the key goals in intelligent control System design is to imitate human behaviour (Buss and Hashimoto, 1996). Buss and Hashimoto said: "If we were able to generate "real" artificial intelligence the resulting behaviour would be naturally perceived as intelligent. But since this seems to be the long-term challenge it is more reasonable to follow the subgoal to imitate humans." Human behaviour has been discussed from various fields. At this point we briefly review the work by Rasmussen (1983) and point to a few insights from cognitive science about "human skill."

When we distinguish categories of human behaviour according to basically different ways of representing the constraints in the behaviour of a deterministic environment or system, three typical levels of performance emerge: skill-based, rule-based, and

knowledge-based performance (Buss and Hashimoto, 1996). These levels and a simplified illustration of their interrelation are show in Figure 2.8.

2.5.2.1 Skill-based Behaviour

Skill-based behaviour represents sensory-motor performance during acts or activities which, following a statement of an intention, take place without conscious control as smooth, automated, and highly integrated patterns of behaviour. There is no conscious thought necessary because control is realized directly between perception and motor action. The sensed information is perceived as time-space signals, continuous quantitative indicators of the time-space behaviour of the environment without symbolic meaning.

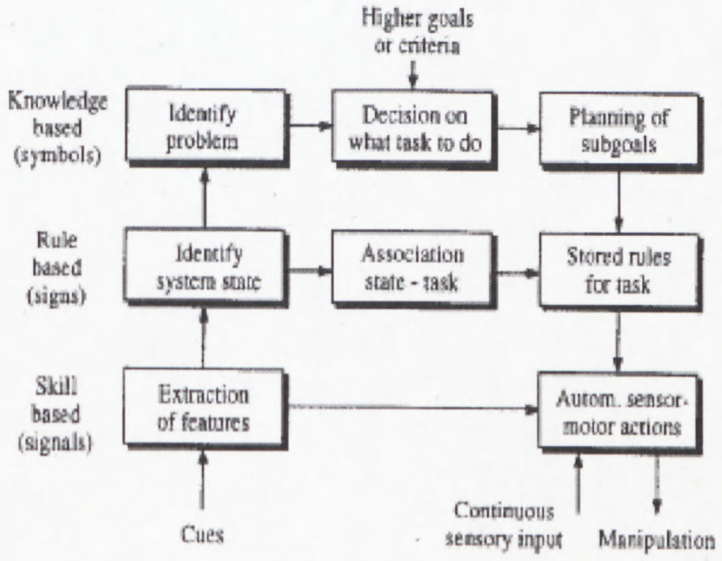


Figure 2.8: Human Behaviour model (Rasmussen, 1983)

2.5.2.2 Rule-based Behaviour

Rule-based behaviour is the composition of a sequence of subroutines in a familiar situation, typically controlled by a stored rule or procedure. These rules or procedures may have been derived empirically during previous occasions, communicated from other persons' know-how, or they may be generated on occasion by conscious

problem solving and planning. The point here is that task performance is goal-oriented but structured by feedforward control through a stored rule. Information is typically perceived as signs, which can be defined as a stimulus to activate or modify predetermined action or manipulation sequences.

2.5.2.3 Knowledge-based Behaviour

During unfamiliar environment situations the control of performance must move to a higher conceptual level, where it is goal-controlled and knowledge-based. The goal is explicitly formulated, based on the person's analysis of the environment and the overall aim. Then a useful plan is developed such that different plans are considered, and their effect is tested against the goal, physically by trial and error, or conceptually by means of an environment model and simulation. Information must be perceived and processed in symbols, which are part of the human world of meaning, without direct meaning in the physical world.

2.5.3 Human-Machine Interface

Intelligent control systems being semiautonomous or at times fully autonomous must interact with human operators, experts and users (Buss and Hashimoto, 1996). To achieve natural interaction the human interface is very important and many technologies have emerged to realize such interfaces. Teleoperation (TO), supervisory control (SC), virtual reality (VR), and computer supported cooperative work (CSCW) systems are being proposed (Sheridan, 1987 and 1992) and (Lee, 1993). As a brief overview we list here a few terms and definitions of these fields and point to important results in the literature.

Teleoperation is the extension of person's sensing and manipulation capability to a remote location. Teleoperation refers to direct and continuous human control.

Telerobotics is a form of teleoperation in which a human operator acts as a supervisor and intermittently communicates to a computer, giving information about goals, constrains, plans, contingencies, assumptions, suggestions, and orders relative to a limited task, getting back information about accomplishments, difficulties, concerns, and sensory data. Figure 2.9 illustrates the concept of telerobotics. The human operator provides some commands as symbolic goals to the intermittent computer and some analogic hand-control movements to point to or grasp objects.

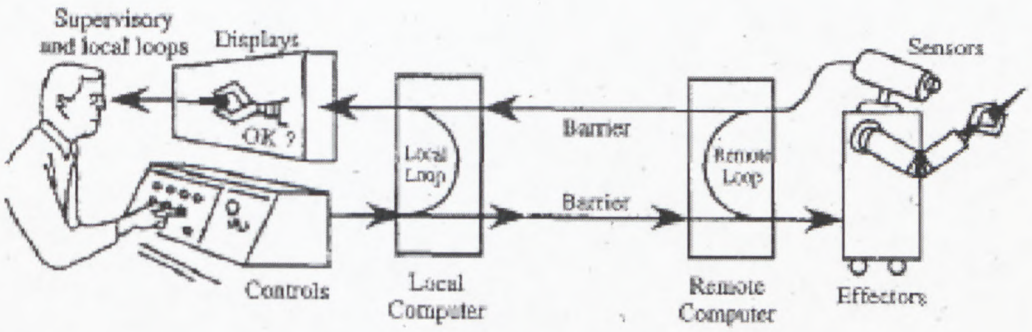


Figure 2.9: Concept of a teleoperation system (Sheridan, 1987)

The "human-interactive" computer should thus be human friendly, able to indicate that it understands the message or to point out that a specification is incomplete. It should be able to interpret signals from the distant telerobot and to generate useful graphic displays. This part of the human interaction is recently often referred to as the human interface (HI) (Askew and Diftler, 1993) and (Neugebauer, Degenhart, and Schraft, 1993).

Supervisory control as a term is derived from the close analogy between the characteristics of a supervisor's interaction with subordinate human staff members and a person's interaction with intelligent automated subsystems (Sheridan, 1987 and 1992). SC in the present context is mostly synonymous with telerobotics, referring to the analogy of a human supervisor directing and monitoring the activities of a human subordinate. Common applications of SC are human supervision of any semiautonomous system, while telerobotics commonly refers to a device having arms

or hands for manipulating or processing discrete objects in its environment.

Figure 2.10 (Buss and Hashimoto, 1996) shows the concept of SC transferred to motion control problems, robotics and manufacturing systems. Motion control methods are well developed and usually much detailed knowledge about the task environment is available. The autonomous part of the SC system gets commands from the supervisory controller.

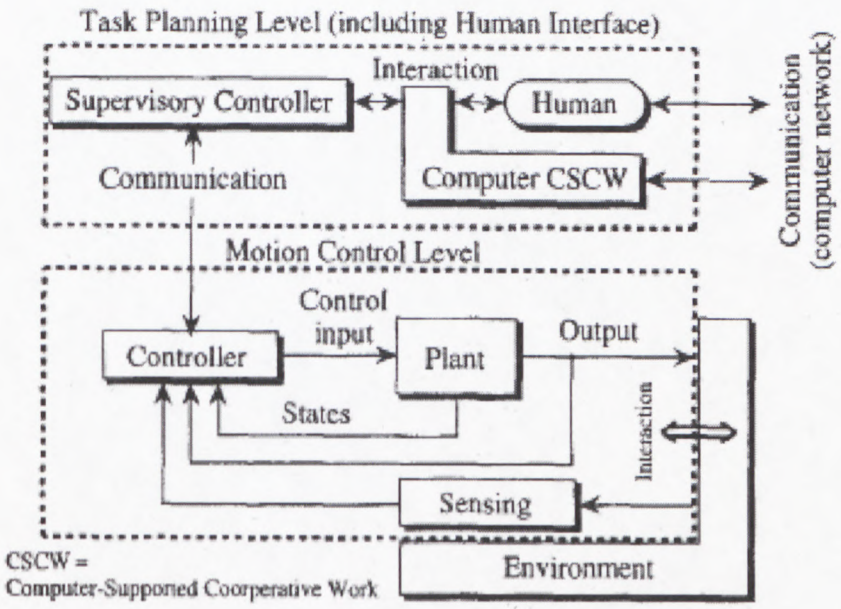


Figure 2.10: Computer-supported cooperative work (CSCW)

The human-interactive-computer interacts with the human operator and can have several subordinate supervisory motion controllers. In normal operation the automation level of the overall system is fairly high, where the operator must interfere with the system only if unknown environment states are detected. Several supervising human operators as well as several human-interactive-computers can communicate over a network to realize control of large scale systems. The computers play an important role in assisting the human operator while realizing or planning complicated tasks by showing necessary data and knowledge.

2.6 Human-Machine Interaction in Information Systems

In the section 2.5, we presented intelligent control for human-machine interface, but we cannot neglect one point is human beings will operate the machine to get a feedback information whether it is correct or not. For this point, we must consider using information system to help user to save and manage information in order to ensure the control system working properly and supply correct information. So, this section we will indicate the effect of human-machine interaction in information systems. Human beings are constantly interacting with systems surrounding them. These include personal computers, automobiles, and smart appliances, among many others. It is expected that in the near future there will be interactions with service robots in homes and offices. A great deal of effort has been made into advancing this interaction using science and technology. The main goal is to make human interactions with systems not only effective and safe, but also enjoyable and entertaining (Agah, 2001). Information systems are designed to solve specific interactive tasks. If all intended tasks can be interactively performed, the information system is complete in that respect. Human-Computer Interaction draws on many disciplines, but it is in computer science and systems design that it must be accepted as a central concern. For systems design it is an essential part of the design process. From this perspective, HCI involves the design, implementation and evaluation of interactive systems in the context of the user's task and work.

2.6.1 Information and Information Systems

Devlin (1991) indicated: "What is information? Many people have attempted to give a definition but most of them are not complete. A typical explanation is that information is processed data that has meanings to its users. But then questions arise as to what meaning is. If information is to the study of information as object is to physics, and

there are many laws by which we can study objects, then what are the laws by which we can study information? What is the study of information anyway?

What can be said here is that information is not a simple, primitive notion. He compares the difficulties for a man in the Iron Age to answer the question 'What is iron?' and for a man in today's Information Age the question 'What is information?' To point to various artefacts of iron in order to answer his question would not be satisfactory; to demonstrate some properties of information as an answer to 'What is information?' is not good enough either. People can feel the possession of information, and can create and can use information. They gather it, store it, process it, transmit it, use it, sell it and buy it. It seems our lives depend on it; yet no one can tell what exactly it is."

In order to understand the nature of information, one may have to find some fundamental and primitive notions with which the question can be investigated and explained. The concept of a sign is such a primitive notion that serves the purpose. All information is 'carried' by signs of one kind or another. Information processing and communication in an organisation are realised by creating, passing and utilising signs. Therefore, understanding signs should contribute to our understanding of information and information systems (Stamper, 1992).

The investigation of information systems is an active area where much attention has been paid by other research and industrial communities. Much discussion has taken place on its pluralistic and interdisciplinary nature and its foundations. An increasing number of researchers and practitioners define information systems as social interaction systems. Social and organisational infrastructure, human activities and business processes are considered as part of information systems. The UK Academy of Information Systems provides the following definition of the domain of information systems (UKAIS, 1996). 'The study of information systems and their development is a multi-disciplinary subject and addresses the range of strategic,

managerial and operational activities involved in the gathering, processing, storing, distributing and use of information, and its associated technologies, in society and organisations.' Many authors emphasise the importance of information systems study in this postmodern society and its interdisciplinary nature. For example, Avison and Nandhakumar (1995) suggest that information systems encompass a wide range of disciplinary areas such as information theory, sociology, information technology, human-computer interaction, and perhaps more.

2.6.2 Models of Interaction

Dix et al. (1998) presented: "Interaction involves at least two participants: the user and the complex systems. Both are complex and are very different from each other in the way that they communicate and view the domain and the task. The interface must therefore effectively translate between them to allow the interaction to be successful. This translation can fail at a number of points and for a number of reasons.

The purpose of an interactive system is to aid a user in accomplishing goals from some application domain. A domain defines an area of expertise and knowledge in some real-world activity. Some examples of domains are graphic design, authoring and process control in a factory. A domain consists of concepts that highlight its important aspects. In a graphic design domain, some of the important concepts are geometric shapes, a drawing surface and a drawing utensil. Tasks are operations to manipulate the concepts of a domain. A goal is the desired output from a performed task. For example, one task within the graphic design domain is the construction of a specific geometric shape with particular attributes on the drawing surface. A related goal would be to produce a solid red triangle centred on the canvas. An intention is a specific action required to meet the goal."

The interaction framework attempts a more realistic description of interaction by including the system explicitly, and breaks it into four main components (Dix et al., 1998), as shown in Figure 2.11. The nodes represent the four major components in an interactive system – the System, the User, the Input and the Output. In addition to the User's task language and the System's core language, there are languages for both the Input and Output components to represent those separate, though possibly overlapping, components. Input and Output together form the Interface.

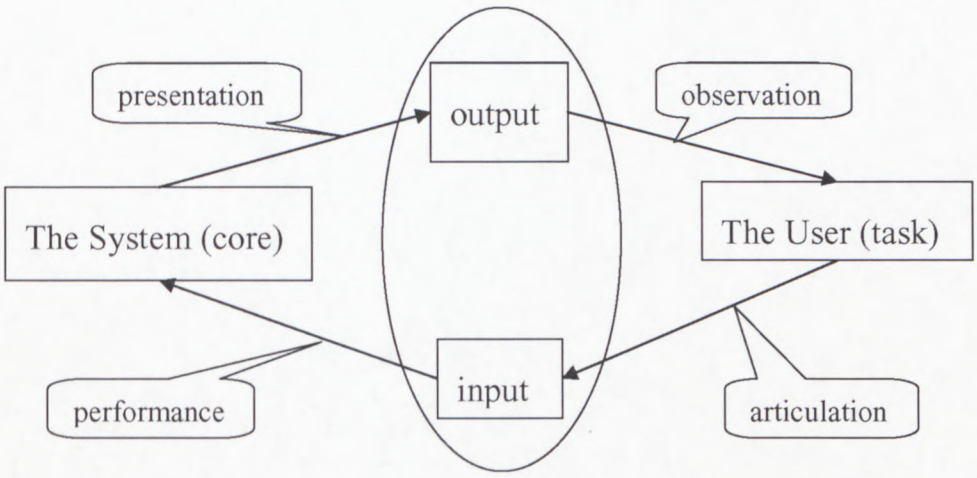


Figure 2.11: The General Interaction Framework (Dix et al., 1998)

The concept of an interactive system seems to appreciate the duality and the need for integration between systems development and interface design. Managing this integration as well as their separate concerns is the key to successful development of interactive systems.

2.6.2.1. The need to study HCI

I appreciated Lif's (1998) opinion when he says: "Computerisation has sometimes changed the work as well as the work environment for people beyond recognition. Several industrial processes were run earlier by hundreds of people who knew the different steps of the process in detail. These people could get an understanding of the

processes involved by just watching, or even listening, to the different machines. Today, it is common that factories are run by only a few computer operators. The computer system automatically controls some parts of the process, while the users manually control other parts of it by reading and changing parameters on the computer screens. The interface towards the process has moved away from the machines to the computer screen.

Administrative work involves decision-making based on huge amounts of information. Traditionally, this information was carefully recorded in documents and in forms. Tools and methods for storing, sorting, searching, reading and writing information have developed based on knowledge gained through decades of accumulated experience. When making a decision, it is important to have the necessary information readily at hand. Previously it was possible to get an overview by simply spreading all papers, books, maps, etc. on the desk. Today, however, almost all information is stored in immense information systems. The same amount of information has to be read from a computer screen. According to Henderson and Card (1986), a standard office desk has the area of approximately 22 standard size PC displays. Obviously, the way information is presented on the screen will have an important impact on the decisions that are made by the user. The appropriate information and functionality have to be available in an adequate way to enable the user to make proper decisions. Designing systems that effectively support the user during decision-making processes is a major goal within HCI.

A user interface may be described as the appearance and the behaviour of the information system (i.e., the only part of the computer system with which the user is in direct contact). In the early days in the history of computers, there was no need, or even no possibility, to put much effort into the design of the user interface. The number of design options was rather limited. It was not possible to use graphics, sound or speech recognition in the user interface. User and programmer was usually the same person. There was no need to bother about making the system useful for

other people. Today, most computer systems are intended for users with little or no skills in programming or even in using computers as such. Graphical user interfaces and various techniques for interaction have increased the possibilities for creating information systems that are easy to learn and efficient in daily use. However, the number of possible design options has increased, which can be a problem if they are not carefully examined. It is necessary that the user interface is designed to meet the requirements of users working within a certain context.

Understanding the user's interaction with the computer has been a central goal of HCI. Norman (1986) has described this interaction in a model including seven goal-driven user activities. A user performs a task by specifying an intention to achieve a goal. This intention is translated into an action that is then executed (e.g., pressing "enter" on the keyboard). The action may cause a change in the state of the computer, which is perceived and interpreted by the user. The user then evaluates this state with respect to goals and intentions. The potential difficulties when working with the system have been described as the "gulfs" of execution and evaluation. The gulf of execution is the case in which the user knows what goals to be achieved but does not know which physical variables to adjust, or in what way to adjust them. The gulf of evaluation is where the system has altered, usually because of the user's action, but the user cannot easily understand the change in the system's state. These difficulties may depend on the design of the user interface.

The area of HCI is extensive. It is necessary to understand how users think and perceive information, how people co-operate during work, how to create hardware and software to support users, how information systems affect people, how to create methods for building the software, how to evaluate the applications, etc. HCI is a multidisciplinary topic based on knowledge gained from several disciplines such as cognitive psychology, software engineering, computer science, organisational theory, sociology, ergonomics, systems analysis, process control and industrial design."

2.6.2.2. Systems analysis

He also presents that another background discipline is systems analysis which is concerned with understanding systems by building models. A system is a limited part of the environment, and is separated from its environment by the systems boundary. According to this approach, the computer, the user and the environment can be regarded as one system, a system that can be represented by a model. Such a model can be modified and tested, e.g. through simulations, without affecting the real world, where the result of this process can be employed to improve the actual system. A model is never a true or complete representation of a system, but describes aspects of the system relevant to a specified purpose. A model is extremely useful when several people need to have a common view of a system.

Systems analysis gives a theoretical framework for how models are formulated and validated, how they can be analysed and how conclusions concerning the behaviour of the studied system can be drawn from this analysis.

Models are frequently occurring within HCI. The users' interaction with the computer can be described with a model (Lif, 1998). A user has a conceptual model of the information system, but also of the "real" system that is the focus of interest in a work situation. An information system is usually tested with a model (e.g., a prototype). Systems analysis, and the use of formal models, has an important role in information system development. Here, systems analysis involves analysing and modelling present and future work situations. The result of this process are models suitable for developing the computer support, such as data models, data flow diagrams, use cases etc. Each model represents one view of the analysed system, is never equal to the real system and is only valid within the area defined by the purpose of the study. When formulated in a correct way and with a known and controlled quality, formal models are, however, extremely useful for specification of requirements, documentation of data flows and information utilisation, etc.

2.6.2.3. Usability

Nielsen (1993) claims that the term "user friendly" was introduced when developers of information systems first realised that their systems were to be operated by users with demands on the products in terms of access, etc. Nielsen means that this term is not appropriate for several reasons. "Users don't need machines to be friendly to them, they just need machines that will not stand in their way when they try to get their work done" (p. 23). Another reason is that different users have different needs in the sense that a machine can be "friendly" to one user but tedious to another. Instead, Nielsen proposes the term "usefulness" that relates to whether the system can be used to achieve some desired goal. Usefulness can be divided into two categories: Utility and usability (Grudin, 1992). Utility corresponds to whether the needed functionality is included in the system (i.e., if all tools needed to perform a given task exist). Usability is more closely related to how well a user could use this functionality.

Another important criterion is the extent to which the user can interact efficiently with the system without unnecessary mental efforts that are caused by cognitive work environment problems. Limitations of the cognition in the work environment that hinder the users to use their skills efficiently may cause cognitive work-environment problems (Lind, Nygren, and Sandblad, 1991). Such impediments are often caused by the human-computer interface and can lead, in addition to somatic and mental health problems, inefficient work procedures, bad performance and low user acceptance.

2.6.3 User interfaces

The research area of computer user interfaces is a field where the technology of human- system interaction has direct applications, encompassing interface for human and robot through robot understanding of human intentions (Sato et al., 1994), human gesture recognition for human-robot interaction and communication (Tsukamoto and

Lee, 1995), multi-modal communication systems for human-computer interface and dialogue (Mori, Dohi and Ishizuka, 1995), interactive communications between human and computers considering the human feelings and emotions (Iwano et al., 1995), user interface for human communication with robots (Kurata, Chang and Hashimoto, 1995), human-computer interface through the computer understanding of human motions (Stoll and Ohya, 1995), volume visualization and presentation of multiple dimensional data for scientific simulation (Noma and Iwata, 1993), volume visualization application of multimedia technology to user interface design (Rodriguez and Rowe, 1995), user interface using a human-like face generated by the computer (Hiramoto, Dohi and Ishizuka, 1994), multimedia user interfaces for interactive teaching systems, composition, collaboration, and explanatory systems (Woolf and Hall, 1995), and utilization of multimedia technology in user interface design (Maybury, 1993).

User Interface Modelling (UIM) is a method for gathering user requirements that are directly applicable when designing the user interface of an information system (Lif, 1999). UIM is basically a complement to traditional use-case modelling. UIM specifies an actor model, a goal model and a work model in participatory sessions with end-users, software developers and user interface designers.

Lif summarized that information systems construction and user interface design have traditionally been undertaken using different methods and techniques, and the people taking part in these activities have often been part of different academic traditions. Such a separation is unfortunate, since every information system has a user interface which for many purposes is perceived as the whole system. For the user, the distinction between interface and system is, for most practical purposes, meaningless. In addition, most application usage is part of a larger information processing context, whether computerised or not. Thus, isolating the design of the application's user interface from the information system development is clearly undesirable. However, the conceptual distinction between user interface and information system remains

useful and interesting. The proliferation of different connected electronic devices necessities the development of many front-ends for the same information system. The different capabilities of these devices result in differing interfaces, while the information system remains essentially the same. Furthermore, many future products may serve as new interfaces for existing and largely unchanged systems. Hence, speaking about the information system and its separately attached user interfaces is clearly meaningful.

2.7 Conclusion

In this chapter, we fully investigated the key concepts for intelligent machines, as Artificial Intelligence, Intelligent Control, Semantics, Intelligent Machine Architecture, Human-Machine Interaction, Information Systems and Graphical User Interface. These literatures of the key concepts provided enough knowledge to guide us to work out a suitable structure for developing a GUI integrated with an intelligent semantic machine control system.

Chapter 3: Framework and Methodology

This chapter starts presenting Usability Engineering drawn from the user-centred design standard and Object-Oriented Methodology with Unified Modelling Language (UML). Usability is put into the wide context of system acceptability and characterized in terms of a set of attributes (learnability, efficiency, memorability, error prevention and satisfaction) that enable a systematic approach to usability as an engineering discipline (Nunes, 2001). We describe the major theoretical frameworks of usability from the definition of usability. From the description of the framework of model human processor we indicate the Goals, Operators, Methods and Selection rules (GOMS) family of models, one of the most widely accepted theoretical models to predict and evaluate human performance. We address Object-Oriented Methodology and Unified Modelling Language (UML) in software development. And then, we briefly present the most complete proposal for a usability engineering lifecycle. The description includes the tasks required to redesign the software development lifecycle and introduce usability methods. From this detail lifecycle we introduce the broad standard for user-centred design, and describe the main principles and activities involved.

3.1 Usability Engineering

This section discusses usability of software systems, its definition and major theories, models and principles. For the purpose of this chapter we are concerned with usability in the context of the application of engineering principles to systematically build usable and economically viable interactive systems that support real work in an effective and efficient way that promotes satisfaction in use (Nunes, 2001). However,

this definition of usability engineering arises from the wide perspective of human-computer interaction (HCI), which is the discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them. The discipline of human-computer interaction is shown in Figure 3.1 (ACM, 1992):

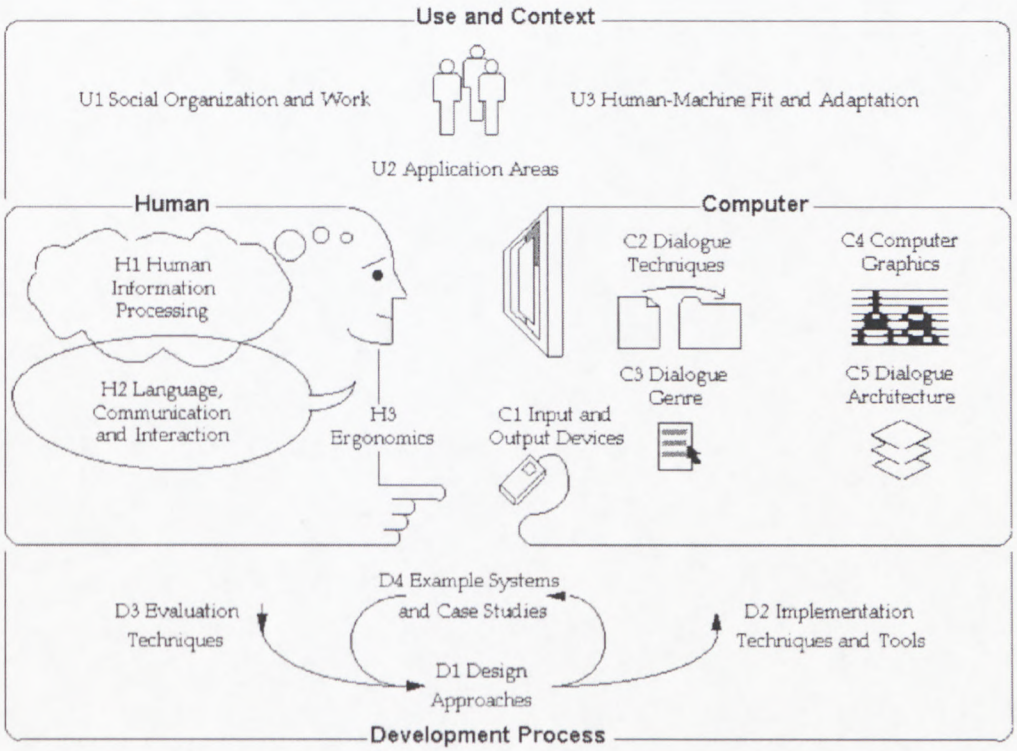


Figure 3.1: The discipline of human-computer interaction (ACM, 1992)

3.1.1 Definition of usability

The usability of a product is defined in the ISO 9241 (1998) standard, part 11 as: "the extent to which a product can be used by specific users to achieve specific goals with effectiveness, efficiency and satisfaction in a specific context of use".

This definition relates to the quality of the interaction between the person who uses the product to achieve actual work and the product itself. The important features of

the interaction are effectiveness, efficiency and satisfaction. Effectiveness concerns how well the user accomplishes the goal he wants to achieve with the system. Efficiency relates to the resources consumed in order to achieve the goals. Finally, satisfaction concerns how the user feels about use of the system (ISO, 1998).

Usability has multiple components and is not single one-dimension property of a user interface. According to Nielsen (1993) usability is traditionally associated with the following attributes:

- Learnability - the system should be easy to learn, enabling even inexperienced users to perform rapidly the supported tasks;

- Efficiency - the system should be efficient in use, so that once the users have learned the system they should be able to achieve a high level of productivity;

- memorability - the system should be easy to remember, allowing causal users to reuse the system without having to learn the system again;

- Error prevention - the system should prevent users from making errors, in particular, errors that damage users' work must not occur. The system should enable users to recover from errors;

- Satisfaction - the system should be pleasant to use, fostering subjective satisfaction in use.

The precise definition of such usability attributes enables a systematic approach to usability as an engineering discipline. The components described above can be improved, measured and evaluated, and hence, constitute an alternative to speculative approaches to usability.

Usability is typically measured by testing a number of representative users performing a predetermined set of tasks. To determine the system's overall usability we can take a mean value of the scores of a set of usability measures, or, recognizing that users are different, consider the entire distribution of usability measures (Nielsen, 1993). Furthermore, a number of methods for analyzing user interface quantitatively are also available, such as GOMS (Card et al., 1983).

Constantine and Lockwood (1999) also provide a set of five rules of usability that point out the general direction towards usability. They provide a general framework for the effectiveness of modern user interfaces in the form of the following usability rules (Constantine and Lockwood, 1999):

- Access rule - the system should be usable without help, prior experience or instruction by an experienced user in the application domain;

- Efficacy rule - the system should not interfere or impede efficient use by an experienced and skilled user;

- Progression rule - the system should accommodate and facilitate continuous advancement in knowledge, skill and facility as the user gains experience;

- Support rule - the system should support real work by making it simpler, faster, easier and more fun for the users to perform the tasks they are trying to accomplish and by creating new possibilities;

- Context rule - the system should be suited to the operational context within which it will be deployed.

3.1.2 Cognitive Framework of Usability

The dominant frameworks used in human-machine interaction to understand and represent how humans interact with computers or machines are based on the cognitive psychology perspective. Cognitive psychology is a theoretical perspective that focuses on how human beings achieve their goals in terms of cognitive tasks that involve transforming and processing information from the sensory input (Nunes, 2001).

3.1.2.1 The model Human Processor

The Model Human Processor is an extension of the basic information-processing model that sees the human mind as a series of ordered processing stages (Preece, 1995). This simplified model starts with the sensory input stimuli and comprises a four-stage model of encoding, comparison, response selection, response execution, and finally ends with the output.

The two main extensions of the basic information-processing model are the inclusion of attention and memory processes. They form the basis of the Model Human Processor (see Figure 3.2) which consists of three interacting systems: the perceptual system, the motor system and the cognitive system - each with its own memory and processor (Card et al., 1983).

The perceptual system carries sensations from the physical world through sensors (eyes, ears, etc.) and buffers that information while its being symbolically coded.

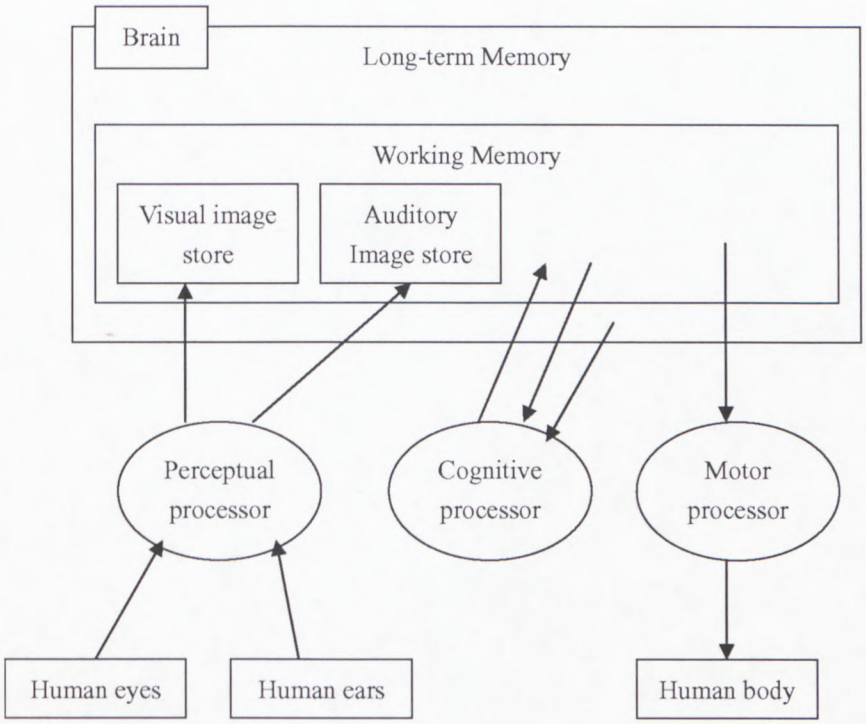


Figure 3.2: The Model Human Processor

Card et al (1983) pointed out: "The cognitive system connects the inputs from the perceptual system to the right outputs of the motor system. The process consists in receiving symbolic coded information from the working memory, eventually combining that information with previously coded information in the long-term memory, and decides how to respond. Since most of the human tasks are complex and involve learning, retrieval of facts, or problem solving; the memories for the cognitive system are more complicated. The working memory holds the intermediate products of thinking and the symbolic representations produced by the perceptual processor. The working memory consists of a subset of the symbolic elements (chunks) in the long-term memory that are active to support are its limited capacity to hold information, in amount and time. The long-term memory holds the available mass of knowledge in the form of a network of related chunks accessed associatively from the contents of the working memory. The contents of the long-term memory must be encoded in symbolic form - there is no direct way of storing information from the

sensory inputs into the long-term memory.

The motor system is responsible for carrying out responses to the sensory inputs. Thought is transferred into action, using information in the working memory, by activating patterns of voluntary muscles."

3.1.2.2 The GOMS Family of Models

GOMS is a method for describing a task and the user's knowledge of how to perform the task in terms of Goals, Operators, Methods and Selection rules (John, 1995). Since the concepts underlying the GOMS model are widely used in the HCI field, it is useful to conceptualize those terms.

John says: "A goal can be defined as the state of the system that the human wishes to achieve. A goal, sometimes called an external task, can be achieved using some instrument, method, agent, tool, technique, skill, or generally, some device that enables the human to change the system to the desired state. A task (or more specifically an internal task) is defined as the activities required, used or believed to be necessary to achieve a goal using a particular device. A task is, hence, a structured set of sequential activities that a human has to do in order to accomplish a goal. Finally an action is defined as an internal task that involves no problem solving or control structure component."

In GOMS, goals are what the humans want to accomplish with the software. Since they involve problem solving and control structure, they relate to goals and tasks. Conversely operators are the actions that the software allows the user to perform. The operators usually map to typed commands, menu selection, button presses, gestures or spoken commands in concrete user interface. Methods are well-learned sequences of sub-goals and operators that can accomplish a goal. Finally, if there is more than one method of accomplishing the same goal, selection rules apply, that is, decide what

method to use in a particular circumstance (John, 1995).

3.1.3 Usability principles and rules

Many sources define general rules and principles for usability. Those principles and rules, also called usability heuristics, provide design guides for interactive system development and can be effectively used in discount usability engineering heuristic evaluation (Nielsen, 1993).

In a study of the impact of the explanatory power of usability heuristics, Nielsen performed a factor analysis of the scores of 249 usability problems with respect to a list of 101 usability heuristics (Nielsen, 1994). From this statistical analysis, Nielsen derived a candidate set of 10 heuristics providing a broad coverage of the usability problems found in common interactive applications. This candidate set of heuristics include the following principles:

H1. Visibility of system status - the system should always keep users informed about what is going on, through appropriate feedback within reasonable time;

H2. Match between system and real world - the system should speak the users' language with concepts familiar to the user in a logical and natural order, rather than system-oriented terms;

H3. User control and freedom - provide ways for the user to escape from unwanted states without going through extended dialogues. Supports undo and redo;

H4. Consistency and standards - avoid different terms, situations and actions with the same meaning. Follow platform conventions, standards and guidelines;

H5. Error prevention - design the system carefully to prevent problems from occurring in the first place;

H6. Recognition rather than recall - make objects, actions and option visible. The user should not have to recall information from one part of the dialogue to another. Instructions for use of the system should be visible and retrievable whenever appropriate;

H7. Flexibility and efficiency of use - accommodate different experience levels providing accelerators for expert users that are invisible to novices. Allow users to tailor frequent actions;

H8. Aesthetic and minimalist design - avoid information in dialogues which is irrelevant and rarely needed, hence diminishing the visibility of relevant information;

H9. Helping users recognize, diagnose and recover from errors - error messages should be expressed in simple language, precisely indicating the problem and constructively suggesting a solution.

H10. Help and documentation - documentation and help, when required, should be focused on user tasks, concise and easy to search.

There are five main clusters of heuristics corresponding to the following concerns (Nielsen, 1994):

- The importance of matching the user's conceptual model with the system image, thus leveraging on the previous experience the users acquired in the application domain. This cluster involves the heuristics related to providing good conceptual models, that yield well-structured user interfaces based on clear mappings

between the actions and representations of the user interface;

- Visibility and feedback as major factors enabling users to perceive the system state and the alternatives for action through clear, concise and unambiguous language. This cluster involves the heuristics related to feedback, visibility and closure that enable the users to feel in control of the user interface.

- Flexibility and efficiency enabling the system to accommodate different user experience level. This cluster involves heuristics related to providing infrequent users with simple user interface, while leveraging shortcuts for frequent users.

- Consistency as a way of avoiding different situations, actions and terms with the same meaning. This cluster involves heuristics related to following standards, guidelines and platform conventions that should be reused with purpose to avoid arbitrary consistency.

- Error prevention, tolerance and reversibility of actions as a way to reduce the cost of mistakes, errors and misuse. This cluster involves heuristics related to error prevention, error handling, undoing, redoing and tolerance to varied input sequences.

3.2 Object-Oriented Methodology

Object-oriented programming offers a new and powerful model for developing computer software. The object-oriented methodology speeds the development of new programs, and, if properly used, improves the maintenance, reusability, and modifiability of software. Object-orientation is a new technology based on objects and classes. It presently represents the best methodological framework for software

engineering and its pragmatics provides the foundation for a systematic engineering discipline. By providing first class support for the objects and classes of objects of an application domain, the object-oriented paradigm precepts offer better modelling and implementation of systems. Objects provide a canonical focus throughout analysis, design, and implementation by emphasizing the state, behaviour, and interaction of objects in its models, providing the desirable property of seamlessness between activities (Basic of Object-orientation, 2004).

The contents of Basic of Object-orientation also represented that object-orientation proceeds from the very beginning of the engineering, problem and solution, or needs fulfilment process to the very end and comprises a complete and comprehensive systemic approach replete with a paradigm of techniques, methods, processes, standards, models, notations, tools, components, languages, environments, examples, community, practice, and skills. Its application domain is very large and ranges from real-world application domain modelling and enterprise engineering to the use of polymorphism in programs and the architecture, modelling and implementation of the systems that run beneath them and the distributed systems that connect them. Object-orientation offers a better paradigm, a pattern of practice and thought through which we apply the traditions of the discipline, a means through which we can view the world and a fundamental shift in the philosophies of computer science and engineering, replacing the older paradigm of structured techniques by providing fundamental improvement and superior solution spaces. Object-orientation provides for better modelling of the real world by providing a much-needed improvement in domain analysis and then integration with system design, which is itself improved by the same technique.

3.2.1 Object-Oriented Methodology (HKSAR)

Object-oriented Methodology (OOM) is a new system development approach encouraging and facilitating re-use of software components. With this methodology, a computer system can be developed on a component basis which enables the effective re-use of existing components and facilitates the sharing of its components by other systems. By the adoption of OOM, higher productivity, lower maintenance cost and better quality can be achieved.

The ultimate objective of OOM is application assembly - the construction of new business solutions from existing components. The components are combined in different ways to meet the new requirements specified by the user community. Only completely new functionality will have to be built to complete the solution.

OOM applies a single object model that evolves from the analysis and design stage and carries all the way down to the programming level. An object contains both the data and the functions that operate upon that data. An object can only be accessed via the functions it makes publicly available, so that all details of its implementation are hidden from all other objects. This strong encapsulation provides the basis for the improvements in traceability, quality, maintainability and extensibility that are key features of well-designed Object-oriented systems.

3.2.2 Object-Oriented Design Theory

Object-oriented software is all about objects. An object is a "black box" which receives and sends messages. A black box actually contains code (sequences of computer instructions) and data (information which the instructions operate on). Traditionally, code and data have been kept apart. For example, in the C language, units of code are called functions, while units of data are called structures. Functions and structures are not formally connected in C. A C function can operate on more than

one type of structure, and more than one function can operate on the same structure (Montlick, 1999).

Not so for object-oriented software! In O-O (object-oriented) programming, code and data are merged into a single indivisible thing -- an object. This has some big advantages, as you'll see in a moment. But first, here is why we developed the "black box" metaphor for an object. A primary rule of object-oriented programming is this: as the user of an object, you should never need to peek inside the box (Montlick, 1999)!

Why shouldn't you need to look inside an object? For one thing, all communication to it is done via messages. The object which a message is sent to is called the receiver of the message. Messages define the interface to the object. Everything an object can do is represented by its message interface. So you shouldn't have to know anything about what is in the black box in order to use it (Montlick, 1999).

And not looking inside the object's black box doesn't tempt you to directly modify that object. If you did, you would be tampering with the details of how the object works. Suppose the person who programmed the object in the first place decided later on to change some of these details? Then you would be in trouble. Your software would no longer work correctly! But so long as you just deal with objects as black boxes via their messages, the software is guaranteed to work. Providing access to an object only through its messages, while keeping the details private is called information hiding. An equivalent buzzword is encapsulation (Montlick, 1999).

Why all this concern for being able to change software? Experience has taught us that software changes. A popular adage is that "software is not written, it is re-written". And some of the costliest mistakes in computer history have come from software that breaks when someone tries to change it (Montlick, 1999).

Object-oriented programming (OOP) is a computer programming paradigm in which a software system is modelled as a set of objects that interact with each other. Object-oriented programming emphasizes the following concepts (Object-oriented programming, 2004):

Objects: Packaging data and functionality together into units within a running computer program; objects are the basis of modularity and structure in an object-oriented computer program.

Abstraction: The ability of a program to ignore some aspects of the information that it is manipulating, the ability to focus on the essential. Each object in the system serves as a model of an abstract "actor" that can perform work, report on and change its state, and "communicate" with other objects in the system, without revealing how these features are implemented. Processes, functions or methods may also be so abstracted, and when they are, a variety of techniques is required to extend an abstraction.

Encapsulation: Ensures that users of an object cannot change the internal state of the object in unexpected ways; only the object's own internal methods are allowed to access its state. Each class exposes an interface (of an application programming interface, API) that specifies how other classes may interact with it. This prevents users from breaking the invariants of the class, which is useful because it allows the implementation of a class of objects to be changed for aspects not exposed in the interface without impact to user code. The definitions of encapsulation focus on the grouping and packaging of related information rather than security issues. OOP languages do not normally offer formal security restrictions to the internal object state. Using a method of access is a matter of convention for the interface design.

Polymorphism: Different things or objects can have the same interface or answer the same message (based on message name) and respond appropriately depending on the

thing's nature or type. This potentially allows multiple things to be interchangeable with each other. For example, if a bird receives the message "move fast", it will flap its wings and fly. If a lion receives the same message, it will run with its legs. Both answer the same request, but in ways appropriate to each creature.

Inheritance: Organizes and facilitates polymorphism and encapsulation by permitting objects to be defined and created that are specialized types of already-existing objects - these can share (and extend) their behaviour without having to reimplement that behaviour. This is typically done by grouping objects into classes, and defining classes as extensions of existing classes, and thus grouping classes into trees or lattices reflecting behavioural commonality.

OOP is often called a paradigm rather than a style or type of programming to emphasize the point that OOP can change the way software is developed, by changing the way that programmers and Software engineer think about software.

3.2.3 UML-Based Design Methodology

In software engineering, Unified Modelling Language (UML) is a non-proprietary, third generation modelling and specification language. However, the use of UML is not restricted to model software. It can be used for modelling hardware (engineering systems) and is commonly used for business process modelling, organizational structure, and systems engineering modelling. The UML is an open method used to specify, visualize, construct, and document the artefacts of an object-oriented software-intensive system under development. The UML represents a compilation of best engineering practices which have proven to be successful in modelling large, complex systems, especially at the architectural level (Answers, 2005).

UML succeeds the concepts of Booch, OMT and OOSE by fusing them into a single, common and widely usable modelling language. UML aims to be a standard modelling language which can model concurrent and distributed systems (OMG, 2002).

UML is not an industry standard, but is taking shape under the auspices of the Object Management Group (OMG, 2002). OMG has called for information on object-oriented methodologies, that might create a rigorous software modelling language. Many industry leaders have responded in earnest to help create the standard (Biography.ms).

The primary artefacts of the Unified Modelling Language (Rational Software)

This question can be answered in two ways: first in terms of the artefacts that constitute the Unified Modelling Language (the inside view) and second in terms of the artefacts that users apply to model systems using the Unified Modelling Language (the outside view).

From the inside, the Unified Modelling Language consists of three things:

- A formal meta-model
- A graphical notation
- A set of idioms of usage

The purpose of the meta-model was to provide a single, common, and unambiguous statement of the syntax and semantics of the elements of the Unified Modelling Language. The presence of this meta-model has made it possible for us to agree on semantics, decoupled from the human factors issues of how those semantics would best be rendered. Additionally, the meta-model has made it possible for us to explore ways to make the modelling language much simpler by, in a sense, unifying the elements of the Unified Modelling Language. The graphical notation is the most

visible part of the Unified Modelling Language, and it constitutes the graphical syntax that humans and tools use to model systems. Lastly, the Unified Modelling Language encompasses a set of idioms that describe common usage (by humans) and degrees of freedom (by tools).

From the outside, the Unified Modelling Language encompasses a number of models that can be rendered in a variety of projections:

- Use-case diagrams
- Class diagrams
- State-machine diagrams
- Message-trace diagrams
- Object-message diagrams
- Process diagrams
- Module diagrams
- Platform diagrams

Use-case diagrams are as found in Objectory, and they serve to organize the use cases that encompass a system's behaviour. Class diagrams, derived from Booch and OMT, capture the static semantics of the classes that constitute a system. State-machine diagrams, derived from David Harel, capture the dynamic semantics of a class. Message-trace diagrams, object-message diagrams, and process diagrams, derived from Booch, OMT, and Fusion, capture the dynamic semantics of collaborations of objects. Module diagrams serve to model the development view of a system, whereas platform diagrams serve to model the physical computing topology upon which a system executes.

These are the primary artefacts that a modeller sees, although the method (and tools) provides for a number of derivative views.

The Unified Modelling Language meta-model (Rational Software)

A meta-model is a model of a model.

Think of it this way: Developers create things like class diagrams and object diagrams to model a system under development; this resulting model describes a system. A meta-model is a model of that model.

Meta-models are important, because they can provide a single, common, and unambiguous statement of the syntax and semantics of a model. We began our work on the Unified Modelling Language by starting with a meta-model, because it let us come to rapid agreement about the meaning of things (and the very things themselves) that were to constitute our unification. Thus, our meta-model includes some obvious things like classes and objects (which are isomorphic to the elements of the modelling language) and some nonobvious things like uninterpreted and nonclass decls (which are artefacts of the meta-model).

For most users, the meta-model is invisible (as it should be). It's valuable to us, because it lets us communicate our intended semantics to each other and to tool builders. It also gives us something to throw stones at as we try new modelling problems: If we can model something complex easily, then we know we are on the right path. Similarly, it gives us something to evolve. A large part of our latest work has been making the meta-model simpler (which is a very hard thing to do).

Currently, the meta-model for the Unified Modelling Language is described in a combination of English text and class diagrams using the Unified Modelling Language itself. For those purists out there: Yes, Godel's Theorem does apply, which means that there are theoretical limits to what we can express about the meta-model using the meta-model itself. However, we think combination of text and diagrams strikes the right balance between expressiveness and readability.

Currently, we are working to make this model more formal, in two ways. First, we are writing a number of use cases against the meta-model, which lets us try out common and not-so-common modelling problems. Second, we are writing a number of use cases against the dynamic semantics of the meta-model, using formal logic. This is definitely not for the meek, but again it helps us be sure we are generating models that are self-consistent and precise.

The notation for the Unified Modelling Language (Rational Software)

The Unified Modelling Language notation is truly a melding of the graphical syntax of Booch, Objectory, and OMT. It largely draws from the renderings of each one of these methods, with a number of symbols thrown out (because they were confusing, superfluous, or little-used) and with a few new symbols added.

From the outside, the Unified Modelling Language encompasses the following models:

- Use-case diagrams
- Class diagrams
- State-machine diagrams
- Message-trace diagrams
- Object-message diagrams
- Process diagrams
- Module diagrams
- Network diagrams

Use-case diagrams look pretty much as they do in Objectory.

Class diagrams look much like OMT class diagrams (classes are rendered as rectangles) but with most relationships drawn from Booch.

State-machine diagrams, as developed by David Harel, are the same as in Booch and OMT.

Message-trace diagrams were in all three methods (and have thus been unified). Object-message diagrams are derived from Booch (with the change that objects are no longer clouds, but structured clouds; we also adopt the numbering convention from Fusion). Process diagrams are a new invention, and module diagrams and network diagrams derive from Booch.

Thus, if you are a Booch user, you'll need to get over the use of clouds (rectangles are more space-efficient), but pretty much things look the same. If you are an OMT user, you'll see a number of niggling changes (such as the multiplicity balls replaced with more explicit expressions) but no fundamental changes (though there are a number of additions). If you are an Objectory user, you'll see new notation (because Objectory didn't deal with much graphical modelling post analysis). For all three kinds of users, there will be new things to learn (such as stereotypes and properties) but nothing that a few minutes of reading won't do.

The process associated with the Unified Modelling Language (Rational Software)

It's architecture-driven, and it's incremental and iterative.

Booch, Objectory, and OMT all have well-defined processes, and these are indeed sufficient for the Unified Modelling Language. Beyond that, we don't yet specify a process. Our focus first is on a common, standard modelling language. Processes by their very nature must be tailored to the organization, culture, and problem domain at hand. What works for one context (shrink-wrapped software development, for example) would be a disaster for another (hard-real-time, human-rated systems, for example).

This does not mean that we are blind to the importance of process. We do think we understand what such a process looks like, but it is unnecessary for us to bind the Unified Modelling Language to a particular process, because experience has shown that the modelling languages of Booch, Objectory, and OMT are amenable to a spectrum of different processes. Thus, we have kept in mind which artefacts are important for visualizing different aspects of a system under development and which artefacts are critical for controlling and measuring the progress of a development team. Doing so has been sufficient for us to know what to keep and what to throw out yet still permit diverse organizations to apply the Unified Modelling Language successfully.

3.3 Usability and Software Development

Methodological work about usability engineering and the software development process started in the 1970s. One of the first references to usability engineering methodology was provided by Gould and Lewis (1985) and described an approach involving three global strategies: early focus on users and tasks, empirical measurement, and iterative design. Since this initial approach others have provided general frameworks for usability engineering (Shneiderman, 1998). According to Button and Dourish (1996) those approaches can be distinguished into three categories, regarding their integration with the software development process (Mayhew, 1999):

- Introducing usability experts to design teams with the aim of providing input to the development process;

- Introducing usability engineering methods and techniques that are able to produce specific work artifacts to the development process;

- Redesigning the whole development process around usability engineering expertise, methods and techniques;

Mayhew explained: "From the early focus on introducing usability experts and specific techniques and methods to the development process, a shift to redesign approaches occurred. This shift is consistent with the acceptance that usability engineering requires an active involvement of both usability experts and corresponding methods and techniques throughout the development lifecycle - from early inception to transition."

3.3.1 The Usability Engineering Lifecycle

The usability engineering lifecycle (UEL) (Mayhew, 1999) is one of the most recent and complete approaches consistent with the principle of redesigning the software development process to integrate usability engineering (Nunes, 2001). The UEL consists of several types of tasks, as follows:

- Structured usability requirements analysis;
- Explicit usability goal setting task, driven directly from requirements analysis data;
- Task supporting a structured top-down approach to user-interface design driven directly from usability goals and other requirements data;
- Objective usability evaluation tasks for iterating design toward usability goals;

The usability engineering lifecycle is divided into three phases each one involving a set of tasks (Mayhew, 1999). The lifecycle is illustrated in Figure 3.3, where dependencies between tasks are depicted.

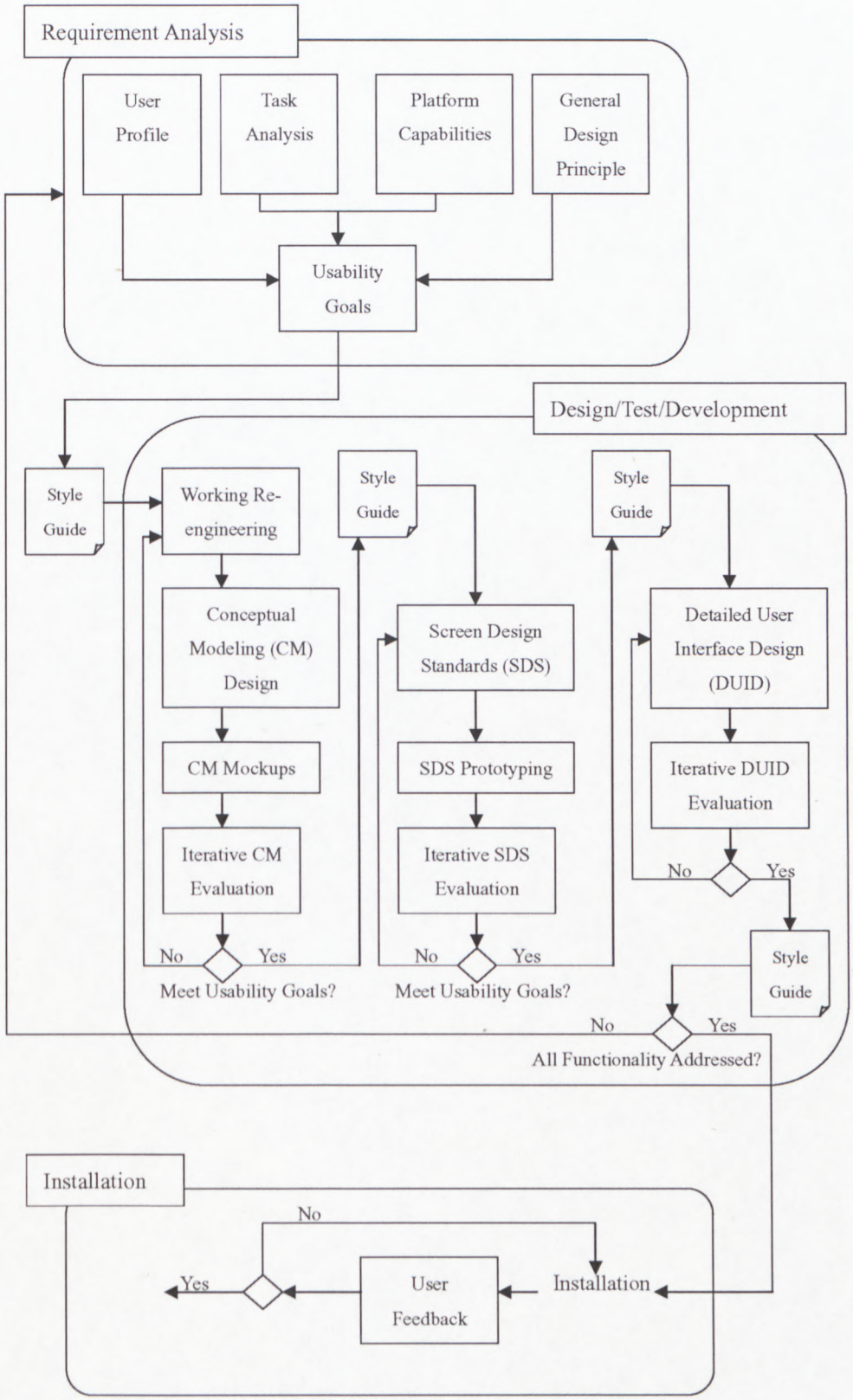


Figure 3.3: The usability engineering lifecycle (Mayhew, 1999)

Phase one is called "requirements analysis" and involves the following tasks:

- User profile - involves obtaining a description of the specific user characteristics relevant for user interface design from the intended user population;

- Contextual task analysis - involves studying the users' current tasks, workflow patterns and conceptual frameworks with the aim of producing a description of current tasks and workflows that underlie the user goals;

- Usability goal setting - this task involves defining qualitative goals reflecting usability requirements extracted from the previous tasks; and also quantitative goals defining minimal acceptable user performance and satisfaction criteria based on a subset of high-priority qualitative goals;

- Platform capabilities and constraints - involves determining and documenting the user interface capabilities and constraints that define the scope of possibilities for user interface design;

- General design principles - involves gathering and reviewing relevant user interface design principle and guidelines.

Phase two is called "design/testing/development" and divided into three levels each one dealing with different design issues. Level 1 is concerned with high-level design issues and contains four tasks:

- Work reengineering - involves redesigning users tasks at the level of the organization and workflow to streamline work and exploit the capabilities of automation;

- Conceptual Model Design - involves generating high level design alternatives

such as navigational pathways, major display and rules for consistently presenting work products, processes and actions;

- Conceptual Model Mock-ups - involves generating paper-and-pencil prototype mock-ups of high level design ideas about organization and conceptual model design;

- Iterative Conceptual Model Evaluation - involves the evaluation of the prototype mock-ups, through iterative evaluation techniques such as formal usability testing, and of real-users attempting to perform representative tasks with minimal training and intervention.

The second level of the design/testing/development phase involves four addition tasks as follows:

- Screen design standards - involves the development of a set of product-specific standards and conventions, for all aspects of detailed screen design, and based on mandated industry and corporate standards;

- Screen design standards prototyping - involves applying the screen design standards, and the conceptual models design to the detailed design of selected subsets of product functionality, implemented as a prototype;

- Iterative screen design standards evaluation - involves evaluating the screen design standards prototype, with an evaluation technique such as formal usability evaluation, and then iteratively redesigning and re-evaluating until all the major usability problem are solved;

- Style guide development - involves capturing in a document the set of standards and conventions validated and stabilized in the previous tasks.

Finally level 3 of the design/testing/development phase involves two tasks, as follows:

- Detailed user interface design - involves designing the complete product user interface based on the conceptual model and screen design standards documents;

- Iterative detailed user interface evaluation - involves evaluating the previously unassessed subsets of functionality, using a technique such as formal usability evaluation, iterating the process until all usability problems are addressed.

The final phase of the Usability Engineering Lifecycle is called "Installation" and involves only one task corresponding to user feedback. This task involves gathering feedback from users working with the product after it has been installed with the aim of resolving prevalent usability problems, enhancing the design or designing new releases or products.

There are also some general principles behind the Usability Engineering Lifecycle approach (Mayhew, 1999). According to Nunes (2001) described, the UEL user requirements and user interface drive the entire development process because it focuses on interactive systems to serve the users. Furthermore, the UEL fully adopts the idea that the integration of usability engineering with software engineering must be tailored in a way that enables different techniques to overlap and flow in parallel with development tasks. In addition, UEL fosters an iterative top-down structured process. Each level of design/test/development phase include an iterative task where evaluation feedback specification and development. This characteristic is supported by a flexible and adaptable allocation of alternative usability methods and techniques, selected in conformance with the characteristics of the project. Finally, the UEL layers the development lifecycle across subsets of functionality, or in other terms, the development process progresses incrementally.

The UEL approach also provides guidelines for integration with object-oriented software engineering (OOSE) (Mayhew, 1999). The integration is based on coupling OOSE models and workflows (Jacobson, 1992) with UEL tasks. Therefore, the user profile and contextual task analysis tasks should parallel the development of the requirements model in OOSE. The work re-engineering task corresponds roughly with the development of the analysis model. Finally, the actual user interface design, which starts with the conceptual model design task, enhances the design model. In the words of Mayhew this kind of integration is necessary because "In OOSE, many fundamental Usability Engineering goals are still not addressed, and many fundamental Usability Engineering techniques are still not applied."

3.3.2 User-centred Design

User-centred design is currently defined in the ISO 13407 (1999) Standard (Human Centred Design Processes for Interactive Systems). User-centred design (UCD) focuses specifically on the process required to build products usable with respect to the definition of usability. The user-centred approach typically entails involving users in the design and evaluation of the system so that feedback can be obtained. Therefore, UCD produces software products that are easier to understand and use; improves the quality of life of users by reducing stress and improving satisfaction; and improves the productivity and operational efficiency of individual users and the overall organization (Daly-Jones et al., 1999).

3.3.2.1 Principles of User-centred Design

The ISO standard for human-centred design processes defines a set of principles that aim at incorporating the user perspective in the software development process. The principles are outlined as follows (ISO, 1999):

- Appropriate allocation of function between user and system, determining which aspects of the job or task should be handled by people, and which can be handled by software and hardware. This division of labour should be based on the appreciation of human capabilities;

- Active involvement of users - utilize people who have real insight into the context in which an application will be used, therefore taking advantage of enhanced acceptance and commitment to the new software;

- Iteration of design solutions - entails the early and continuous feedback of end-users, through different prototyping techniques;

- Multi-disciplinary teams - fostering a collaborative development process which benefits from the active involvement of various parties, each of whom has insights and expertise to share.

3.3.2.2 Key Activates in User-centred Design

According to the ISO 13047 Standard there are four essential user-centred design activities that should be undertaken in order to incorporate usability requirements into the software development process, as follows (ISO, 1999):

- Understand and specify the context of use - the quality of the system, including user health and safety, highly depends upon the context in which the system will be used. Aspects that should be understood about the context of use include the characteristics of the intended users, the tasks the users will perform, and the environment in which the users will use the system;

- Specify the user and organizational requirements - the user-centred requirements should be explicitly stated alongside the technical and functional requirements.

These include, identification of the range of relevant users and other personnel in the design, provision of a clear statement of design goals, an indication of appropriate priorities for the different requirements, provision of measurable benchmarks against which the emerging design can be tested, evidence of acceptance of the requirements by the stakeholders, acknowledgement of any statuary or legislative requirements;

- Produce designs and prototypes - to simulate the design solution using paper or computer based prototypes from the earliest stages of design as well as during later stages. Prototyping enhances communication between the design team and the end-users and provides an inexpensive way to test and explore design alternatives;

- Carry out user-based assessment - to confirm the extent to which user and organizational objectives have been met, as well as provide further information to refine the design. Evaluations should be carried out from the earliest opportunity and involve the following steps: develop an evaluation plan, collect and analyze data, report the results and recommendations for changes, iterate the activity until design (and usability) objectives are met, and finally tack changes, maintenance and follow-up.

3.4 Conclusion

In this chapter, we investigated a suitable methodology and framework of integration of Usability Engineering and object-oriented programming for designing a GUI of an intelligent machine control system. There are good and complete guidelines for providing a proper method about how to design and redesign a GUI application in our project. The step of 'reengineering' in Usability Engineering lifecycle plays a very

important role in our project, because the users' needs are always changing. Object-oriented methodology with UML is an effective method to analysis the system and the tasks.

Chapter 4: Case Study

In this chapter, we take an intelligent exercise machine as a case study. The intelligent exercise machine is composed by three major systems as an intelligent control system, an information system, and an interactive system. The intelligent control system will respond and execute a task or a function of the exercise machine immediately in terms of user's action. Another words, in the exercise machine, the term "intelligent" means designers must consider all users' needs when they are doing exercise in order to control the machine in time. Fail-safe measure and electro-pneumatic control are the core applications in the intelligent control system, but how do we measure the value of fail-safe and the piston position of electro-pneumatic? There needs an information system to manage data which determine "fail safe" happened and piston position changed. And another reason we bring an information system out is to record all users' exercise data for customizing their future exercise plans and items by coaches and doctors. Human-machine interface provides an interactive style between a user and an exercise machine. User communicates the exercise machine by an effective interface which is consisted of input device and output device.

We will describe a proposal of an intelligent exercise machine. A mind map of an exercise unit control is illustrated in Figure 4.1. This idea is derived from "Computer-supported cooperative work (CSCW)" Figure 2.10 in chapter 2. In the section of Environment, we briefly describe the design process of the electro-pneumatic control component. In the section on implementation technology, we present the technology of human-machine interface from input/output to interactive style. In the design section, the system's and task's analysis will be presented firstly. Secondly, semantic network must be taken into account in the whole

information system. Finally, we will design the software features and the flow logic of graphical user interfaces. The end of this chapter, we will evaluate our design with evaluation technique.

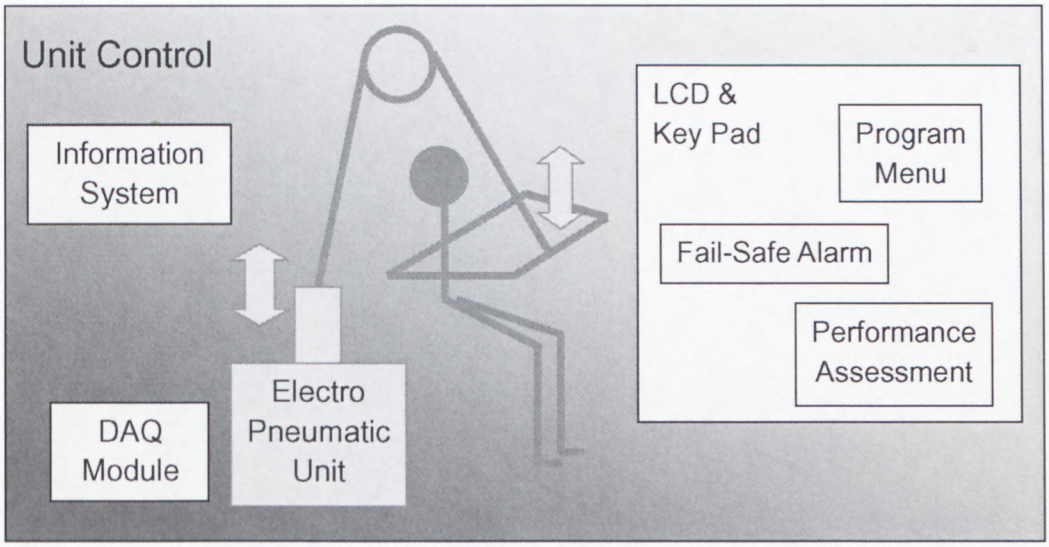


Figure 4.1: A mind map of an exercise unit control

4.1 Environment

4.1.1 The Electro-pneumatic Control Component

T The FX arm and leg components are controlled by software through electro-pneumatic direct control. Similar technology is applied in industry to accomplish high-speed and accurate positioning in repetitive movement.

The principle known as FX-control relies on Differential Pressure Control as represented in Figure 4.2. FX-control can be described as follows:

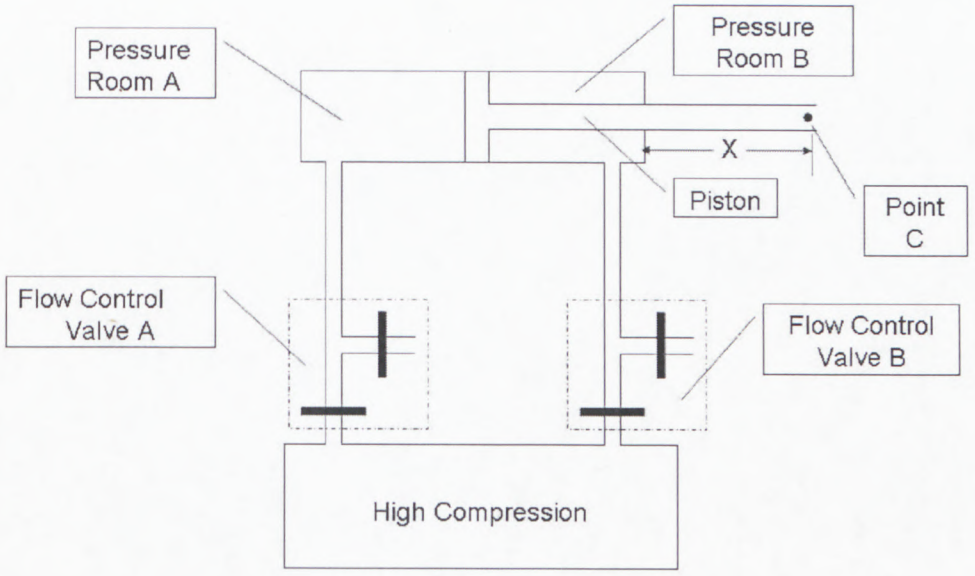


Figure 4.2: Differential Pressure Control Principle

A bidirectional pneumatic cylinder has two chambers, each chamber connected via an electronically pressure control valve to a high pressure reservoir.

Pressure room A can be increased or reduced by Flow control valve A, Pressure room B also can be changed by valve B. When pressure of room A is higher than room B, the piston will be moved to the right by the margin of pressure. Thus the force can be exported via the piston. The valve A and valve B incept different voltage signals from DAQ card to control the different forces.

The situation of the piston must be measured by a distance sensor. The sensor will be fixed in the piston, thus it is able to catch the situation of piston relative to the pressure room. The sensor will export a different voltage signal to DAQ card when it is in a different situation.

In FX control, this is a specific control using a computer-based profile which determines the differential pressure in the cylinder in terms of the spatial position of the mechanical linkage. An Exercise Trajectory Control System uses mathematical

algorithms to deliver the required pressure to achieve the desired force trajectories.

We establish an intelligent machine control structure to control electro-pneumatic to export force and catch the situation of piston, the relation of the physical part is showed in Figure 4.3. From Figure 4.2 we can see the valve A and valve B incepted different voltage signals from DAQ card to control the different forces which depend on exerciser's needs. So we acquire knowledge from "SMPA approach" (Figure 2.4) in chapter 2. In figure 4.3, a loop links sensor generating feedback information of motion control from an exercise machine, DAQ card processing the feedback and conveying the next task, Actuator receiving the new task and transforming the signal to electro-pneumatic, and Exercise machine executing motion control. The electro-pneumatic is set up in the exercise machine.

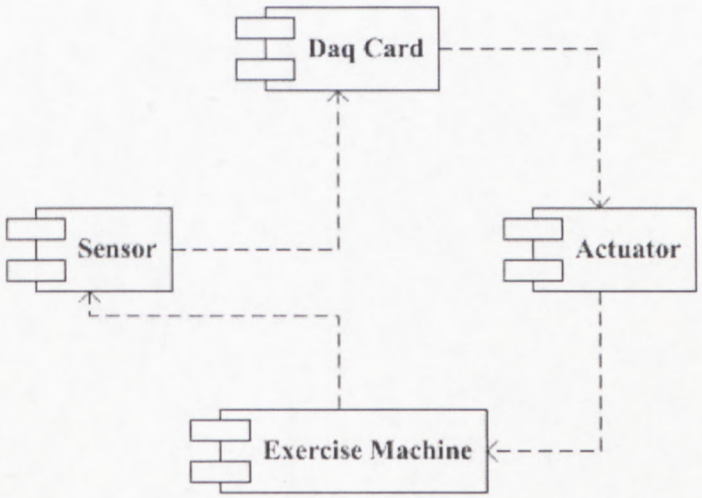


Figure 4.3: The connection of physical part

4.1.2 The Formulation of FX-control

For exercise special muscle with dynamic resistance exercising, a formulation of FX-control must be established. It supplies a function to exercise special muscle through offer difference force at difference situation.

At first a series of mode should be developed to exercise special muscle for different exercisers, this work must be done by coacher, doctor etc. It is able to be upgraded when other new mode is developed. In this case, the possible mode is $A()$, the general formula of FX-control is:

$$F() = A(x) \tag{1}$$

where $F()$ is the exported force,

x is the real-time position of the arm or leg which needs to be determined.

Then for individual, according to different intensities and ranges of exercisers, some parameters should be considered. Therefore, the formula of FX-control now is:

$$F() = C + B \times A(x/y) \tag{3}$$

where $F()$ is the exported force,

C is the initial force which can be given by coacher or exerciser himself,

B is a multiple coefficient,

A(x/y) is the exercise mode given by coacher or doctor,

x is the real-time position of arm or leg which needs to be determined,

y is the maximum position of arm or leg which is a known parameter.

The derivation of equation (3) is as following:

As mentioned above, equation (1) is:

$$F() = A(x) \tag{1}$$

To consider different exercise ranges, a real-time position of arm or leg (x) is changed to be a comparative position (x/y).

Substitute (x/y) to equation (1) to get equation (2):

$$F() = A(x/y) \tag{2}$$

To consider different exercise intensities, the mode $A()$ will multiple a coefficient (B) and plus a initial force (C) to get exported force $F()$. Thus equation (3) becomes as:

$$F() = C + B \times A(x/y) \tag{3}$$

In this case, one formula also can be used repeatedly and continuously, it will be shown in the way as *D-F()*, where D is the number of repeating one action.

In one exercise, different formula of FX-control is able to be used. In a plan of exercise, the formula is listed below:

$$D_1 - F_1() = C_1 + B_1 \times A_1(x/y)$$

$$D_2 - F_2() = C_2 + B_2 \times A_2(x/y)$$

...

$$D_n - F_n() = C_n + B_n \times A_n(x/y) \quad (4)$$

The formula of mode must be installed in Function module, the function module is an upgradeable module, and the mode of exercise can be increased.

4.2 Human-machine Interface Technology

4.2.1 Input

Input is concerned with recording and entering data into the computer system and issuing instruction to the computer (Preece, 1995). In order to interact with computer systems effectively, users must be able to communicate their intentions in such a way that machine can interpret them. Therefore, we can define an input device as: a device that, together with appropriate software, transforms information from the user into data that a computer application can process.

One of the key aims in selecting an input device and deciding how it will be used to control events in the system is to help users to carry out their exercise safely,

effectively, efficiently, and enjoyable. The choice of input device should contribute as positively as possible to the usability of the system. In general, the most appropriate input device will be the one that (Preece, 1995):

- matches the physiological and psychological characteristics of users, their training and their expertise.
- is appropriate for the tasks that are to be performed.
- is suitable for the intended environment.

He also presented that frequently the demands on the input device are conflicting, and no single optimal device can be identified: trade-offs usually have to be made between desirable and undesirable features in any given situation. Not only must an input device be easy to use and the form of input be straightforward, there must also be adequate and appropriate system feedback to guide, reassure, inform, and correct user's errors. This feedback can take various forms. It can be a visual display on a screen: a piece of text appears, an icon expands into a window, or a complete change of screen presentation occurs. It can be tactile: the feel of a button being depressed, or a change in pressure. In many cases feedback from input can be a combination of visual and tactile responses.

Input device: keyboard

The most common method of entering information into the computer is through a keyboard (Preece, 1995). Broadly defined, a keyboard is a group of on-off push buttons, which are used either in combination or separately. Such a device is a discrete entry device. These devices involve sensing essentially one of two or more discrete positions which are either on or off.

When considering the design of keyboards, both individual keys and grouping arrangements need to be considered. The physical design of keys is obviously

important (Preece, 1995). For example, if keys are too small this may cause difficulty in locating and hitting chosen keys accurately.

Response Time and Display Rate

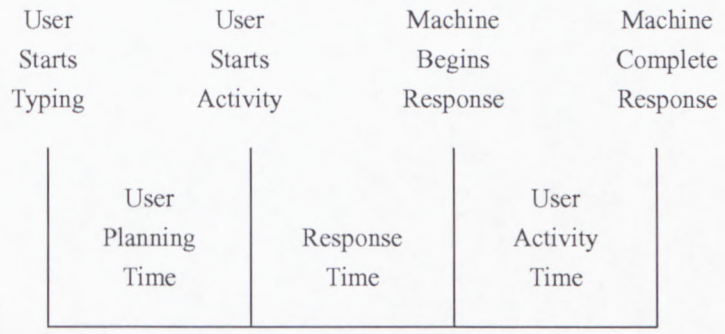


Figure 4.5: Response Time

There are two important things about response time (see Figure 4.5).

1. The exerciser starts to type before the activity that is being requested is initiated.
2. The machine's response is neither instantaneous nor all at once.

Design of user interface must take this into account.

The challenge then is for the system to have long term memory and present the short term needs "Just in time" so that exerciser does not need to maintain information in long term memory.

Long delays in system response can cause the exerciser to forget his plan with a subsequent further slowdown.

On the other hand if the exerciser does too fast, he is more prone to errors brought on by excessive haste. There is some balance between speed and accuracy that needs to be maintained.

4.2.2 Output

According to Preece (1995), Output devices are those devices that convert information coming from an electronic, internal representation in a computer system into some from perceptible by a human, which is known as output. Until recently almost all computer output was visual and two-dimensional; it was presented on a screen or as hard copy from a printer. These are, and are likely to remain for some time, two of most common output devices. However, recently, various new trends and possibilities in the managing of output have started to emerge. For example, Graphical user interfaces (GUIs) have become commonplace.

Visual output

Visual display of text or data is the most common form of output. We expect to see output displays in most interfaces, except for tasks where visual attention has to be focused elsewhere, such as operating a machine. The key design considerations are that the information displayed be legible and that it be easy for a user to locate and process.

Visual feedback on user actions

Users need to know what is happening on the machine's side of an interaction. In particular, a system needs to provide high quality, timely responses to keep users well informed and feeling in control (Preece, 1995). And he said this includes providing both information about normal processes and warnings if there is a problem. For example, in a way similar to the manner in which human communication progresses, where possible, a system should be able to:

- Tell the user where he is in a state or process,

- Indicate how much progress through a process has occurred,
- Signify that it is the user's turn to provide some input,
- Confirm that input has been received,
- Tell the user whether input just received is unsuitable.

If feedback is not current, correct or clearly expressed a user may think or do the wrong thing. Moreover, if there is no feedback the user will be left wondering what is happening of where a user is in the state on which he is exercising. In the intelligent control system of exercise machine, the user's data must be captured, and the data which speed up changes over time is dynamic display in LCD screen. It can be auditory: an indicator of alarm warning from green to red.

4.2.3 Human Factors

Dix et al. (1998) pointed out that Ergonomics (or human factors) is traditionally the study of the physical characteristics of the interaction: how the controls are designed, the physical environment in which the interaction takes place, and the layout and physical qualities of the screen. A primary focus is on user performance and how the interface enhances or detracts from this. In seeking to evaluate these aspects of the interaction, ergonomics will certainly also touch upon human psychology and system constraints.

Sets of controls and parts of the display should be grouped logically to allow rapid access by the user. This may not seem so important when we are considering a single user, but it becomes vital when we turn to safety-critical applications such as exercise machine control. Users are under pressure and are faced with a huge range of displays and controls. Here it is crucial that the physical layout of these be appropriate.

We have already touched on the importance of grouping controls together logically

(and keeping opposing controls separate). The exact organization that this will suggest will depend on the domain and the application, but possible organizations include the following (Dix et al., 1998):

- Functional controls and displays are organized so that those that are functionally related are placed together;

- Sequential controls and displays are organized to reflect the order of their use in a typical interaction;

- Frequency controls and displays are organized according to how frequently they are used, with the most commonly used controls being the most easily accessible.

In addition to the organization of the controls and displays in relation to each other, the entire system interface must be arranged appropriately in relation to the user's position.

4.2.4 Interaction styles

According to Preece (1995) indicated that to make sense of different interaction styles, it is useful to take a historical perspective. Consider, for example, early command-driven applications and form-fill applications. They matched the user and task requirements at the time relatively well. The early command-driven applications tended to be used by expert users or at least technical, knowledgeable people who were not afraid of computers and could be expected to overcome any obstacles by sheer perseverance. The development of the dialogue mode of interaction was aimed at a completely different set of users and tasks. This type of interface was designed using dialogue modelling to enable users to carry out their tasks. We will indicate dialogue modelling in the section of software testing and evaluation and apply the

standard interaction objects typically found in GUI builders, toolkits against the GUI of the intelligent exercise machine.

4.3 Software Design

In this section, we will present a design process which includes system analysis, task analysis, semantic network, software features and flow logic for GUI. In order to describe the design of human-machine interface, we must also take an information system into account. A proposed intelligent semantic machine control system of our whole project is shown as below (Figure 4.6):

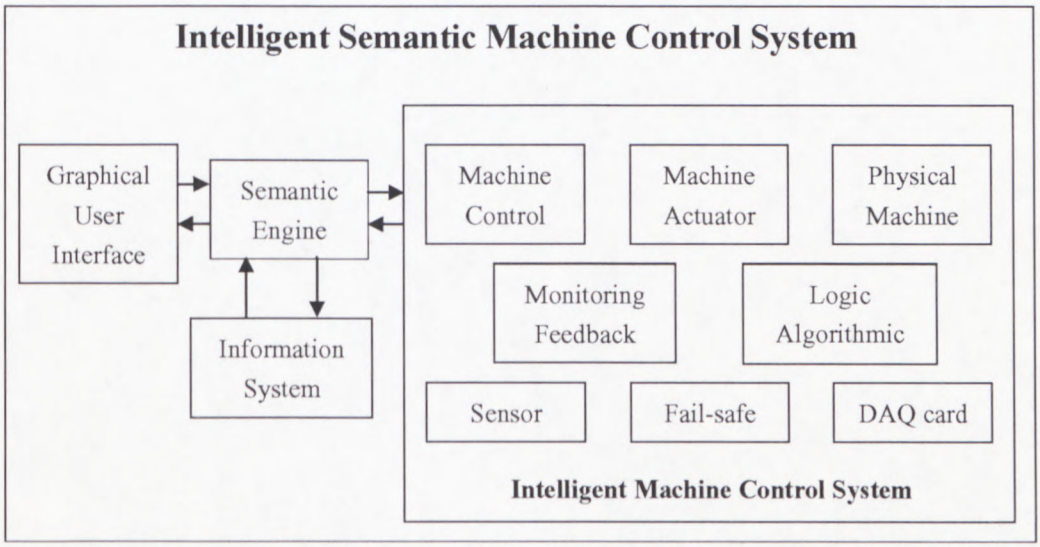


Figure 4.6: A Proposed intelligent semantic machine control system

In Figure 4.6, we draw the knowledge from "Concept of a teleoperation system" (Figure 2.9) and "The general interaction framework" (Figure 2.11) in chapter 2. In Figure 2.9, the concept of a teleoperation system is suitable for our project which the loop line indicated user sends a task to a robot or a machine and gets the feedback information from it. In Figure 4.6, the semantic engine/network, as the information of articulation, performance, presentation, and observation in Figure 2.11, plays very important role in the central position of the project. The information system records

and manages all users' profiles and their exercise data in order to evaluate their health status. The GUI provides users' hierarchical interfaces in terms of the hierarchical access control. The user interface must be user-centred thinking. However, we should also take the capability of the system into account. Based on the usability of the system, we maximize to consider all users' needs.

4.3.1 System Analysis with Object-oriented and UML

4.3.1.1 Unified Modelling Language

In software engineering, Unified Modelling Language (UML) is a non-proprietary, third generation modelling and specification language. However, the use of UML is not restricted to model software. It can be used for modelling hardware (engineering systems) and is commonly used for business process modelling, organizational structure, and systems engineering modelling. The UML is an open method used to specify, visualize, construct, and document the artifacts of an object-oriented software system under development. The UML represents a compilation of best engineering practices which have proven to be successful in modelling large, complex systems, especially at the architectural level (UML, 2005).

UML succeeds the concepts of Booch, OMT and OOSE by fusing them into a single, common and widely usable modelling language. UML aims to be a standard modelling language which can model concurrent and distributed systems.

UML is not an industry standard, but is taking shape under the auspices of the Object Management Group (OMG). OMG has called for information on object-oriented methodologies, which might create a rigorous software modelling language. Many industry leaders have responded in earnest to help create the standard.

4.3.1.2 Creating use case diagram

The first step is to specify with a use case diagram what users desire such that different users can use various functions under different execution environments that best fit what they really need. Six concepts that result in the extensions on UML are employed in creating a use case diagram suitable for our interactive system (Lin and Lee, 2004). As illustrated in Figure 4.7, we presented these concepts include classifying users, categorizing users' execution environment, building client unit, and specifying trigger event, customization, and personalization as follow.

- **Classifying users**

Classifying users is a generic concept in our user management. When designing the interactive system, we should comprehend first who gets benefits from the functions and features of the system. Therefore, it is better to classify prospect users of the system into different kinds in advance before the design work gets started. Therefore, as shown in Figure 4.7, we classify two kinds of general users which are general exercisers and general administrators in our project. The general administrators can be further classified into some special kinds of administrators as users' profile and exercise data management, users' status record management, machine monitoring, and machine functions upgrading. The special kinds of exercisers are the way of self exercise and exercise with remote coaches. As a common knowledge for such a resultant generalization-specialization structure, a lower-level kind of users could possess (i.e. inherit) the characteristics of its parent kind(s) of users.

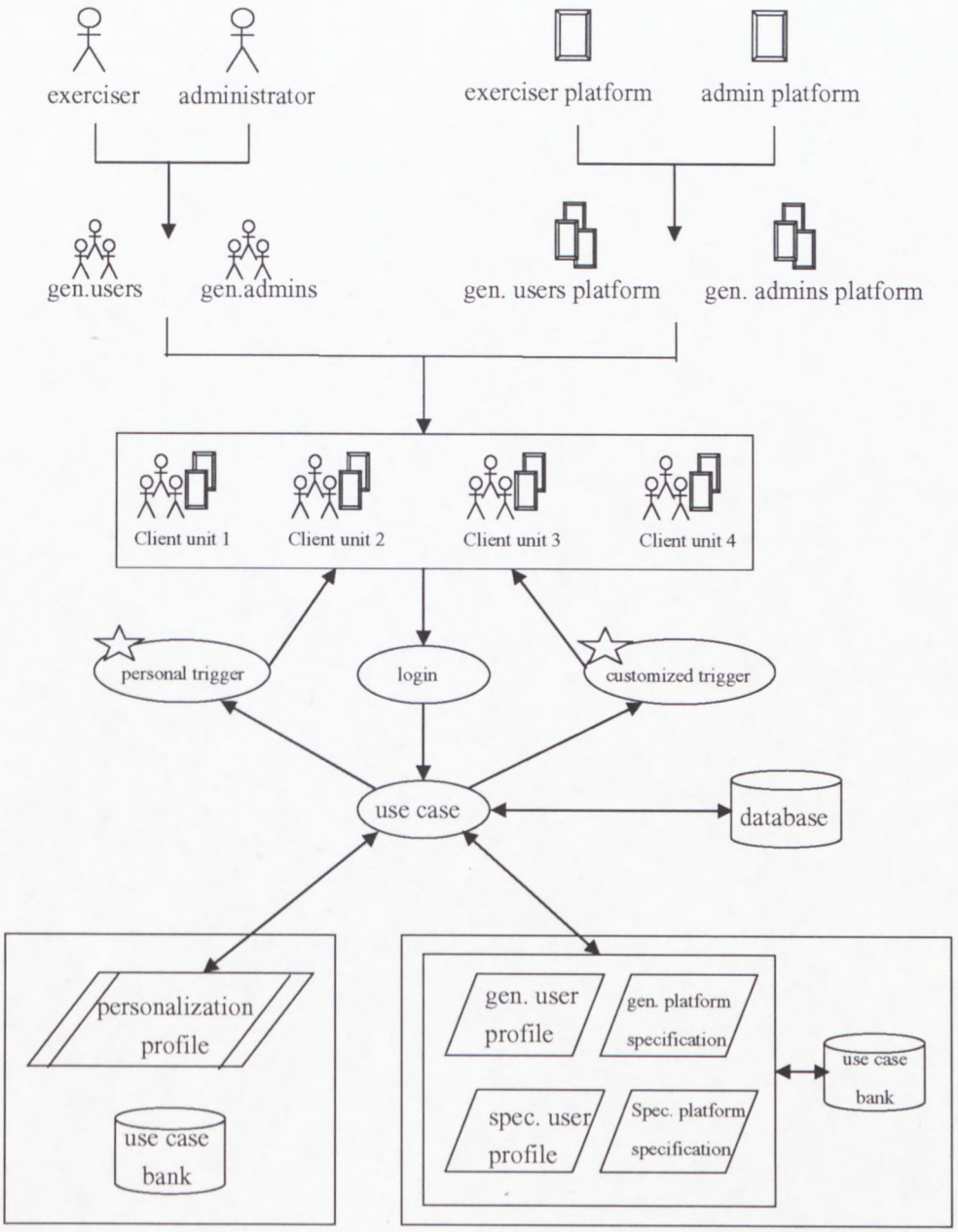


Figure 4.7: use case diagram

- **Categorizing users' execution environment**

Many media can be used for providing users with effective interfaces. Since the hardware and operating systems (i.e. execution environments, or say, platforms) that support these media are different, it is needed when designing our system to consider all possibly preferable execution environments that prospect users of the system may use, and then categorize them into some special kinds as if these users may be classified into corresponding special kinds. As above, this also results in a generalization-specialization structure: general execution environments are matched to the general users by way of general exercisers' platforms and general administrators' platforms where each one is further categorized into the special kinds of exercisers' and administrators' platforms.

- **Building client unit**

Client unit is a mixture of classified users and categorized execution environments. For instance, in Figure 4.7, four kinds of client units can be built up by the mixture of two kinds of users and two kinds of platforms. In client unit 1, in particular, it specifies the situation that general users will use general (all available) platforms. For this most general situation, of course, we need to make the system support well all available platforms possibly used by each prospect user. In contrast, client unit 3 specifies that a special kind of users will use a particular kind of platforms. For this most special situation, the system should be designed more focusing on how to provide these special users with desired and convenient functions under such a special kind of execution environments.

- **Specifying trigger event**

Trigger event is a special use case that enacts users to accomplish some specific operations that may result further in other use cases to be performed. As shown in Figure 4.7, a customized trigger event is specified to model such a desired service for a kind of users, and a personal trigger event is specified for a specific user of this

kind.

- **Specifying customization**

With various kinds of client units through the categorization of users and platforms, the system is then able to offer customized functions under preferable platforms for different kinds of users. The objective of customization focuses on special kinds of users and the users will be offered with these customized functions under their platforms, for example, in our case, the exerciser can choose the customized mode as 'Exercise with remote coach'. Therefore, the system purposefully offers different functions to its users based on their characteristics and expected platforms. As shown in Figure 4.7 that four kinds of client units are specified, the system is therefore customized to have four kinds of profiles or specifications that provide guidance for determining suitable use cases for these four kinds of client units, respectively.

- **Specifying personalization**

After achieving customization, the system may be further refined as a personalized mode if it allows users to designate their preferable functions by themselves. For this purpose, the system, as shown in Figure 4.7, is specified with a personal profile for each specific user that records the functions (use cases) designated by this user.

4.3.1.3 Creating activity diagram

As defined in UML, an activity diagram can be used to specify the possible workflow of system functions. In this step, we can use it to specify the workflow of the customization and personalization processes presented in the previous step. As shown in Figure 4.8, we follow the six concepts used above in creating a use case diagram to create our UML activity diagram with the adapted mechanisms described below.

In Figure 4.8, a rectangle with solid lines is used to hold use cases to be selected by system users. For the rectangles with dotted lines, they specify a range that holds use cases obtained from such resources as use case bank and customization/personalization profiles. For example, use cases in the top rectangle are obtained from the use case bank; the two profiles that fill in the middle and bottom rectangles will control these use cases to determine if they are retained during the processes of customization and personalization.

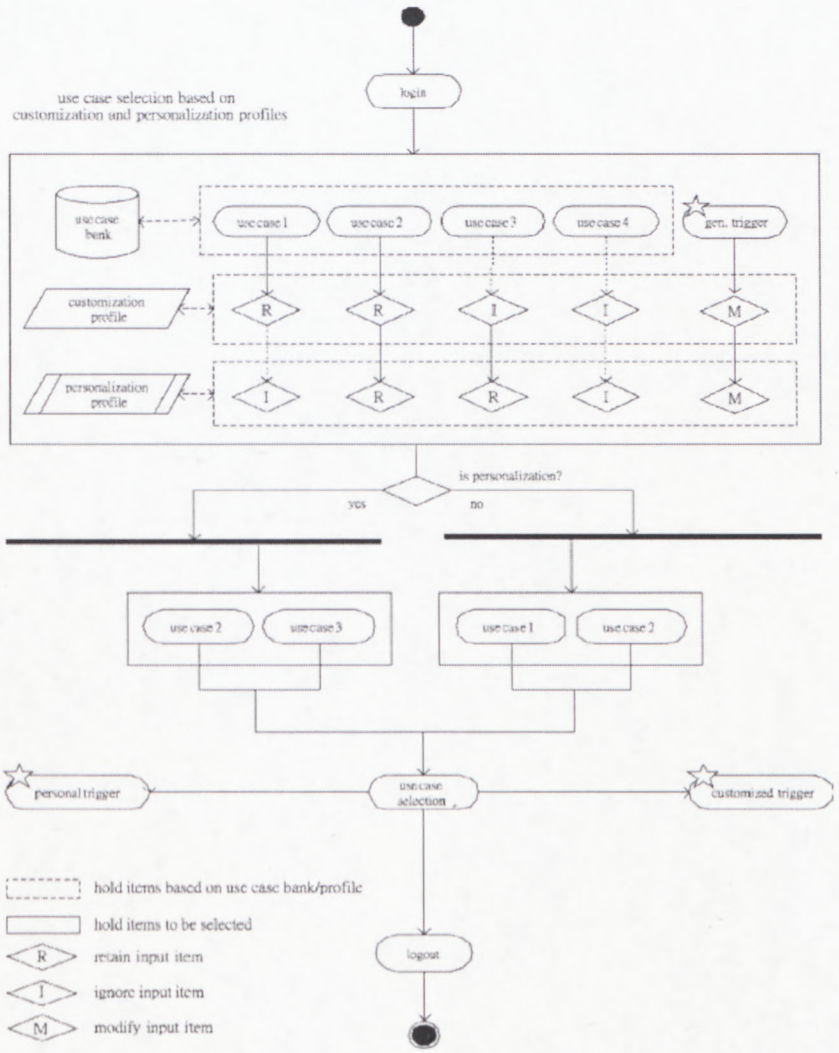


Figure 4.8: Activity diagram

- **Specifying the workflow of customization and personalization**

Here in the activity diagram, customization and personalization processes are

modelled by first retrieving use cases from the use case bank. There are two kinds of use cases, exercisers' use cases and administrators' use cases, in our project. Two decisions of these use cases will be made consecutively based on the customization and personalization profiles respectively, and there are three possible choices to select at each decision: (1) retain the use case, (2) ignore the use case, and (3) retain but modify (part of) the use case. The arrows entering decisions represent selections on relevant use cases: a solid arrow means that the relevant use case is retained (with or without modification) at the customization level for a prospect kind of users or at the personalization level for a specific user of this kind; a dotted arrow means that the relevant use case is ignored at the customization level since this kind of users do not need it, or at the personalization level the specific user does not need it. For illustration in Figure 4.8, use cases 1 and 2 are retained, together with the general trigger event that is modified as a customized one, at the customization level for a kind of users; then, at the personalization level, use cases 2 and 3 are retained, together with the customized trigger event that is modified as a personal one, for a specific user of this kind; these results are specifically shown at the lower part of Figure 4.8.

4.3.1.4 Modelling class diagram

A class diagram is used to describe any system-internal objects/entities that collaborate together to support desired system behaviours (support the workflow of the customization and personalization processes). In UML, it can be derived from the use case diagram with three stereotypes: boundary, entity, and control classes - a boundary class represents an interface used to interact the system with an actor as a bridge; an entity class models the information and associated behaviours in the real world; and a control class provides the desired behaviours for accomplishing one or several use cases (Lin and Lee, 2004).

In addition to these three kinds of class, however, a new kind class of 'trigger control'

is introduced in the class diagram. This is because in Lin and Lee's use case diagram, trigger event is a special use case that can be used in an active or passive way to make other use cases to be performed. That is, in addition to the three UML stereotypes, we represent a particular control class that is responsible for the execution of a trigger event. Figure 4.9 shows the class diagram derived from the use case diagram in Figure 4.7. It is noticed that as shown in Figure 4.9, various relationships may occur between classes, such as association and inheritance relationships. As a common recognition for object-oriented paradigm, these relationships are particularly useful for making the system constructed much easier to understand, maintain, and reuse.

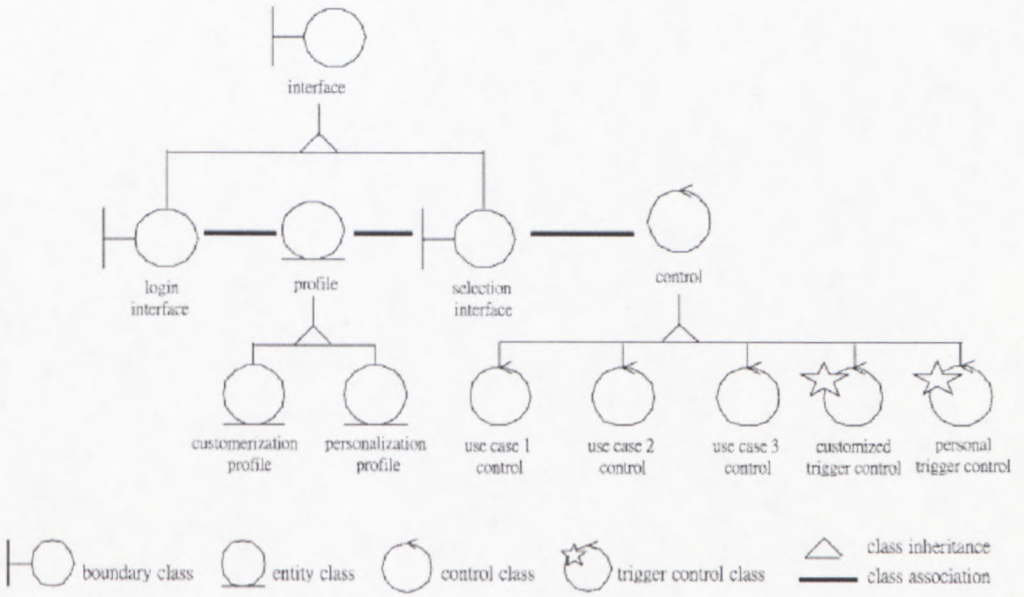


Figure 4.9: Class diagram

4.3.1.5 Modelling sequence diagram

With classes defined and specified for creating objects to support desired system behaviours, it is now possible to depict a sequence diagram that specifies how such objects collaborate to support these behaviours. Figure 4.10 is Lin and Lee's sequence diagram where there are three differences from the traditional UML sequence diagram: (1) client unit is used that replaces actor; (2) trigger control object is specifically used for accomplishing the execution of a trigger event in an active or passive way; and (3)

There is considerable overlap between task analysis and system analysis, as both attempt to define functions needed by the system. However, the difference between the two is significant. The system analysis addresses major functionalities within the entire organization. On the other hand, task analysis addresses the tasks of individual users and human-machine interaction at their level. Personal information processing through the interface is the major interest of task analysis (Wang, 1995).

According to Wang's opinion, there are similarities shared by the task analysis approaches: specification of human knowledge in terms of task structure and users' activities to accomplish the task though the human-machine interface. Accordingly, three types of formal descriptions are fundamental for task analysis: task, user, and interface descriptions.

Before object-oriented models for these are presented, our position of "analyst/designer's view" must be clarified. In the human-computer interaction field, researchers usually differentiate "user modelling" from "designer modelling", however, for the purposes of system analysis and design, the user's conceptual model of the system has to be acquired and transformed into a designer's model (Wang, 1995).

4.3.2.1 Task descriptions

A central part of task analysis is to provide task descriptions for an extant human-machine interactive system. One of the most popular approaches to task descriptions is hierarchical task analysis (HTA) (Annett and Duncan, 1967). It was intended this to use a general approach to task descriptions and has been applied to a wide range of solutions (Shepherd, 1989). Using HTA, a task is formally described as a hierarchical structure, and a task can be represented by a tree of subtasks. In human-machine interactive systems, a primitive subtask could be a machine subtask or human subtask, as shown in Figure 4.11. A machine subtask is a machine procedure.

dotted arrows pointed to a rectangle are used to specify which objects are required for customization and personalization— those (boundary and control) objects in the rectangle are needed to support customized behaviours (customized use cases) for a prospect kind of users or personalized behaviours (personalized use cases) for a specific user of this kind. As illustrated in Figure 4.10, after a client unit logs in the system, the login interface object will verify the customization and personalization profiles to determine which objects are required for desired customized and personalized behaviours: control objects for use cases 1 and 2 and the customized trigger event are required to support the customized behaviours, and control objects for use cases 2 and 3 and the personal trigger event are required to support the personalized behaviours. With required control objects, use cases 1 and 2 and the customized trigger can then be performed to support the customized behaviours for a prospect kind of users, and use cases 2 and 3 and the personal trigger can be performed to support the personalized behaviours for a specific user of this kind.

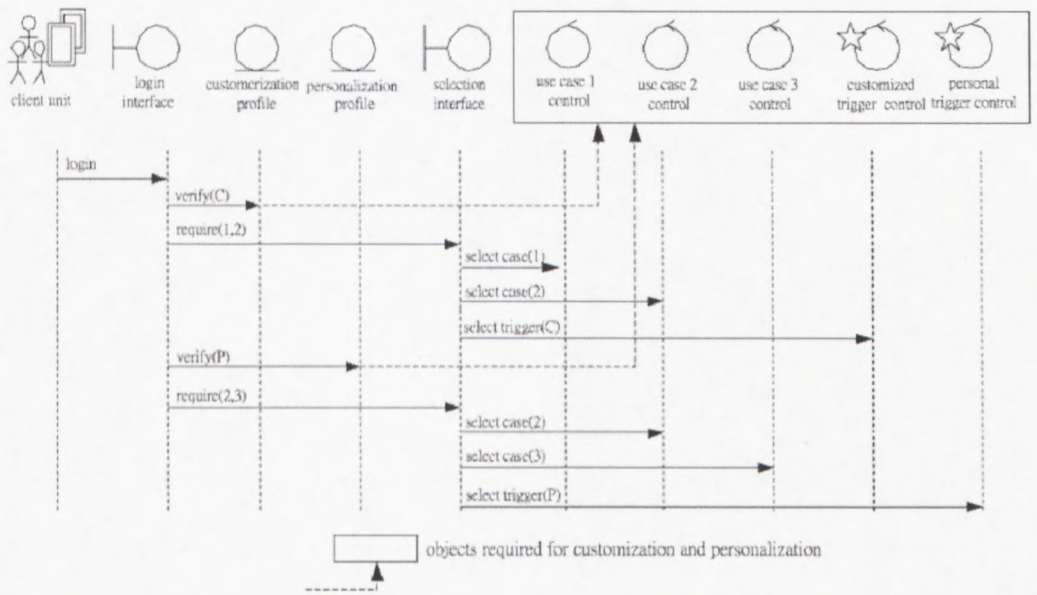


Figure 4.10: Sequence diagram

4.3.2 Task Analysis and Object-Oriented Perspective

A human subtask could be physical-motor operators, such as keystrokes and memory activities. In a highly interactive human-machine environment, when the user presses a keystroke or input the data, the machine will accept and record it and must quickly respond to execute the task or change the machine status. For my purpose, inputting data are defined as human tasks supported by machine procedures and semantic network. This is illustrated in Figure 4.11.

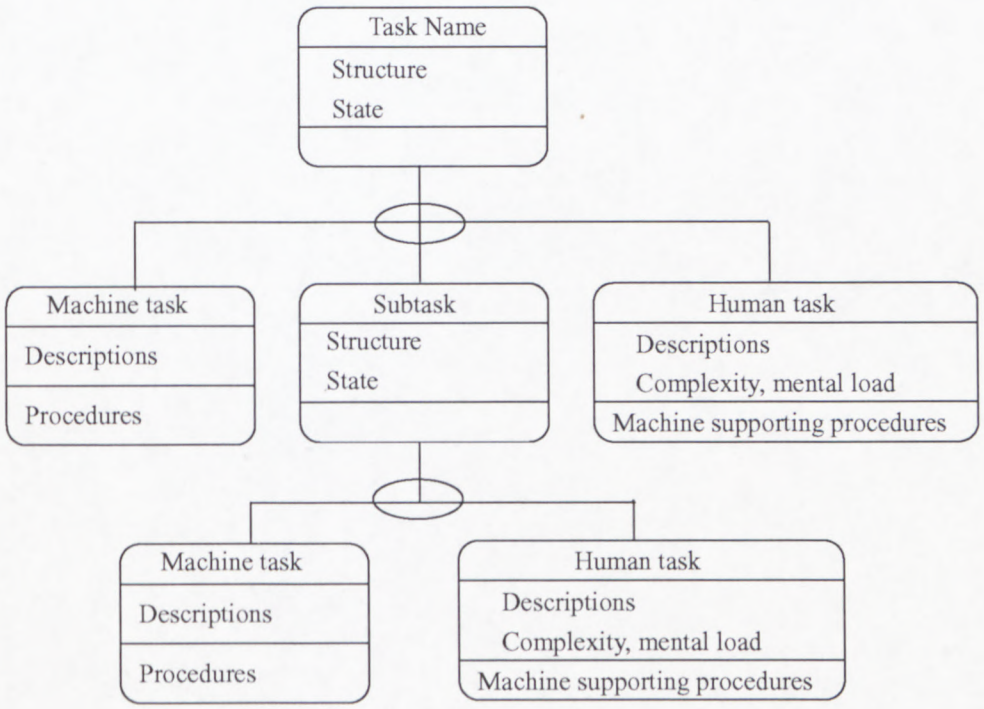


Figure 4.11: Task objects

Research (Walsh, 1989) has argued that task analysis should be based upon principles that have increased the usability of system specifications. Therefore, if object-oriented system specifications are applied, a task or a subtask is an object, and a task structure is a type of assembly structure. In this study, it is assumed that a task object can have more than one assembly structure to meet the requirements for designing a human-machine interface.

Two characteristics of object specifications make the representation of a task unique

to others in task analysis. First, since a task (or subtask) is an object, it possesses its own operations (the set of machine procedures to accomplish it), while a human task object contains a set of human operators and machine support procedures. Second, a task object can send messages to trigger other objects; it can also be triggered by other objects. Consequently, in the object-oriented frame, the structure of a task is dynamic. Such a dynamic structure is different from the traditional static structure. Accordingly, the dynamic method makes the representation of a task structure more flexible than the traditional static method of designing the human-machine interface.

4.3.2.2 User descriptions

One of the major activities of task analysis is to analyze user characteristics in order to provide more information about the users for task design and strategic choice of interface. Many classification schemes use dimensions of exercisers' skill and expertise in the system, for example novices, experts, and intermittent users. However, there is little consensus about what characteristics should be described. On the other hand, cognitive models describe the human behaviour during information processing. Among the human cognitive models, GOMS is one that is widely used. In this, the user's cognitive structure has four sets of components: goals, operators, methods for achieving the goals, and selection rules for choosing among competing methods for goals. A method is a conditional sequence of goals and operators, with conditional tests of the contents of the user's immediate memory and of the state of the task environment. Thus, method descriptions play a central role in this model, and they are the core of user descriptions in task analysis. Because the term "method" has its own particular meaning in the object-oriented community, and to avoid confusion, "task solver" is used instead of "method" in the GOMS model (Wang, 1995).

Wang (1995) described that a task solver can be represented as an object. A major component of the attributes of a task solver object is a goal stack, which retains the step-by-step behaviour of the user in reaching a top level goal. In the operation part, a

task solver object is assumed to have a generic processor for goal stack processing. The operation part of each task solver object also contains a set of processes called "operators" in the GOMS model. From the point of view of object-oriented analysis, those operational processes are messages to task objects. The operation part of a task solver object usually also contains a set of selection rules that control the user's decision path, while accomplishing a top level goal. An example of task solver object corresponding to a goal of EXECUTE-UNIT-TASK is shown in Figure 4.12(a), and an object-oriented frame for user descriptions in TA is shown in Figure 4.12(b).

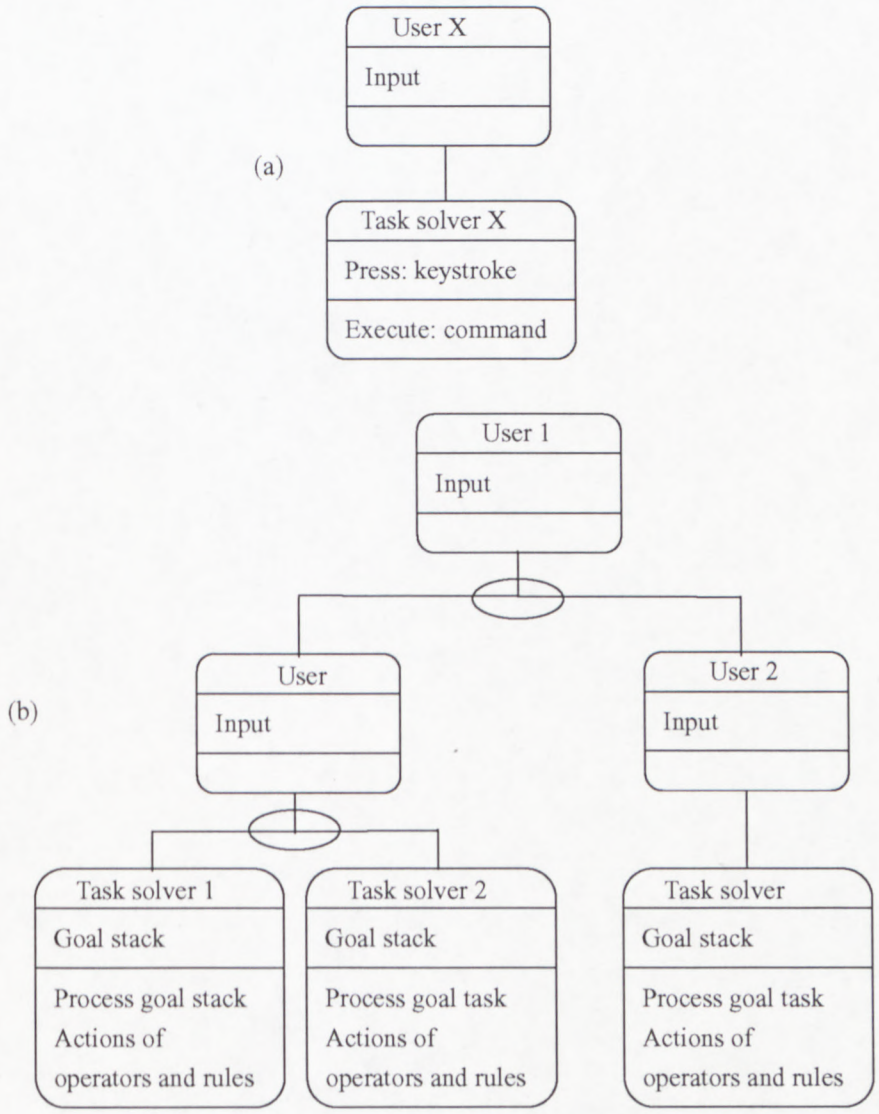


Figure 4.12: User objects

4.3.2.3 Interface descriptions

Interface descriptions specify the dialogue between user and machine. Presentations of information to the user and request for input from the user are the two aspects of human-machine interface from the machine side (Wang, 1995). In a contemporary computer programming language, the programming environment often provides high level interface implementation functions to allow easy development of a variety of interface formats for the user. In the present object-oriented context, interface descriptions for the project are organized into a structure, as shown in Figure 4.13. This structure is a type of classification structure. There are two subclasses of the general interface object of a particular information system, one is requests for input, and the other is information presentation. Each subclass in turn has its subclasses of interlace objects. An interface object possesses its own values of attributes and its event driven operations.

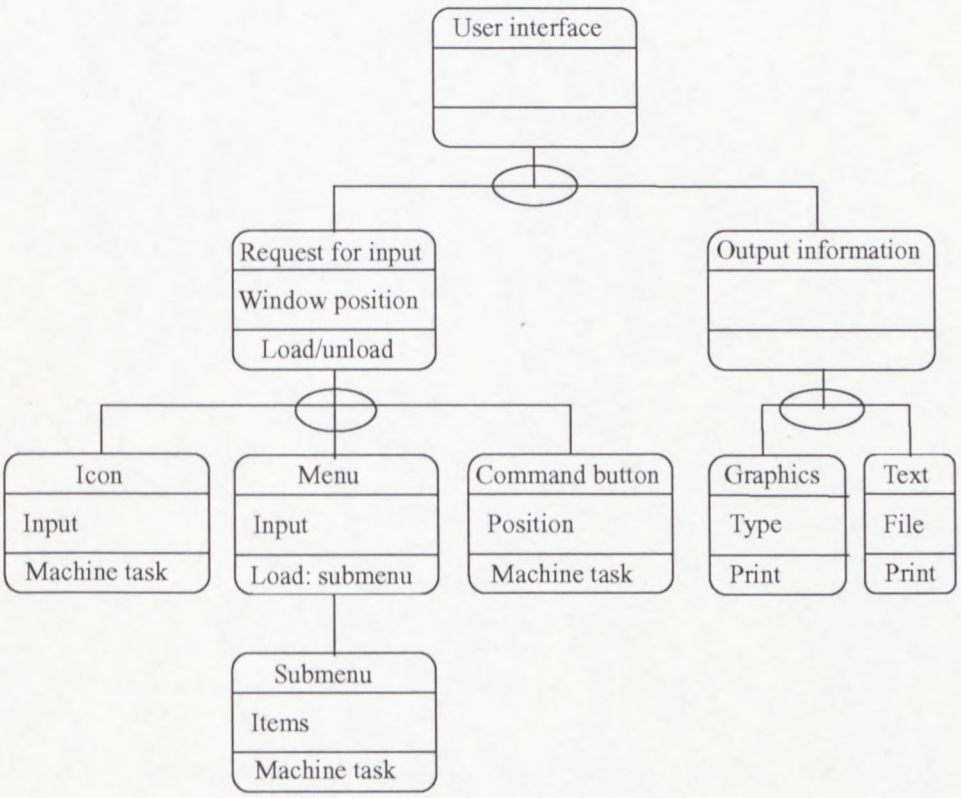


Figure 4.13: Human-computer interface objects

4.3.3 Semantic networks

A semantic network is "a structure for representing knowledge as a pattern of interconnected nodes and arcs" (Sowa, 1991). As a psychological model of human associative memory, semantic networks are very general (Quillian, 1968).

A semantic network is made up of nodes and arcs (Nault and Storey, 1998). Nodes represent concepts of entities, attributes, events, and states. Arcs are conceptual relations representing relationships between the concept nodes. The nodes may be physical things, conceptual entities, or descriptors. Nault and Storey explained the links can be is-a links, representing subclass-superclass relationships, or has-a links, indicating that one thing is an attribute of another. Links may also be definitional; for example, "Exercisers use exercise system". They can even capture heuristic information; for example, "over-exercising causes fatigue" (Harmon and King, 1985). However, there are no well-defined formal semantics for network representations (Partridge, 1991). Figure 4.14 shows an example of a semantic network in the project.

The nodes in a semantic network can be arranged into an inheritance hierarchy, whereby nodes lower in the hierarchy automatically inherit properties of the higher-level nodes. Inheritance gives new nodes the ability to know information about many attributes, capabilities, and constraints as soon as these nodes are created. For example, if a new exerciser is added, he or she automatically inherits the information from the nodes above it (Mockler, 1989). Nodes can be used to represent both instances of things (for example, `William Wu') and the type of a thing (for example, Coach). Moreover, inheritance is through inference rather than shared variables.

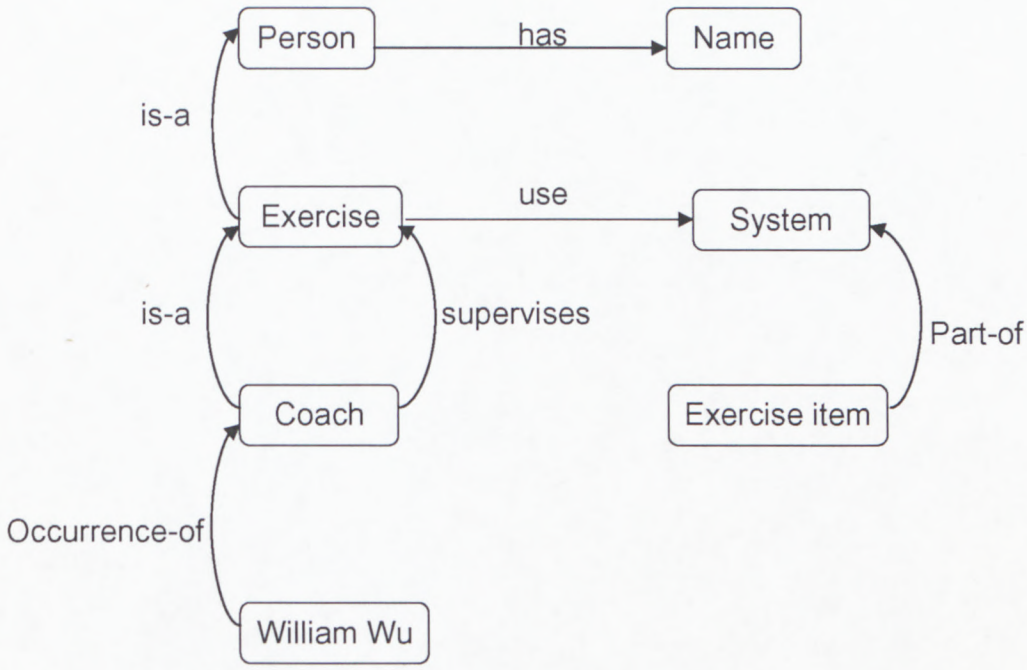


Figure 4.14: Semantic network

4.3.2.1 Semantic network object concepts

A node, along with its related arcs, is compared to an object (Nault and Storey, 1998).

- **Data abstraction**

Nodes represent things, either by variables (for example, Coach), values (for example, `William Wu'), or as compositions (aggregations) of other things (for example, System has links to its parts; Exercise item is one of them). Variables or constants that represent values are both stores; thus, a node can act as a single store. Methods, such as semantic integrity constraints, can be defined as relationships (for example, Coach supervises-maximum-7 Exercisers) between nodes. In this way, multiple methods can easily be defined. Methods cannot operate on data in the nodes.

- **Classification/instantiation**

Classification and instantiation are accommodated in a semantic network by designating nodes such that one node defines the type and another its subtype, through

an is-a arc (that is, specialization). An arc labelled occurrence-of indicates instantiation.

- **Inheritance**

The main quality of semantic networks is inheritance. Nodes can be arranged into subtypes and supertypes using is-a links. This specialization is used to infer properties, rather than directly specifying properties. Multiple inheritance is supported; for example, an exerciser is-a coach and an exerciser is-an amateur can both be defined on the node, Exerciser. Partial inheritance is not accommodated.

- **Independence**

Semantic networks do not operate on each other, nor do they share variables directly; therefore, one aspect of independence is implicit. To illustrate, suppose a particular individual is represented in a semantic network and changes her name (for example, from `Joanne' to `Joan'). This will change the value in the name node, but it will not directly affect other nodes.

- **Message passing**

Semantic networks do not support message passing. To achieve this objective, message passing channels would need to be represented by different kinds of arcs than have traditionally been used.

- **Homogeneity**

Semantic networks exhibit some degree of homogeneity because all things are represented by nodes. For example, both attributes (for example, `colour') and values of attributes (for example, `green') can be organized in a semantic network. Relationships, however, are not represented by nodes.

- **Composition**

Composition can easily be represented. A thing that is an aggregation of other things

can be captured by has-a arcs (for example, person has-a name, age, weight, and frame). Aggregation of physical objects can be represented by labelling each directed arc from a component to an aggregate as part-of. Emergent properties can be added as new attribute nodes and associated arcs.

4.3.4 Software Features and Functionality

4.3.3.1 Set-up Paradigm

In the section of set-up paradigm, User's name and User ID would be recorded when the user registries a gym club and the privilege would be allotted to the user. And then, user will take a series of body tests which include user's weight, body fat index, Cardio Vascular fitness index, and heart-rate. The user's file will be created in a database of the application server of gym club. The exercise machines in the gym club are allocated machine ID by administrator. When user selects a machine to do his exercise, the machine will send the user ID to the application server in order to fetch the user's body data. While user is doing exercise, the machine is monitoring user's body state. The machine will alarm if user's body state exceed the range of health zone.

What is Body IQ? Body IQ is an innovative approach to mind-body exercise training that builds a smarter and stronger body (Body IQ, 2005). Body IQ draws on the use of intuitive awareness to visualize how movement and the use of breath can come together to create greater strength, stability, and a stronger mind connection to user's body. Body IQ is about using the body to move the mind. The Body IQ is designed to make taking user's health measurements quick and easy. It includes measure weight, percent body fat, and blood pressure. The user can immediately view his results. His data is transmitted to the Body IQ database overnight, for future viewing in his personal folder. The ultimate goal focuses on helping users improving or losing the

weight, reducing the blood pressure, stopping smoking, and reducing the stress.

Set-up Paradigm

- Biography
    - User Name, User ID
    - Machine ID
    - Hierarchical Access Control – User GUI, Admin GUI
- Body State
    - Dimensions include – Age, Gender, Weight, Frame
    - Extended Dimension – "Body-IQ" Technology or Medical Doctor
    - Extended Dimension – Body Fat Index
- Cardio Vascular (CV) Fitness Index
    - CV Fitness Index
    - Based on heart-rate monitoring – Polar Technology

4.3.3.2 Control Paradigm

In the section of control paradigm, user's safety is the first thing to an intelligent machine control system. The button of Fail-Safe is designed in the keypad, and a security indicator is devised on the LCD screen. The term "Fail-safe" is used to describe a device which, if (or when) it fails, fails in a way that will cause no harm or at least a minimum of harm to other devices or danger to personnel. The machine will alarm if user's body state exceed the range of health zone which depends on the evaluation of user's body, at the same time, the security indicator will become red from green. User can easily push the button of Fail Safe to stop the machine.

The software affords a series of items of exercise, and each one has different FX profiles (curves) and Intensity (scale). A user can choose the repetitions and tempo, but they are restricted by maximum.

Control Paradigm

- Design Principles re Artificial Intelligence
    - Safety First
    - Fail Safe
    - Parameter Suggest
    - Manual Override
- Manual / Direct re Exercise Event
    - Input device/s: Keypad & LCD
    - Single Exercise Event
    - Machine Configuration – Exercise Context, User Range
    - Exercise Scope – FX Profile, Intensity (Scale)
    - Event Activity – Repetitions, Tempo
- Menu Driven re Exercise Routine
    - Input device/s: Keypad & LCD
    - Ordered set of Exercise Events
    - Dimensions include – Body State, CV Fitness Level, Goal, Duration
    - Body State include – Age, Gender, Weight, Frame
    - Goals include – Warm-up, Cool-down, CV Fitness, Muscle Toning, Strength, Weight Loss

4.3.3.3 Monitoring Paradigm

In the section of monitoring paradigm, all machines' states are awaiting input. When user selects a machine to do his exercise, the machine will send the user ID to the application server. In terms of user's body data, machine will monitor the change of user's body state, and accordingly records the FX profile, intensity, repetitions, and Tempo. And one of the most important aspects must be watched is user's heart-rate. These data will monitor in LCD screen.

Monitoring Paradigm

- User ID, Machine ID
- Date, Start Time, End Time, Total Time (Calculated)
- Machine State – Active / Standby / Awaiting Input
- Machine Configuration – Exercise Context, User Range
- Exercise Scope – FX Profile, Intensity
- Event Activity – Repetitions, Tempo, Tempo Deviation (Calculated)
- Heart-rate – Instantaneous, Min, Max, Average

4.3.3.4 Output Paradigm

In the section of output paradigm, all monitors dimensions must be shown in LCD Display. Tempo and User Range are to be graphically displayed with (coded) audible beeps. All users' data and controlled and monitored dimensions are recorded by the application server in event journal database.

Output Paradigm

- LCD Display – All monitored dimensions, Tempo and User Range to be graphically displayed with (coded) audible beeps
- Event Journal Database (EJdB) – All User, Controlled and Monitored dimensions

4.3.5 Flow Logic of GUI for a stand-alone exercise machine

In this section, a logic flow of user interface will be designed for an intelligent exercise machine. We follow the software features and functionalities to achieve the requirement analysis.

Hierarchical access control has two privileges, one is Administrator's privilege, and the other is user's level (see Figure 4.15).

4.3.5.1 Administrator's Privilege

Administrator's privilege includes Exercise Mode (see Figure 4.16 and 4.17) and Administration Mode (see Figure 4.18 and 4.19).

- Administration Mode:

1. When the gym machine has some trouble, the administrator can reset the machine and enters Administration Model to fix them.

2. Administrator can acquire the record of the trouble including the time when the machine made a mistake and the reason why it happened.

3. Administrator can customize the exercise plan for himself. There is no need for him to choose the options every time before starting to do the exercise.

4. Administrator can customize the dimension of Exercise Model.

5. There is a situation when one customer buys this gym machine for his family, and he could be an administrator. He can customize the exercise for his family with coach's or doctor's recommendation; taking his child for example he can set the limitation of the time and the weight for his child, also for elder people.

- Exercise Mode:

There are two options user can choose. One is "Exercise with remote coach", the other is "Self exercise". It depends on whether user has a personal trainer or not. He can just select the Self exercise if he does not have a coach, and the exercise machine can supply parameter suggest depending on user's data (age, gender, weight, frame, and heart-rate).

1. Exercise with remote coach. User will follow the items of an exercise plan which is customized by his coach. When he feels that he is not comfortable with the weight or curve, he can adjust it expediently, and the system will record it and make a report to his coach.

2. Self exercise. After user selects a motion exercise, the system will recommend some parameter related to a weight and a curve, etc. He can choose the parameter by himself if he does not think the suggestion is good for him. All the suggestions are depending on the user's details, like age, gender, weight, heart-rate and so on.

4.3.5.2 User's Level

User's Level only has Exercise Mode. They must ask for help from administrator, when they have trouble with the machine. The users can do all motion exercises, but it depends on their details like age, weight, gender to give some restrictions. Just as different ages have to select different weights, the limitation is strictly restricted.

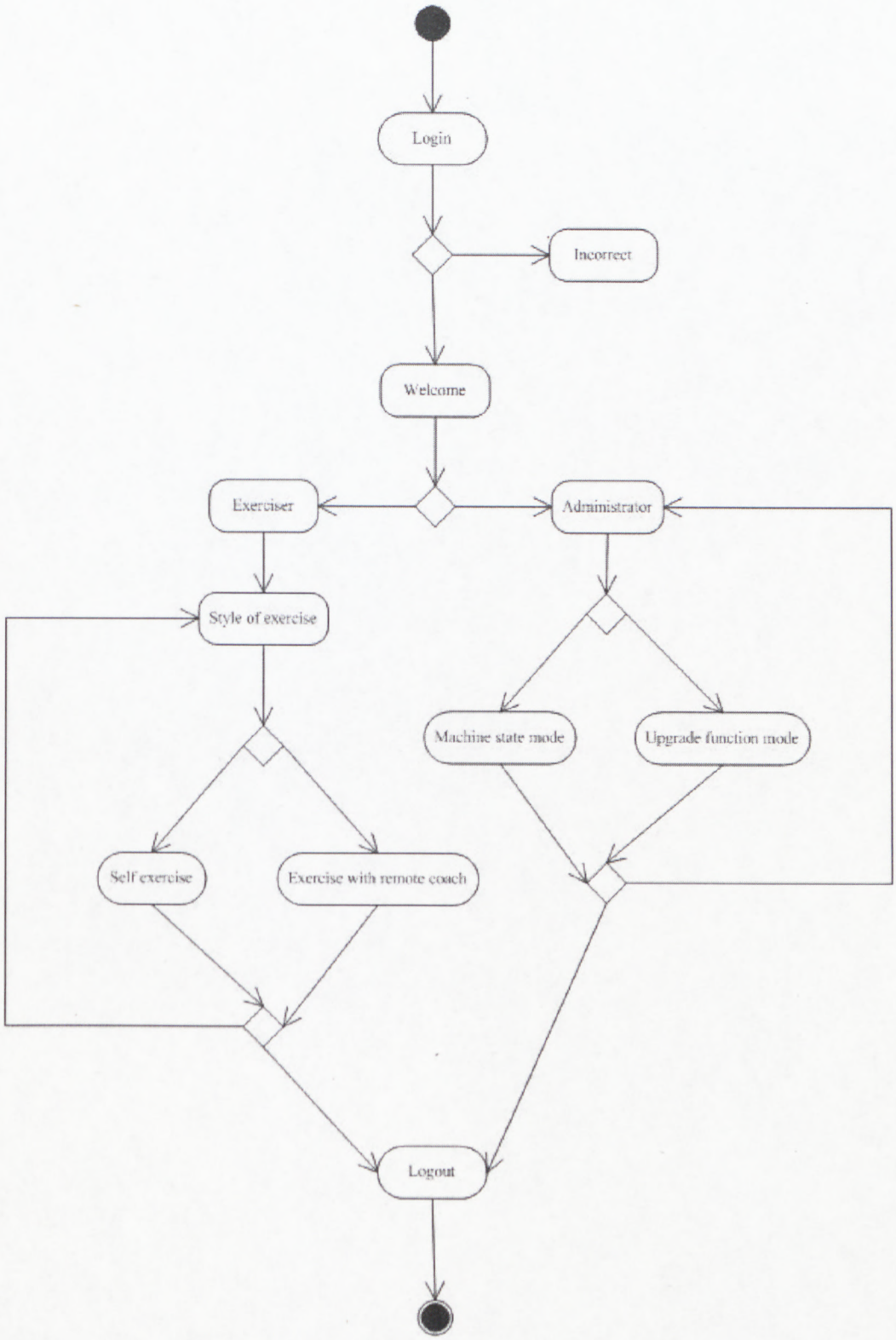


Figure 4.15: Flow logic of the exercise software system

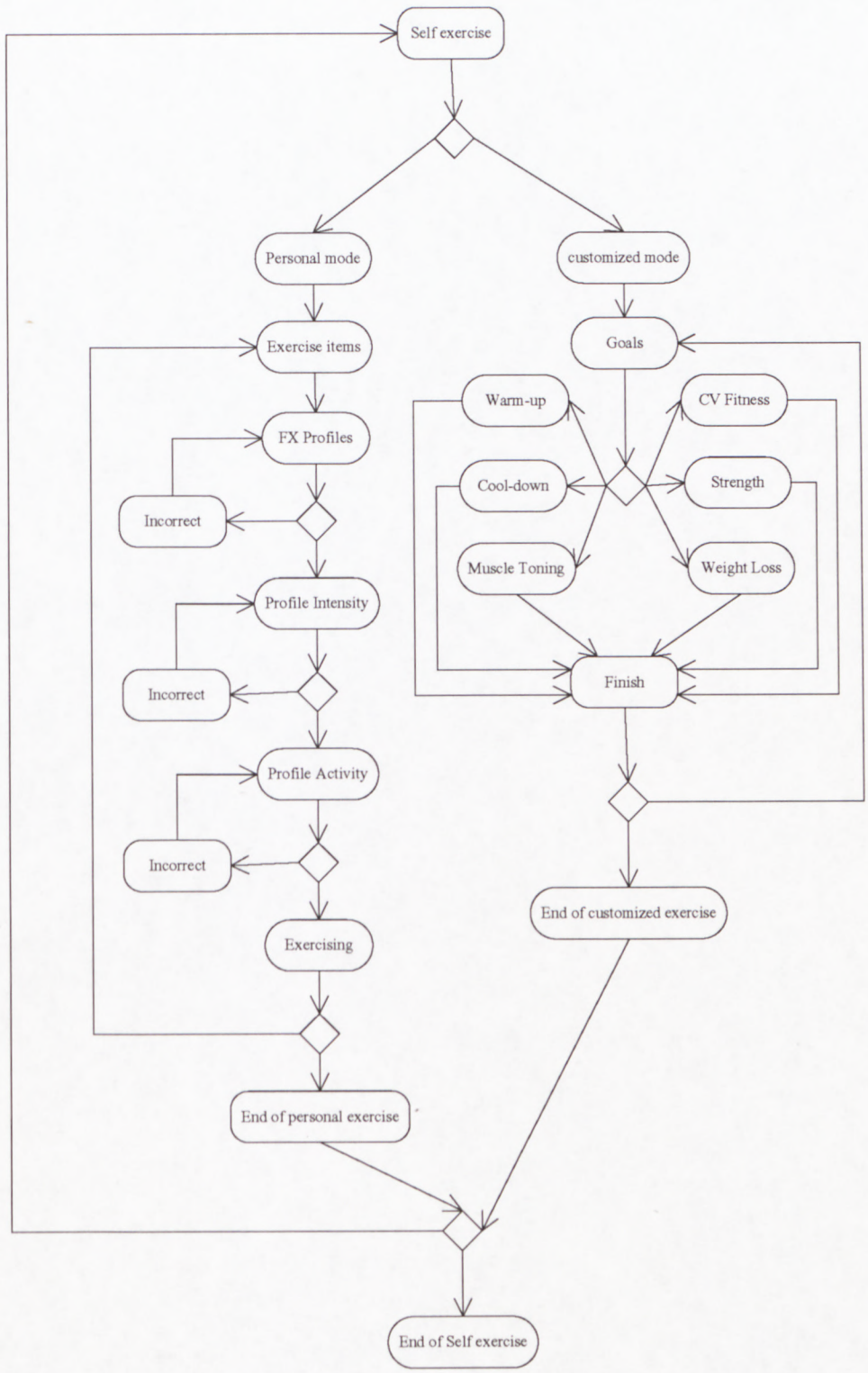


Figure 4.16: Flow logic of "self exercise" mode

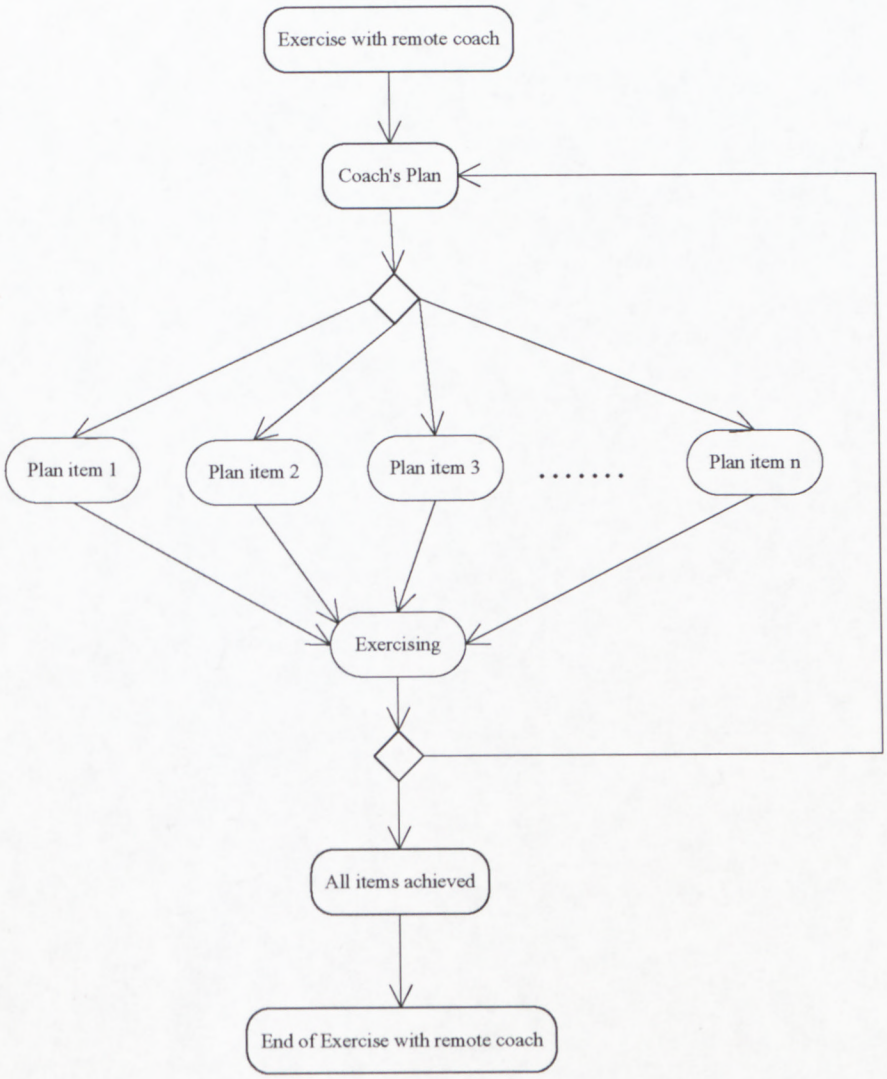


Figure 4.17: Flow logic of "exercise with remote coach" mode

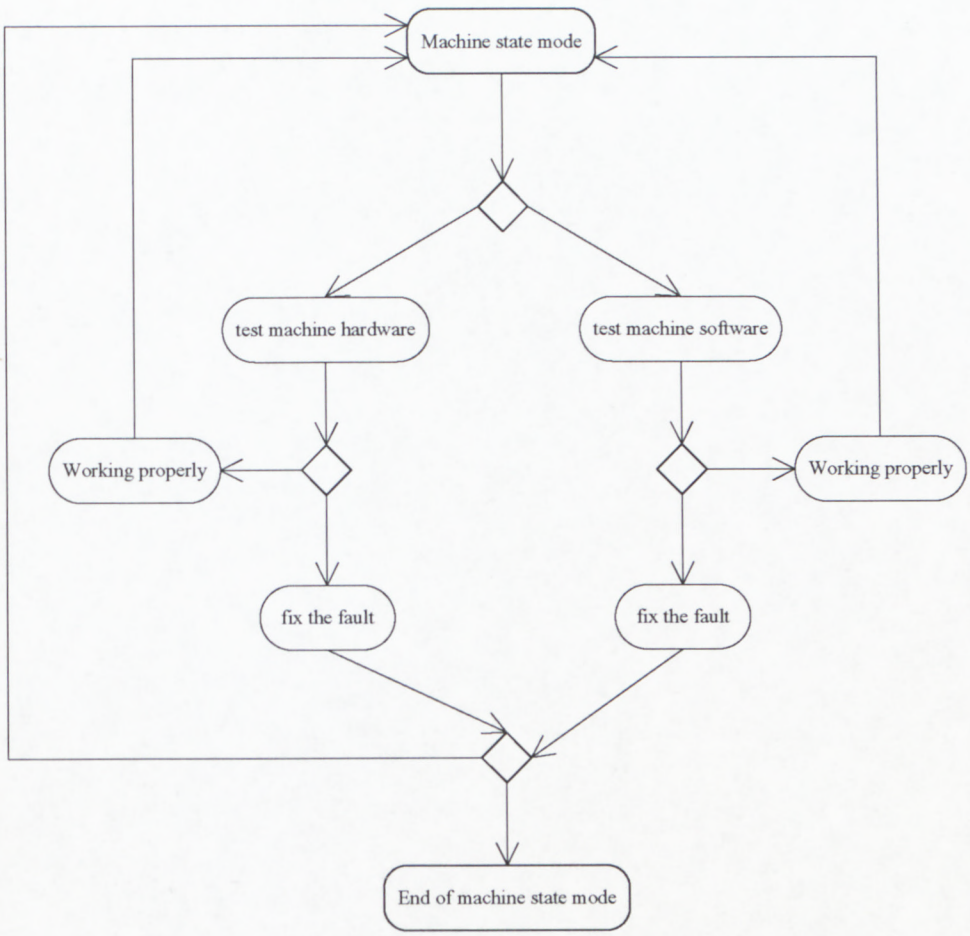


Figure 4.18: Flow logic of "machine state" mode

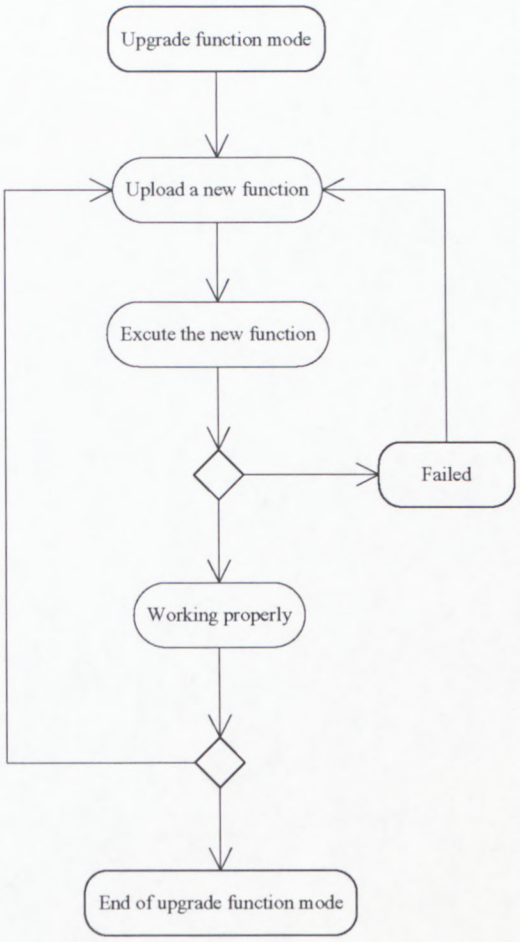


Figure 4.19: Flow logic of "upgrade function" mode

4.3.5.3 Screenshots and Keypad

Here we illustrate some important screenshots in the exercise software system, which are simply described the key functionalities of the software. In terms of the screenshots, the keypad is designed for user input.

In the software of the intelligent exercise machine, we take the stage of "style of exercise" (see figure 4.15) for example to illustrate how the screen design relys on the GUI flow logic. Two options indicate in this stage, one is "self-manage exercise", and the other is "doing exercise with coach plan" (see figure 4.20). User enters his choice from the keypad, then press the keystroke "OK". When the user wants to change his choice, he can press a button "Cancel" in order to enter a number again. A time

counter is necessary added in the screen which reminds user how long they have doing the exercise.

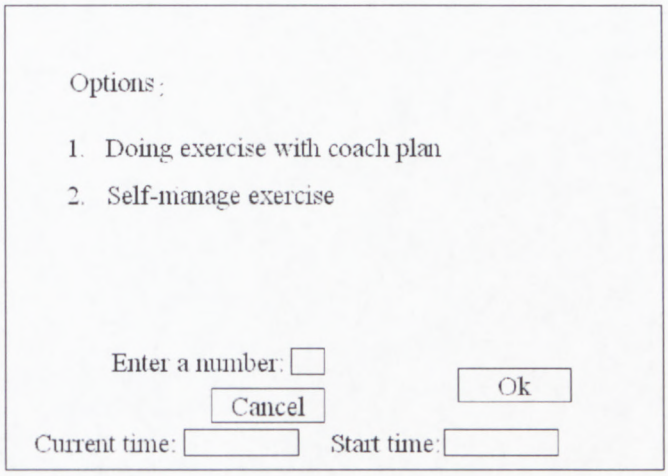


Figure 4.20: the style of options in our exercise software

The option 2 "self-manage exercise" includes the options "Personal mode" and "Customized mode" (see figure 4.16). The option of FX Profiles as the core in the "Personal mode" is shown in Figure 4.21.

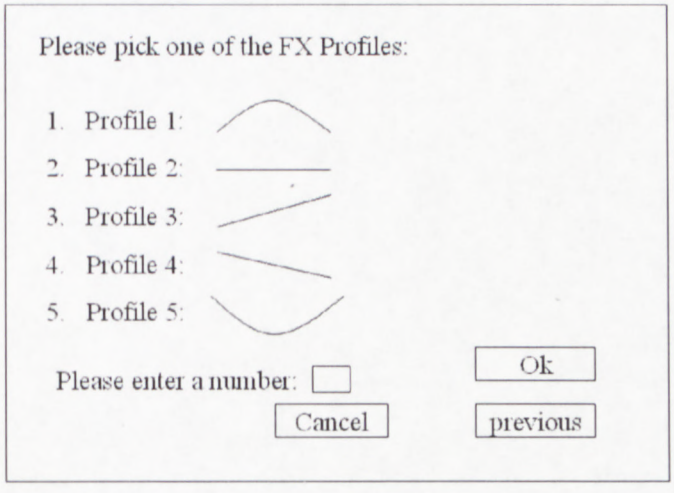


Figure 4.21: the option of FX Profiles

The list of items about "Customized mode" is illustrated in Figure 4.22.

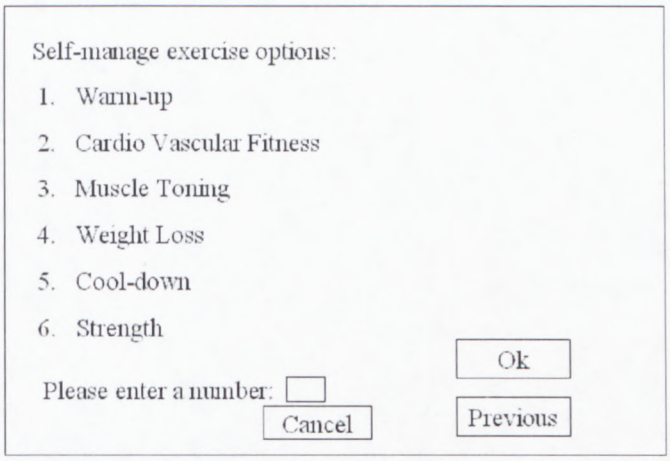


Figure 4.22: the screenshot of "Customized mode" options

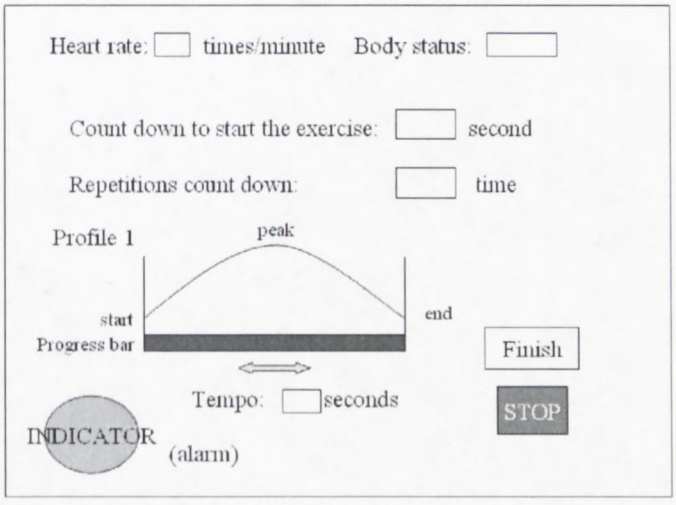


Figure 4.23: the exercising screen

As shown in figure 4.23, the indicator is green, which indicates the user data of heart rate and body status are normal. The initial user data of them are supplied by his doctor. And if the exercise is overloaded beyond the standard health status, the indicator will immediately become red and flicker as fail safe indication, which matched the red colour of the "STOP" button. The fail safe indication also measures machine status. When the machine encounters the trouble, the indicator also becomes red.

In terms of the screenshots we illustrated, the keypad is come out, shown in Figure

4.24.

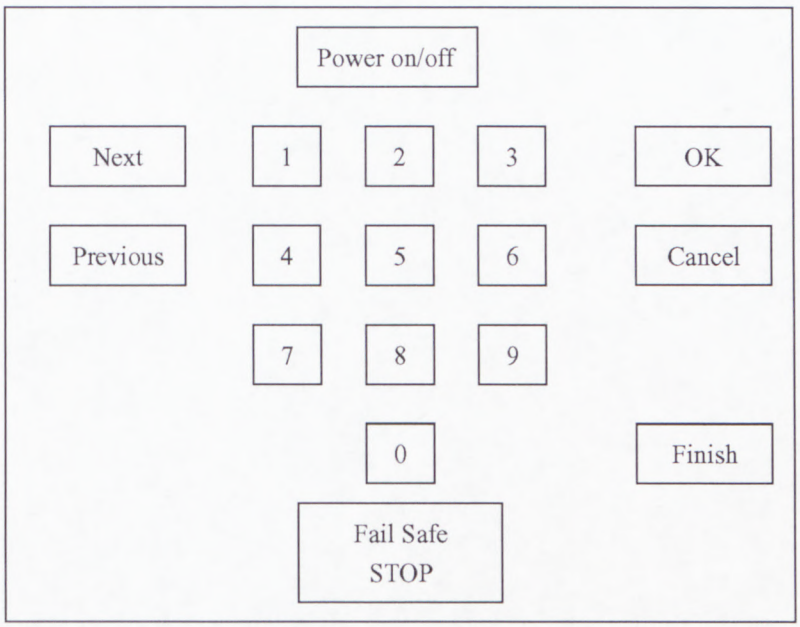


Figure 4.24: the keypad, input device of the exercise machine

4.4 Software Testing and Evaluation

4.4.1 Software quality

What is quality?

Standard Glossary of Software Engineering Terminology:

Quality: 1) The degree to which a system component, or process meets specified requirements. 2) The degree to which a system, component, or process meets customer or user needs or expectations.

Pressman: Software Engineering 1992:

Quality: conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are

expected of all professionally developed software.

Dimensions of software quality (Figure 4.25)

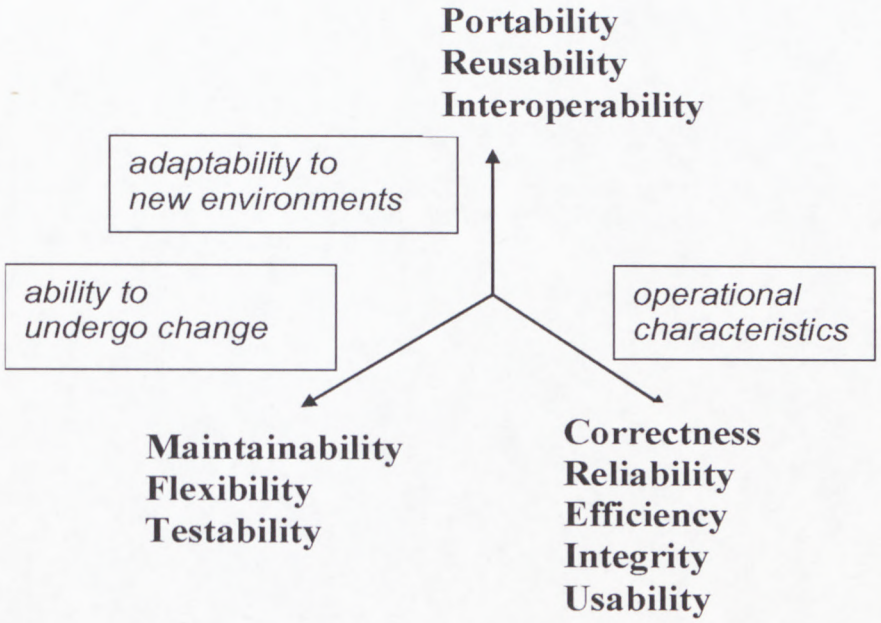


Figure 4.25: Dimensions of software quality

Portability: The effort required to transfer the system from one hardware and/or software environment to another.

Reusability: The extent to which the system (or part of it) can be reused in other applications.

Interoperability: The effort required to couple the system to another.

Maintainability: The effort required to introduce a modification (usually a correction) into the system.

Flexibility: The effort required to modify or customize the system in operation.

- software testing
- enforcement of standards
- documentation
- control of change
- extensive measurement
- record keeping and reporting of the process

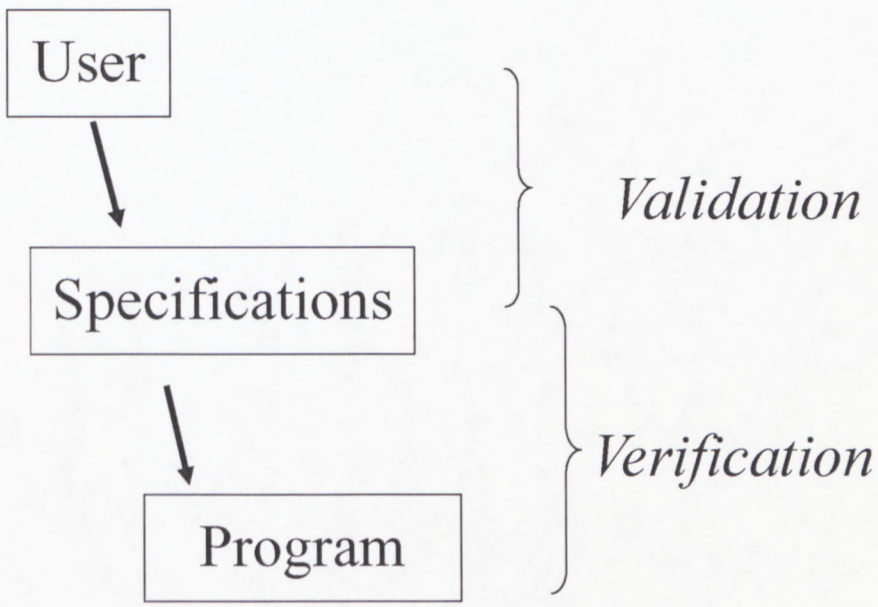


Figure 4.26: Factors of Verification and Validation

Verification: are we building the product right? (Figure 4.26)

Validation: are we building the right product? (Figure 4.26)

4.4.2 Principle of software testing

(1) Testing is a process of executing a program with the intent of finding a defect. So, there must be some program code to be executed.

Testability: The effort required to test the system to ensure that it performs its intended function.

Correctness: The extent to which the system satisfies its specifications and fulfils the users' needs.

Reliability: The extent to which the system can be expected to perform its intended function with required precision and without failure.

Efficiency: The amount of computing resources (space, time) required by the system to perform its function.

Integrity: The extent to which access to the system or its data by unauthorized persons can be controlled.

Usability: The effort required to learn, operate, prepare input and interpret output of the system.

Measurement

Standard Glossary of Software Engineering Terminology:

Quality assurance: 1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. 2) A set of activities designed to evaluate the process by which products are developed or manufactured.

- application of sound technical methods and tools
- formal technical reviews and inspections

(2) A good test case is one that has a high probability of finding an as yet undiscovered defect. So, the test cases (the program input) should be selected systematically and with care, both for correct and incorrect behaviour.

(3) A successful test is one that uncovers an as yet undiscovered defect. So, testing is psychologically destructive since it tries do demolish the software that has been constructed.

(4) Testing cannot show the absence of defects, it can only show that they are present. (Testing is not formal verification)

(5) Testing is quite an ineffective method of quality assurance. (Though, usually the only applicable one)

(6) Successful testing shall be followed by a separate debugging phase.

(7) Testing is also by itself a process that must be systematically managed (and assisted with special testing tools)

Error: A mistake (human action) made by a software developer. It might be a total misinterpretation of user requirements, or a simple typographical misprint. An error introduces a defect into the software code.

Defect, fault, bug: A difference between the incorrect program and its correct version; a coding error. A defect in the software, if encountered during execution, may cause a failure.

Failure: An externally observable deviation of the functional software from its specification; an incorrect result of computation.

4.4.3 Dialog Modelling Testing and Evaluation

4.4.3.1 Labels and Icons

The text label concrete interaction object is used for most short and static texts in a user interface, like attribute names that proceed text input fields. According to the application of GUI in exercise machine, a text label consists of 1) a label string, 2) a font (specification), and 3) two colours, one for the text itself and one for the background. It is a better approximation of modern toolkit implementations.

In most cases, the label string itself carries most of the information for the user, and has a dual use. First, it represents some data in the domain of discourse, e.g. User ID, User Password or Weight Value. Second, it refers to elements of the underlying model, e.g. concept/class, relation and attribute names, as an aid for making the interface easier to understand. The latter use is interesting since it highlights the user interface model as a source of output information about user exercise data.

While the label string is considered the main part of the text label, the text label's relations and attributes represent a considerable state space that can be utilised for e.g. classification purposes. The transitions are controlled by how the output/receive ELEMENT value is classified into specialization of a general concept, where each group of states may represent one disjoint specialisation. In terms of a signal transition between the machine and the GUI, We used 0~5 Voltages as control signals. Consider a three-axis classification of a process state, into either Normal, Fast or Slow (Speed), either Heavy or Light (Weight), and finally whether the rate is Increasing or Decreasing (Time). If quantitative measurements are represented by the label string, the class membership is mapped to each of the state groups, with Normal, Fast or

Slow mapped to the three colours, Heavy or Light mapped to bold on/off and Increasing or Decreasing to italic on/off.

A similar reasoning can be tested to sets of related icons. Each icon is modelled as an or-composition of colour states, like for text labels. A set of related icons are or-composed with each other, and finally and-composed with the composite colour state, giving a total state space of #icons times #colours. Icons are often combined with labels, and it might seem that the label discussed above could be (and-)composed with an icon, to give a total state space of #label states times #icon states. However, when composing concrete dialogues and state spaces with "physical" attributes like bounding boxes and colours, we should consider whether *merging* would make more sense:

- If the two parts, the label and icon, are supposed to represent one element in the domain, it seems reasonable to always use the same (background) colour

- Alternatively, if the label and icon represent independent domain elements, they should themselves be independent, and an and-composition seems reasonable.

This testing and evaluation shows that composition of concrete interaction objects is not as simple as the abstract model suggests, and that the concrete and "physical" aspects of the elements provide natural constraints for composition that must be considered.

4.4.3.2 Buttons

Due to the special limited circumstance of the exercise machine, we just provide keypad for user input. We describe the testing and evaluation about the design of a

fully functional keypad controller.

One of Many

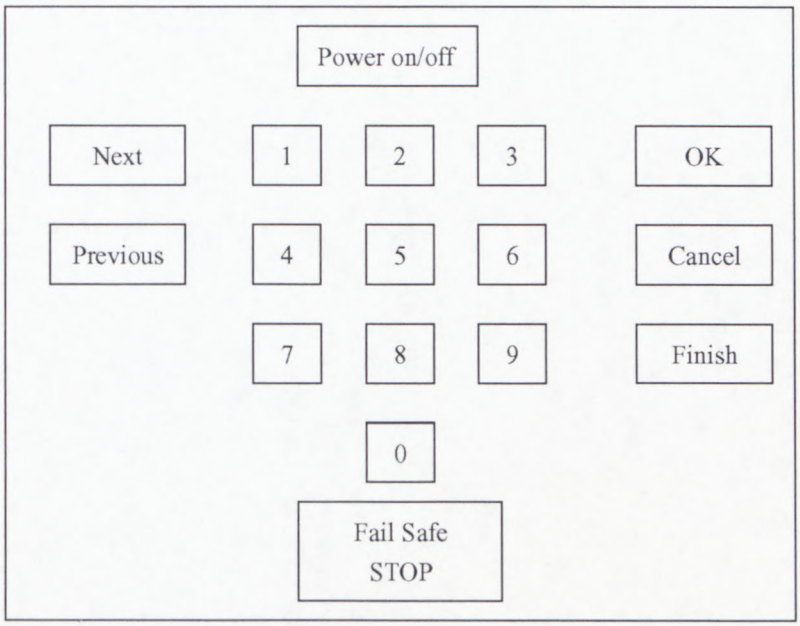


Figure 4.27: The keypad design for an intelligent exercise machine

Option buttons are best for selecting among a fixed set of alternatives as long as the number is relatively small. In our project, we designed six options at most in each dialog shown below (Figure 4.28). The number 0~9 of the keypad supports user input the user id, password, and a weight (Figure 4.27). They have the advantage that all options are visible – which makes visually scanning the choices fast and easy – but, for selection within large sets, too much screen real estate is used. Option buttons are rarely used where the number of options must vary because either the amount of screen space taken must also vary or all options have to be shown, with unavailable options grayed out.

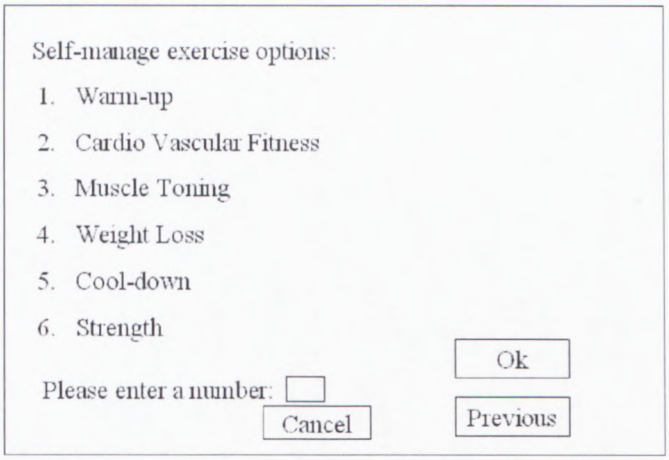


Figure 4.28: a screenshot of the GUI application

While text labels and icons are used for presentation only, buttons are interactive and can be used for input. The basic behaviour of buttons is the switching between the released and pressed states. When used for *control* purposes, the transition is typically triggered by a keypad button. The two states may also be used for information mediation, where the pressed and released states are mapped to the boolean values true and false, respectively, e.g. based on an attribute value, element classification or the presence of a relation.

To test and evaluate the basis for the functionality of buttons, we again start with a model of the static structure, as shown Figure 4.29 (a). A BUTTON contains either an ICON or a TEXT LABEL, as well as a border of which there are (at least) three kinds. This means that for presentation purposes, the BUTTON's maximum state space is the states of the border and-composed with an or-composition of the ICON and TEXT LABEL, i.e. a considerable state space. At a minimum, the BUTTON needs two states to distinguish between the two boolean values, while in practice, some additional intermediate states are used for user feedback. Figure 4.29 (b) shows one possibility, including the presentation-oriented states INTERIOR and BORDER. These are driven by the CONTROL states, which limit the allowed state sequences to those providing the desired interactive behaviour. The PRESS state/condition resource in turn drives the CONTROL state and provides the link from an external state, a keypad button. In

the typical button implementation, the PRESS resource is bound to left-button and space-bar keystroke, the latter requiring that the button has the keypad focus. The typical state sequence is INACTIVE, ACTIVE.RELEASED, ACTIVE.PRESSED, ACTIVE.RELEASED and INACTIVE. By adding appropriate events and conditions to the transitions in the CONTENT and BORDER states, variants of the basic look and feel are implemented.

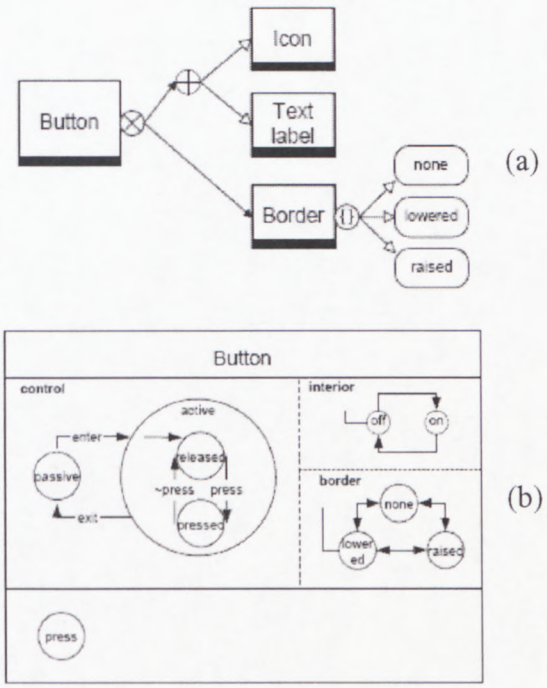


Figure 4.29: Button Structure and States

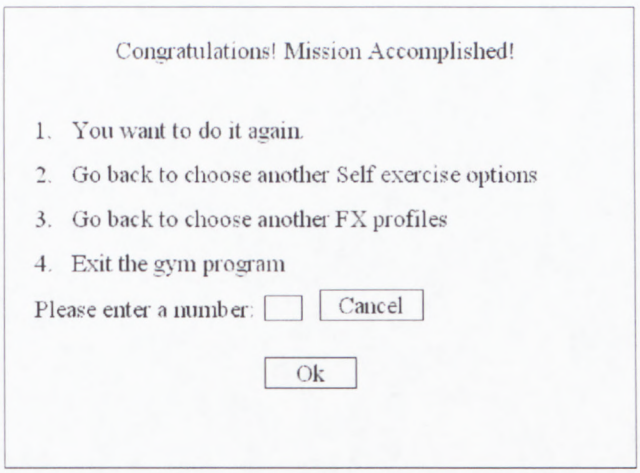
Two basic look and feel variants of the button are normally provided. In the first variant, the border is instantly raised after the button is pressed, returning to the released state. This variant is mainly used for control, by triggering other transitions when reentering the ACTIVE.RELEASED state. The pressing of the button can be seen as an utterance or act, e.g. "NEXT", directed towards the application or the context of the button. If the context of the button is changed by moving the text cursor or selecting a different region, the button is updated to reflect the state of the new context.

The interpretation of these two variants as either control- or information-oriented, should be reflected in the corresponding interactor model. The control case is shown in Figure 4.29 (b), while the information interpretation is indicated by adding output/receive and input/send gates. The input/send gate's tip must be connected using a complement connection to its own base.

4.4.3.3 Popups dialogs and menus

Menus are special purpose composite interactor that implements a neat way of quickly accessing functionality without using too much screen space in our application. Menus are typically used for providing access to the main application functions and are often hierarchical. Popups dialog achieve application communicated with user. Figure 4.30 (a) and (b) show how an error message and a feedback message are set by popping up an application-user dialogue shown below. This message interactor consists of a trigger and the popup dialogue, shown here in the active state.

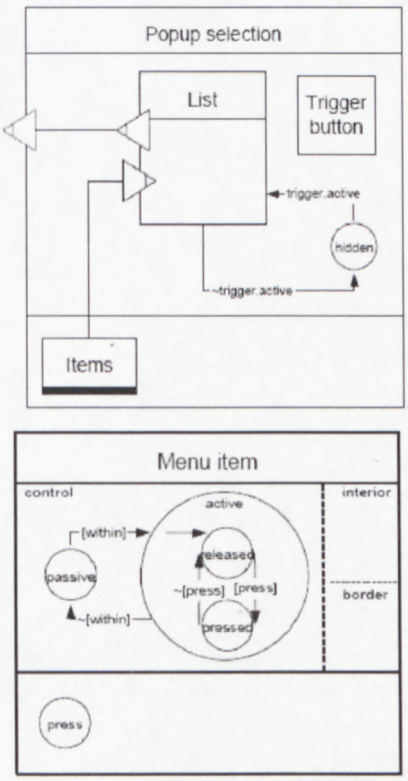
(a)



(b)

Figure 4.30: Popup Dialogs

Menus of the application is compositions of a trigger part and a selection part, where interacting with the trigger makes the selection part ready for accepting a selection, as shown in Figure 4.31 (a). The interactor signature of menus is used for inputting data and invoking actions.



(a)

(b)

Figure 4.31: Popup selection and menu items interactors

The way that the menu trigger activates a hidden dialogue, but there is one important difference: The functionality of menu items is limited, since it must be possible to both trigger and operate them using a single device or event, typically a keypad button press, press and release. This means that in general, menu items essentially are various kinds of buttons, e.g. boolean buttons, act buttons and sub-menu triggers.

Thus, a menu represents a composition of a trigger button and a set of item buttons, where both control and layout is coordinated. The composition of the control aspect must ensure that the trigger button activates the set of item buttons and that these react in a coordinated manner. The layout must support the presentation of a uniform set of elements. Figure 4.31 (b) shows an alternative menu item model of the application, where conditions are used for triggering transitions, rather than events.

The menu of the application illustrates several important points:

- Special purpose interactors may be modelled as tightly coupled simpler interactors.

- Composition if interactors require careful consideration of how each part contributes to the overall behaviour.

- Abstracting away the details of the parts, may prevent the integration needed for achieving the desired behaviour.

Menus have a special role in user interfaces, as it the main repository of predefined acts. The menubar is the main container for menus, which may contain either acts (buttons) or other sub-menus. As Figure 4.32shows, there are few constraints for how the menu hierarchy is composed composition. Hence, it is easy to add or remove acts from this kind of container. In practice, there are two limiting factors: 1) The user will

have problems navigating in deep structure or menu with many entries, and 2) there are conventions for naming and positioning menus, as well as populating them with acts.

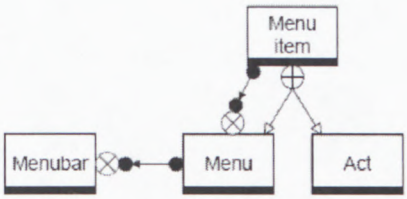


Figure 4.32: Menu aggregation

4.4.4 The results of Testing and Evaluation

In the section of Testing and Evaluation, we illustrated the application of graphical user interface in an intelligent exercise machine has been tested and evaluated by using a technique dialog modelling from functionality and usability. From the part of our whole project, the user interface modelling is designed effectively in quality. And my colleague, who designed the system part, accepted a well-structured electro-pneumatic application. These two parts of the whole project is properly tested and evaluated together, and we worked out the result achieved our goal to design and implementation an intelligent exercise machine.

The important thing we have done is we rely on the Usability Engineering lifecycle and user-centred principle to design the GUI software. At the first stage "requirement analysis", we analyzed the system requirements, as classifying users, categorizing user's execution environment, building client unit, specifying trigger event, customization, and personalization, and investigate the tasks from three different aspects like task descriptions, user descriptions, and interface descriptions. At the second stage "Design/Testing/Development", we worked out software features and functionalities by user specific requirements, and relied on the "requirement analysis" to design the flow logic of GUI for the intelligent exercise machine, and tested and

evaluated the GUI of the intelligent exercise machine from software functionalities to hardware equipment. At the last stage, we installed the software in the intelligent exercise machine and recorded user's feedback. During the first and second stages, Object-oriented modelling and UML supplied an approach to help us for analysis and design of the software reliably and fast.

4.5 Conclusion

In this chapter, Case Study, we fully illustrated the design and implementation of an intelligent exercise machine with human-machine interface as a specific practical IT application. The design process of the application successfully applied the Usability Engineering and Object-oriented modelling with UML from system analysis to task analysis, from software features to flow logic, from testing to evaluation. We approved Usability Engineering and user-centred design with Object-oriented modelling and UML benefit for develop a suitable and effective GUI for an intelligent machine.

Chapter 5: Conclusion and Future Works

This chapter concludes this thesis. It summarizes the previous chapters that discussed various aspects of the design and implementation of an intelligent semantic machine control system with specific reference to human-machine interface design. On the one hand this thesis shows the benefits of usability design method but on the other hand it shows that many improvements can still be made. We first summarize the work and state the main contributions of this thesis.

As we presented, this work is a development of a practical IT application. We have focused on human-machine interface design as one of a repertoire of techniques that are necessary when developing interactive systems. Developing interactive systems that are efficient and reliable in use, easy to learn and remember, and that provide user satisfaction is one of the greatest challenges of the software industry. During this study, we have learnt how to develop a practical GUI application, which includes preparing the literature, customizing a suitable methodology and framework as a guideline, and merging the method and framework into the design process. As a practical research, the product must be tested and evaluated properly before it enters into the market. We did it! But there is not at the end of the journey. After the product is purchased, we need to keep on upgrading the new functions and fixing the redundant functions due to customers' needs. We used object-oriented modelling and UML to design the application, which are easy for stretching the new modules.

5.1 Conclusion

The primary purpose of this research project was to develop a novel dynamic

resistance exercise machine with an effective graphical user interface. The purpose is composed by two aspects. One is building an intelligent semantic machine control system which is specific reference to an intelligent exercise machine with regard to electro-pneumatic machine control; the other is designing a suitable human-machine interface for interaction between users and the system.

In chapter 2, we described the related work about intelligent machine control system and human-computer interaction paradigms. Artificial intelligence is a very important concept in our project. It supplies a way to communicate between user and the system easily. Semantics, which is one main field of the AI, is conveying information in between user and the intelligent system. That is the point why we use this word in the title. The other concepts, which are Intelligent Control, Semantics, Intelligent Machine Architecture, Human-Machine Interaction, Information systems and Graphical User Interface, also play important roles for integrating the design of machine control system and human-machine interface.

My focus concentrates on human-machine interface design methodology. Building interactive systems that are efficient and reliable in use, easy to learn and remember, and that provide user satisfaction is one of the greatest challenges of the software industry. These requirements are increasing with the widespread use of intelligent machines, the advent of the information system and the increasing connectivity of existing machines and computers. In the past decades, usability engineering was consistently neglected in software development in field of user interface. However, we are rapidly moving away from a technology-centred foundation into a user-centred maturity. While in the beginning computing technology could not meet all the needs of the customers, and software developers could take advantage of the relative stability of the object-oriented programming paradigm with UML; today customers demand a diversity of highly usable and specialized information access that are efficient, reliable and convenient.

In chapter 4, we explored how object-oriented modelling and UML could contribute to integrate usability engineering into modern software development, providing benefits for both disciplines. The underlying principle was that some of the problems that software engineering faces today are the essence of what usability engineering promoted for years. Conversely, some of the problems that usability engineering currently faces are already successfully solved in the software engineering field. The design process of the case study, an intelligent exercise machine, is fulfilled properly from the system analysis and the task analysis to the software features and functionalities. This design process illustrates how usability engineering can take advantage of object-oriented modelling, and of the UML in particular, to leverage many of the new and emergent problems in user interface design for machine control system and information system. In particular we demonstrate how the UML interchange mechanisms can: leverage automatic and flexible generation of user interface from high-level models; tool interoperability promoting artifact change between UML modelling tools and task modelling tools. We also illustrate how semantic network can describe the interaction design between the user and the system. At the step of the testing and evaluation, we approved that we achieved our goal to develop an effective and suitable GUI for an intelligent semantic machine control system.

Although considerable work remains, this thesis demonstrates the feasibility of applying sound and systematic engineering principles to build interactive software of quality in turbulent and competitive environments. Today software is one of the most important industries in the world. Developing software of quality is hard and the complexity to ensure that engineering prevails over high-speed hacking.

Larry Constantine says: "Ultimately, the true pace of change is not dictated by the evolution of science or technology or of ideas, but by the capacities of human and human social systems to accommodate change. A product, a service, a practice, or a perspective - however new and innovative - can have no impact without acceptance;

no significance without change in people and their institutions".

In the end, it's all about the users, but developers are also users of the modelling languages, techniques, and methods. Ensuring that they have a proper set of manageable tools, methods, and techniques is the first step toward our quest for quality and usability in software.

5.2 Future Works

Developing a design method almost never ends. There are still many areas that need to be improved both on the theoretical as the practical side. Theories are important to link together many aspects in the process of design. With the help of sound theories we can develop practical techniques that make improvements of the process and product possible. On the other hand, many techniques are also developed ad hoc when designers face problem. Studying them can also contribute much to theoretical understanding.

The work leading to this thesis has generated many interesting and promising ideas. Some of those future developments are promising ideas that we believe are worth exploring, others are equally interesting ideas that we have dropped during the research program due to different reasons.

References

ACM (1992) *ACM SIGCHI Curricula on Human-computer Interaction*, Curricula Recommendation, Association Computing Machinery, http://www.acm.org/sigchi.

Agah, A. (2001) *Human interactions with intelligent systems*. Computers and Electrical Engineering, 27:71-107.

Agre, P. and Chapman, D. (1987) *Pengi: an implementation of a theory of activity*. Proceedings of the Sixth National Conference on Artificial Intelligence, pp. 268-272. CA: Morgan Kaufmann.

Albus, J. (1991) *Outline for a Theory of Intelligence*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 3, May/June, pp. 473-509.

Albus, J., McLean, C. R., Barbera, A. J., and Fitzgerald, M. L. (1985) *Hierarchical Control for Robots and Teleoperators*, Proc. of the IEEE Workshop on Intelligent Control, Eds. G. Saridis, A. Meystel, Troy, NY pp. 39-49.

Anderson, J. R. (1983) *The Architecture of Cognition*. Harvard University Press.

Annett, J. and Duncan, K.D., (1967) *Task Analysis and Training Design*, Occupational Psychology, 41, pp. 211-221.

Answers.com Retrieved: Oct 16 2005, from
http://www.answers.com/topic/unified-modeling-language

Antsaklis, P. (1994) *Defining Intelligent Control*, Report of the Task Force on Intelligent Control, IEEE Control Systems, 0272-1708/94, June, pp. 1-5 and 58-60.

Arkin, R. C. (1989) *Towards the unification of navigational planning and reactive control*. In AAAI Spring Symposium on Robot Navigation.

Arkin, R. C. (1995) Just what is a robot architecture anyway? *Turing equivalency versus organizing principles*. In AAAI Spring Symposium: Lessons Learned from Implemented Software Architectures for Physical Agent.

Askew, R. S. and Diftler, M. A. (1993) *Ground control testbed for space station freedom robot manipulators*, IEEE Virtual Reality Annual, Seattle, WA.

Avison, D. and Nandhakumar, J. (1995) *The Discipline of Information systems: Let Many Flowers Bloom*! In Information System Concepts: Towards a consolidation of views, Proceedings of the IFIP International Working Conference on Information System Concepts (Falkenberg, E., Hesse, W. and Olivé, A., Eds.), 1-17, Chapman & Hall, London.

Bagchi, S., Biswas, G., and Kawamura, K. (1996) *Interactive task planning under uncertainty and goal changes*. Robotics and Autonomous Systems, 18:157-167.

Basic of object-orientation. (2004) http://www.objectfaq.com/oofaq2/body/basics.htm

Bellman, R. E. (1978) *An Introduction to Artificial Intelligence: Can Computers Think*? Boyd & Fraser Publishing Company, San Francisco.

Bender, E. A. (1996) *Mathematical Methods in Artificial Intelligence*. IEEE Computer Society Press.

Biography.ms Retrieved: Oct 15 2005, from http://unified-modeling-language.biography.ms/

Body IQ. (2005) *The relationship between the Sports Science Institute of South Africa and Body iQ*. http://www.bodyiq.co.za/ssisa.html/.

Booch, G. (1994) *Object-Oriented Analysis and Design.* Addison-Wesley.

Brooks, R. A. (1986) *A robust layered control system for a mobile robot*. IEEE Journal of Robotics and Automation, RA-2(1), March.

Brooks, R. A. (1991) *Intelligence without representation.* Proceedings of IJCAI-91, pp. 569-595. San Mateo, CA: Morgan Kaufmann.

Brown, C. M., (1989) *Human-Computer Interface Design Guidelines*, Ablex Publishing, Norwood, N J.

Buss, M. and Hashimoto, H. (1996) *Intelligent Control for Human-Machine Systems*. IEEE Transactions on Mechatronics, Vol.1, No.1, March.

Button, G. and Dourish, P. (1996) *Technometodology: Paradoxes and Possibilities*, in Proceedings of CHI'96, ACM Press.

Bytheway, Andy. (n.d.) *The Logic Model for research students adopting an "interventionist" or "action research" approach to their work*. University of Western Cape (UWC).

Cacciabue P.C., and Hollnagel E. (1998) Human Factors Methods for Safety assessment in Highly Automated Environments: Task Analysis, Modelling and Data. In proceedings of International Conference on Probabilistic Safety Assessment and

Management, New York, USA, pp. 741-745.

Card, S. K., Moran, T. P., and Newell, A. (1983) *The psychology of human-computer interaction*, L. Erlbaum Associates, Hillsdale, N.J.

Charniak, E. and McDermott, D. (1985) *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts.

Chiacchio, P., Chiaverini, S., and Siciliano, B. (1993) *User-Oriented Task description for Cooperative Spatial Manipulators: One-Degree-of-Freedom Rolling Grasp*, IEEE Proceed. of the 32nd Conf. on Decision and Control, San Antonio, TX, Dec. pp. 1126-1127.

Constantine, L. L., and Lockwood, L. A. D. (1999) *Software for use: a practical guide to the models and methods of usage-centered design*, Addison-Wesley.

Correa, M. and Coelho, H. (1993) Around the architectural agent approach. In Cristiano Castelfranchi and Jean-Pierre Muller, editors, From Reaction to Cognition: 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, pages 172-185. Spring-Verlag.

Cotton, S., Bundy, A., and Walsh, T. (2000) *Automatic invention of integer sequences*. Proceedings of the AAAI-2000, Cambridge: MIT Press.

Courtois, P. (1985) *On time and space decomposition of complex structures*. Communications of the ACM, 28(6):596.

Daly-Jones, O., Bevan, N., and Thomas, C. (1999) *Handbook of User-centered Design*, Serco Usability Services.

Davis, R. and Lenat, D. B. (1982) *Knowledge-based Systems in Artificial Intelligence.* New York: McGraw-Hill.

DeJong, G. and Mooney, R. (1986) *Explanation-based Learning: An alternative view.* Machine Learning, 1(2): 145-176.

Delson, N. and West, H. (1993) *Robot programming by human demonstration: Subtask compliance controller identification*, IEEE Conference Intelligent Robots Systems.

Dennett, D. C. (1991) *Consciousness Explained.* Little, Brown and Company.

Devlin, K. (1991) *Logic and Information.* Elsevier Handbook of Logic and Language, Cambridge.

Dix, A., Finlay, J., Abowd, G., and Beale, R. (1998) *Human-Computer Interaction,* second edition. Pearson Education Limited. Second Edition.

Fu, K.-S. (1971) *Learning Control Systems and Intelligent Control Systems: An Intersection of Artificial intelligence and Automatic Control*, IEEE Trans. on Automatic Control, Vol. AC-16.

Gall, J. (1986) *Systemantics: How Systems Really Work and How They Fail.* The General Systemantics Press, Ann Arbor, MI.

Garlan, D. and Shaw, M. (1994) *An introduction to software architecture*. Technical Report CMU-CS-94-166, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, January.

Gould, J. D. and Lewis, C. (1985) *Designing for Usability: Key principles and what*

ISO (1998) ISO9241 - *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs)* - part 11 Guidance on Usability, International Organisation for Standardisation.

Iwano, Y, Sukegawa, H, Kasahara, Y, and Shirai, K. (1995) Extraction of speaker's feeling using facial image and speech. In: Proceedings of the 4th IEEE International Workshop on Robot and Human Communication, Tokyo, Japan, July. p. 101-6.

Jacobson, Ivar. (1992) *object-oriented software engineering: a use case driven approach*, ACM Press - Addison-Wesley.

John, B. E. (1995) *Why GOMS?* ACM Interactions, 80-89.

Jurafsky, D., and Martin, J. H. (2000) *Speech and Language Processing*, Upper Saddle River, NJ: Prentice Hall.

Kahneman, D., Slovic, P., and Tversky, A., editors (1982) *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge.

Klein, W. B., Stern, C. R., Luger, G. F., and Pless, D. (2000) Teleo-reactive control for accelerator beamline tuning. Artificial intelligence and Soft Computing: Proceedings of the IASTED International Conference. Anaheim: IASTED/ACTA Press.

Klein, W. B., Westervelt, R. T., and Luger, G. F. (1999) *A general purpose intelligent control system for particle accelerators*. Journal of Intelligent & Fuzzy Systems. New York: John Wiley.

Kuipers, B. and Åström, K. (1994) *The Composition and Validation of Heterogeneous Control Laws*, Automatica, Vol. 30, No. 2, pp. 233-249.

designers think, Communications of the ACM, 28, 300-311.

Grudin, J. (1992) *Utility and usability: research issues and development contexts.* Interacting with computers, 4 (2), 209-217.

Harmon, P. and King, D. (1985) *Expert Systems,* Addison-Wesley.

Haugeland, J. (1985) *Artificial intelligence: The very idea.* Cambridge, MA: MIT Press.

Henderson, A., and Card, S.K. (1986) Rooms: *The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface.* ACM Transactions on Graphics, 5 (3), 211-243.

Hesselroth, T., Sarkar, K., van der Smagt, P. P., and Schulten, K. (1994) *Neural Network Control of a Pneumatic Robot Arm*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 24, No. 1, Jan. pp. 28-38.

Hiramoto, Y, Dohi, H, and Ishizuka, M. (1994) *A speech dialogue management system for human interface employing visual anthropomorphous agent.* In: Proceedings of the 3rd IEEE International Workshop on Robot and Human Communication, Nagoya, Japan, July. p. 227-82.

HKSAR (2005) *OOM Procedures Manual.* Retrieved: Oct 21 2005, from http://www.ogcio.gov.hk/eng/prodev/eg52.htm

Holland, J.H. (1998) *Emergence.* Oxford University Press.

ISO 13407, (1999) *Human Centred Design Process for Interactive Systems*, International Organisation for Standardisation.

Kurata, T, Chang, D, and Hashimoto, S. (1995) *Multimedia sensing system for robot.* In: Proceedings of the 4th IEEE International Workshop on Robot and Human Communication, Tokyo, Japan, July. p. 83-8.

Kurzweil, R. (1990) *The Age of Intelligent Machines.* MIT Press, Cambridge, Massachusetts.

Laird, J. and Newell, S. (1991) *An analysis of SOAR as an integrated architecture.* In SIGART Bulletin 2, pages 85-90.

Laird, J. and Newell, S., and Rosenbloom, P. (1987) *SOAR: an architecture for general intelligence.* Artificial intelligence, 33:1-64.

Lee, S. (1993) *Intelligent sensing and control for advanced teleoperation*, IEEE Control Systems Magazine.

Lenat, D. B. (1977) *On automated scientific theory formation: a case study using the AM program.* Machine Intelligence, 9:251-256.

Lenat, D. B. (1982) *AM: an artificial intelligence approach to discovery in mathematics as heuristic search.* In Davis and Lenat (1982).

Lewis, J. A. and Luger, G. F., (2000) *A constructivist model of robot perception and performance.* In Proceedings of the Twenty Second Annual Conference of the Cognitive Science Socity. Hillsdale, NJ: Erlbaum.

Lif, M. (1998) Adding Usability. *Methods for Modelling, User Interface Design and Evaluation.* Acta Univ. Ups., Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 359, VIII+42 pp. Uppsala. ISBN

91-554-4186-6.

Lif, M. (1999) *User Interface Modelling* - adding usability to use cases, International Journal of Human-Computer Studies.

Lin, J. and Lee, M, C. (2004) *An object-oriented analysis method for customer relationship management information systems*. Information and Software Technology, pp. 433-443.

Lind, M., Nygren, E., and Sandblad, B. (1991) *Cognitive work environment problems and design of user interfaces* (Rep. No. 20, CMD). Uppsala, Sweden: Uppsala University.

Ling, N. (1993) *A Simple Expert System for theReasoning and Systolic Designs*, IEEE Comp. Soc.Press, CCC: o 8186 3492 8/93, pp. 128-131.

Liu, K. and Lewis, F. L. (1993) *Some Issues About Fuzzy Logic Control*, IEEE Proceed. of the 32nd Conf. on Decision and Control, San Antonio, TX, Dec. pp. 1743-1748.

Luger, G. F. (1994) *Congnitive Science: The Science of Intelligent Systems*. San Diego and New York: Academic Press.

Luger, G. F. (2002) *Artificial Intelligence structures and strategies for Complex Problem Solving*. ADDISON WESLEY.

Luger, G. F. and Stubblefield W. A. (1993) *Artificial Intelligence*. The Benjamin Cummings Publishing.

Luo, R. C. and Kay, M. G. (1989) *Multisensor integration and fusion in intelligent*

systems. IEEE Transactions on Systems, Man and Cybernetics, 19(5).

Ma, X.-J., Sun, Z.-Q., and He, Y.-Y. (1998) *Analysis and Design of Fuzzy Controller and Fuzzy Observer,* IEEE Trans. on Fuzzy Systems, Vol. 6, No. 1, Feb. pp. 41-51.

Maes, P. (1993) *Behavior-based artificial intelligence*. In Proceedings of the 2nd Conference on Adaptive Behavior. MIT Press.

Manning, C. D. and Schutze, H. (1999) *Foundations of Statistical Natural Language Processing*. Combridge, MA: MIT Press.

Marcus, M. (1980) *A Theory of Syntactic Recognition for Natural Language*. Cambridge, MA: MIT Press.

Maybeck, P. S. (1979) *The Kalman Filter: An Introduction to Concepts*, Stoch. Models, Estim. And Control, Vol. 1, pp. 3-16.

Maybury, M. T. (1993) *Intelligent multimedia interfaces*. Menlo Park. California: AAAI Press.

Mayfield, J, Labrou, Y, and Finin, T. (1996) *Evaluation of KQML as an agent communication language*. Proceedings of the 1995 Workshop on Agent Theories, Architectures and Languages. Springer-Verlag.

Mayhew, D. J. 1999. *The usability engineering lifecycle: a practitioner's handbook for user interface design*, Morgan Kaufmann Publishers, San Francisco, Calif.

McFarland, D. and Bosser, T. (1993) *Intelligent Behavior in Animals and Robots*. The MIT Press.

Meystel, A. (1985) K.-S. Fu: *Life Devoted to the New Frontier of Science*, Proc. of the IEEE Workshop on Intelligent Control, Eds. G. Saridis, A. Meystel, Troy, NY, pp. iii-vi.

Meystel, A. and Messina, E. (2000) *The Challenge of Intelligent Systems*. Proceedings of the 15th IEEE International Symposium on Intelligent Control (ISIC 2000). Rio, Patras, Greece.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986) *Explanation-based generalization: A unifying view. Machine Learning*, 1(1): 47-80.

Mockler, R. J. (1989) *Knowledge-based Systems for Management Decisions*, Prentice-Hall, Englewood Cliffs, New Jersey.

Montlick T. (1999) What is Object-Oriented Software?

Mori, M, Dohi, H, and Ishizuka, M. (1995) *A multi-purpose dialogue management system employing visual anthropomorphous agent.* In: Proceedings of the 4th IEEE International Workshop on Robot and Human Communication, Tokyo, Japan, July. p. 187-92.

Muller, J. P. (1996) *The Design of Intelligent Agents*. Springer-Verlag, Berlin.

Musto, J. C. and Saridis G. N. (1998) *A Reliability- Based Formulation for Intelligent Control,* International Journal of Intelligent Control and Systems, Vol.2, No. 2, pp. 193-209.

Nault, B. R. and Storey, V. C. (1998) *Using object concepts to match artificial intelligence techniques to problem types*, Information & Management, pp. 19-31.

Neugebauer, Degenhart, and Schraft. (1993) *Advances in virtual reality techniques for robot simulation and control,* in proceeding IEEE Conference Advanced Robot.

Nielsen, J. (1993) *Usability Engineering*. San Diego: Academic Press, Inc.

Nielsen, J. (1994) *Enhanceing the Explanatory Power of Usability Heuristics*, in Proceedings of CHI'94, ACM.

Nilsson, N. J. (1994) *Teleo-Reactive Programs for Agent Control*. Journal of Artificial Intelligence Research, 1:139-158.

Noma, H. and Iwata, H. (1993) *Presentation of multiple dimensional data by six degree-of-freedom force display*. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan, vol. 3, July. p. 1495-500.

Norman, D.A. (1986) *Cognitive Engineering*. In D.A. Norman, & S.W. Draper (Eds.), User Centered System Design (pp. 31-61). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.

Nunes, D. N. (2001) *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach*. Universidade Da Madeira. Funchal, Portugal.

Object-oriented programming. (2004) From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Object-oriented_programming/

OMG 2002 Retrieved: Oct 15 2005, from
http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?UML

Pack, R. T. (1998) *IMA: The Intelligent Machine Architecture*. PhD in Electrical Engineering, Vanderbilt University, Nashville, Tennessee.

Pang, G. K. H. (1991) *A Framework for Intelligent Control,* Journal of Intelligent and Robotic Systems, No. 4, pp. 109-127.

Partridge, D. (1991) *A New Guide to Artificial Intelligence*, Ablex Publishing Corporation, Norwood, New Jersey.

Preece, J. (1995) *Human-computer interaction*, Addison-Wesley.

Quillian, M. R. (1968) *Semantic Memory*, in: M. Minsky (Ed.), Semantic Information Processing, MIT Press, Cambridge, Mass.

Quinlan, J. R. (1986) *Induction of decision trees. Machine Learning*, 1(1): 81-106.

Rao, A. S. (1996) *Agentspeak: BDI agents speak out in a logical computable language*. Agents Rreaking Away: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, pages 42-71. Springer-Verlag.

Rasmussen, J. (1983) *Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models*, IEEE Transitional System Machine.

Rational Software Retrieved: Oct 17 2005, from
http://www.microgold.com/version3/faq_provided_by_rational_softwar.htm

Rich, E. and Knight, K. (1991) *Artificial Intelligence*. McGraw-Hill, New York, second edition.

Rodriguez, A. A. and Rowe, L. A. (1995) *Multimedia systems and applications*.

Computer; May: 20-22.

Russell, S. and Norvig, P. (1995) *Artificial Intelligence: A Modern Approach.* Prentice-Hall, New Jersey.

Rzevski, G. (2003) *On conceptual design of intelligent mechatronic systems.* PERGAMON Mechatronics.

Saridis, G. N. (1977) *Self-organizing Control of Stochastic Systems*, M. Dekker, NY.

Saridis, G. N. (1985) *Foundations of the Theory of Intelligent Controls*, Proc. of the IEEE Workshop on Intelligent Control, Eds. G. Saridis, A. Meystel, Troy, NY, pp. 23-28.

Sato, T, Nishida, Y, Ichikawa, J, Hatamura, Y, and Mizoguchi, H. (1994) *Active understanding of human intention by a robot through monitoring of human behavior.* In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Munich, Germany, vol. 1, September. p. 405-14.

Schalkoff, R. J. (1990) *Artificial Intelligence: An Engineering Approach.* McGraw-Hill, New York.

Schweitzer, G. (1995) *Motion control for human-oriented manchines,* Federation of Autonomous Control on Motion Control, Germany.

Shaw, M, Deline, R., Klein, D., Ross, T., Young, D., and Zelesnik, G. (1995) *Abstractions for software architecture and tools to support them.* IEEE Transactions on Software Engineering, April.

Shepherd, A., (1989) *Analysis and Training in Information Technology Tasks*, in D.

Diaper (Ed.), Task Analysis for Human- Computer Interaction, Ellis Horwood, Chichester, UK, pp. 15-54.

Sheridan, T. B. (1987) *Supervisory control*, in Handbook of Human Factors, New York: Wiley.

Sheridan, T. B. (1992) *Telerobotics, Automation, and Human Supervisory Control.* Cambridge, MA: MIT Press.

Shneiderman, B. (1998) *Designing the user interface: strategies for effective human-computer interaction,* Addison Wesley.

Shoham, Y. (1993) *Agent-oriented programming.* Artificial Intelligence, 60:51-92.

Simon, H. (1982) *The Sciences of the Artificial.* The MIT Press, Cambridge, MA.

Sloman, A. (1994a) *Semantics in an intelligent control system.* In philosophical transactions of the Royal Society, 349, 1689, pp 43-58.

Sloman, A. (1994b) *Representations as Control Substates in preparation.* To appear in proceedings 11th European Conference on AI Amsterdam.

Song, Y. D. and Mitchell, T. L. (1993) *A new and Simple Control Strategy for Multiple Robots Involving Redundant Motion*, IEEE Proceed. of the 32nd CDC, San Antonio, TX, Dec. pp. 1124-1125.

Sowa, J. F. (1991) *Issues in knowledge representation, in: J.F. Sowa (Ed.), Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann Publishers, Inc., San Mateo, California, pp. 1-11.

Stamper, R. (1992) *Signs, Organisations, Norms and Information Systems,* Proc. 3rd Australian Conference on Information Systems, Wollongong.

Stewart, J. (1995) *The implication for understanding high-level cognition of a grounding in elementary adaptive systems*. Robotics and Autonomous Systems, 16(2-4):107-116.

Stoll, P. A. and Ohya, J. (1995) *Applications of HMM modeling to recognizing human gestures in image sequences for a man-machine interface.* In: Proceedings of the 4th IEEE International Workshop on Robot and Human Communication, Tokyo, Japan, July. p. 129-34.

Sussman, G. J. (1975) *A Computer model of Skill Acquisition.* Cambridge, MA: MIT Press.

Thomas, S. R. (1993) PLACA, *An Agent Oriented Programming Language*. PhD thesis, Stanford University.

Tsukamoto, A. and Lee, C, W. (1995) *A methodological approach on real-time gesture recognition using multiple silhouette models*. In: Proceedings of the 4th IEEE International Workshop on Robot and Human Communication, Tokyo, Japan, July. p. 123-8.

UKAIS. (1996) *Information Systems Journal.* January. Vol. 6: Issue 1, Page 3-81.

UML (2005) *Unified Modelling Language.*
http://www.reference.com/browse/wiki/Unified_Modeling_Language

Walsh, P., (1989) *Analysis for Task Object Modelling (ATOM): Towards a Method of Integrating Task Analysis with Jackson System Development for User Interface*

Software Design, in D. Diaper (Ed.), Task Analysis for Human-Computer Interaction, Ellis Horwood, Chichester, UK, pp. 186-209.

Wang, S. (1995) *Object-oriented task analysis*. Information & Management. pp. 331-341.

White, D. A., and Sofge, D. A. (1992) *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, Van Nostrand Reinhold, NY.

Winograd, T. (1973) *A procedural model of language understanding*. In Schank and Colby (1973).

Winston, P. H. (1975) Learning structural descriptions from examples. The psychology of Computer Vision. New York: McGraw-Hill.

Winston, P. H. (1992) *Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition.

Wooldridge, M. (1994) *Time, knowledge and choice*. Technical report, Department of Computing, Manchester Metropolitan University, United Kindom.

Woolf, B. P. and Hall W. (1995) *Multimedia pedagogues-interactive systems for teaching and learning*. Computer; May: 74-80.

Yang, B. and Asada, H. (1990) *Skill acquisition for robot grinding: Learning of high-level feedback laws from teaching data.* Flexible Automation.

Yang, J., Xu, Y., and Chen, C. S. (1993) *Hidden Markov model approach to skill learning and its application in telerobotics,* IEEE Conference Robot Automation, Atlanta.