TEAMDEC: A GROUP DECISION SUPPORT SYSTEM

by

Qian Chen

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

Mark Jones, Chairman
A. Lynn Abbott
Eloise Coupey

July 20, 1998
Blacksburg, Virginia

Keywords: Group Decision Support System, Human Computer Interaction, TEAMDEC

TEAMDEC: A GROUP DECISION SUPPORT SYSTEM

by
Qian Chen
Mark Jones, Chairman
Electrical Engineering

(ABSTRACT)

TEAMDEC is a Group Decision Support System (GDSS). The development of a GDSS is supported by a broad spectrum of theories and techniques. Two major aspects of GDSS development were considered in TEAMDEC design: HCI and decision-making assistance. These two aspects interact to promote an interactive group decision support system with high quality.

Decision guidance using a script-based knowledge representation improves the GDSS's efficiency, effectiveness, and flexibility. The traditional script, however, is relatively inflexible. The proposed application, TEAMDEC, provides a set of solutions to support customization in a script system to enhance the decision guidance utilization.

The user interface design plays an important role in the overall system design. Two software development models (lifecycle model and V-model with backtracking) are adopted for TEAMDEC development. The user interface design of TEAMDEC is considered from three perspectives: functional, aesthetic, and structural.

Quality is emphasized in the development of the interactive system. It can be measured from two perspectives: those of the user and the designer. The quality measures of TEAMDEC are categorized into external properties and internal properties, corresponding to the two perspectives.

Acknowledgements

I would like to express my sincere appreciation to my advisor, Dr. Mark Jones, for his guidance, encouragement, and continued support throughout this work. His extensive knowledge and creative thinking have been an invaluable help.

I am grateful to Dr. Eloise Coupey and Dr. A. Lynn Abbott for their suggestions and valuable contributions as members of my advisory committee.

I gratefully thank Ms. Haiyuan Wang for her contributions to this research project.

I would like to thank my husband, Yiqing Zhao, for his love, support, and understanding during the past years.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS AND TABLES

Chapter 1 Introduction

1.1 Overview

Decision Support Systems (**DSS**s) are computer-based information systems that aid decision-makers with semistructured and unstructured tasks[1][22][24]. Decision support systems have a wide range of application areas, including manufacturing, finance, marketing, human resources management, and strategic planning[1]. The human decision-making process is knowledge-based, hence information-related operations are fundamental in DSSs. In recent years, the technology innovations of the Internet, networking, communication, and multimedia promote the improvement of Information Technology (**IT**).

Human Computer Interaction (**HCI**) design for DSSs has a major impact on the acceptance and effectiveness of DSSs[22]. As a medium between users and computers, the user interface is the focus of HCI. The user interface provides the physical means (such as visual, audio, and tactile) and the facilities for communication and interaction between the human user and the computer. The interactive software applications ensure the desired human computer interaction.

The proposed application, **TEAMDEC**, focuses on the implementation of interactive Group Decision Support Systems (**GDSS**s). This research investigates the use of GDSSs as a practical aid for an organization. The development of **TEAMDEC** is concerned with the factors influencing the quality of an interactive software system supporting team-based decision-making.

1.2 Overview of TEAMDEC

TEAMDEC is an interactive software system that facilitates solving problems by a group of decision-makers working together. New technologies, such as the Internet, multimedia information presentation, video conferencing, and network messaging security, are integrated into **TEAMDEC**. The objective in developing **TEAMDEC** was to create a collaborative decision-making support system with safety, utility, efficiency, effectiveness, and usability. **TEAMDEC** concentrates on software implementation for GDSSs because software plays a fundamental role in human computer interaction. The influence of the user interface on human computer interaction reflects the importance of software design and development in a HCI system. The related concepts and aspects of HCI and software development will be presented in Chapter 3. **TEAMDEC** has several unique and attractive characteristics. The major characteristics include that ease of use, security, platform independence, multiple facilities, and abundance of information.

Usability is a key concern of **TEAMDEC**. The essential goal of usability is to make a system easy and pleasant to learn and use[10]. **TEAMDEC** is easy to use because of its user interface, such as a succinct menu and a familiar Internet browser. **TEAMDEC** runs in Netscape Communicator, which is well accepted by Internet users and familiar to them. The user can easily go to the system in the same way as opening any web page with its URL address. To some extent, this feature can reduce a system novice's learning time. The other reason for choosing the Internet is to take advantage of the global information space on the Internet. The Internet not only provides a huge amount of information that covers nearly everything in the world, but also offers various search engines. Obviously, it is a valuable external information source for decision-makers in **TEAMDEC**.

TEAMDEC helps decision-makers by providing multiple facilities, including the ease of communication (i.e., video conferencing, group discussion board, whiteboard, and notice sending), and information operation (i.e., searching data, retrieving data, capturing procedures automatically, and scripting). These facilities contribute to information processing, information dissemination, data retrieval, and decision guidance generation that are necessary functions in group decision support systems.

TEAMDEC can be used safely. It is able to prevent or minimize damage from the outside. **TEAMDEC** is equipped with a security capability to protect against undesirable intrusion. Moreover, even the communication contents of a group meeting are well protected, which makes it difficult for any undesirable audience to acquire information.

Platform independence is an underlying factor which supports system flexibility. **TEAMDEC** is a cross-platform software system. It can work well on most of the popular platforms, such as Windows 95, Windows NT, and UNIX. A change of target environment will not cause system malfunction.

1.3 Contributions

The research focuses on implementation of a high quality group decision support system. **TEAMDEC** is a group decision support system for the Internet. It integrates diverse software components and is platform independent. An important contribution of **TEAMDEC** is that the **TEAMDEC** script system combines the idea of scripts and a suggestion generation mechanism to improve the quality and efficiency of decision-making. The system is able to predict the user's next possible action goal and provide the action suggestion scripts based on the user's past interaction and knowledge stored in databases. The user is allowed to customize the scripts to carry out a new task.

1.4 Overview of content

The following sections of this paper are centered around **TEAMDEC**, an interactive group decision support system. As mentioned in the previous section, the goal of **TEAMDEC** is to develop an interactive system with safety, utility, efficiency, effectiveness, and usability. The development of **TEAMDEC** will involve knowledge of decision support systems, human computer interaction, script, script-related concepts, and

software development. The related concepts and principles will be addressed in Chapters 2, 3, and 4. Chapter 2 focuses on decision support systems. This paper contains a review of DSSs, including definitions, characteristics, the human decision-making process, and subclasses. One important characteristic of DSS, customization, is closely related to the systems' flexibility which affects software quality. Customization is addressed in Chapter 2. This chapter discusses three subclasses of DSS: Expert Systems (ES), Executive Support Systems (ESS), and Group Decision Support Systems (GDSSs). The context emphasizes features and key aspects of GDSSs. Chapter 3 focuses on human computer interaction, HCI software development, and quality properties. The context is HCI-related sciences, research on HCI factors; the HCI development process, methods and related techniques; user interface design; and the issues of interactive software quality. The quality is represented by two types of properties according to different perspectives. These properties are categorized into external properties and internal properties corresponding to the user's and designer's points of view. Scripts are an important feature of **TEAMDEC**. Scripts contribute to efficiency improvement in the interactive system and suggestion utilization in the decision support system. The purpose of Chapter 4 is to address the significant characteristic of **TEAMDEC**: scripts. A script is a form of knowledge representation which influences the quality of decision-making outcomes. **TEAMDEC** is illustrated in Chapter 5. Chapter 5 gives an overview of the **TEAMDEC** environment (software and hardware), task analysis, and quality properties of **TEAMDEC**. Because the achievement of the system's goals depends on quality analysis of the software properties of **TEAMDEC,** the interaction between each property will be the emphasis of Chapter 5. Chapter 6 is a short discussion of what has been achieved. It summarizes the HCI and GDSS principles applied in the implementation of an interactive group decision support system.

Chapter 2 Decision Support Systems

2.1 Introduction

TEAMDEC is a group decision support system, thus the relevant knowledge and principles of group decision support systems have been applied in the implementation of **TEAMDEC**. In addition, the ideas of expert systems and executive support systems are useful to **TEAMDEC.** The knowledge of the human decision-making process underlies the development and design of decision support systems. Therefore, the objective of this chapter is to discuss decision support systems, including the definition of decision support systems, the human decision-making process, and three subclasses of decision support systems (expert systems, executive support systems, and group decision support systems). Among the three subclasses of decision support systems, the issues of group decision support systems will be discussed in the greatest detail.

Because of the rapid development in areas such as networking, communications, multimedia, and databases, ***Information Systems*** (***IS***) have been improved and the range of application has been expanded. Moreover, ***IS*** innovations enhance the importance and capability of ***Decision Support Systems*** (***DSS****s*).

A ***Decision Support System*** is a computer-based information system that affects or is intended to affect how people make decisions[24]. *DSSs* aim to take advantage of the strengths of humans and computers, including human decision-making capability and computer data processing capability. *DSSs* provide computer-based assistance to human decision-makers to supplement the decision powers of the human with the data manipulation capabilities of the computer. *DSSs* primarily consist of hardware, software, and the human element. They are designed to assist decision-makers at any organizational level, depending on particular application requirements.

Confronted with the dynamic information world, one expects a large amount of information to provide sufficient material for analyzing problems deeply, completely, and

rapidly. Therefore, an efficient and effective information processing and decision-making assistance system becomes necessary. The principles and concepts involved in developing a high quality information processing and decision-making assistance system will be discussed in this chapter.

In this chapter, the process of decision-making will be briefly introduced. One important feature, customization, needs to be mentioned. In designing a flexible and adaptive decision support system, customization is a key factor. Customization implies that the ***DSS*** must be tailored to specific decision-making environments[24]. Customization will be discussed in Section 2.3. Section 2.4 will give an overview of three special subclasses of *DSSs*: ESs, ESSs, and GDSSs.

In the following section, examples will be used to illustrate characteristics of *DSSs*. Although the illustrations are not exhaustive, they do show some advantages of *DSSs* in human decision-making, such as efficiency and processing speed improvements, data accuracy guarantees, and human workload reduction.

Examples:

1. *Ad hoc* data retrieval

A computer works much faster and more accurately than a human does in handling massive quantities of data. By means of a computer and application software, *DSSs* help the decision-maker to reduce the workload of handling a large amount of data, while improving efficiency and accuracy. The basic functionality of *DSSs* is to provide the decision-maker with the ability to retrieve information selectively on an *ad hoc* basis. *DSSs* are not only able to get data from internal databases (e.g., the internal data of an organization), but also from external databases (e.g., the information from other web databases). *DSSs* have the capability to retrieve data selectively, as well as to aggregate and summarize data.

2. Information presentation

DSS often present computational results in a variety of formats. The formats include traditional ones (e.g., tables and graphics) as well as new patterns (e.g., animation, audio, and video). In addition, the color and the shape of graphical

components are factors of information presentation. By means of the visual aids, users can recognize the relationship of a collection of data and understand some complicated results more easily. Another goal of visual-aided information presentation is to highlight the emphasis of information.

3. Multiple decision aids

DSSs provide interactive decision aids that combine data retrieval, stylized displays, and model-based processing to satisfy some particular decision needs. For example, a decision aid can help a decision-maker to choose from many alternative solutions that might come from different databases, or be drawn out under different control conditions or by using different parameters. Therefore, many alternatives may need to be taken into account or alternative needs may need to be considered in detail when solving complicated problems. The pre-defined decision rules usually play an important role internally. Eventually, the decision outcomes depend on human analytical ability.

2.2 The process view of decision-making

Knowing something about the process of decision-making can help software designers understand how a computer-based system affects human decision-makers and the roles human decision-makers play in *DSSs*. Some experiments present convincing evidence that *DSSs* intervene in the way decisions are made. These experiments include a number of outcome-oriented empirical studies (e.g., Sharela, Barr, McDonnell (1988)[45], and Benbasat and Nault (1990)[30]), and process-oriented studies (e.g., Jarvenpaa (1989)[32], and Todd and Benbasat (1988)[52]). Moreover, these studies show that human and system-based decision-making processes have both strengths and weaknesses. In some cases, *DSSs*-aided decisions might not be as good as users expect, perhaps even be worse than non-*DSSs*-aided decisions. For example, *DSS* intervention may cause decision-makers to rely on ineffective processes. If decision-makers adopt inferior processes, it may lead to worse consequences[24]. The *DSSs* designers, therefore,

need to pay attention to how to amplify the *DSS*' strengths and attenuate their weaknesses for desirable performance. Understanding decision-making procedures and careful design are prerequisites to good *DSSs*.

Hence, the decision-making procedure will be briefly explained. Human decision-making procedure has two major characteristics[24]:

- Decision-making is not a point-event. It is a complex sequence of differentiated activities occurring over time. For example, it might relate to individual knowledge and experience, and
- Decision-making is not monolithic. There could be a number of distinct paths to reach a decision. It is important but difficult to figure out the optimal path.

The above two characteristics essentially affect the overall development of *DSSs*. Because decision-making involves a complex sequence of activities over time, it implies there are at least three functions that should be assigned to *DSSs*: 1) the capability of capturing and saving information from previous activities; 2) data processing capability; and 3) data retrieval capability. A decision support system can provide several ways to support the second characteristic of decision-making procedure, such as multiple facilities to access various information, and decisional guidance or suggestion. The practical implementation will be presented in Chapter 4 and Chapter 5.

2.2.1 The process of decision-making

The decision-making process can usually be decomposed into four phases [22]:

(1) Intelligence

(2) Design

(3) Choice

(4) Review

Intelligence

This phase consists of two parts: *recognition* and *diagnosis*. "*Recognition*" is a need for decision activity that triggers the decision-making process[38]. "*Diagnosis*" is a

period during which the decision situation is clarified and defined[38]. This phase can be described as the process of problem finding.

Design

In the *Design* phase, decision-makers prepare alternative courses of actions in response to the situation diagnosed in the *Intelligence* phase. Decision-makers might respond by searching for ready-made solutions, by modifying ready-made solutions, or by developing custom-made solutions[38]. Searching solutions can be done with the aid of information searching engines in *DSSs*. The alternatives found can be displayed in various presentation formats for further analysis.

Choice

The *Choice* stage is the final phase to reach a decision. If the *Design* phase generates only one option, the decision-makers take action of either acceptance or rejection in the *Choice* phase. If the *Design* phase generates a set of potential outcomes, the decision-makers need to choose one under certain rules. Sometimes decision-makers are confronted with multiple, conflicting objectives; they must either choose from or trade off these objectives. So, multiplicity of criteria becomes a principal factor contributing to the difficulty of the *Choice* phase. In practice, some decision rules for multi-attribute problems have been applied in solution strategies, such as holistic evaluation methods, heuristic elimination and holistic judgment (see Table 2.1) [45].

Besides multi-attribute problems, another obstacle between a decision-maker and a decision is the probabilistic nature of the world. Although we assume alternatives under certain premises, in fact, we still can not take for granted what the future state will be and what outcomes will follow from our action. There are two identified classes of structural problems that lack of certainty: *decision under risk* and *decision under uncertainty* [38]. *Decision under risk* represents those cases for which the decision-maker knows the set of possible outcomes and their probabilities, although they don't know with certainty the outcomes that will follow from actions. *Decision under uncertainty* represents those cases for which the decision-maker does not even know the probabilities of the outcomes that follow from actions.

Table 2.1 Some decision rules for a multi-attribute problem

Holistic evaluation	Multi-attribute utility theory Additive linear models
Heuristic elimination	Lexicographic rules Elimination by aspects Conjunctive elimination Disjunctive elimination Dominance Additive difference
Holistic judgment	Standard operating procedures Intuitive affect Reasoning by analogy

Post-decisional activities

There are several interpretations of *post-decisional activities*. Simon identifies it as *Review*, wherein past choices are assessed[50]. Sprague and Carlson add an *implementation* phase[51]. Mintzberg *et al*. identifies an *authorization* routine[42]. Indeed, a set of activities following *Choice* can be regarded as this phase:

- *authorization* or *ratification*, which requires the presentation and defense of decisions to higher levels of the organization;
- *implementation*, which often triggers other lower level decision-making processes;
- *review* and *control*, which also may trigger new decision-making activity if the actual performance does not conform to plans.

2.2.2 Structuring the process

Within each phase of the decision-making process, there are competing approaches. For example, during the *Design* phase, decision-makers can generate alternative actions by searching for ready-made solutions, by modifying ready-made solutions, or by developing custom-made solutions. For the *Choice* phase, various techniques are identified for solving multi-criteria problems. Different strategies and techniques will probably lead to different solutions, so defining the decision-making process is necessary and pivotal (see Figure 2.1). The process of decision-making can be decomposed into structuring the decision-making process and executing the decision-making process. Structuring the decision-making process not only defines how the decision will be made, but also specifies the information-processing and problem-solving activities to be performed, and the order of activities as well. Of course, human judgment is a crucial part of the decision-making process. The role of human judgment includes predictive and evaluative judgments of process execution, deciding and structuring the decision-making process, and selecting the techniques, models, and data, which are appropriate for the task. Predictive judgments reflect decision-makers' expectations on future conditions[36], e.g., claiming that the next month's sales will exceed this month's. Evaluative judgments express decision-makers' preference[36], e.g., trading off performance and cost. Execution of the process entails actual performance of the various information-processing and problem-solving activities. Structuring the decision-making process and executing the decision making process are not independent; in other words, they often interact. Sometimes the decision-making process is dynamically structured along with the procedure towards a decision. Even if a complete process plan has been made before beginning execution, modification of the original plan may be required as the results are obtained.

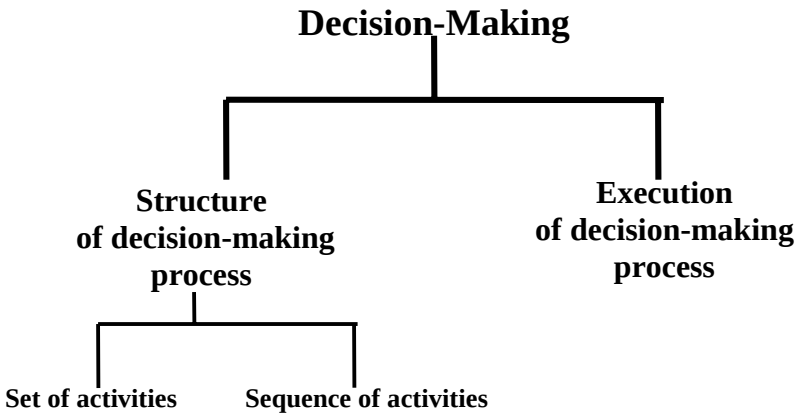


Figure 2.1 Process view of decision-making

2.2.3 Increasing process complexity

The previous section emphasizes individual decision-making behavior. Individual decision-making is just one of several contexts of *DSSs*. *DSSs* have been used to support five decision-making contexts: individual, group, organizational, interorganizational, and societal. Essentially, various contexts are collections of individual decision-making activities in different manners and/or using different processing techniques. In complex decision-making contexts, the decision-making process is not a simple sequence of information-processing activities any more, but rather a network of activities, a collection of sequences that intersect at many points. The participants might be at different levels, performing different/relevant tasks.

2.3 Customization

In this section, two system attributes of *DSSs*, *precustomization* and *customizability*, will be briefly introduced and the relationship between them will be discussed.

Precustomization and customizability are two system attributes of *DSSs*. These two attributes are not mutually exclusive. A *DSS* can be both precustomized and customizable. For example, if a system provides access to predefined datasets, it is precustomized; if a system allows its users to modify those datasets or create new datasets, it is customizable. To support a specific decision-making environment, a *DSS* might be precustomized with respect to some aspects of the environment, such as predefined datasets, users, tasks, settings, models, visual representations, functional capabilities, and so forth. For example, a *DSS* is specifically designed for a unique decision-making environment, a set of particular people performing particular tasks in a given setting. The additional assumption is that there will not be other people performing the same tasks in other settings, or someone performing other tasks in the same setting. That *DSS* will be highly precustomized. The highly precustomized system restricts its users' capability to modify settings or extend its features. A customizable system, however, allows its users to tailor it to meet the needs of its environment. The factors and parameters of the environment may vary in terms of some aspects of the environment. Typically, highly customizable systems are general or multi-purpose decision support systems. Precustomization and customizability co-exist. In general, two systems can not be compared in order to draw a conclusion on which is more precustomized or customizable; because a system can be precustomized or customizable in so many ways depending on the many aspects of decision makers, tasks and settings. However, the comparison can be how two systems differ in precustomization or customizability.

2.3.1 Definitions

Precustomization: "the degree to which, and the manner in which, at the time it is delivered to the user, some or all of the features of a *Decision Support System* have already been tailored to the specific decision-making environment that it is intended to support"[24].

Customizability: “the degree to which, and the manner in which, a *Decision Support System* empowers its users to specialize it as needed to fit the environment that it supports”[24].

2.3.2 Relationship

Precustomization and *Customizability* are a pair of system attributes. *DSSs* can be both precustomizable and customizable because a precustomized system does not preclude further customization. Precustomizing a *DSS* and making the system customizable are two complementary ways of facilitating system specialization. For example, users can build a set of new activities based on previous activities if the actions that are to be taken are similar to the previous ones. Users retrieve data according to a set of requirements. Some of the parameters and control conditions are used to adapt to the given environment. These parameters and control conditions are usually predefined, which manifests precustomization. On the other hand, users are allowed to modify the sequence order of the previous events, and change some parameters of procedures. In other words, the users can customize new action sequence according to their current needs. Although precustomization and customizability co-exist in *DSS*, precustomization takes away some freedom that would otherwise be available for customizability.

2.4 Several subclasses of *DSS*

There are three identifiable subclasses of *DSSs*: *Expert Systems (ESs), Executive Support Systems (ESSs),* and *Group Decision Support Systems (GDSSs).* Emphasis will be placed on *GDSSs* because **TEAMDEC** is a *GDSS* software system. On the other hand, the design of **TEAMDEC** also refers to the ideas in *ESs* and *ESSs*, such as decision guidance, key items reporting in an understandable format to increase executive productivity, and timely data delivery.

2.4.1 Expert Systems (ESs)

Expert Systems (***ESs***) are an instance of *DSSs* with unique characteristics. *ES* are designed to solve a problem in a particular well-defined area which has been successfully solved by a human expert. A typical ES generally consists of six components: knowledge acquisition facility, knowledge base (rule base and database), knowledge-base management system, inference engine, user interface, and explanation facility[1]. Two types of *ESs*:

- Stand-alone ESs

  Stand-alone ESs are computer-based systems that acquire knowledge from users or experts in this area. By invoking their inference engine and knowledge bases, they produce solutions for users[1]. ES are highly precustomized and highly restrictive because their design objective is to solve a particular problem. User-control is limited to providing facts and asking simple questions.

- Expert support system

  An expert support system is a computer-based support system that is embedded within *ES* or *ES* (inference) technology. The expertise or inference might be embedded either as a conclusion producer or as suggestive decision guidance[23]. The first approach imposes the conclusions which are drawn by the embedded ES on the decision-makers and constrains the use of system features. The latter approach provides the decision-makers with a recommendation on how to proceed in using the system.

2.4.2 Executive Support Systems (ESSs)

The definition of ***ESSs*** is " *the routine use of a computer-based system, most often directly access to a terminal or personal computer, for any business function. The users are the CEO or a member of the senior management team reporting directly to him*

or her. Executive support systems can be implemented at the corporate or divisional level."[45]

ESSs are able to easily obtain predefined displays of current data that provide a snapshot of the organization's key status indicators. There could be two factors influencing and stimulating the development of an ESS. One is the combination of communication and database technology which allows a much more comprehensive and immediate snapshot of the status of the organization and the environment. The other factor is that of the accelerating pace of business and the simultaneous need to make organizations more flexible. *ESS* seem similar to the ***Information Reporting Systems*** (**IRSs**)[24]. There are some differences between *ESSs* and IRSs.

- Data from ESSs are more current than the data from IRSs, since an ESS's data are continuously updated whereas IRSs give periodic reports.
- ESSs contain key status indicators and are tailored to meet the needs of executives, whereas IRSs are not focused on any managerial needs.
- ESSs provide easy, on-line access to the relevant displays, whereas IRSs provide hardcopies.

2.4.3 Group Decision Support Systems (GDSS)

Group decision support systems (GDSSs), a subclass of *DSSs*, are defined as information technology-based support systems that provide decision-making support to groups[24]. They refer to the systems that provide computer-based aids and communication support for decision-making meetings in organizations. The group meeting is a joint activity in which a group of people is engaged with equal or near-equal status. The activity and its outputs are intellectual in nature. Essentially, the outputs of the meeting depend on the knowledge and judgment contributed by the participants. Differences in opinion may be settled by negotiation or arbitration.

Components of GDSS

The difference between *GDSSs* and *DSS*s is the focus on the group versus the individual decision-maker. The components of a *GDSS* are basically similar to those of *DSS*, including hardware, software, and people; but in addition, within the collaborative environment, communication and networking technologies are added for group participation from different sites. Moreover, compared with *DSSs*, *GDSSs* designers pay more attention to the user/system interface with multi-user access and system reliability because a system failure will affect a multi-user group, rather than just an individual. There are three fundamental types of components [1] that compose *GDSSs*:

1. Software

   The software part may consist of the following components: databases and database management capabilities, user/system interface with multi-user access, specific applications to facilitate group decision-makers' activities, and modeling capabilities.

2. Hardware

   he hardware part may consist of the following components: I/O devices, PCs or workstations, individual monitors for each participant or a public screen for group, and a network to link participants to each other.

3. People

   he people may include decision-making participants and /or facilitator. A facilitator is a person who directs the group through the planning process.

Benefits claimed for GDSS

There are three benefits claimed for *GDSSs*: increased efficiency, improved quality, and leverage that improves the way meetings run[22].

Due to increasing computer data processing power, communication and network performance, the speed and quality for information processing and information transmission create the opportunity for higher efficiency. Efficiency achievement depends on the performance of hardware (e.g., PCs, LAN/WAN) and software. With regard to the software aspect of *GDSSs*, the software architecture with database management and an interactive interface affects system run time efficiency and

performance. Improved quality of the outcomes of a group meeting implies the increased quality of alternatives examined, greater participation and contribution from people who would otherwise be silent, or decision outcomes judged to be of higher quality. In a *GDSS*, the outcome of a meeting or decision-making process depends on communication facilities and decision support facilities. Those facilities can help decision-making participants avoid the constraints imposed by geography. They also make information sharable and reduce effort in the decision-making process. Therefore, those facilities contribute to meeting quality improvement. Leverage implies that the system does not merely speed up the process (say efficiency), but changes it fundamentally. In other words, leverage can be achieved through providing better ways of meeting, such as providing the ability to execute multiple tasks at the same time.

Factors that affect GDSS

Research indicates there are usually several factors affecting *GDSSs,*

- *Anonymity*
- *Facility design*
- *Multiple public screens*
- *Knowledge bases and databases*
- *Communication network speed*
- *Fixed versus customized methodology*
- *Software design*
- *Group size and composition*
- *Satisfaction*

Information needs of groups

It is fundamental and important to clearly understand what groups do and which of their activities and procedures can be and should be supported by *GDSSs*. Also, it is necessary to know the information needs of groups and examine how best to support these information uses with *GDSSs*. The information needs of groups covers a broad spectrum.

- Database access

Databases are one of the basic components of *GDSSs*. *GDSSs* offer groups the advantage of accessing databases or some on-line service for the lastest information. The databases can be internal or external databases. This is a key element in information retrieval and sharing in a group meeting. The requirements on the presentation and functions of the obtained information can be summarized as follows[24]: information should be presented in clear and familiar ways; information presentation and all other associated management control aspects should assist the decision-maker to guide the process of judgment and choice; with an explanation facility, information containing an advice or decision suggestion enables users to know how and why results and advice are obtained; and information should be helpful to improve the precision of task situation understanding. Moreover, information needs are based on the identification of the information requirements for the particular situation.

- Information creation

In addition to a decision, the output of the meeting is new information. In a *GDSS*, all input into the computer is usually captured. In some cases, the actions of individual members of the group are stored in a database, file or some other storage format. Making a decision is not a point-event. The decision is produced based on valuable knowledge. It is worthwhile to save the valuable information in efficient ways which make it convenient for further use.

- Dissemination of information, decisions, and responsibilities

An often-cited advantage of *GDSSs* is that the participants are allowed to know what new information was created, what decision was reached, and who is responsible for follow-up or for implementing decisions.

- On-line modeling

On-line modeling is the next step beyond sharing existing data. For example, the participants can perform on-line analysis and send out their results or ideas to a public board.

- Visual decision-making

Some decisions involve visuals rather than words or numbers. Intuitively, graphics with shape, size, and color, might make it easier and faster for users to have an overall view of the information.

- Multimedia information presentation

  The combination of visible and audible information presentation format impacts the traditional information presentation format. The benefits of multimedia presentation include better interaction, more straightforward and effective communication in the group, and decreased learning time.

- Idea generation

  A variety of idea generation packages or methods exists for *GDSSs* use.

- Voting

  This implies the ability to vote, rank, or rate.

*GDSS*s have an impact on the work of individuals, groups, and organizations. In general, the performance improvement and satisfaction of individuals will lead to the improvement of the group. Both hardware and software will influence *GDSSs*. For example, the performance of a network will directly affect data transmission. If the network slows down, it will constrain the *GDSS*'s capability of on-line data processing. Video and audio devices are adopted to make it more straightforward for users to recognize multimedia information, which results in the improvement of efficiency and in effectiveness, as well as in the quality of meeting outcomes. Hardware development and innovation are significant for *GDSSs* performance. Software is another factor that has an impact on *GDSSs* performance. The quality properties of an interactive software system and the software design principles will be discussed in Chapter 3. Software and hardware interact, and, to a certain extent, trade off performance. Because of either software or hardware, the performance can be enhanced or inhibited depending on the target environment.

2.5 Summary

This chapter focuses on decision support systems. It provides an overview of *DSSs*, including concepts, features, subclasses, and relevant knowledge. *DSSs* are computer-based systems for supporting human decision-making. *DSS*s benefit decision-makers in various ways, such as faster decision-making, more comprehensive information, improved accuracy, improved communication, improved customer service, decreased user workload, and an increased user-satisfaction level. Understanding of the human decision-making process underlies the development and design of decision support systems. Customizability and precustomization, a pair of attributes of *DSSs*, affect flexibility and adaptivity of a decision support system. Three subclasses of *DSSs* were discussed: *Expert Systems*, *Executive Support Systems*, and *Group Decision Support Systems*. The emphasis is put on *GDSSs*. Analyzing the factors and components of *GDSSs* contributes to further practical design and development.

Chapter 3 HCI and Interactive Software design

3.1 Introduction

This chapter concentrates on the issues of interactive software design and development. The concepts and related science of human computer interaction are fundamental knowledge. Specifically, the theories on human factors lead to an understanding of human behavior that is helpful to the interactive software designer. The software development process and relevant techniques will be briefly introduced. The issues of software design, including user interface design, interactive software quality, and interactive software design principles, are principal topics in this chapter.

Human computer interaction (***HCI***) is an exciting and important area of computer science. It combines the physical, logical, conceptual, and language-based interactions between the human user and the computer for achieving some goals[11]. It involves the interaction of person, task, and computer. The interactive system is composed of users, software, and hardware. *Human factors* refers to all human characteristics (psychological, physiological, social, etc.) that have the potential of influencing the design of human-oriented *HCI*[20]. It is used to describe the study of people and their behavior in the context of using machines, tools, and other technological developments with which to carry out work.

A key focus of *HCI* the design for providing solutions to identified problems, taking full account of all the problem constraints and requirements. The purpose of design is to enable work and other activities to be performed more efficiently, effectively, and with more satisfaction. In human computer interaction, software plays a fundamental role. The primary objective of design for interactive systems is to implement user interactions with computers for the desired performance. The overall concern, therefore, is quality of work. The rapid proliferation of interactive systems has resulted in increasing interest in the quality of the user interface of interactive systems. The quality

is considered from several aspects, such as usability, effectiveness, efficiency, flexibility, security, and maintainability.

Interactive software development is discussed in Section 3.2.3. The quality of an interactive software system is discussed in Section 3.3. In the next section, concepts and characteristics of *HCI* are briefly addressed.

3.2 Human computer interaction

HCI is a comprehensive study of interaction between human, computer and task. There are four aspects with which *HCI* is principally concerned[11]:

- understanding the people who will use the computer program (the users);
- the domain and institutional structures that might affect how or when the users would use the program;
- the tasks that are carried out / are going to be carried out; and
- the hardware and software solutions that intend to meet the requirements and constraints from users and tasks.

The major implementation concern of *HCI* is the design that draws out solutions according to the various requirements and constraints from users and tasks, and eventually implements the application in practice. The design aspect of *HCI* is crucial. It does not, however, just apply to the software and hardware. Other factors, such as users and tasks, must be taken into account within the design stage. The issue of the relationship between the users, tasks, hardware, and software needs to be considered from theoretical, methodological, and practical perspectives. Moreover, *HCI* is a multidisciplinary subject that is based on knowledge derived from the subject areas of science, engineering, and art. It encompasses computer science, psychology, ergonomics, linguistics, and sociology. *HCI* involves the development and application of principles, guidelines, and methods to support the design and evaluation of interactive system.

3.2.1 Theory

In *HCI*, humans play a very important role. The fundamental purpose of the computer system is to complement and extend human capability, and improve work performance and quality. To accomplish these goals, research must be conducted to determine optimal *HCI* design. The following discussion is focused on the psychological and physiological factors pertinent to effective *HCI* design.

Psychological factor

Psychology is a science dealing with the mind and mental processes, feelings, desires, etc[20]. Psychology is concerned with understanding, modeling, predicting, and explaining what is perhaps the most complicated phenomenon of all, namely human behavior. Psychology approaches the study of human behavior from the perspective of attempting to identify the mental structures and processes that must exist in order to account for behavior. Psychological methods include observation, surveys, laboratory experiments, case studies, simulation, and other forms of investigating the many different aspects of human behavior[11]. Psychological theories cover a wide range of topics, including motivation, emotion and cognition; social, biological and organizational aspects; human development and maturation from birth to death; and aspects of normal and abnormal human behavior.

To an interactive system designer, the knowledge of psychology theories and research on human behavior is very helpful. How the computer user thinks and behaves in interaction with the computer is a factor which *HCI* designers need to take into account. For example, human mental activities, knowledge representation, and skill acquisition are usually involved in human computer interaction. Much research has been carried out on the nature of human memory from philosophical, anatomical, physiological, and psychological perspectives[11]. People have the ability to remember, recognize and recall information. An understanding of human memory has much to contribute to *HCI* because it can help in designing a computer system to overcome the

limitation of human memory ability and to guide human behavior. Human memory can be thought of in terms of prepositional networks[30]. The structuring of knowledge in memory has consequences for the way activation spreads through the network. Naturally, memory is a dynamic store. New information is added and knowledge in the memory is reorganized. Memory can not only be thought of as storage of knowledge of events, actions, and images, but also as processes operating upon representations[32]. In all of this mental activity, knowledge is critical. The knowledge people acquire and use is represented in the form of knowledge structures. As mentioned in Chapter 2, in the human decision-making process, activities are undertaken based on knowledge. A computer-based system can help people fulfill and expand their capabilities of the knowledge acquisition and storage. Mental faculties can be categorized into three areas: cognitive, affective, and co-native; or knowing, feeling, and doing[11].

The study of cognition is primarily concerned with knowing and includes the nature of human intelligence, understanding, and problem solving. It also includes the study of how people learn and develop their skills, and use their knowledge. Cognitive psychology plays a major role in the understanding of human behavior and the mental processes that underlie it. Thus, cognitive psychologists have attempted to apply relevant psychological principles to *HCI* by a variety of methods, including development of guidelines, the use of models to predict human performance, and the use of empirical methods for testing a computer system[15]. Figure 3.1 illustrates the simplified view of the human information processing system[14].

Psychology contributes a lot to the understanding of *HCI*. The computer system designer and developer are required to make decisions based on common assumptions about the user's prior knowledge, experience, and ability to learn. These assumptions directly influence the quality of the interaction between a human and the computer.

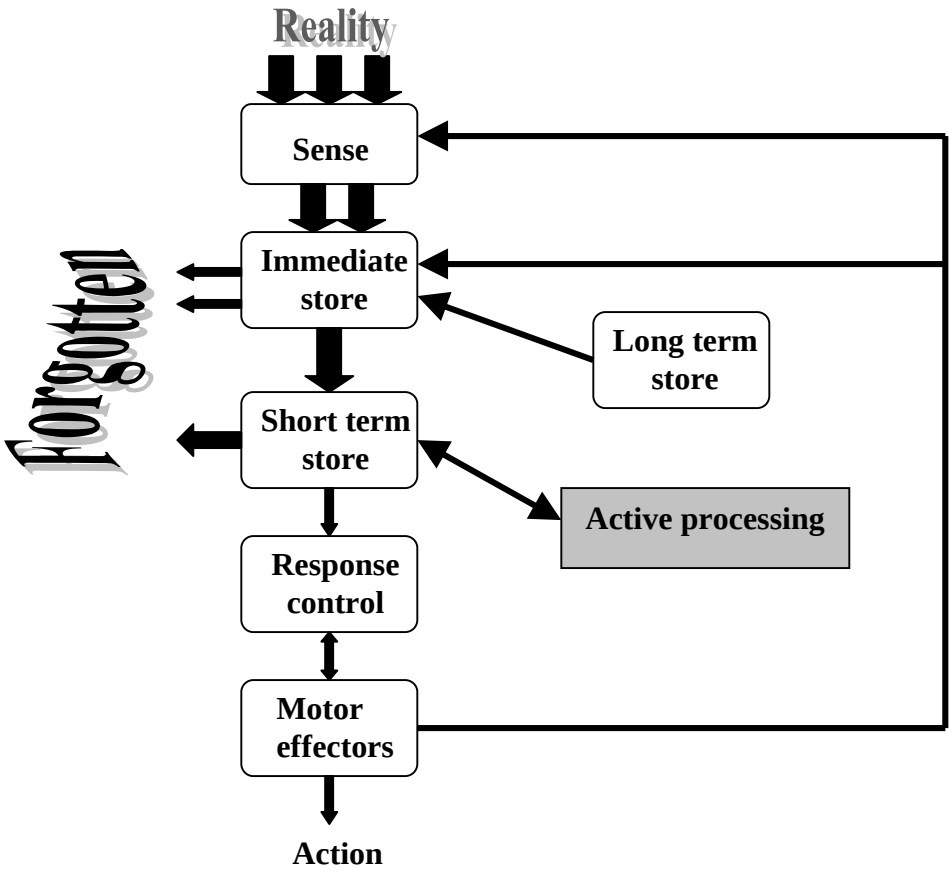


Figure 3.1 Human information processing model

.

Physiological factor

Physiology is the science which examines the functions and vital processes of living organisms and their parts[20]. It emphasizes physical characteristics of humans, rather than the cognitive characteristics. “Ergonomics” is the study of the relationship between users and their computer-based work and work environment, especially fitting the job to the requirements and abilities of computer users[13]. Thus the designers need to consider some factors that are known to affect the way people use machines. For example, the physical characteristics are taken into account for input/output devices. The

size and force requirements of a key are tested to ensure the optimum range for users to press the key comfortably and easily. In addition to the concern with physical characteristics, the information representation on the screen, the training provision, the on-line help message, etc., are factors affecting use of the computer system.

3.2.2 Methods in HCI development

The methods applied in *HCI* development and design are based on computer science. Computer science is concerned with the theory, methods, algorithms, and the practice of computation. Computer science includes the study of the computer languages for writing programs, hardware environments upon which programs can execute, structural properties of programs, and architectures for designing programs. *HCI* not only depends on computer science, but also raises special concerns for computer science.

For example, the user interface design involves the understanding of human factors, investigating and analyzing user tasks and requirements, and developing techniques and methods of software design to accommodate and improve the design of user interfaces. Many issues arise from *HCI*:

- requirements for fast response,
- convenient and efficient input manners,
- advanced forms of output (for instance, graphical output),
- the use of logic and mathematics to specify the complicated nature of user interaction, and
- development of new programming environments to allow user interface to be constructed more easily.

From a technical standpoint, those requirements can be implemented by those methods from computer science which enable software and hardware applications to carry out their intended functions.

Consequently, *HCI* must identify the special requirements of an interactive system and must also ensure there are adequate and reliable ways to know whether the design meets the requirements. In this section, the user interface issue is simply stated for

interpreting the relationship between *HCI* and computer science. Because the user interface is a very important and necessary part of an interactive system, user interface design and user interface related issues will be described in more detail in Section 3.3.

Conceptual framework

Given the different disciplines and problems in *HCI*, some form of framework for *HCI* is useful.

(1) Long and Dowell (1989) have proposed a framework for *HCI* that abstractly deals with the different representations and processes that occur in applying knowledge from an appropriate discipline to a particular problem[38]. This is a general framework that is not just particular to *HCI*, but can also be used in other areas of research.

(2) Diaper and Johnson (1989) show how the design of an information technology-training syllabus can be understood in terms of a framework[33].

(3) Carroll (1990) has developed a more detailed framework for *HCI* [32]. Carroll's framework is essentially a task-artifact cycle that views *HCI* as requiring an understanding of tasks and designs and the way they cyclically influence each other. Artifacts are products developed and maintained as part of the organization's process. A task implicitly sets design requirements for the development of artifacts and the use of artifacts redefines the task for which the artifact was originally designed. To design more useful artifacts, the designer should better understand tasks that people are undertaking and apply the understanding of tasks to the process of design. Carroll further redefined the task-artifact cycle to include the design activities of computer science and the psychological basis for understanding tasks. Figure 3.2 and Figure 3.3 show the task-artifact cycle and an elaboration of the task-artifact cycle.

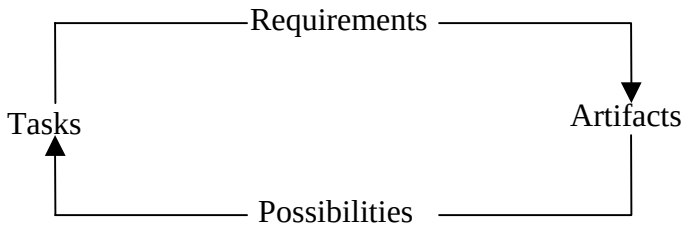


Figure 3.2 Task-artifact cycle

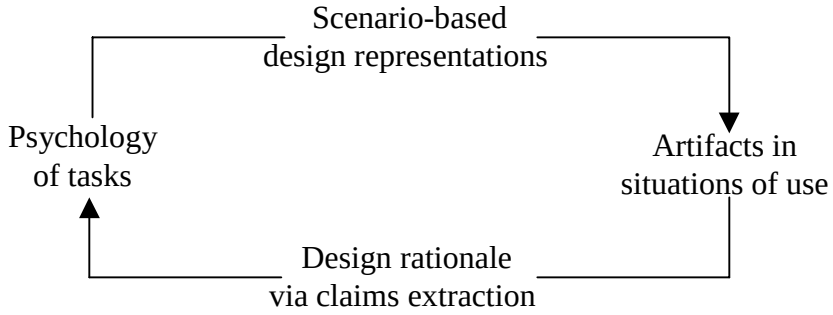


Figure 3.3 An elaboration of the task-artifact cycle

3.2.3 Practice

In practice, the purpose of developing interactive computer systems is to help system users achieve their specific goals as efficiently as possible. Donald Norman identified two key principles to help ensure good *HCI*: *visibility* and *affordance* [43][44]. *Visibility* means that controls need to be visible with a good mapping to their effects; their design should also suggest their functionality. For example, the two software properties (see Section 3.3.3), honesty and observability, reflect the requirements of *Visibility*. *Affordance* refers to what sort of operations and manipulations can be done to a

particular object. *Perceived affordance* refers to what a person thinks can be done with the object. For example, the user's interaction requirements include that the user interface is friendly, reliable, comfortable and understandable; the system has sufficient functionality; and the interaction response time is short. *Affordance* plays an important role in object design. The operations and manipulations are based on task needs and interaction requirements. From the software engineer's point of view, the implementation of the interactive system needs to be concerned with hardware aspects and software aspects in the design environment and the target environment; certain properties can be used to measure the system's quality, such as modifiability, maintainability, and run time efficiency. The content of the following sections covers the issue of how to structure interactive systems to support user goals and concepts related to the construction and the quality of interactive systems. This section is focused on the issue of interactive software design.

Development process

The development process is regarded as a phase structure. Development models integrate a collection of methods to support different phases. Software development models allow the quality and production of software to be efficiently managed and controlled. The models of software development require systematic, sequential approaches to system development. The different models will have slightly different phases. This section will introduce two models which are applied to the **TEAMDEC** development process. Figure 3.4 shows the classic lifecycle or waterfall model[11]. The *lifecycle or waterfall* model is popular. It connects the phases into a pipeline: all prerequisite work for each phase is undertaken before the phase starts.

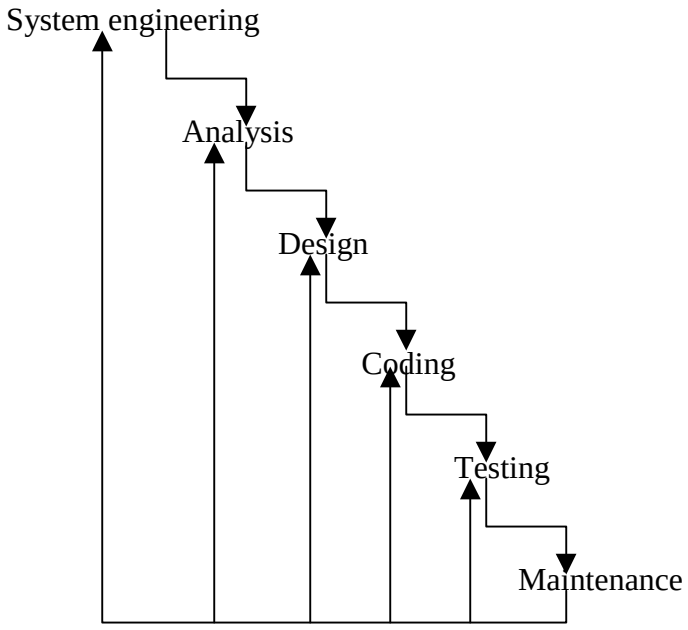


Figure 3.4 The waterfall model

An alternative structure is the *V-model*. The *V-model* relates each development phase not only to its immediate predecessor and successor, but also to the construction and testing phase on the same level. Both *waterfall* and *V-model* have the limitation that all project steps are carried out as single steps in a forward sequence. A solution to the limitation is applied in *V-model with backtracking* (see Figure 3.5)[8]. Each backtracking step results in some recovery that extends, corrects or refines existing inadequacies in the previous phase.

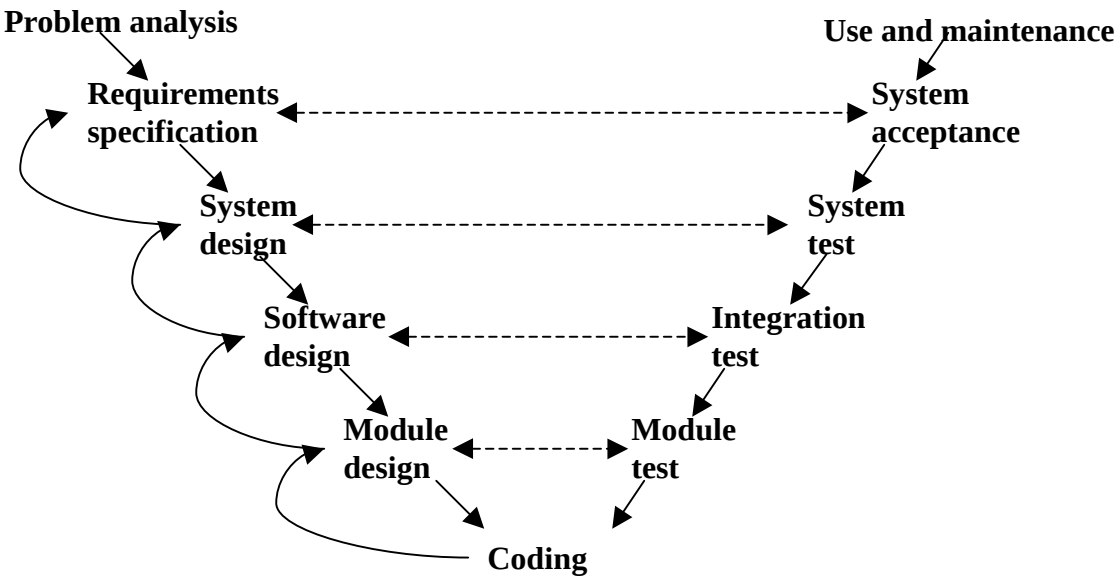


Figure 3.5 V model with backtracking

Generally, the identified phases in most development models include problem analysis, requirements specification, system design, software design, module design, coding, module and integration test, system test, system acceptance, and maintenance[8]. Problem analysis, the first necessary phase, identifies the problems or needs of its users in a given domain. This phase is related to the goal completeness of a software system. During the requirement specification phase, problem analysis always draws out a description of the functionality of the system, constraints in the system environment, and quality goals. System design transforms the specified requirements into solutions that outline the system design. During the software design phase, the software structure is determined, and the main components and their interfaces are specified. Module design refines the software structure and details the design. During the coding phase, implementation and debugging are performed. Module and integration test is a phase in which implemented modules are progressively tested and integrated into the system. System test is to test the final system against the system design. During system acceptance, the final system is installed and tested against the requirements.

3.3 User interface design

User interface is the main point of contact between the user and the computer system. The consideration of the user interface is critical in the interactive software design. The purpose of this section is to emphasize the aspects of the user interface with which the software designers are most concerned. This section addresses the issue of user interface design and related issues of quality properties of the user interface.

The user interface is a part of the system that users can touch, hear, see, and communicate with. Interface design and development is arguably the most labor-intensive and difficult part of the software development system. The reason for this is that interface design involves making a variety of different design decisions, many of which involve users and tasks for which the consequences of these decisions on both users and tasks are unknown. Because the effectiveness of the user interface is difficult to predict, the iterative design method may be used in the development process in response to the difficulty of prediction. The procedure of iteration design is the use of evaluation techniques to gather feedback of each development version from user representatives or specialists, then evaluation results are used to design the next version[8]. In addition, a number of design skills are required to develop a good interface. User interface development usually involves designing communication and discourse, graphical and textual material, and information and tasks.

3.3.1 Perspectives on user interface design

Design of a user interface has to be considered from different perspectives, each of which interacts to affect the quality of the overall design. In Section 3.3.3, the properties that measure the design quality will be identified. Three perspectives to be considered in interface design [11]: functional, aesthetic, and structural perspective.

The *functional perspective* is concerned with whether or not the design is serviceable for its intended purpose[11]. This perspective concentrates on the issues of usability and task (or goal) completeness. However, it is inevitable that designing a new system will have unpredictable and unknown effects on the tasks that users can perform. So the evaluation feedback and cyclical design methods will be employed against unpredictability.

The *aesthetic perspective* is concerned with whether or not the design is pleasing in its appearance and conforms to any accepted notions of design[11]. This perspective concentrates on the design of visual appearance and interface. It leads to the consideration of layout on the screen, icons, graphical and textual figures, the style of menus and buttons, animation, and interactive video. The aesthetic aspects of design contribute to the effectiveness of the overall interactive system. For example, colors and shapes are used to assist decision-makers with understanding the perceived information, to reduce the decision-making time, and to improve the decision outcome quality. Sometimes when necessary information must be noticed by the user, termed “insistence” (see Section 3.3.3), the aesthetic aspects of design can help achieve the desired effect.

The *structural perspective* is concerned with whether or not the design has been built in a manner that will make it reliable and efficient to use and can be easily maintained and extended[11]. The essence of the *structural perspective* is related to several internal properties of the interactive system, such as system modifiability, portability, maintainability, run time efficiency, and development efficiency (see Section 3.3.3). There are several approaches to provide a good structural perspective on the overall quality of design. One solution is the use of object-oriented programming which enables user interface design to be based on sets of primitive objects that have their own behaviors and connections to others.

3.3.2 Design approaches

There are a number of different design approaches for user interface design depending on different design criteria. Of course, different design criteria lead to different

ways to reach the design, which will be illustrated by several examples of user interface design. The task-oriented approach is a fundamental user interface design approach which involves understanding users and tasks, and the sequencing of user actions[18]. A graphical user interface (GUI) uses an object-action paradigm where the user recognizes an object first, and then decides the action.

Moreover, several design approaches are associated with certain special purposes, such as network-oriented design and adaptive design. Network-oriented interface is applied to support user-network interaction. The designers of a network-oriented interface extend the task analysis with the considerations related to a network environment[20]. For example, in CSCW (computer supported cooperative working) systems, the interface is different from the user interface in single-user systems. A CSCW system may combine various groupware technologies depending upon the objectives of group. From the user point of view, a CSCW system should support some of the following functions: presentation support software, computer-support meetings, group discussion support, public windows, computer conferencing, group writing, group memory management, and project management.

Adaptation refers to the adjustment to different circumstances or conditions. The importance of adaptability has been recognized. Its relationship to the quality of the interactive system will be addressed in Section 3.3.3. An adaptive interface is designed to allow the user to modify its appearance and/or behavior based on explicitly entered information or instructions for doing so; or, it is designed to make changes automatically in its appearance and/or behavior, based on knowledge that is stored, collected, updated, and analyzed dynamically[20]. Adaptive interface design strategies can be knowledge-based (e.g., ES), function-based, or condition-based which allows the automatic adjustment of components in accordance with the different conditions.

In an interactive software system, the design and development of the system may involve more than one user interface design approach. The determination of what kind of design approaches will be involved depends on tasks, users, quality goals, and the environment.

3.3.3 Quality properties of user interfaces

The quality of a user interface should be measured with properties of the interface and the computer system. Some of the properties can be defined and measured by taking the user's cognition and understanding into account. Some of the properties can be measured by using standard software engineering methods. There are several ways to categorize these properties depending on the relationship between them. In this paper, the properties are grouped into two types: *internal properties* and *external properties*.

The internal quality is presented as a list of software development-related properties. These properties are used to judge the quality of an interactive software system from the software engineer's perspective. These kinds of properties are *internal properties*. From the user's perspective, high quality means that the interface is pleasant, reliable, easily understood, and has sufficient functionality. These characteristics can be defined as *external properties*.

Internal properties

The basic objective of a system is to design and develop an acceptable product that matches users' requirements. The user's satisfaction is a primary concern of an interactive system. Some problems are not visible to the user; however, they must be considered by the designer, such as the difficulty of actually constructing the desired system and determining the actual effectiveness of the end result. This kind of consideration will eventually affect the software and hardware architecture chosen, which, in turn, influences how the desired user-detectable properties are to be achieved.

Internal properties are quality attributes from the software designer's perspective. There are some approaches for satisfying the internal quality goals throughout the life cycle of the system. Three facets of the designer's work: methods, software contents, and tools, are termed software techniques for software designers. Note that several different techniques may contribute to any one internal property. A particular technique for achieving one property may produce side-effects making it more difficult to achieve

some other properties. So the existence of such interactions makes it necessary to study the inter-relationship between properties.

The internal properties will be discussed from eight aspects: functional completeness, system modification, portability, evaluability, maintainability, run time efficiency, user interface integratability, and development efficiency[8]. These eight aspects reflect a complete life cycle view, from the conception of a system, beyond construction to modification and maintenance.

Functional completeness

Functional completeness is one of the basic internal properties. A particular system is constructed for satisfying a set of task requirements. Functional completeness is conformance to the specifications resulting from task analysis. The corresponding external property is task completeness for which designers need to describe the necessary interactions for all the required tasks.

Modifiability

Once a software product has been released, system modifiability becomes necessary when new or additional requirements arise. The ease of system modification is an important factor for improving life cycle effectiveness. Modifiability is influenced by several different factors, such as the available development environment, the target environment, the re-use of existing specifications and code, clean abstractions for system components, and the software architecture. The target environment in which a system will be used, offers hardware and software facilities together with constraints. The degree of ease of modification is influenced by the target environment. The well-parameterized module and well-designed library of code offer the opportunity for re-use of existing codes and specifications. For example, modification can be effected by amendment of the actual parameters used by a module or by changing one or more modules. Otherwise, a new code needs to be constructed with additional development and test work. This may increase cost and reduce development efficiency. The ability to produce separately generated modules is essentially related to ease of modification, because modification of one module has no effect on other components. The implementation of separation of

concerns of components can be supported by the overall software architecture. The separation and consequent encapsulation of functions benefit not only future modification, but eventually maintenance. In practice, the user interface is the most highly modified portion of an interactive system.

Portability

Portability is the ease and expense with which a system is moved to different environments. Portability covers situations where the target platforms are changed, including change of target hardware and change of target software. The target platform change may have profound effects on the user interface and the functionality of the system. Portability enables the system to behave consistently for the user across platforms.

Evaluability

Evaluability is how easy it is to evaluate the system against quality goals. One method to enhance evaluability is to integrate facilities into the system for obtaining metrics related to the behavior and performance in use, such as effectiveness, efficiency, error visibility, maintainability, and other software properties.

Maintainability

Maintainability means whether the system is easy to maintain and manage once the system is installed. Maintenance is the effort necessary to keep the system running in a given environment. It includes system administration, installation of new devices, tuning of the system, and error correction[8]. Maintainability can be measured from three aspects: whether it is easy for a system administrator to keep the system running, whether it is easy to detect errors which could cause failures, and whether it is easy to correct errors when failures do occur. A clear system structure as well as systematic and accurate documentation are helpful to system administration. Errors may be caused by the underlying operating system and hardware, by the user application, or by the interface system itself. The primary solution to deal with these errors is to prevent them and make

it easy to correct them. The principal means to support this solution is to re-use existing code and make use of standards and standard development toolkits.

Run-time efficiency

To the system user, an obvious measure of run time efficiency is the response time of the system to user input. There are several factors that influence run-time efficiency: the adopted software architecture, the incorporated algorithms and heuristics, and the underlying software and hardware[8]. Achieving some quality goals may result in the reduction of run-time efficiency. For example, improving deviation tolerance will make extensive demands on capturing more information. This may influence run-time efficiency.

User interface integratability

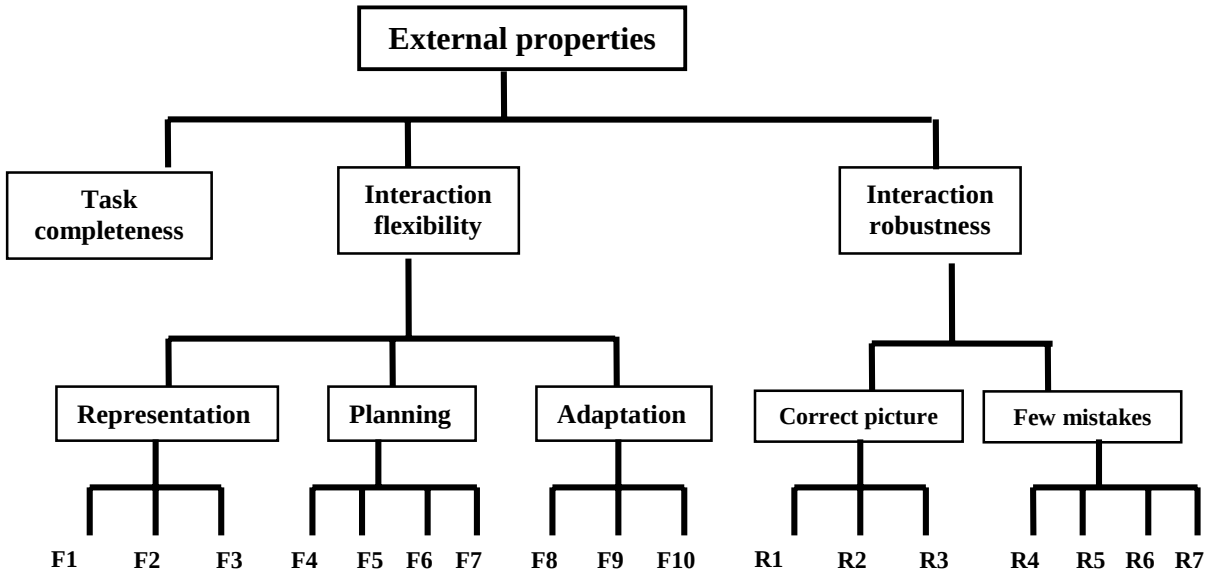
A integratable user interface system means that the interactive system is easily integrated with the existing user facilities or new user software applications. User interface integratability is obtained in three ways: (i) there is no significant difference in apparent behavior between the interface of the new system and the existing systems; (ii) the new system can run correctly with existing software; (iii) the new system must not disrupt the target software and hardware environment in such a way that the behavior of other existing systems is affected perceptibly.

Development efficiency

The entire development process includes construction and testing. To some extent, the methods and techniques selected for the design may influence the overall development efficiency. The factors that influence development efficiency should be considered in the development process. These factors include the complexity of the development methods, the available development environment and tools to the engineers, the software architecture being developed, the target platform which may place more or less severe restrictions on the available options for implementing facilities, the need to adhere to published standards or local software engineering practices, and the size and composition of the development team[8].

External properties

The external properties are user-centered properties of an interactive system that promote high quality from the perspective of users. The external usability properties can be loosely characterized as three main principles: task completeness, interaction flexibility, and interaction robustness. Interaction flexibility and robustness are further represented by more basic properties. Figure 3.6 illustrates the external properties with their relationships.



F1: Device multiplicity	F2: Representation multiplicity	F3: Input/Output re-use	
F4: Human role multiplicity	F5: Multithreading	F6: Non-preemptiveness	F7: Reachability
F8: Reconfigurability	F9: Adaptivity	F10: Migratability	
R1: Observability	R2: Insistence	R3: Honesty	
R4: Predictability	R5: Access control	R6: Pace tolerance	R7: Deviation tolerance

Figure 3.6 External properties

Goal and task completeness

The principal purpose of an interactive system is to allow users to reach their goals within a specific application domain. Task completeness allows users to perform their tasks and achieve their goals. Problem analysis is an essential process in interactive system design. It results in the relevant tasks. The descriptions of tasks are analyzed to isolate common goal states, typical and problematic initial task states, and a regular procedure for task execution[8]. A system exhibiting task completeness should support all of the tasks that have been identified. It allows user choice during task execution, facilitates the users' actions, and helps users to recover from mistakes.

Interaction flexibility

Interaction flexibility refers to the multiplicity of ways in which users and the system can exchange information during task execution. Flexibility properties can be divided into three groups: representation of information, planning of task execution, and adaptation of dialog forms.

(1) Representation of information

Device multiplicity

A system is able to provide multiple input and output devices for user-computer communication. Device multiplicity refers to the physical level of interaction flexibility. Media refers to the device or physical level.

Representation multiplicity

A system with multiple representations provides an alternative representation for both input and output. Multiple representation covers the variation of information content as well as the presentation of the information. For example, the fluctuation of a stock price over a time period can be represented by a table with the actual numerical values, or it can be represented by a curve. Representation multiplicity can achieve a desired effect by

means of multiple devices. For example, a sound effect may be issued when a warning message is promoted.

I/O re-use

The application of I/O re-use is the articulation of the current input expression referring to a previous input or output expression. Two issues may be involved in I/O re-use: how to convert objects with different types, and how to record interaction and store the interaction history for future use.

(2) Planning of task execution

Human role multiplicity

In multi-user systems, different users may serve different roles in their interaction with the system. They have different tasks and goals. They expect to use different methods and information to implement their tasks. Multi-user systems should support multiple human roles simultaneously. Moreover, the users should be able to change roles. Changes in roles can be either user-initiated or system-initiated, corresponding to *Reconfigurability* and *adaptivity* discussed below. For example, in a decision-making team, the manager can assign roles to the participants. The member sends a request to the manager, but it is up to the manager to make the appropriate role assignments. So the manager has the capability to reconfigure the system.

Multithreading

In multi-user systems, users want to execute individual tasks in parallel, so the system needs simultaneous multithreading. In a single-user environment, multithreading can satisfy the need of the user to execute multiple concurrent tasks.

Non-preemptiveness

This is a property to distinguish a system-driven model of interaction from a user-driven model of interaction. Non-preemptiveness refers to the degree of freedom the user

has for deciding the next action to be performed. The system-driven interaction has a preemption restriction. The user-driven system tends to be non-preemptive, which allows the user to have more freedom to choose the next action. Non-preemptiveness is one of the key factors contributing to the user's feeling of flexibility. From the user's perspective, the system-driven interaction hinders flexibility whereas a user-driven interaction favors it. Although designers want to minimize the system's ability to preempt users, some situations (e.g., safety reasons) may require preemptiveness to prevent errors and to recover from error states by error correction.

Reachability

Reachability refers to the capability of reaching any system state regardless of the current state. From the user's perspective, the manners of state navigation are backward reachability and forward reachability. So users can either get back to the previous system states or proceed to the next desired state. Backward reachability requires sufficient history information of system actions.

(3) Adaptation of dialog forms

Reconfigurability

Reconfigurability allows users to modify the form of input and output, thus customizing the manner of interaction.

Adaptivity

Adaptivity refers to the capability of the system to automatically customize the user interface of the system. The difference between reconfigurability and adaptivity is the role of users. In reconfigurable interface systems, the user plays an explicit role in customization. An adaptable system has the ability to detect the user's behavior, current settings, and environment.

Migratability

Migratability refers to the system capability for supporting user-initiated or system-initiated transfers of task responsibility. In other words, task migratability concerns the transfer of control for events or execution of tasks between the system and the user[8].

Interaction robustness

A robust interactive system should be able to detect and correct mistakes, have fault and deviation tolerance, and eventually minimize the risk of task failure. The following seven properties contribute to interaction robustness. Observability, insistence, and honesty ensure a correct and complete picture of the system. Predictability, access control, pace tolerance, and deviation tolerance contribute to reduce the risk and cost of mistakes.

Observability

An observable system should allow users to inspect all the information relevant to the tasks. Usually, critical information should be available immediately. The perceivable information should be relevant and sufficient to current tasks. Although ideally users can access all of the information, the capability of observing needs to be taken into account to avoid information overload. In a CSCW system, the user also is aware of the actions of other users and of the external system[35]. This observability is related to data transmission over the network.

Insistence

Insistence ensures that the available information can be actually perceived or noticed by users. Insistence can be achieved by a variety of means, such as increasing visual salience[16], interrupting the user with preemptive dialogs, adopting aural signals[16], or leaving a persistent event indicator. The choice of mechanisms depends on two aspects: where the user's attention is likely to be, and the required timeliness and salience of different system elements[8].

Honesty

The honesty of a system refers to conformation of user's interpretation and the designer's intended interpretation of the interface[43]. It helps users obtain a correct and complete picture of the system. Honesty is a fundamental factor underlying interaction robustness. If the user misinterprets the information that an interactive system provides, then it will lead to trouble and inconvenience.

Predictability

Predictability provides users with the ability to determine future actions of the system based on the past interaction and the current observable state. Consistency is one heuristic that is often applied to increase the predictability of an interface. It allows users to generalize from specific situations to similar situations.

Access control

Access control sets constraints on the scope of information and functions that can be available to and processed by a user. One concern of an access control mechanism is system robustness. The main reasons for access control are human role multiplicity and damage prevention.

Pace tolerance

A pace-tolerant system is concerned with the match between the user's expectancy and the system demands. There are some external factors that will affect system demand, such as network latency and hardware failure. The designer needs to consider how to meet the user's expectations when the system is too fast or too slow.

Deviation tolerance

A deviation-tolerant system may have functions such as error detection, error occurrence prevention, and error correction[8]. Although the system is carefully designed for error detection and prevention, users will inevitably commit errors. Recovery is, therefore, the most important aspect of deviation tolerance. Recovery is relevant to reachability because it is necessary to reach any system state for a recovery strategy.

3.4 Summary

This chapter introduced HCI concepts and relevant theories. HCI involves the interaction of person, task, and computer; thus, this chapter addressed the issue of the relationship between users, tasks, hardware, and software from theoretical, methodological, and practical perspectives. Because of the importance of the user interface in human computer interaction, the issues of user interface design were discussed from several aspects, such as development perspectives, design approaches, and quality properties. All of the issues in interactive system development are based on HCI concepts, theories, and research findings.

Chapter 4 Scripts

4.1 Introduction

This chapter illustrates a practical application of scripts in **TEAMDEC**. The functionality of the scripts is one of the significant characteristics of **TEAMDEC**. Based on previous knowledge and a suggestion-generation mechanism, the **TEAMDEC** script system supplies decision-makers with improved decision-making quality and efficiency. This chapter discusses the functions of scripts in a group decision support system and the interactive software quality properties of the script system in **TEAMDEC**. The fundamental concepts of script-based knowledge organization underlie the script capability development. The conceptual issues of scripts are explained in Section 4.2. Section 4.3 describes how the script system in **TEAMDEC** achieves its functional goals. Section 4.3 addresses the application of the principles of GDSSs and interactive software design in the **TEAMDEC** script system.

4.2 Terminology and concepts

The following definitions and concepts are related to the construction and functionality of a script system. The relationship between the script and human computer interaction determines the choice of principles, techniques, and goals in script development.

4.2.1 Terminology

- ***Schema***

A *schema* is “a mental representation which consists of general knowledge about events, objects or actions.” [13]

- ***Script***

A s*cript* refers to “a specific version of a schema consisting of general knowledge about likely outcomes.”[13]

4.2.2 Concepts

One characteristic of knowledge is that it is highly organized. Knowledge is assumed to be organized in the form of a network. A *schema* is a network of general knowledge based on previous experience. A *script* is a special subcase of a schema. It describes a characteristic scenario of behavior in a particular setting. The underlying assumption is that people develop a *script* for a frequently occurring set of events in a given setting. Schank and Abelson (1977) developed a framework for *scripts* and gave a well-cited example of a schema which is a restaurant *script[47]*. *Scripts* have two categories of variables: props and roles. Props are related to objects and roles are related to people. In addition to props and roles, scripts have three other elements: *entry conditions*, *scenes*, and *results*. *Entry conditions* are the premises under which a script is used. A s*cene* refers to a particular group of activities within a script. The activities of a *scene* normally occur together and constitute a recognizable subset of the main activity. The restaurant script [47] is described in Table 4.1.

Table 4.1 Components and features of a restaurant script (adapted from Schank and Abelson [47])

Script relevant elements	Entry conditions	Hungry, had money, restaurant open
	Roles	Diner, waiter, cashier
	Props	Tables, money, chairs, menu, cutlery, food
Script components	Entry scene	Enter restaurant
		Waiter seats diner at table
		Waiter places menu on table
		Read menu
	Ordering scene	Select food from menu
		Signal to waiter
		Waiters approaches table
		Order food
		Waiter leaves
	Eating scene	Waiter brings food to table
		Waiter leaves
		Eat food with cutlery
		Finish eating food
	Leaving scene	Signal to waiter
		Waiter approaches to table
		Ask for bill
		Waiter writes bill and gives to customer
		Customer checks bill
		Customer approaches cshier
		Customer gives cashier bill and money
		Cashier checks money
		Leave restaurant

Similarly, when people interact with a computer, a schema can be viewed as guiding people's behavior. For example, for file operation, a specific script may be developed for creating a document, one for editing a document, one for saving a document, and an overall schema for using the computer, such as turning on or off the computer, and inserting or removing a disk. Scripts allow people to take advantage of the regularities of events and situations to reduce the effort in repeated or frequently occurring events. When people are dealing with a new but familiar situation, a script can help them know how to behave appropriately and know what to look for. The features of a *script* imply that an interactive system could provide a script or activate a schema similar to a set of "new" activities for offering users behavioral suggestions. In addition to the advantage of saving effort, *scripts* improve efficiency and, to some degree, reduce error.

4.3 TEAMDEC script system

As a decision support system, **TEAMDEC** is concerned with how to provide decision guidance and how to help the decision-maker utilize the suggestion efficiently. Based on the user's cognition and knowledge, the script (a knowledge representation), facilitates decision-making and human computer interaction. As stated in Section 2.2, the human decision-making process has two major characteristics: (i) decision-making is not a point-event; (ii) decision making is not monolithic. In other words, decision making is based on previous experience and knowledge; a decision can be reached through multiple paths. The functionality of **TEAMDEC** satisfies the needs of the human decision process, from the *intelligence* phase to the *choice* phase, as discussed in Section 4.3.1. The design of the script system in **TEAMDEC** is based on a consideration of the features of the human decision-making process, task analysis, and interactive software quality

goals. Section 4.3.1 will give a description of how the **TEAMDEC** user reaches his decision goal by means of scripts. Section 4.3.2 will analyze features of the **TEAMDEC** script system from a decision support aspect and a software quality aspect.

4.3.1 Script utilization process

The user is able to obtain action suggestions in various ways. People can ask questions with certain key words to acquire solutions or decision guidance. This guidance is generated by invoking the inference engine and knowledge bases. Alternatively, the suggestion provision is an automatic procedure. The system gives the decision-maker a recommendation on how to proceed with using the system based on the decision-maker's previous actions, knowledge from databases, and the inference engine. The **TEAMDEC** script system focuses on the latter suggestion provision approach.

There are three factors affecting suggestion generation: the user's goal for the next set of activities, previous action records, and rules that are applied to problem analysis and data processing. Suggestion generation is a knowledge-based procedure conformant to decision-making phases. The *Intelligence* phase is a problem-finding process during which **TEAMDEC** detects current situations for the environment and the user, and collects information on the user's current task. During the *design* phase, **TEAMDEC** generates alternative solutions by searching previous action series with similar task goals and situations. During the *choice* phase, multi-criteria problems are solved with various decision rules. The abilities of prediction and evaluation play important roles in the suggestion generation procedure. Predictive judgment leads to expectances on future conditions, such as the next possible action goal. Evaluative judgment assists in determining the optimal solution. After **TEAMDEC** has produced the action recommendation, the solutions are represented by scripts.

A script is a knowledge organization form which describes characteristics of a scenario of behavior in a particular setting. It contains various knowledge components of a particular sequence of events. For example, a user intends to carry out a discussion with a number of on-line users about the issues of flight safety. The user must first set the

option for suggestion acceptance to “ON”, which indicates that the user wishes to get action guidance. After that step, the user’s activities are traced. Simultaneously, the system invokes its inference engine to derive timely action suggestions based on the user’s real-time activities, knowledge from the action database and other databases, and rules which are related to predictive judgment, evaluative judgment, and decision making. The system consequently produces advice on the outline of possible actions. Generally, the suggestion information is organized in the structure of the script as shown in Table 4.2. The script consists of two scenes: develop a discussion group and begin a group discussion. However, the actual components of the script will be slightly different depending on the data stored in database.

Table 4.2 Components of a group discussion script

Develop a discussion group	Open a window for selecting users
	Display the user information
	Select the user and add to the discussion group
	Remove a user from the discussion group
	Confirm and quit
Begin a group discussion	Open a window for group discussion
	Give a topic
	Send an invitation with the topic to group members
	Get agreement replies from members
	Setup communication channels and begin discussion
	Edit the opinion expression and pose it out
	Send an off-line message
	End the discussion

In practice, a suggestion may not exactly match the user’s expectation or the requirements and constraints of the next series of actions. Reconfiguration and adaptation affect the overall process, including structuring and executing the script. Structuring the decision-making process has been discussed in Section 2.2.2. Similarly, constructing a new script determines what activities will occur and the order in which they will occur. In **TEAMDEC**, the script is displayed schematically with explanation facilities (see Figure

4.1). The information representation form enables the user to easily understand the structure of the script. The explanation facilities help users acquire task-relevant information and develop a new script. A message window can be activated by clicking the mouse on each script element (see Figure 4.2). The message window shows a variety of information about the item, such as an action description, whether the action is selected, whether it is executable, whether it is editable, and whether it is removable. The perceivable information enables users to understand what the advice is and how and why results are obtained. The action description offers a brief overview to help the user know what the item does and what will happen after the execution of this script element. The status of *executable* indicates whether this step of the action can be invoked by executing the script. If it is not executable, the user can still follow the advice to fulfill his task outside the script system. A status of *selected*, *editable*, or *removable* guides customization. The status of *selected* indicates whether this script element has been chosen for use in the script. The status of *editable* indicates whether the variables of this script component can be modified. The status of *removable* indicates whether the element can be deleted from its action group.

The existence of some script elements is precustomized in a given action group. For example, because the interface is necessary for selecting a user, the step of opening a window can not be removed from this action group. These presentation forms of the **TEAMDEC** script system aim to provide sufficient relevant task information to enable the user to easily understand the structure of the script and assist him to conveniently develop a new script to carry out new tasks.

TEAMDEC allows the user to reorganize the script. Thus the user can reconfigure the script and modify the variables of the script, including props and roles. The script containing advice just provides a structure for the temporal order of the elements of the activities. **TEAMDEC** offers two ways for the user to reorder the elements of the script: typing a sequence number or using a mouse to change the element positions so as to change their sequence order. The unwanted components can be deleted from the script if the components are removable. In the **TEAMDEC** script system, props refer to the parameters related to objects, such as the topic of a group meeting, the IP address, and the socket port number. Roles refer to the parameters related to people, such

as the group members. **TEAMDEC** provides the interface for modifying the variables of the element if the element is selected and editable. Thus a new script is generated for a given action goal.

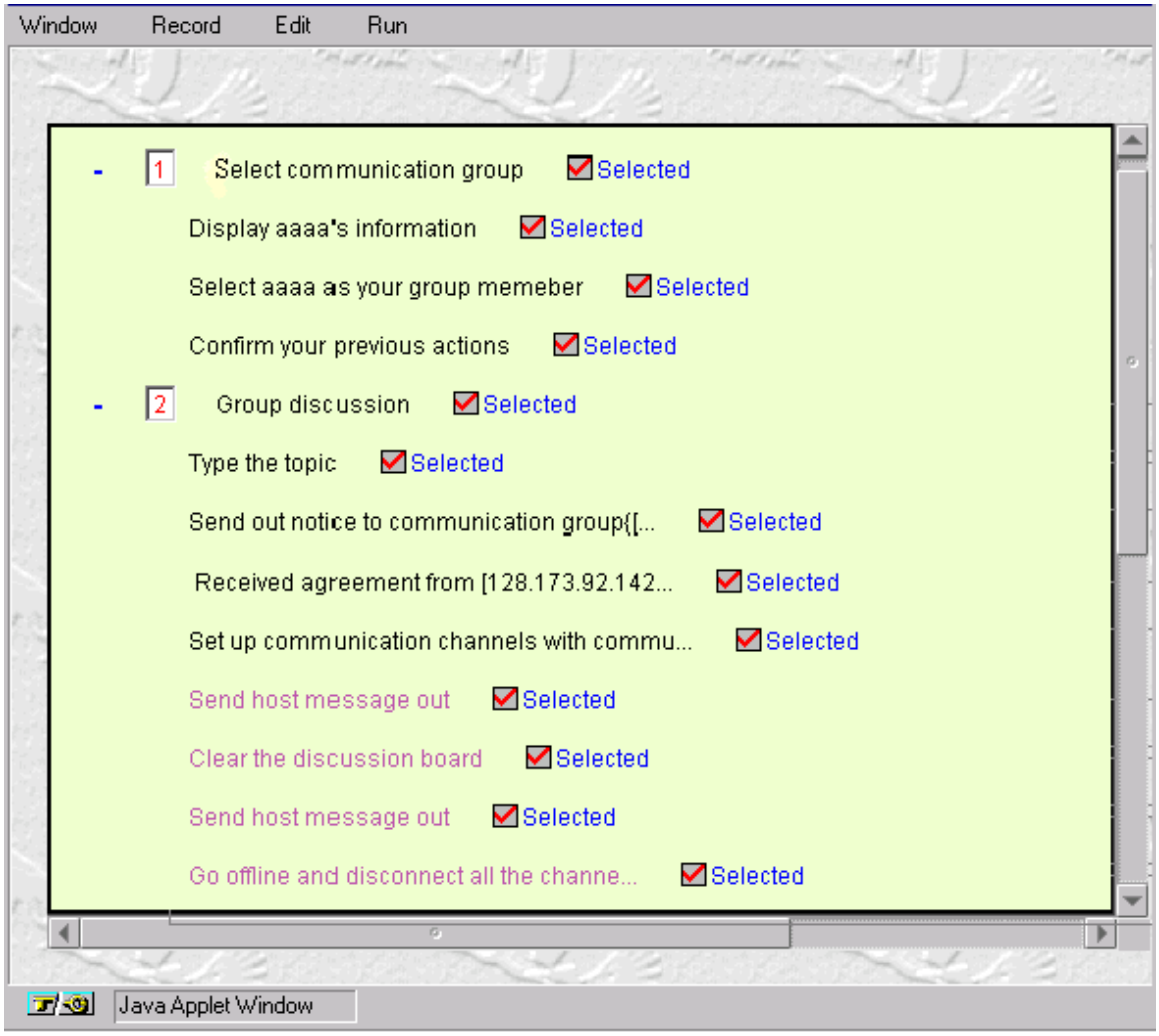


Figure 4.1 The user interface of TEAMDEC script system

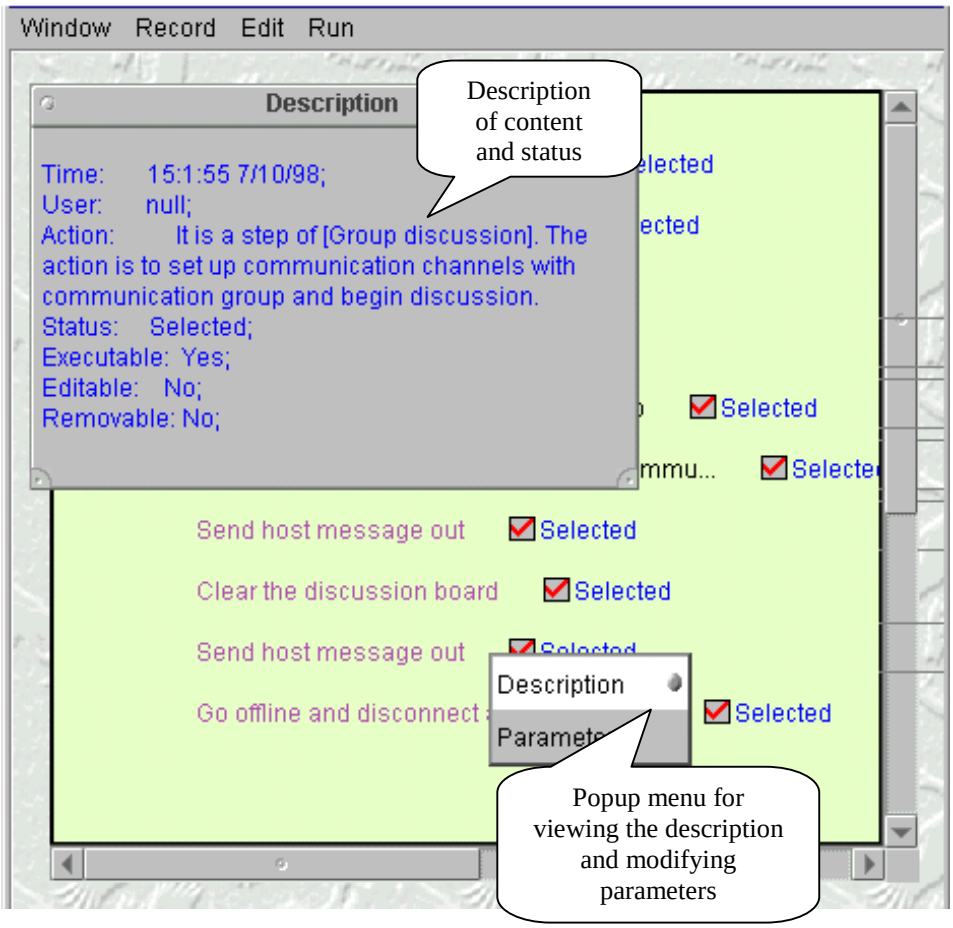


Figure 4.2 The interface of the explanation facility

4.3.2 Quality analysis

The primary advantages of the **TEAMDEC** script system are: (i) guiding the user's behavior; (ii) improving work efficiency; (iii) reducing errors. The significance of a script is its ability to minimize the effort of constructing a new action series. However, the conventional script-based knowledge representation is inflexible. It is very useful in dealing with similar situations, but it is not focused on reconfiguration and adaptation. **TEAMDEC** provides a set of solutions to go beyond the limitations of a conventional

script-based knowledge representation form; for example, the adoption of the concept of mental models, the capability of interaction flexibility (i.e., reconfigurability and adaptivity), and the capability of interaction robustness (i.e., deviation tolerance, honesty, and observability).

To develop a flexible script application, **TEAMDEC** adopts a schemata-related approach to a script interface design. Mental models are used to account for the dynamic aspects of the cognitive activity. A mental model utilizes knowledge of past events to examine various alternatives, conclude which is the best, and predict future situations. It provides ways to deal with the present and the future[33]. It can be dynamically constructed by activating stored schemata. It is either an analogical representation or a combination of an analogical and a procedural representation[38][39]. Analogical representations are assumed to be responsible for the representation of mental images[11]. The most common image form is the visual image. The procedural presentation form is assumed to include the representations of action procedures; these forms are tailored for performance of specific actions[11]. The scripts in **TEAMDEC** are knowledge-based with a representation form that combines analogical and procedural representation. The visualization form of scripts in **TEAMDEC** is a hierarchical schema which clearly illustrates the structure of a script, such as scenes and specific action elements. Each action element corresponds to a series of action procedures. Users can tailor those procedures by activating a parameter customization function.

Based on the script and script-related concepts and software development techniques, **TEAMDEC** can achieve quality goals in these major properties: reconfigurability, adaptivity, observability, honesty, and deviation tolerance. These properties contribute to the interaction flexibility and interaction robustness which ensure a good human computer interaction.

The capabilities of reconfiguration and adaptation conform to the customization requirement of a decision support system. They play roles in structuring and executing a script. A script containing action suggestions consists of a set of precustomized action information. The precustomized data are derived from the previous experience under certain advice generation rules in a given setting. The precustomized data may not exactly match the user's expectation or newest requirements. To deal with unmatched

situations, **TEAMDEC** supports customization of the structure of a script and variables of the script elements. The user is able to adjust the order of script components and remove the elements that are not related to his purpose. In addition, an automatic customization is applied to make the new script adaptive to the current environment and settings. This adaptation relies on the system's detection ability for current environment and action-related variables. For example, the activities of "develop discussion group" result in selecting certain users into the user's discussion group. But some of these members may not be online. The system can automatically adjust the group's composition to match current users' online status during script execution. Reconfigurability and adaptivity contribute to interaction flexibility.

When assisting **TEAMDEC** decision-makers in the process of judgment and choice, the information presentations and the associated explanation facility are critical. **TEAMDEC** is observable because it allows users to inspect all information relevant to their tasks. The **TEAMDEC** script system provides sufficient task-related information to decision-makers. The task-related information includes the structure of a script corresponding to the particular tasks, explanation of script elements, and the parameters involved in task execution and control. The types of information that are presented depend on the situation. The information of a script's structure is utilized to assist the user's judgment, choice, and customization activities; it needs to be continuously presented. The script element explanation and parameter-related information can be presented at the required times.

Correct interpretation of the perceivable information influences the quality of decision-making activities. One of the information needs of a group decision support system is that the information should be helpful in improving a precise understanding of the task situation (see Section 2.4.3). So it is important to ensure that users interpret the symbols in the interface in the way that they are intended. The **TEAMDEC** script system develops several ways to ensure the honesty of the system. The symbols on the button represent specific notions. For example, "+" suggests mouse clicking to expand its action group. "-" means that the following set of actions belong to the script scene. "x" means that the item is not selected (see Figure 4.1). In addition, different colors represent different states of an item. The color of an executable item is different from the color of

an unexecutable item. Furthermore, honesty is supported by observability. The information of each script element can be obtained by rendering corresponding information.

The **TEAMDEC** script system is concerned with interaction robustness. The previous two properties: observability and honesty contribute to interaction robustness. These two properties enable users to be aware of the task situation and help improve the precision of understanding the decision guidance. Deviation tolerance is a solution to ensure that the system takes appropriate actions to guide the user away from errors. The **TEAMDEC** script system is a deviation tolerant system because it can detect errors, correct errors, and prevent getting into error states. There are various ways to prevent getting into error states. For example, the user can reconfigure the script by using the mouse to drag the components freely. But the elements of a scene belong to the given action group so they are prevented from leaving the range of the group. The logical relationship may determine the sequence of script components. To implement a task such as “sending notice”, the window containing the text editor should be opened first to provide a user interface for typing in a notice. The last step of this task should be quitting and closing the window. Rearranging these two elements will lead to execution error. Fortunately, the system can detect the error and prevent it by disabling this kind of unexpected reordering.

The user can modify the variables of a script. For example, the user can modify the URL address, a parameter for opening a web page. The system will give out a warning message if the URL address is not available or not correct because of a bad format. A warning is usually an effective way to prevent further mistakes. Note that the robustness property of deviation tolerance must be balanced against the flexibility property of non-preemptiveness (see Section 3.4.1).

4.4 Summary

The script capability of **TEAMDEC** is designed to improve interaction efficiency and minimize the risk of errors. The **TEAMDEC** script system overcomes the limitation

of inflexibility by adopting the concept of mental models. It can not only generate decision suggestions to guide the decision-maker's behavior, but also supply customization for constructing a new script. By means of a script, the user can carry out tasks efficiently. The primary objectives of the script system are to support decision-making and relevant activities, ensure a good human computer interaction, and satisfy all the functional requirements. There are several factors taken into account, such as structuring the decision-making process, the information needs of group decision-making activities, interaction flexibility, and interaction robustness.

Chapter 5 TEAMDEC

5.1 Introduction

A significant characteristic of **TEAMDEC**, the script system, was discussed in Chapter 4. This chapter gives an overview of **TEAMDEC** and analyzes the overall interactive quality from several perspectives, such as hardware, software architecture, software development, human factors, and decision support system development.

The primary objective of **TEAMDEC** is to provide users with computer and communication support for team-based decision-making. **TEAMDEC** is designed to be a collaborative decision-making support system with safety, utility, efficiency, effectiveness, and usability. The basic design idea of **TEAMDEC** is guided by concepts of HCI and decision support systems. The development of **TEAMDEC** is based on the principles of GDSSs, interactive software, and related development techniques. By taking advantage of abundant information on the Internet, networking, and database technologies, **TEAMDEC** provides decision-making participants:

- comprehensive information access, including internal data and external data,
- communication facility,
- conferencing facility,
- activity guide, such as the script system and suggestion producer,
- friendly interface with multiple-user access, and
- multiple I/O devices.

Because the multimedia technology is used in **TEAMDEC**, the decision-maker can also plug in video and audio to an information presentation screen.

To make full use of Internet information sources, **TEAMDEC** is designed to run in Netscape Communicator. **TEAMDEC** is primarily written in Java and JavaScript[27]. LiveConnect[29] is used for interaction between Java, JavaScript, and plug-ins. The

reason for choosing Java as the implementation language is because it provides platform independence, supports interactive content on Web pages, and provides database connectivity. It is well suited for web applications, GUI-based applications, object-oriented applications, multithreaded applications, distributed networking applications, secure applications, multiplatform applications, and mission-critical applications[27].

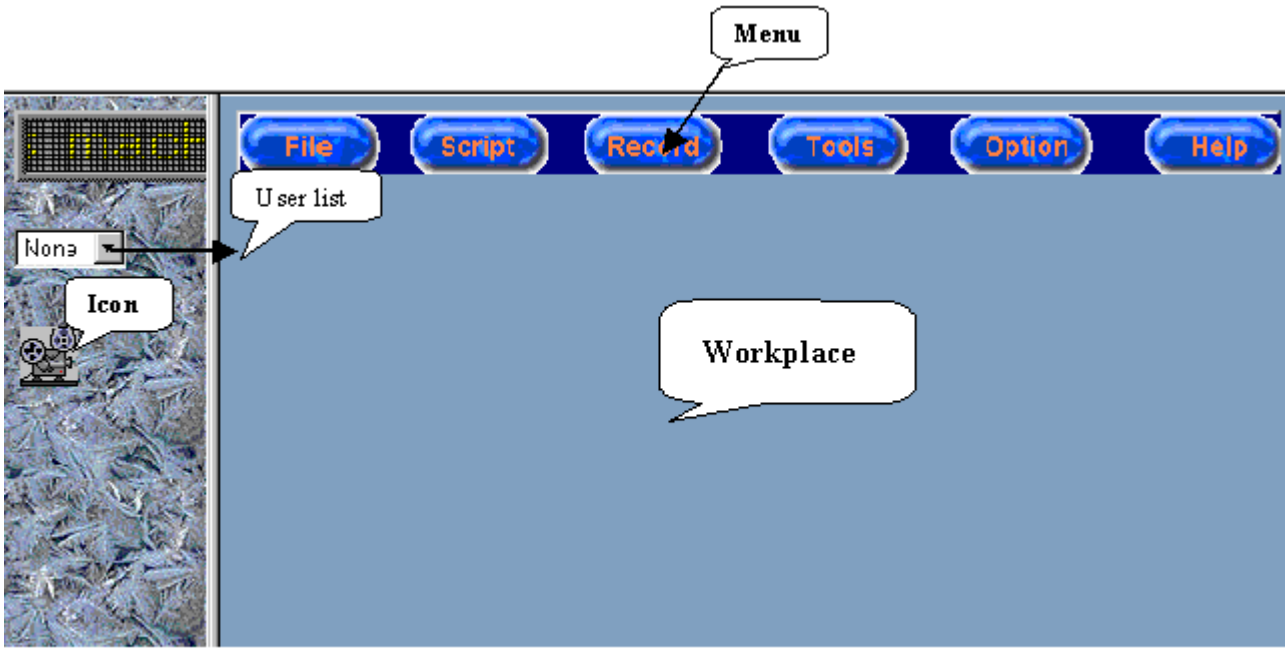


Figure 5.1 TEAMDEC

5.2 Environment

This section gives an overview of **TEAMDEC** components, including hardware and software. The software architecture of **TEAMDEC** is an underlying factor that affects the performance and quality of an interactive system. A three-tiered distributed

architecture is adopted for desirable software performance. This architecture is described in Section 5.2.3.

5.2.1 Hardware

The hardware components of **TEAMDEC** include the following:

- PCs or workstations,
- I/O devices (video cameras, voice I/O, etc),
- a network system that links different sites and participants to each other (LAN/WAN), and
- an individual monitor (for individual) or a public viewing screen (for group).

5.2.2 Software

The software components of **TEAMDEC** include:

- database and database management capability,
- user/system interface (see Figure 5.1), and
- applications which provide facilities for decision-making for individuals and the group, such as information searching, group discussion board, whiteboard, video conferencing, messaging, the script system, action capturing, and developing a communication group.

The database employed in **TEAMDEC** is mSQL[28]. It is a lightweight database engine designed to provide fast access to stored data with low memory requirements. The mSQL language offers a subset of the features provided by ANSI SQL[28]. JDBC is an API for connecting databases with Java applications and applets[27]. JDBC can be implemented on top of most ANSI SQL92 Entry Level compliant databases. The JDBC driver[27] is able to build a connection to mSQL; the JDBC DriverManager is part of the java.sql package.

5.2.3 Three-tiered distributed application architecture

The design of a user interface needs to be considered from three perspectives: functional, aesthetic, and structural (see Section 3.3.1). The structural aspects of design are related to the internal properties of an interactive system. Software architecture is a key factor which affects an interactive system's attributes, such as run time efficiency, modifiability, and maintainability. A three-tiered distributed architecture is adopted in **TEAMDEC** for desirable performance. The central idea of a three-tiered distributed architecture is the separation of concerns of modules to provide clear abstractions for system components, one of the major factors that influences the internal properties of an interactive software. The three-tiered distributed architecture first entered the Information Technology (IT) world in the early 1990's[26]. It proposes the idea of separating the components of an application into three functional layers according to the different goal of each layer. The application using a three-tiered architecture can be divided into three layers: *data management*, *logic*, and *user interface* (see Figure 5.2), each with its own function and design constraints.

The *Data management layer* is the bottom layer, which is responsible for handling data storage. Data should be stored in a raw form, and no process operation is performed on these data. *Logic layer* is the second layer, which implements rules, applies these rules to the raw data by adding value to them, then makes the processed data available to the third layer. *User interface layer* is the third layer, which is responsible for formatting and displaying the data from the Logic layer in a way that is appropriate for the requirement of interaction on the client-side.

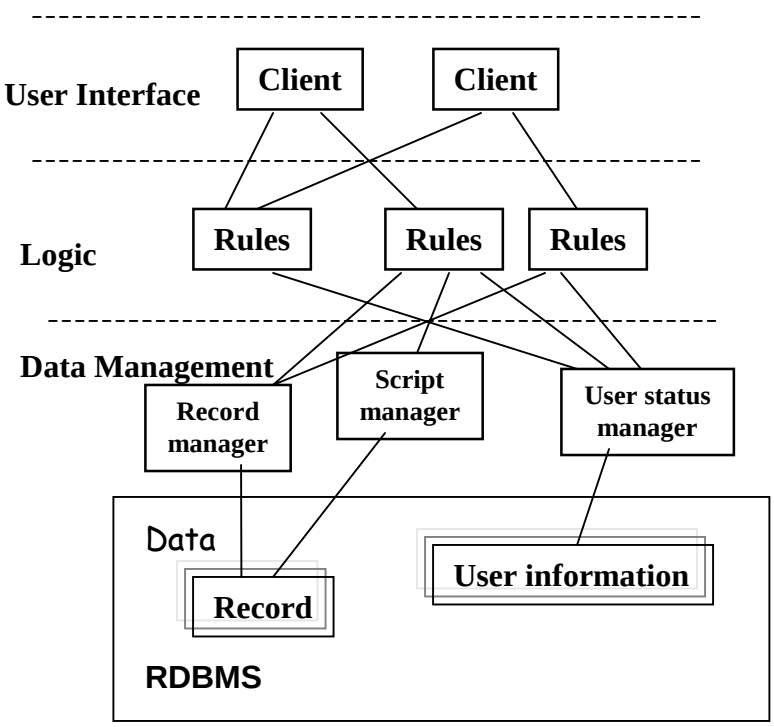


Figure 5.2 Three-tired architecture in TEAMDEC

The use of a three-tiered architecture in **TEAMDEC** enhances the internal properties of the system, such as modifiability and maintainability. The factors which affect the modifiability of an interactive system have been stated in Section 3.3.3, such as clear abstractions for system components, re-use of existing specifications and codes, and the (re)composition of system components. The components of the three-tiered architecture software system are treated as three stand-alone parts: data provider, service provider, and data consumer. So the functional modules are produced separately. Modification on one module has no effect on the other modules. This is an essential adjunct to ease of modification. In addition, a three-tiered architecture creates a software infrastructure of reusable elements, which allows software engineers to assemble pre-existing code into the current system. The features of the **TEAMDEC** software architecture ensure ease of modification, and, eventually, ease of maintenance. Because the three-tiered architecture supports modularity and functional encapsulation, it leads to relatively easier maintenance and system update.

5.3 User's tasks and system functionality

TEAMDEC is an interactive software system that aids the decision-making participants in various aspects of decision-making, such as information acquisition, decision guidance/suggestion, and group meeting by means of communication facilities and information presentation facilities in a collaborative environment. It aids both the individual decision-maker and the decision-making team.

5.3.1 User's tasks

Decision-making is a knowledge-based behavior. For group meeting and group decision-making participants, the basic task is to access information with the assistance of a computer system. Information needs of **TEAMDEC** users can be categorized into four types: information from the Internet (a global information source), information from previous actions, information that contains an advice or decision suggestion, and other task-related information.

Decision-making is not a point-event. It is related to previous experience and knowledge. Previously generated ideas and individual action records can both be used as reference for further decision actions. Searching information, therefore, is necessary to assist decision-making. Naturally, the user needs to understand the search results by means of an interpretation facility. Replaying "old" events with relevant information provided helps the user understand the whole event procedure in an effortless and straightforward way.

A communication group is a basic factor in all the collaborative activities. Determining a communication group is a necessary task in team-based activities. To build his meeting/decision-making group, the participant will acquire the latest information of other on-line users. **TEAMDEC** traces the status of users and updates the

information of users who are on-line or off-line. The user information is broadcast to all of the on-line users if any change of user status occurs. Once the content of the user information database is updated, the on-line user information is sent to all users in real time. The existing communication group list will be modified following the user database update.

Action suggestion is used to guide the user's decision-making activities. Derived from the user's actions and knowledge in the database, action suggestion generation depends on the system's predictive and evaluative judgment. Confronted with the decision suggestion information, the user needs an explanation facility to help him understand the suggestion and what to do next to reach the goal. Furthermore, the user is able to customize a script, which represents the suggestion information, to construct his next series of actions in a given environment.

Group meeting is another fundamental task of a decision-making group. As long as a decision-making group is set, the participants can hold a meeting. A group meeting involves various I/O devices, data transmission over a network, individual or common screens to display the contents of the current meeting and meeting-related process, and multiple information presentation forms to enhance the effect on information utilization. There are several ways to have a group meeting, such as video conferencing, textual group discussion, and a whiteboard which combines textual and graphical information presentation formats. According to certain meeting requirements, specific I/O devices and multimedia boards will be used. For example, participants in video conferencing use video cameras as input devices to capture voice and image. A common screen is a necessary component in a group meeting. It is a dissemination tool to allow all of the participants to see meeting contents – a kind of real-time information sharing. By means of the common screen, participants can perform on-line modeling (see Section 2.4.3). Besides I/O devices, network performance directly influences the meeting quality. For example, a low data transmission rate results in a long communication delay which has a negative effect on the meeting.

Note that all of the above tasks rely on the ability to retrieve data selectively. In addition to loading existing data from databases, users also need to save data to databases

and process data by means of software applications (e.g., record actions, process records, and edit records).

5.3.2 System functionality

System functionality development is based on user's task analysis. It is a reliable way to achieve *functional completeness*. In Section 5.3.1, the user's tasks are briefly described. This section compares the main functions of **TEAMDEC** against the task requirements. The main functions are system access control, developing communication group, action record management, information searching, the script system, and meeting facilities.

The system access control mechanism restricts the range of system functions that a user can use or the scope of information that a user can view, access or alter. Two concerns are taken into account: user role multiplicity and security. **TEAMDEC** is a multi-user interactive system, so different users may play different roles. Each role may have different goals which result in different functional requirements and information requirements. The functional and information requirements determine the access range for the individual user. Security is a pivotal issue of system robustness. Access control is an effective way to prevent damage, or at least minimize the risk of damage.

Building a communication group is the first step for any team-based activities. **TEAMDEC** offers an interface (see Figure 5.3) for selecting communication group members. One user is able to check the other users' information and their on-line status by activating a message window in which the relevant information is reported. After the user finishes setting up his communication team, the names of on-line group members will be displayed in the team member list (see "User List" in Figure 5.1). If one member is off-line, that member will be deleted from the communication group and his name will be removed from the member list. The correct report of user-relevant information and on-line group member conforms to the concern of the correct picture properties in terms of *observability* and *honesty*.

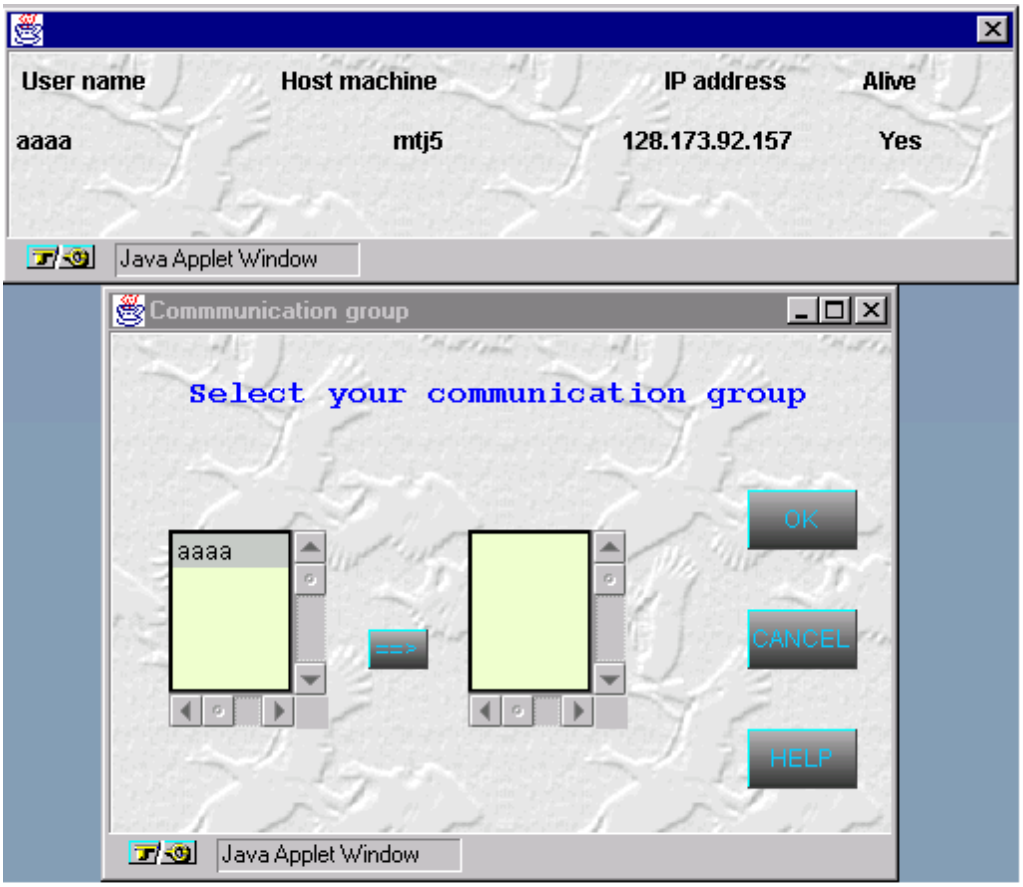


Figure 5.3 The interface for selecting communication group members

Due to the consideration of information re-use, it is necessary to capture the user's computer inputs and save them in a reliable way and in rational formats which make them convenient for further use (see section 2.4.3). The data that represent the actions of individual members of the group are required by GDSSs. **TEAMDEC** stores the captured actions to an action database. Setting the record option to "ON" can activate the action-recording function to capture user actions automatically. Once the action recorder is on, whenever the user begins a new series of activities by clicking a menu item, there will be a pre-emptive message box to tell the user that the recorder is on. That is a way to help

the user be aware of the current state of the system. In addition to the individual's actions, the system also saves the action suggestions which are accepted by users. The action record manager not only can activate action capturing, but also assist the user to retrieve existing records according to particular requirements. Once the necessary data have been delivered to the client-side, several operations can be carried out. For example, the user can replay the "old" series of actions to study the "old" procedures in detail; the user can redo a series of previous actions to repeat the operations when the user faces the same decision; and the user can delete unwanted records.

Computer-assisted information processing is a critical function of a decision support system. It enhances human ability by taking advantage of the powerful data processing ability of a computer. Computer-aided information searching helps users get information accurately and quickly. It helps users to obtain sharable data, which is very important for GDSSs. Information searching can be divided into two types according to their different data sources: internal data searching and external information searching. The databases internal to **TEAMDEC** include the user information database and the action record database. **TEAMDEC** provides an interface for searching information from internal databases. Figure 5.4 shows the interface of action records searching. An important feature of the Internet is that it provides a wide space for global information and is a very valuable external information source. Therefore, as a software application running on Netscape Communicator, **TEAMDEC** can take advantage of this huge information resource.

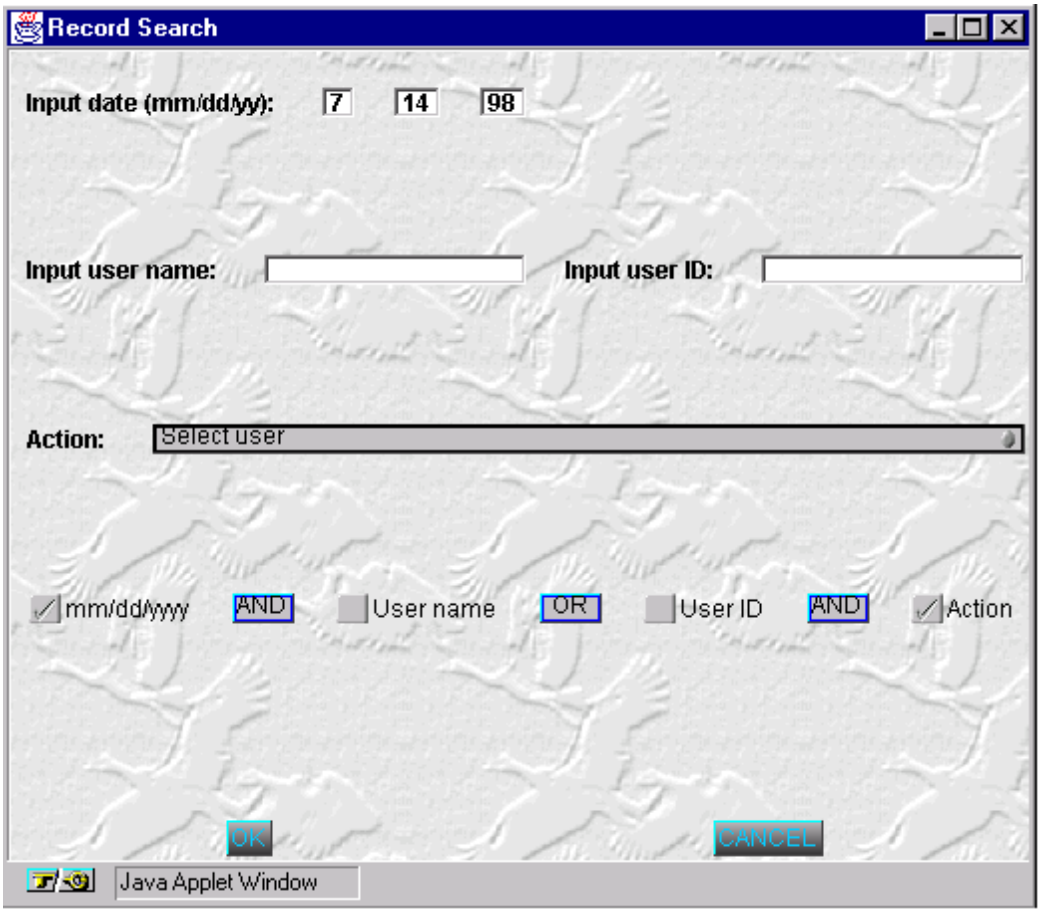


Figure 5.4 The interface for record searching

The script system provides functions that are related to decision guidance and reflect the customizability of **TEAMDEC**. Because suggestion is knowledge-based, it is derived from knowledge and experience stored in the databases under predefined decision rules. A script offers a knowledge representation form for suggestion that reduces the effort in carrying out a new task. **TEAMDEC** supports customization, which means that the user is able to construct a new sequence of actions by modifying the suggested script. This customization includes reconfiguration and parameter modification. Users can reorder the sequence, select the useful items, delete useless items, or modify parameters of certain items. In other words, users can reconstruct the

script. The script system improves individual working efficiency to improve group working efficiency.

TEAMDEC provides various meeting facilities to satisfy various meeting requirements in a flexible fashion. Compared with traditional meeting tools, a GDSS not only significantly improves effectiveness and efficiency, but also break time, space, and geographical limitations. Computer-based meeting facilities are important GDSS software components (See section 2.4.3 *Components of GDSS*). The facilities in **TEAMDEC** include the following:

- Group discussion board (see Figure 5.5),
- Whiteboard (see Figure 5.6),
- Notice sender (see Figure 5.7), and
- Video conferencing (see Figure 5.8).

The group discussion board is used in group discussions in which the information is represented by text. As long as a group has been built, a meeting host sends out a notice to ask his group members to attend a meeting. Then connections will be established between the participants via network protocols. Each participant is able to view the meeting contents and ideas posed on the discussion board through a common window. Each participant is allowed to write his words onto the board as long as his connection is not broken.

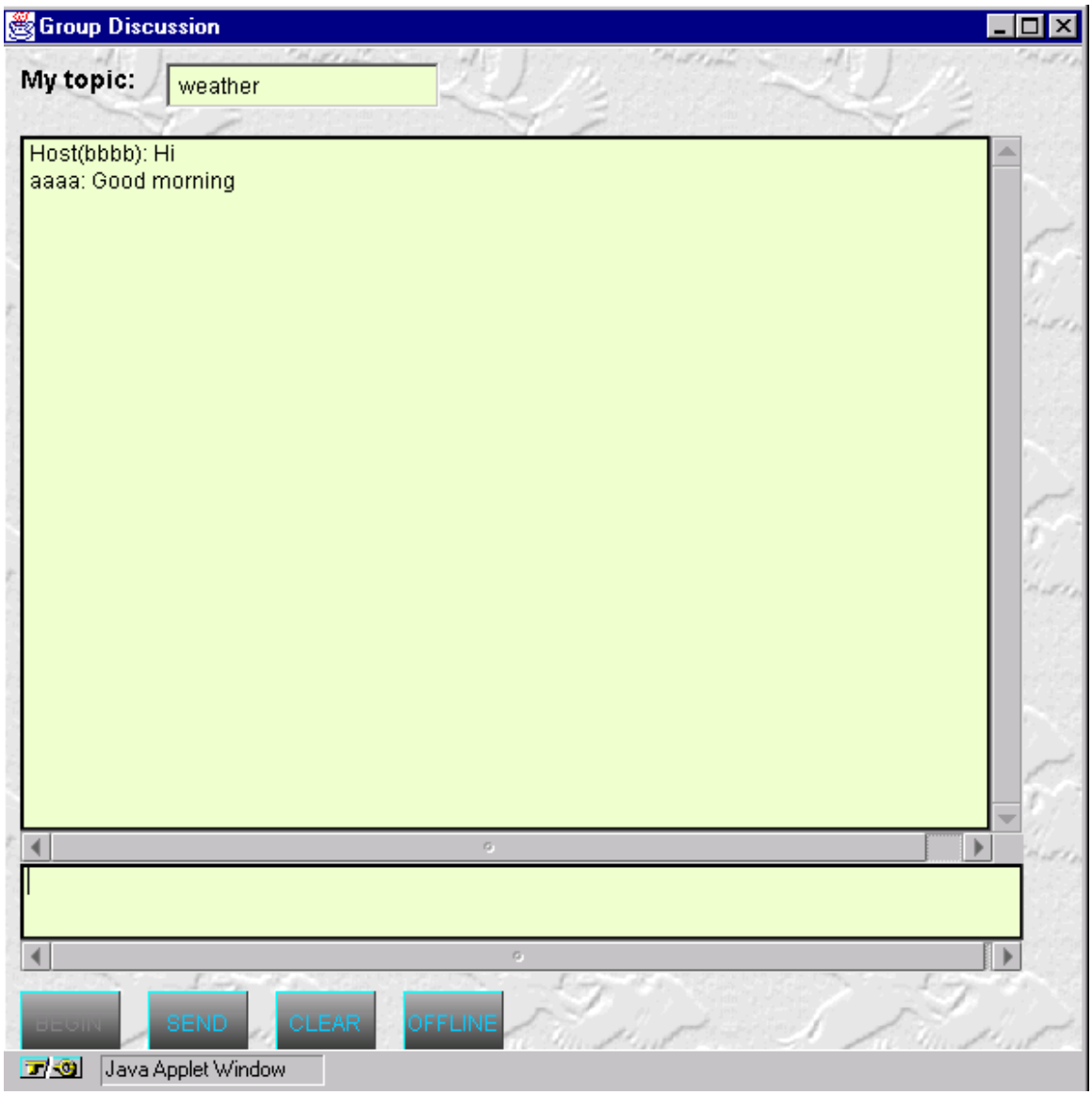


Figure 5.5 The interface of a group discussion board

From the user's view, the difference between a whiteboard and a discussion board is that the information presentation of discussion board is in text, and the information presentation of whiteboard is in both text and graphics. It allows participants to express their ideas in flexible and multiple ways. The color and font on the screen can be changed. Geometrical figures can be drawn on the board. By using different fonts,

colors, and figures, participants can highlight their key points and explain their viewpoints more easily. In addition, if the participant needs to discuss a topic related to a document and a web page, he can display a text file or a web page on the whiteboard and mark his points of interest on the item. This assists visual decision-making (See section 2.4.3).

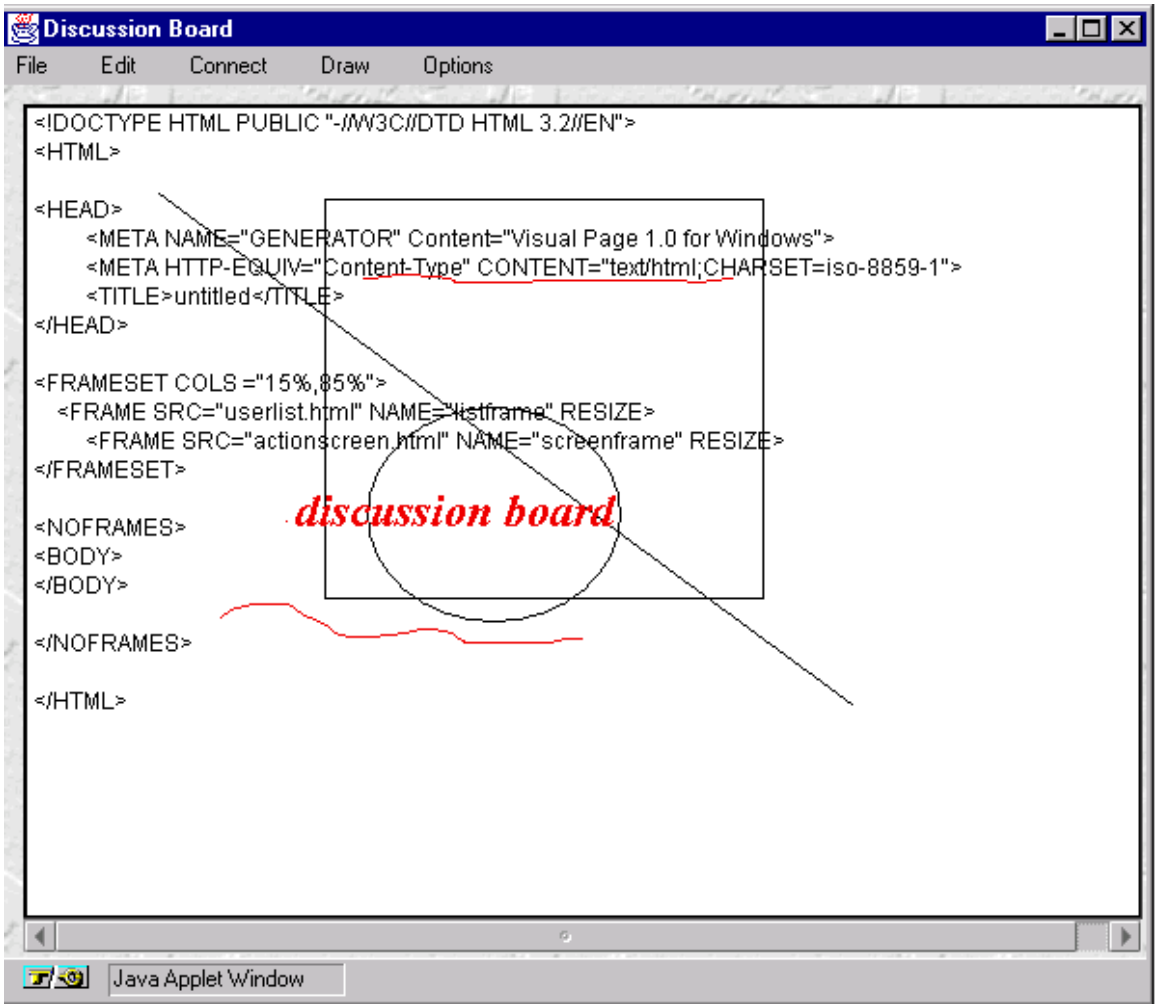


Figure 5.6 The interface of a whiteboard

A user can broadcast a notice to every group member by means of the notice sender.

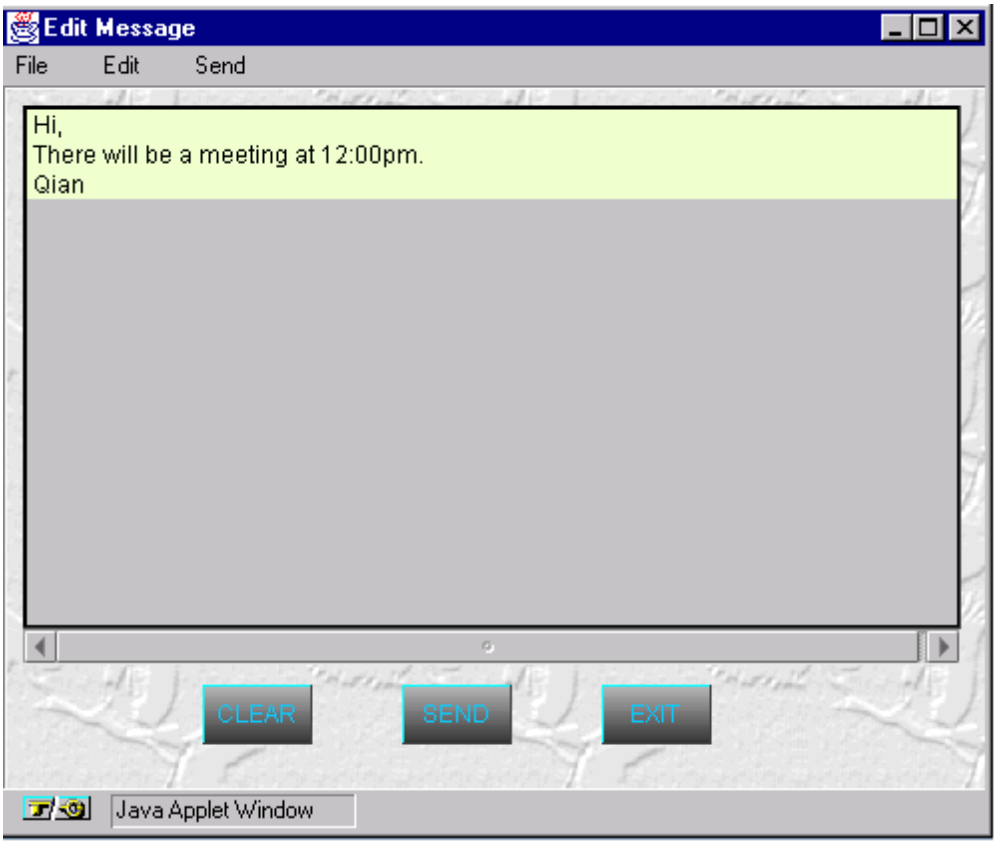


Figure 5.7 The interface for notice editing and sending

Video conferencing is a face-to-face visible meeting form. It requires video cameras to capture participants' images and voices. Video conferencing is a vivid group meeting method to allow participants to communicate with each other directly and conveniently.

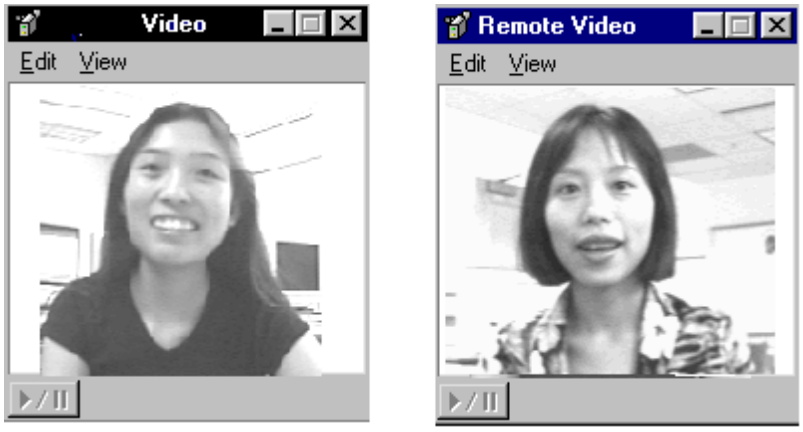


Figure 5.8 Video conferencing

5.4 Quality analysis of TEAMDEC's user interface

TEAMDEC is an interactive software system for GDSSs. The goal of **TEAMDEC** is to develop a group decision-making support system with safety, utility, efficiency, effectiveness, and usability. To achieve this goal, the principles of GDSSs, interactive software, and software development techniques are applied in the development of **TEAMDEC**. **TEAMDEC** is concerned with decision making, suggestion generation, system performance from the user's point of view, and the system internal attributes from the designer's point of view. The next two subsections analyze the system external and internal properties in the application domain.

5.4.1 External properties

The external quality properties of the system are measured from the user's perspective. This section discusses the quality properties in three major forms: goal completeness, interaction flexibility, and interaction robustness.

Lifecycle model and V-model with backtracking are two reference models adopted in the development process of **TEAMDEC**. Basically, the task-analytic approach

leads to goal completeness. The principal objective of **TEAMDEC** is to support team-based decision-making in an efficient, effective and safe way. Although a full judgment is not possible because **TEAMDEC** has not yet been put into use beyond the development team. Based on current task analysis, **TEAMDEC** has achieved software goal completeness because it enables users to interact with it to achieve their purposes.

Interaction flexibility means that multiple ways are available for the user and system to exchange information. Interaction flexibility depends on multiple I/O devices, multiple representations for I/O, I/O reuse, non-preemptive interaction, reachability of system states, reconfigurability, and adaptivity. The use of good interface design methods and the use of a well-structured architecture are important for almost all the flexibility properties. These issues of the flexibility properties of **TEAMDEC** are discussed next.

TEAMDEC has a multi-device capability which means that it uses multiple devices to get data in and out. For example, images are captured by a video camera, sounds are put out by speakers, results can be seen on a monitor, the mouse is used to click menus and draw figures, and the keyboard is used to type characters and numbers into the computer. The multi-device capability of **TEAMDEC** provides many ways for users to interact with the computer, and it puts communication redundancy to full use in the meeting facilities. The multi-device capability of **TEAMDEC** is needed not only for operation convenience, but for system fault tolerance as well. The multi-device capability ensures that group-based activities will not be inhibited because of some (hardware) faults. Representation multiplicity along with multiple devices provides more than one way to present the information about an object. For example, when the user receives a message, not only the text in a message box displays the content of the message, but also a brief sound represents the arrival of the message.

TEAMDEC offers alternative representations for both input and output. For example, **TEAMDEC** supports alternative representations of the notion of a “history” event. The components of action “history” records can be presented in a hierarchically schematic form, if the structure and organization of the action group is important. The action records may also be presented as a series of snapshots (e.g. record replay) if the user wants to know the event in a more straightforward way. The multi-representation

capability provides the user with the opportunity to choose the most suitable information presentation form for the task.

I/O reuse is an important convenience factor to users. In **TEAMDEC**, the earlier input can be used again if the same situation occurs. For example, the message that is sent this time can be saved and loaded to re-send in the future. Databases provide a good way to re-use input or output information. The individual user's interaction can be saved in databases for future knowledge-based activities. Action suggestion is related to I/O re-use. It is based on a collection of the user's current interaction and the knowledge in databases. By means of database access, the decision-making participants are able to share information, allowing the inference engine to derive decision guidance from the knowledge in the databases.

Along with introducing the computer into group decision-making activity, GDSSs improve the way that meetings run. Through the assistance of the computer, multiple tasks can be executed simultaneously. Multithreading is an attribute of **TEAMDEC** to support this. For example, the basic tasks for every **TEAMDEC** client-side application are to keep track of other users' information to update the communication group, send a signal to the server to keep the client status as "alive", and keep listening for notices from other users or the server. All these three basic tasks are executed simultaneously. In addition to the above three basic tasks, once the record option has been set to "on", a thread starts to handle recording specified events. Multithreading is involved in most of the meeting facilities. For example, during a group discussion, the client listens on multiple communication channels to allow for multiple inputs simultaneously.

TEAMDEC is a non-preemptive interactive system because it can tolerate any permissible event occurring at any time. A preemptive system will enforce a sequence of interaction that is not necessarily expected by users. **TEAMDEC** users can choose the next available action freely. However, in order to prevent the system from error states, some illegal operations are not permitted. As mentioned in Section 4.3.2, the robustness property of deviation tolerance must be balanced against the flexibility property of non-preemptiveness.

Reachability refers to the possibility of navigation through system states. This property can not be fully met in **TEAMDEC**, because the states of **TEAMDEC** are real

time states, e.g. users' on-line status and meeting procedure states. The user can not go backwards to a previous state with an "old" meeting group and the same meeting content if some group members are not available and other settings have been changed. Even if the user can still have the same organization of his group, the contents of a new meeting are unpredictable. It is impossible to guarantee that every group member will say the same words and have the same ideas. The meeting goal can not be simply reached just by repeating the previous state. However, it is essential to be able to reach the "historical" records of the actual state of **TEAMDEC**. The reachability of **TEAMDEC** is not real time information reachability, but "history" reachability instead. The user is able to search "history" events by means of the **TEAMDEC** searching engine (see Section 5.3.2). Although the action record manager allows users to redo an "old" series of actions, the task can only be executed according to current settings.

Reconfigurability refers to the user's ability to modify representations and operations. The reconfigurability of **TEAMDEC** supports the customizability of **TEAMDEC** which is an important attribute of a decision support system. Chapter 4 has illustrated a practical example of reconfigurability in the script system of **TEAMDEC**. In the script system, users are allowed a certain range of customization on analogical representation and procedural representation, such as the order of components, the number of action items that are selected for execution, and the parameters of procedures of each script element.

Adaptivity is one of the aspects of customizability. Adaptivity refers to automatic customization of a user interface by a system. This property can be reached by surveying the situation and the interaction pattern of **TEAMDEC**. Chapter 4 has illustrated this property in the script system. Not only the customization on a script, but other interactions in **TEAMDEC** also reflect adaptivity. The **TEAMDEC** interface will work on any computer with a different monitor size. Moreover, **TEAMDEC** can be adapted to various platforms because of Java's platform independence feature.

Interaction robustness

Interaction flexibility emphasizes the system's usability. Interaction robustness focuses on the system's ability to minimize the risk of task failure. To guide the user's

behavior, it is important to give out a correct and complete picture of the system, such as observability, insistence, and honesty. In addition, predictability, access control, pace tolerance, and deviation tolerance are properties to reduce the risk and cost of mistakes.

Observability means that system makes all relevant information perceivable to users. However, the observing capability of a human is a factor that should be taken into account, because information overload should be avoided. To avoid information overload, there are three alternatives:

- make information relevant to the current task perceivable to users,
- select necessary information to display continuously,
- display some information only when it needs to be observable.

Information relevant to the current task should be readily available to users. For example, when the user wants to select on-line users into his communication group, the information of other users is relevant to the task. In **TEAMDEC**, the user can click on other user's name, then a message window will be opened to show that user's name, machine name, IP address and on-line status. Each whiteboard discussion participant can check his team information, such as who is in the team now (see Figure 5.9).

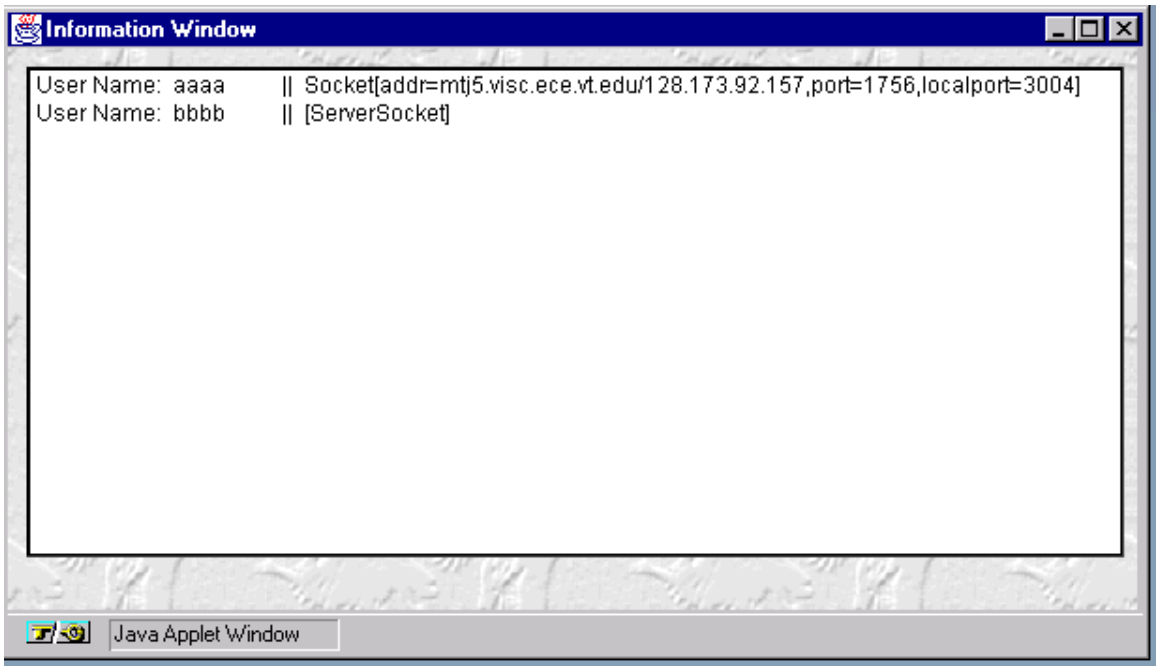


Figure 5.9 The information of on-line team members

The necessary information is required to be displayed continuously. If the user has established his communication group, then the name of his group members will be displayed in the user list on the left part of screen (see Figure 5.1). The user list is continuously observable. The user list is dynamic. If that user is considered as off-line, then the corresponding user name will be removed from the list. Another continuous information display is the record icon. The animation icon indicates that the action recorder has been activated. This kind of continuously displayed information is inexpensive in terms of the perceptive load.

Although it is not necessary for certain kinds of information to be displayed continuously, they still need to be observable at times. When a certain kind of information is required to be observable, this need can be accommodated by making searching or browsing facilities available. For example, **TEAMDEC** provides an action record searching facility to support the user's particular goals. The view of the record list helps the user to know how many action series are temporarily saved in current memory.

The user can browse the actions by invoking the record play function. To load a previous action series from database, the record searching engine is invoked (see Figure 5.4).

Section 5.3.2 has addressed the reasons for the property of access control. First, the user should be authorized before he enters **TEAMDEC**. Second, because of the multiplicity of human roles with different goals, each role is permitted access to a certain range of information. Eventually, **TEAMDEC** will integrate complete information access control.

Honesty is a system capability to ensure users correctly interpret perceived information. Honesty is a fundamental feature of all aspects of **TEAMDEC** because it enables tasks to be executed in a correct way. Furthermore, it reduces the negative effect on quality of group meeting outcomes. For example, a problem will occur if the system shows clearly that the user has successfully established a connection to his communication partner but, in fact, his connection has failed, and the expected communication partner does not exist at all. To avoid dishonesty, which may be caused by system malfunction or user misunderstanding, **TEAMDEC** is designed to detect dishonesty and update all task-relevant information in real time. A warning message or explanation aims to prevent the user's misunderstanding, e.g. a message says that someone is off-line, or a message announces that the connection has failed.

Predictability is a capability based on the system, user knowledge, and user expectation. It means that the user can know the future behavior of the system depending on past interaction and the current system state. Predictability involves psychological factors that account for human predictive and evaluative judgment. Consistency and temporal stability are factors that affect predictability. For example, the previous system using experience makes the user build expectations of execution time for a certain operation. The group discussion host sends out an invitation to a group of people for agreement acknowledgements. He thinks that normally the other people will reply to the notice within a certain amount of time; otherwise it should indicate that they don't plan to attend the meeting, or that the system has malfunctioned. Honesty and observability have been provided in the above example by giving the user information; such as a real-time information table indicating whether the invited people are online, whether the invitation notice has been sent out, and whether the invited people have acknowledged (see Figure

5.10). Based on the user's interaction experience and the observable information, he can make a decision on when to begin the meeting.

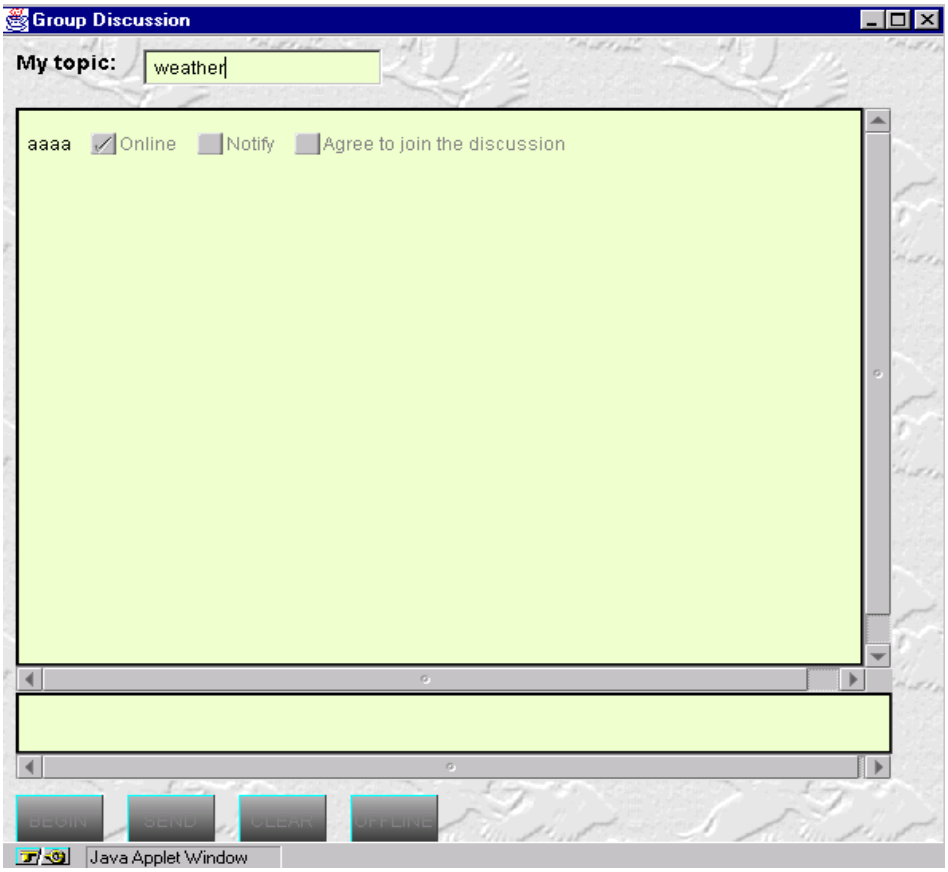


Figure 5.10 The interface supporting predictability of a group discussion

Pace tolerance contributes to interaction robustness. A pace-tolerant system allows users to control the pace of interaction. Pace tolerance is not feasible for real-time procedures. But for non-real-time procedures, **TEAMDEC** supports pace tolerance in playing action records. The user can easily control interval time between two records. Of course, a text editor is another typical example, because it is generally purely reactive and allows users to choose their own pace for typing. Text editors are often used in **TEAMDEC** meeting facilities.

It is inevitable that the user will commit errors, even if the system is well designed. A system can be called deviation tolerant system if it can (i) prevent users from getting into error states and (ii) correct errors[8]. This property requires the system to "force" users not to do something or prompt them with a warning message. The **TEAMDEC** user may make obvious mistakes, such as the user name, ID, or password that he types does not meet length regulation. In these cases, **TEAMDEC** will give users a warning message with a brief explanation and will not accept the incorrect operation. Sometimes errors may not be obvious. For example, scripts enable users to develop a new series of activities. Users are supported to reconstruct the action suggestion results. **TEAMDEC** will detect the errors according to the logical relationship between elements of scripts and the status of each element. A detailed discussion of deviation tolerance of the script system has been presented in Section 4.3.2.

Insistence ensures that the necessary information be noticed by users. In **TEAMDEC**, insistence is achieved by (i) increasing visual salience; (ii) interrupting users with pre-emptive dialogs; and (iii) using aural signals. When the action recorder is set to "on", an animation icon (see Figure 5.1) works better to attract a user's attention than a static icon. If the specified actions are going to begin and they will be captured, a message box is prompted with a brief sound to tell the user that the recorder is turned on. When **TEAMDEC** users receive a notice, they will not only see a message box (a visual signal), but also hear a brief alarm sound (if that machine is installed with sound device) to achieve an aural effect. Intuitively, people's attention will be attracted more easily by sound and movement.

5.4.2 Internal properties

Section 5.4.1 analyzes the quality properties of **TEAMDEC** from the user's point of view. This section aims to discuss some internal attributes of **TEAMDEC** from the designer's point of view. The discussion is carried out from four perspectives: functional completeness, modifiability, portability, and maintainability.

Functional completeness is an internal property corresponding to goal completeness -- an external property of **TEAMDEC** (see Section 5.4.1). If the system can satisfy the requirements of all identified tasks, then this system is viewed as a functionally complete system. Careful task analysis leads to goal completeness, which leads to functional completeness through the effort of designers. However, it can not cover all the further task requirements, because some of tasks can not be predicted within a certain period of time. In response to these unpredictable factors in functional development, the iterative design method is used in **TEAMDEC**. The procedure of iteration design is the use of evaluation techniques to gather feedback of each development version from user representatives or specialists, then such evaluation results are used to design the next version. The choice of design approaches depends on functional perspective. For example, the network-oriented design is suitable for an interactive system with extended network and collaboration requirements. Adaptive design is used for the adaptive user interface requirements. Thus various design approaches are involved in **TEAMDEC** development according to the functional requirements.

Re-use of code, clear abstractions of the system components, and software architecture are factors that influence the ease of modification. Section 5.2.3 introduces the three-tiered distributed architecture and explains the relationship between modifiability and the specific software architecture employed in **TEAMDEC**. The three-tiered distributed architecture divides the structure into three abstract function layers, supports code modules generated separately, and provides infrastructure to support code re-use. This feature of **TEAMDEC** software architecture ensures the system's modifiability.

Portability refers to the system capability of keeping the "same" interface when the environment has been changed. Changes in target environment may cause incompatibilities. **TEAMDEC** is primarily written in Java which is an object-oriented language and platform independent. **TEAMDEC** runs in Netscape Communicator which can work well on most popular platforms. These features can help reduce inevitable problems that are caused by the difference between the development environment and the target environment. **TEAMDEC** is almost platform independent which minimizes the

risk of system failure because of the change of target environment. Moreover, the adaptive design helps the user interface keep in its “original shape” corresponding to the current settings and environment. The GUI or multimedia related components will perhaps be influenced by the change of environment, e.g. if the new environment can not satisfy the device requirements of video conferencing.

Maintainability refers to the ease of keeping a system running in a given environment. Maintainability depends on the system structure and user interface. Good maintenance is supported by a clearly structured system with separate functional parts, which has been discussed in Section 5.2.3. Among **TEAMDEC** external properties, *honesty* and *deviation tolerance* are responsible for user error reduction. The error reduction methods in **TEAMDEC** include warning messages, correct interpretation of perceived information, and error detection by commonsense and logical rules.

5.5 Summary

This chapter describes a software application of an interactive team-based decision support system -- **TEAMDEC**. Based on the principles discussed in Chapters 2, 3, and 4, this chapter details the practical application of those principles. **TEAMDEC** consists of three components: hardware, software and people. The focus of this chapter is software development. The essential objective of **TEAMDEC** is to fulfill the group decision support assistant functions with a good interactive interface. Based on the requirements of GDSSs, task analysis is the first phase of a software system development process. The system software architecture is an important factor influencing system quality. The software properties are used to measure the system quality. This chapter analyzes the quality properties from two perspectives: the user’s and the designer’s, corresponding to external properties and internal properties. The illustrated examples of **TEAMDEC** show that some principles are not suitable for real time processing and that there are conflicts among some principles, which necessitate tradeoffs in system design.

Chapter 6 Conclusions

6.1 Summary

The focus of this research is on how to develop a high quality group decision support system. Based on the theories and concepts of HCI and human decision-making, the design principles of interactive software and GDSSs are applied to the development of an interactive team-based decision support system – **TEAMDEC**. The fundamental objective of **TEAMDEC** is to assist decision-making with the aid of the computer and computer-based technologies in a collaborative environment. Several concerns are taken into account, primarily including usability, efficiency, effectiveness, and security. **TEAMDEC** development is based on these concerns, and these concerns influence the quality of **TEAMDEC**.

6.2 Conclusions

There are two major aspects in the development of a high quality group decision support system: user interface and decision support. These two aspects are overlapped and interact. The decision support aspect involves psychological factors, especially cognitive psychology. For the decision support aspect, **TEAMDEC** provides various computer-based aids, such as data retrieval, information searching, multiple information presentation forms, action suggestion, and script-based knowledge representation. The action suggestion provision has a significant impact on guiding the decision-maker's behavior. Human decision-making is a knowledge-based cognitive activity. Similarly, **TEAMDEC** produces suggestion based on knowledge (from databases and the user's

current actions) by using the inference engine that derives results from knowledge under a set of decision rules. The concept of mental models is adopted into suggestion generation and the **TEAMDEC** script system. The suggestion producer has the ability not only to predict the user's future behavior, but also to evaluate the optimal method.

To fulfill the functions of group-decision-support-related facilities, the user interface design must be taken into account. Knowledge of psychology and physiology helps the designer to understand human behavior, which is intertwined with system usability. Besides usability, the software designer considers effectiveness, efficiency, and security. Based on the task requirements and quality goals, the relevant issues of interactive system development are determined from a functional perspective, an aesthetic perspective, and a structural perspective; such as design approaches, development process, development techniques, software architecture, hardware components, and hardware-related technologies. The design approach in **TEAMDEC** is chosen according to actual situations and purposes. Basically, the design approach of **TEAMDEC** is a task-oriented approach. Because **TEAMDEC** runs in a network and its users are a group of people who need to interact and collaborate with each other, the network-oriented approach is employed to extend the task analysis with the considerations related to network environment.

The quality of **TEAMDEC** can be measured from two perspectives: the user's and the designer's. The quality measures are categorized into external properties and internal properties. From the user's perspective, the quality of an interactive system can be measured in terms of external properties. From the designer's perspective, the interactive software quality can be measured in terms of internal properties. The design of **TEAMDEC** ensures interaction flexibility and robustness. The system provides multiple I/O devices for communication, alternative representations for I/O, and functions of I/O re-use. These properties contribute to the flexibility of information representation. **TEAMDEC** is a non-preemptive system that allows the user to choose the next action freely. It supports multithreading in task execution, which contributes to the flexibility of planning of task execution. Customizability is a significant characteristic of **TEAMDEC**. Certain settings of the system can be reconfigured by the user to satisfy the user's requirements or by the system internally to be adaptive to its new environment.

TEAMDEC is safe to use because of its properties of interaction robustness. It provides a correct and complete picture of task progress and constrains the access range of the user to prevent the user from misunderstanding or making mistakes. Even if the error occurs, it can correct the error to minimize the risk of system malfunction. The software architecture, development methods, and design approaches are considered in the design of **TEAMDEC**. These concerns ensure quality of the internal properties of the system, such as functional completeness, modifiability, portability, and maintainability.

TEAMDEC is the result of a comprehensive process to develop a high quality group decision support system. It involves people, computers, and tasks. Various theories and technologies contributed to the development of this group decision support system.

Reference

[1]. H. Bidgoli, *Intelligent management support systems*, Quorum books, Westport, Connecticut, 1998.

[2]. J. Burger, *Multimedia for decision makers*, Addison-Wesley, Reading, MA, 1995.

[3]. G. R. Ellis, "Application of experiment design in an activity-based environment," M.S. Thesis, VPI&SU, Blacksburg, February, 1993.

[4]. B. Ruf, "Decision making in a decision support systems environment: an evaluation of spatial ability and task structure," Ph.D dissertation, VPI&SU, Blacksburg, June, 1990.

[5]. W. A. Ceccucci, "Decision support systems design: a nursing scheduling application," Ph.D dissertation, VPI&SU, Blacksburg, January, 1994.

[6]. B. Henderson-Sellers and J. M. Edwards, *Booktwo of object-oriented knowledge: the working object*, Prentice Hall, Englewood, New Jersey, 1994.

[7]. M. Shepperd and D. Ince, *Derivation and validation of computer metrics*, Oxford University Press Inc, New York, 1993.

[8]. C. Gram and G. Cockton, *Design principles for interactive software*, Chapman & Hall, New York, 1996.

[9]. B. Laurel, *The art of human-computer interface design*, Addison-Wesley Publishing, Reading, MA, 1990.

[10]. M. Helander, *Handbook of Human-Computer Interaction*, North-Holland, New York, 1991.

[11]. P. Johnson, *Human-Computer Interaction*, Mcgraw-Hill Book Company, London, 1992.

[12]. C. A. Ntuen and E. H. Park, *Human interaction with complex systems: conceptual principles and design practice*, Kluwer Academic Publishers, Boston, 1996.

[13]. J. Preece, Y. Rogers, H. Sharp, D. Benyon, and T. Carey, *Human-Computer Interaction*, Addison-Wesley, Reading, MA, 1994.

[14]. J. A. Waterworth, *Multimedia interaction computers*, Ellis Horwood, New York, 1992.

[15]. S. Andriole and L. Adelman, *Cognitive systems engineering for user-computer interface design, prototype, and evaluation*, Lawrence Erlbrum Associates, New Jersey, 1995.

[16]. K. Cox and D. Walker, *User-interface design*, Prentice Hall, New York, 1993.

[17]. A. Hutt, *User interface*, Prentice Hall, New York, 1993.

[18]. L. Macaulay, *Human-computer interaction for software designs*, International Thomson Computer Press, London, 1995.

[19]. J. Nielsen and R. L. Mack, *Usability inspection methods*, John Wiley & Sons, New York, 1994.

[20]. S. Treu, *User interface design*, Plenum Press, New York, 1994.

[21]. L. Bass and P. Dewan, *User interface software*, John Wiley & Sons, New York, 1993.

[22]. E. A. Stohr and B. R. Konsynski, *Information systems and decision processes*, IEEE Computer Society Press, Los Alamitos, 1992.

[23]. V. L. Plantamura, B. Soucek, and G. Visaggio, *Frontier decision support concepts*, Sixth-Generation Computer Technology Series, New York, 1994.

[24]. A. P. Sage, *Decision support systems engineering*, John Wiley & Sons, New York, 1991.

[25]. M. S. Silver, *Systems that support decision makers*, Wiley Series In Information Systems, John Wiley & Sons, New York, 1991.

[26]. D. J. Berg and J. S. Fritzinger, *Advanced techniques for Java developers*, John Wiley & Sons, New York, 1997.

[27]. J. Jaworski, *Developer's Guide: Java 1.1 (2nd edition)*, Sams Net, Indianapolis, 1997.

[28]. D. J. Hughes, " Mini SQL 1.0.16 (OS/2 version)", Hughes Technologies Pty Ltd., http://Hughes.com.au/.

[29]. W. R. Stanek, *Developer's guide: Netscape one*, Sams Net, Indianapolis, 1997.

[30]. J. R. Anderson, "Retrieval of propositional information from long-term memory," *Cognitive Psychology*, 1974, pp. 451-74.

[31]. I. Benbasat and R. B.Nault, "An evaluation of empirical research in Managerial Support Systems," *Decision Support Systems*, August, 1990, pp. 203-226.

[32]. J. M. Carroll, “Infinite detail and emulation in an ontologically minimized HCI,” *Empower People*, CHI 90, Addison-Wesley, Reading MA, 1990.

[33]. K. J. W. Craik, *The Nature of Explanation*, Cambridge University Press, Cambridge, 1943.

[34]. D. Diaper and P. Johnson, “Task analysis for knowledge descriptions: theory and application in training,” *Cognitive Ergonomics and Human Computer Interaction*, Cambridge University Press, Cambridge, 1989.

[35]. A. Dix, “ CSCW – a framework”, in Rosenburg, D. and Hutchinson, C., editors, *Design Issues in CSCW*, Springer-Verlag, 1994.

[36]. P. M. Hogarth, *Judgment and choice* (2nd ed.), Wiley-Interscience, New York, 1987.

[37]. Jarvenpaa, L. Sirkka, “The effects of task demands and graphical format on information processing strategies,” *Management Science*, March, 1989, pp.185-303.

[38]. P. N. Johnson-Laird, *Mental Models*, Cambridge University Press, Cambridge, 1983.

[39]. P. N. Johnson-Laird, *The computer and the mind*, Harvard University Press, Cambridge, MA, 1988.

[40]. J.B. Long and J. Dowell, “Conceptions of the discipline of HCI: craft, applied science and engineering,” *People and Computers V*, HCI’89, Cambridge University Press, Cambridge, 1989.

[41]. R. O. Mason and I. G. Mitroff, "A program for research on Management Information Systems," *Management Science*, January, 1973, pp. 475-487.

[42]. H. Mintzberg, D. Raisinghani, and A. Theoret, "The structure of unstructured decision processes," *Administrative Science Quarterly*, June, 1976, pp. 246-275.

[43]. D. A. Norman, *The Psychology of Everyday Things*, New York: Basic Books, 1988.

[44]. D. A. Norman, *Turn Signals are the Facial Expressions of Automobiles*, Addison-Wesley, Reading, MA, 1992.

[45]. J. F. Rockart, and D.W. DeLong, *Executive Support Systems: The Emergence of Top Management Computer Use*, Dow Jones-Irwin, Homewood, Illinois, 1988.

[46]. P. A. Sage, "Behavioral and organizational considerations in the design of Information Systems and processes for planning and decision supports," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1199, September, 1981, pp.640-678.

[47]. R. C. Schank and R. Abelson, *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.

[48]. R. Sharda, S. H. Barr, and J. C.McDonnell, "Decision Support System effectiveness: a review and an empirical test," *Management Science*, February, 1988, pp.139-159.

[49]. H. A. Simon, *The New Science of Management Decision*, Harper & Row, New York, 1960 (revised edition, 1977).

[50]. H. A. Simon, "Theories of bounded rationality," *Decision and Organization*, North-Holland, New York, 1972.

[51]. H. R., Jr, Sprague and E. D. Carlson, *Building Effective Decision Support Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[52]. P. Todd, and I. Benbasat, "An experimental investigation of the impact of computer based decision aids on the process of preferential choice," Working Paper 88-MIS-026, University of British Columbia, Faculty of Commerce and Business Administration, June, 1988.

Vita

Qian Chen received her B.S. in electrical engineering from Xi Dian University, China in 1992. From 1992 to 1994, she worked as a software engineer for The Computer Company of Zheng Jiang Province, China. From 1994 to 1996, she worked as a manager of the computer department of A&J securities company.

She joined the Virginia Tech Information System Center (VISC) in December of 1997 as a graduate research assistant. Her major research interest is interactive software design.