Supporting Information Navigation in Generative Design Systems

Dissertation

Sheng-Fen Chien

Submitted to
the School of Architecture of Carnegie Mellon University
in partial fulfillment of the requirements for the degree of Doctor of Philosophy

School of Architecture
Carnegie Mellon University
Pittsburgh, Pennsylvania
U.S.A.

May 1998

Abstract

Generative design systems make it easier for designers to generate and explore design alternatives, but the amount of information generated during a design session can become very large. Intelligent navigation aids are needed to enable designers to access the information with ease. Such aids may improve the usability of generative design systems and encourage their use in architectural practice.

This dissertation presents a comprehensive approach to support navigation in generative design systems. This approach takes account of studies related to human spatial cognition, wayfinding in physical environments, and information navigation in electronic media. It contains a general model of design space, basic navigation operations, and principles for designing navigation support. The design space model describes how the space may grow and evolve along predictable dimensions. The basic operations facilitate navigation activities in this multi-dimensional design space. The design principles aim at guiding system developers in creating navigation utilities tailored to the needs of individual design systems.

This approach is validated through prototype implementations and limited pilot usability studies. The validity of the design space model and basic navigation operations is examined through the development of a design space navigation framework that encapsulates the model and operations in a software environment and provides the infrastructure and mechanisms for supporting navigation. Three prototype navigation tools are implemented using this framework. These tools are subjected to usability studies. The studies show that these tools are easy to learn and are efficient in assisting designers locating desired information. In summary, it can be demonstrated that through the prototype implementations and usability studies, this approach offers sufficient support for the design and implementation of navigation aids in a generative design system.

The research effort is a pioneer study on navigation support in generative design systems. It demonstrates why navigation support is necessary; how to provide the support; and what types of user interaction it can offer. This research contributes to information navigation studies not only in the specific domain of generative design system research, but also in the general field of human-computer interaction.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Professor Ulrich Flemming, for his confidence in me, and his extraordinary clarity of thinking and patient help that guided me through many critical moments in this work. I am also grateful to my dissertation committee: to Professor Ömer Akın for his thoughtful critiques and suggestions, especially on empirical studies; and to Professor Susan Finger for many constructive comments and her encouragement.

I would like to thank my colleagues: the seedlings, Zeyno Aygen, Shang-chia Chiou, Bongjin Choi, Magd Donia, Rana Sen, Jim Snyder, and Wen-Jaw (Jonah) Tsai, for their ideas and insights that have been crucial to the development of this work and to my personal well-being; Eric Griffith of USA-CERL for his suggestions and encouragement in the beginning phase of this work; and Professor Robert Woodbury in Australia for his advice and encouragement.

In addition, I would also like to say thanks to those who have made my days in Pittsburgh memorable: to Mikako Harada and Ruria Namba for their special friendship; to Cornelia and Michael Cumming, and Patty and Robert Ries for their thoughtfulness and hospitality; to Safwan Aly, Professor Ardeshir Mahdavi, Georg Suter, and friends of weekly volleyball games for the laughter and spikes; and to Lars Baerentzen, Edurado Camponogara, Matt Drown, Sanjay Sachdev, and other tea-time "philosophers" at the EDRC/ICES for stimulating discussions about all matters in life.

I thank mentors and friends who have helped me in various phases of my study: Professor Ming-Hung Wang in Taiwan for introducing me to the field of design research; Professor Jens Pohl in San Luis Obispo for his encouragement to pursue a Ph.D. degree; Professor Mark Gross in Boulder for his advice and encouragement during this work; and Yi-Luen Ellen Do for her intellectual and emotional support throughout these years.

I am deeply indebted to my Mother who has been my constant source of support and inspiration. I am also grateful to my brothers, Sheng-Feng and Sheng-Lin, for their support, love and faith in me. I would like to thank my dearest love, Shinyuan, for being understanding and supportive during all the difficult times in my graduate studies.

Finally, I would like to acknowledge the Engineering Design Research Center (EDRC, a NSF supported engineering research center); Institute of Complex Engineered Systems (ICES); Carnegie Mellon/Building Industry Computer-Aided Design Consortium (CBCC); and U.S. Army Corps of Engineers Construction Engineering Research Laboratory (USA-CERL) for supporting me during my Ph.D. study.

Table of Contents

List of Figures

List of Tables

1. Introduction

Generative design systems assist designers to explore design alternatives through the rapid generation of design models. However, the amount of information generated during a design session can become very large. I believe intelligent navigation aids are needed to enable designers to manage and access the information with ease. Furthermore, such navigation aids may improve the usability of generative design systems and encourage their use in the architectural design practice. In this dissertation, I will demonstrate *why* navigational aids are necessary, *how* to provide these aids, and *what* type of user interaction they can offer.[1]

1.1 Motivation

To date, more and more computers have been used in architectural design offices. The CAD (computer-aided drafting or computer-aided design) tools frequently used in these design offices provide two types of support (see Langdon 1997). These tools are for two-dimensional (2D) drafting and three-dimensional (3D) modeling tasks. The drafting tools replace traditional tools of pencils and rulers to produce detailed design drawings, while modeling tools take over the functions served by traditional physical building models to allow not only the visualization of new designs but also virtual walkthrough of buildings. These tools assist designers in the final production and presentation of the design products.

However, few of those CAD tools have been used to assist designers in the early phases, such as schematic design, of the design process (e.g. see Jog 1993; Leslie 1992; Shu, Neeman, and Langdon 1994). During the early phases, designers explore many design alternatives. Current CAD tools do not provide sufficient design support to create and evaluate alternatives rapidly and manage them. For example, if a designer creates design alternatives by varying the size of a room, the adjacent rooms, or in turn a section of the building, may have to be resized in order to maintain the proper dimensions. Designers using a current CAD tool will have to do these changes manually and keep each alternative as an independent drawing for later references.

Design tasks in the early design phases can be supported by generative design systems. These systems can assist designers to create feasible design solutions rapidly through generative mechanisms, such as design heuristics, shape grammars, constraint solving, and so on. Early research of generative design systems has focused on the representation

1. In this dissertation, the term "design" is used to refer to architectural design, and "designers" to architects and designers in architecture related professions.

of design problems and evaluation of design criteria. The prototype systems have had very limited capabilities, for example having very simple generative mechanisms such that they could only produce solutions for a particular kind of design problems (see, e.g. Eastman 1975; Tzonis and White 1994). Recent advances in this research area, such as SEED-Pro (Akın et al. 1995), SEED-Layout (Flemming and Chien 1995) and SEED-Config (Woodbury and Chang 1995), have been able to provide extensive support, through sophisticated generative mechanisms and evaluation functions, for designers to investigate different formulations and decompositions of design problems and to explore alternative design solutions.

These generative design systems promise to provide superior design aids to current CAD tools, especially during the early design phases. To bring these systems into the architectural practice, first, they should be easy to use. Second, they should provide assistance to manage the information, such as different problem formulations, decompositions, and solutions mentioned previously, generated during the design process. In addition, generative design systems should support user interactions so that designers are encouraged to explore different design concepts when solving a design problem. In short, these are issues regarding the usability of generative design systems, which the research prototypes have yet to address in a systematic manner. I believe these issues can be addressed by providing navigation support in generative design systems. Moreover, I expect that providing good navigation support for generative design systems will improve their usability and thus encourage their uses by designers in architectural practice.

1.2 Generative Design Systems

The idea of generative design systems emerges from the result of early research into the understanding of the design process (for an overview of design theories and models of design process, see Rowe 1987). Among numerous models of the design process proposed by researchers over the past few decades, the three-phase design model—*analysis*, *synthesis*, and *evaluation*—outlined by Asimov (1962) is widely accepted. This model describes design as an iterative process of understanding problem and producing a statement of goals (analysis); finding plausible solutions (synthesis); and judging the validity of solutions relative to the goals and selecting among alternatives (evaluation). Iteration occurs after the evaluation phase; the solution can be revised by reviewing the analysis. Other architectural researchers, such as Akın (1986), Mitchell (1977) and Eastman (1975), consider the design process as a problem-solving process based on information processing theory (Newell and Simon 1972). From this perspective, the design process is a series of problem definition, solution generation and testing cycles. This is usually described as the *specification*, *generation*, and *evaluation* cycle. These two models of the design process share the same view, i.e. design is a staged—and iterative—problem-solving process. This view is fundamental to generative design systems. Although there are many other models of the design process, and problem-solving is by no means the only nor the most important aspect of design, Flemming and Woodbury (1995) state that the "problem-solving aspects are the easiest to support with computational tools and that they are important enough in design to make the development of such tools practically interesting."

Based on the problem-solving view, design is characterized as complex problem solving with the problem undergoing redefinition or restructuring throughout the process (Akın, Dave, and Pithavadian 1992; Simon 1981). During this process, the designer may restructure the design problem several times; and for each problem definition, there will be many alternative solutions generated. Moreover, since architectural problems are complex design problems, they are usually decomposed into tractable subproblems (Akın 1986; Alexander 1964; Simon 1981); these subproblems, in turn, form a hierarchical structure called a *problem hierarchy.* A complete solution is recomposed from subsolutions.

In general, the set of all possible problem definitions and decompositions is referred to as the *problem space*; and the set of all possible solutions as the *design space*.[2] Thus, the design process can also be understood as a transition through states in the design space. The transition from one state to the other is facilitated by generative mechanisms such as rules or actions. Applying candidate rules or actions at each state could potentially produce a vast number of solutions, i.e. a huge design space. These solutions may be feasible designs, partial designs, or even impossible designs. Thus, evaluation functions, such as heuristics or additional rules, must be used to eliminate the generation of impossible designs.

The focus of early research in this area has been to develop a formal method to describe design problems and to build mechanisms to generate alternative solutions, and recent advances have been able to demonstrate the research results through useful software prototypes. More recently, researchers (e.g. Coyne et al. 1993; Harada 1997) have started to address a new type of problem: interactions between people and computers in the process of design exploration. Using a generative design system, designers can be overwhelmed by the amount of information (i.e. different problem formulations and decompositions, and partial and complete solutions) they are able to generate in a short time. One approach to address this issue is to hide or discard the information of intermediate design states so that designers always focus on the active design state (e.g. the one shown on the screen at the time). However in this approach, it is as though designers are exploring a forest in the dark with a flashlight. The flashlight may provide sufficient information to direct the exploration, but designers would have to start from scratch if they wish to further explore a previously abandoned path. Conversely, if all information generated during the design session is made available to designers, it would be like exploring the forest in daylight; designers can make further design decisions based on previous decisions, and their intermediate efforts are preserved and can always be revisited and expanded in new directions. Since the collection of all design states captures indirectly the design process and decisions made by designers during a design session, it may be preferable to keep the intermediate design states. Nevertheless, in this case, the issue of information overload will have to be addressed by providing utilities to manage the information and allow designers to locate desired information with ease.

2. Here, I have separated problems and solutions, which are considered together in the problem space according to the human problem-solving theory by Newell and Simon (1972), into two distinct sets to bring more attention to solutions.

1.3 Information Navigation

The Marriam-Webster Dictionary (1998) defines *navigation* and *navigate* as follows:

nav·i·ga·tion
1: the act or practice of navigating
2: the science of getting ships, aircraft, or spacecraft from place to place; especially: the method of determining position, course, and distance traveled
3: ship traffic or commerce

nav·i·gate
Etymology: Latin *navigatus*, past participle of *navigare*, from *navis* ship + *-igare* (from *agere* to drive)
intransitive senses
1: to travel by water: SAIL
2: to steer a course through a medium; specifically: to operate an airplane
3: GET AROUND, MOVE
transitive senses
1 **a**: to sail over, on, or through; **b**: to make one's way over or through: TRAVERSE
2 **a**: to steer or manage (a boat) in sailing; **b**: to operate or control the course of (as an airplane)

Although the origin of these terms is closely tied to physical movement, people have used them in a more general sense to refer to the activity of moving between locations. For instance, we navigate in cities and in finding our way through buildings; but we also speak of navigating when looking for information in libraries or choosing which television program to watch.

Researchers faced with the problem of information overload have attempted to employ the navigation metaphor to address the problem. They discuss "cyberspace navigation" (Benedikt 1991), "navigating large information spaces" (Nielsen 1995), "navigation in electronic worlds" (Jul and Furnas 1997), and so on. These discussions rest on some important assumptions: first, the physical environment that we live in and the information space where the information resides are similar enough to make it possible to utilize research results of navigation in the physical environment; second, the activities of navigation are similar to the information seeking activities of users of information spaces (Dahlbäck 1998).

Researchers have shown that hypermedia, databases, and hierarchical file systems have a spatial character (Akın, Baykan, and Rao 1987; Benyon and Murray 1993; Dahlbäck, Höök, and Sjölinder 1996; Vicente and Williges 1988). Specifically, the organization of their contents has a spatial structure. Furthermore, there is evidence showing that users with high spatial ability[3] outperform users with low spatial ability for tasks of seeking information in such types of environment (Benyon and Murray 1993; Dahlbäck , Höök, and Sjölinder 1996; Vicente and Williges 1988). Although not all information spaces exhibit this spatial characteristic, many researchers have suggested that imposing a spatial structure to information spaces (regardless of their inherent structures) could make the

3. This is a person's ability to recognize objects in an environment and the spatial relations between these objects. Further discussions regarding this concept is available in Chapter 2.

task of seeking information a more effective and pleasing activity (see Jul and Furnas 1997).

On the other hand, Norman (1993, pp. 175-180) has argued against the use of navigation metaphor. He indicates that this metaphor relies on spatial organization, which only works when the space is small and when a natural mapping between the information items and the spatial location exists. In the context of large information spaces, Norman proposes a retrieval-by-description alternative based on the way people retrieve information from their memory: think of a telephone number, and out it comes. However, he has overlooked the fact that knowing the structure of an information space may be as important as finding a specific piece of information in that space. For example, Doerry et al. (1997) in their work on the design of a database for research geneticists have found that, while users have little trouble finding specific data, they frequently become confused during multi-step data manipulation that requires the understanding of relationships between these data. In generative design systems, knowing the relationships between different pieces of information created during a design session is important to the designers (as discussed in the previous section). Therefore, the retrieval-by-description method suggested by Norman cannot provide adequate support for generative design systems.

Nevertheless, the field of information navigation research to date is still in its infancy. As Lakoff and Johnson (1980) have pointed out, each metaphor hides more than it highlights. Further theoretical and empirical work is needed to determine the appropriateness of employing the metaphor.

1.4 Research Objective and Approach

This research assumes that all information generated during a design session should be preserved and made available to designers. The goal of the research endeavor has been to develop a method that allows designers to *easily access desired information* in a generative design system where all intermediate design decisions and solutions are maintained. In particular, the research seeks to establish a model (i.e. a mathematical formulation) to describe all types of information, in generative design systems, that may interest designers and the relationships between different types or pieces of information. Furthermore, to achieve the ease-of-access idea, the navigation metaphor is employed based on an extensive literature survey. As a result, this research has developed a framework for supporting navigation in generative design systems; demonstrated the approach by implementing it in a generative design system prototype that specialized in 2D spatial layout design; and examined the validity of this approach through the usability evaluation of the prototype.

1.5 Dissertation Road Map

This dissertation is organized as follows:

- In Chapter 2, I examine the background related to the research. This background study covers a survey of literature from areas of human spatial cognition, wayfinding behavior in real-world environments, and navigation in computer environments. In

addition, I provide a comparison between navigation tasks in the real-world environments versus computer environments.

- In Chapter 3, I present the approach to support information navigation in generative design systems. The approach provides a model to describe the information space in generative design systems, defines a set of basic navigation operations for users, and offers guidelines for developers to design navigation support in this type of design system.
- In Chapter 4, I describe a software framework that implements the concepts presented in Chapter 3. The use of this framework is illustrated through an example application.
- In Chapter 5, I introduce a hybrid software engineering and usability engineering process to develop the software framework and the example application described in Chapter 4. This process enables developers to incorporate usability evaluations early in the software design stage with little or no additional effort.
- In Chapter 6, I validate the concepts presented in Chapter 3 and the implementation described in Chapter 4 through a case study. In addition, I discuss findings from pilot usability studies including an empirical study and a formal usability evaluation.
- In Chapter 7, I summarize the dissertation and the research contributions, and discuss a number of future research directions for supporting information navigation in generative design systems.

2. Background

This research is the first study on supporting navigation in generative design systems. However, navigation issues had been studied early on in database systems and recently in hypertext and hypermedia environments. These computing environments manage *information spaces*, where information is organized in certain structures that are meaningful or useful to users. In addition, the human's ability to navigate in a computing environment may rely on human spatial cognition. Therefore, I review the literature related to wayfinding in physical environments (or *physical spaces*) to understand the cognitive aspects of the task and the navigational supports used in physical spaces. Furthermore, I compare navigation tasks in computing environments (information spaces) vs. physical environments (physical spaces). This chapter presents the related background and supporting knowledge for the research.

2.1 Wayfinding in Physical Spaces

When studying the issue of "navigation in information spaces," one cannot overlook its analogous situation of navigation in physical spaces. What enables people to find their ways through cities, jungles, or vast oceans? and what aids do people have to improve their wayfinding and keep them from being lost in these environments? The same questions need to be answered when the surroundings are not physical landscapes but abstract information spaces. Studies of human spatial cognition propose cognitive mapping theory (or variations of it) to answer the first question. Accordingly, researchers of environmental design propose answers to the second question and set up design guidelines for man-made spaces.

2.1.1 Spatial Cognition

Cognitive mapping has been the primary theory of human spatial cognition proposed in the literature. Discussions focus on cognitive mapping as the spatial cognition process and cognitive map as the organization structure of spatial knowledge. These studies are based on insights into human spatial ability.

Developmental psychologist Jean Piaget first reported human spatial ability through his studies of children's cognitive development (see Piaget and Inhelder 1956). The spatial ability is one's ability to recognize objects in the environment and the spatial relations between these objects. To recognize and relate objects in our surroundings, three strategies are believed to be common to human beings. They are egocentric, fixed, and abstract systems of reference (see, e.g. Piaget and Inhelder 1967; Hart and Berzok 1982). For a child, these strategies are developed in the order stated above, from simple to complex.

However, researchers have demonstrated that both children (Down 1985; Hart and Berzok 1982; Mandler 1988; Pick and Rieser 1982; Somerville and Haake 1985) and adults (Gärling, Böök, and Lindberg 1985; Mandler 1988) use all three strategies in various situations. The ability to form topological relations, such as proximity, separation, open, close, and between, is observed as early as the egocentric system of reference; whereas Euclidean relations, such as proportional scale, distance estimates, and coordinates are observed only in the abstract system of reference.

Our daily spatial tasks are based on spatial knowledge which we gather from the environment. Warner Brown's study (1932) on wayfinding behavior of blindfolded subjects indicates that "... subjects seemed to use at least three different kinds of orientation: a memorization of the sequence of movements, usually difficult to reconstruct except in correct sequence; a set of landmarks (rough boards, sound sources, rays of sunlight that gave warmth) which identified localities; and a general sense of orientation in the room space (for example, the solution might be imaged as a general movement around the four sides of the room, with two excursions into the interiors)." Not surprisingly, Thorndyke and Goldin (1983) describe spatial knowledge in terms of three levels of information: landmark, procedural, and survey knowledge. Each level of knowledge builds on previous levels of knowledge. Landmark knowledge represents information about the perceptually salient objects in the environment. Procedural knowledge (or route knowledge) represents information about the sequence of actions required to follow a particular route. Survey knowledge represents topological information.

Spatial knowledge is stored in cognitive maps. Studies (Downs 1985; Hart and Berzok 1982; Pick and Rieser 1982) suggest that a subject may develop different cognitive maps through the use of the three different cognitive mapping strategies (i.e. egocentric, fixed, and abstract systems of reference). Lynch (1960) observes that a person may have different images of a city and that these images are "arranged in a series of levels, roughly by the scale of area involved, so that the observer moved as necessary from an image at street level to levels of a neighborhood, a city, a metropolitan region" (Lynch 1960, p. 86). Tversky (1993) refers to these images as cognitive collages which "are thematic overlays of multimedia from different points of views" (Tversky 1993, p. 15). Furthermore, among the three levels of spatial knowledge, landmark knowledge is the essential knowledge in human cognitive maps or collages (Down 1985; Gärling, Böök, and Lindberg 1985; Golbeck 1985; Hart and Berzok 1982; Lynch 1960; Mandler 1988; Pick and Rieser 1982; Somerville and Haake 1985; Tversky 1993). In addition, landmarks "facilitate the encoding and retrieval of information about spatial location" (Golbeck 1985).

Devlin (1976) categorizes landmarks into two types: physical-perceptual and functional. Physical-perceptual landmarks are usually public facilities. They are well-known and shared by people who are familiar to the area. Functional landmarks are usually developed due to an individual's needs, such as grocery shopping, health care. These landmarks are personal and may be shared among a small group of people. For example, a small coffee shop may be considered a landmark by its frequent visitors, but not the general public. In her study of people navigating in a small town (Idaho Falls, Idaho), physical-perceptual landmarks constitute less then 25% of all landmarks identified by participants. Participants usually utilize functional landmarks in their daily navigation tasks.

Issues of wayfinding and spatial orientations are also examined in the context of urban planning and architectural design. This literature is summarized in the following section.

2.1.2 Environmental Design

Cities and buildings are man-made spaces where most of us perform daunting wayfinding tasks daily. Unfortunately, not all of these spaces provide adequate supports for navigation (i.e. people get lost in the cities or buildings). Lynch (1960) identifies a measure called *legibility* (or *imageability*). Legibility refers to the ease with which a city's parts can be reorganized and organized into a coherent pattern, or the ease with which a place can be mentally represented. Environments with high legibility or imageability are easier for people to navigate in than those with low legibility or imageability. Environmental design studies aim at improving this quality of man-made environments.

Lynch (1960) identifies five functional elements of a city. They are paths, edges, districts, nodes, and landmarks. Paths are channels of movement, such as walkways, streets, railroad, and so on. Edges are boundaries between regions, such as walls or rivers. Districts are sections of a city, which are distinguishable as having some common, identifying characteristics. Nodes are spots in a city where the observer can enter, for example, mass transit stations or bus stops. They are usually the connections of paths. Landmarks are point-references that are external to the observer. Although Lynch emphasizes visual characteristics of landmarks, other studies have shown that the identification of landmarks does not solely base on visual features. For example, Devlin's (1976) study shows that at least 75% of the landmarks identified by participants are based on functional features.

Passini (1984) outlines the elements of environmental design in three categories: signs, organization, and maps. Signs are simply used to communicate information such as direction to a place, identity of a place, or reassurance. Use of organizational elements can improve legibility of a space. They contain the five functional elements of a city described above. He proposes the application of these functional elements in architectural designs. Maps provide survey information. Passini describes three types of map: plans, views, and fantasy drawings.[1]

Lynch (1960) describes design principles for the five functional elements of the city. For example, he advocates *singularity or figure-background clarity* and *form simplicity* for the design of districts, *continuity* and *clarity of joint* for designing edges and paths, and *dominance* for landmark design. In addition, Lynch also emphasizes principles that concern movements and time, since time is often treated as the distance measure in human wayfinding tasks. These principles, in essence, focus on ensuring legibility of each functional element, which in turn will produce legible city environments.

Passini (1984) takes a process-oriented approach to the environmental design issues. He develops a seven-step design method to guide designers. His design process starts with identifying the pertinent *wayfinding tasks* and understanding the *user profile*. In the third and forth steps, the appropriate condition is identified for each task, then the *condition* is analyzed along with the consideration of users to define the *design requirements*. Next, the

1. Plans indicate the metric properties or topological structure of the whole space; views are similar to plans but have a perspective and usually in 3D; and fantasy drawings emphasize parts of a space but giving minimal sense of the whole.

specific *wayfinding solution* for each task is developed according to the design requirements. Based on wayfinding solutions, the *supportive information* is identified and incorporated into the final *design solution*. Passini's goal is to design the spaces which accommodate wayfinding tasks by providing necessary environmental information without sensory or information overload.

Researchers, particularly Lynch and Passini, have shown environmental design principles and methods which reflect their dependency on the human cognitive process in the physical environments. Legibility or imageability of a space is determined by the human's cognitive maps. Conversely, a space with high legibility or imageability can assist the development of the human's cognitive maps of the space.

2.2 Navigation in Information Spaces

The literature reviewed in this section covers research and applications in human-computer interaction (HCI), database management, hypermedia systems, World-Wide Web (WWW), computer graphics, geographical information systems (GIS), and other areas related to information management. There are two fundamental questions of information navigation: what are vehicles for navigation? and how to visualize information spaces? Research prototypes and applications address the first question by providing various types of navigational support. Novel visualization techniques are introduced to address the second question. In addition, specific research efforts that offer answers to both questions are reviewed to understand the related past or on-going efforts.

2.2.1 Navigational Supports

Navigational supports in an information space can be considered at two levels: framework and operation. Navigation *operations* are basic tools, which are analogous to compasses, maps, or vehicles we use to navigate and move around in the physical environment. A navigation *framework* provides a visual representation of an information space and a consistent interaction mechanism which integrates different navigation operations.

Studies about information navigation have been conducted early on in connection with hypertext systems[2] and more recently with the WWW environment. In summary, four types of navigation frameworks have been described:

- **Hyperlinks**: navigation through sensitive objects in an information space, where the information may or may not be structured; WWW browsers (such as Netscape or Mosaic), KMS (Akscyn, McCracken, and Yoder 1988), and InfoGrid (Rao et al. 1992) are of this type.
- **Hierarchies or networks**: navigation through nodes in an information space, where the information is organized into hierarchies (e.g. Rivlin, Botafogo, and Shneiderman 1994) or networks (e.g. Thüring, Haake, and Hannemann 1991) and views of the overall or partial information space are provided.

2. See Balasubramanian (1994) for a review of several hypertext systems, as well as issues concerning navigation in these systems.

- **Portals or wormholes**: navigation through interactive zooming and panning in an information space; the zoom operation can activate a view (portal) which displays the selected information at different scales or in different formats (Perlin and Fox 1993), or a view (through a wormhole) that displays additional data related to the selected information (Woodruff et al. 1994).
- **Architectural places**: navigation through virtual spaces, such as cities, buildings, or rooms. For example, the Information Visualizer (Clarkson 1991) supports navigation through rooms (information clusters) in an information space modeled as the interior of a physical space with rooms and doors displayed in 3D or 2D views.

Different navigation frameworks support different operations to navigate in their respective information spaces; but in general, navigation operations can be categorized into four types:

- **Go to**: moving from place to place; examples are highlighted links, or Home, Forward, Backward commands in WWW browsers.
- **History or Path**: retracing the steps and keeping records for quick accesses, e.g. History, and Bookmark/Hotlist utilities in WWW browsers and retracing in FrameMaker Help.
- **View**: viewing the information space and showing current location, e.g. panning and zooming in Tioga (Woodruff et al. 1994) and location indication in FrameMaker Help.
- **Search**: finding a set of places that satisfy certain criteria, e.g. search facilities in WWW.

2.2.2 Visualization Techniques

Unlike the physical environment where most landscapes and terrains have existed for a long time, the information space has to be constructed. In order to support navigations in information spaces, many researchers focus their efforts on answering the question: *How to visualize it*?

Visualization is the key in supporting navigation in computing environments. Research has shown that human spatial cognition applies to interactions in information spaces (Darken 1996; Mantei 1982). A good visualization of an information space provides a coherent representation of the space, supplies the user with desired information, and minimizes the user's memory load.

The visualization techniques proposed in the reviewed literature can be categorized into several types:

- **Traditional methods**. This type of method is often used to visualize statistical information. Tables, panels, graphs, scatter plots, and maps are products of this visualization technique. It has been used as a common approach to visualize high-dimensional objects (i.e. objects with more than three dimensions of interest) by presenting the two primary dimensions in 2D, and supplying visual cues (such as colors, patterns, and texts) to distinguish additional dimensions. (For examples, see Chimera 1991; Plaisant, Carr, and Shneiderman 1994; Zarmer and Chew 1992.)
- **Nodes and links**. This technique is used to visualize information with hierarchical or network structures, where nodes represent data sets and links between nodes depict

relationships between data sets. The visualization usually takes the form of 2D or 3D trees or networks (see, e.g. Pitkow and Bharat 1994; Wood et al. 1995), 3D cone trees, or structured hyperlinks (Mukherjea and Foley 1995).

- **Multiscale views**. In this technique, the information is presented at different scales to show the data of focus (in the largest scale) as well as contextual information. There are two types of multiscale displays: *focus+context* (Brown, Meehan, and Sarkar 1993; Lamping and Rao 1994; Sarkar et al. 1993) and *pan+zoom* (Bederson and Hollan 1994; Bier et al. 1993; Lieberman 1994; Perlin and Fox 1993; Rennison 1994). A focus+context scheme displays information at continuous scales, while a pan+zoom interface uses discrete scales. However, the pan+zoom interface provides possibilities for many scales overlaying at one time, while the focus+context scheme provides a single view of the information space.
- **Perspective views**. This is used to present multi-attribute information three-dimensionally; for example, the Perspective Wall in Mukherjea and Foley (1995), or the Time Lattice in Mackinlay, Robertson and DeLine (1994).
- **Memory palaces**. This is an ancient architectural mnemonic which associates things to be remembered with architectural spaces (Yates 1966). Although they do not mention it explicitly, information visualization systems use visual metaphors such as the buildings/rooms metaphor (e.g. Clarkson 1991; Staples 1993) and city/landscape metaphor (e.g. Turtiainen and Auer 1995) to organize data spaces.

With the development of the WWW and Internet, local information spaces can now connect with one another. Many researchers recognize this emerging information universe of interconnected information spaces. This emerging information universe is generally referred as *cyberspace*. Unlike the information space discussed in the information visualization literature, where the structure of an information space (relationships between data) is well-defined, the structure of the cyberspace is still under investigation. Recent cyberspace research has focused on modeling the cyberspace.

2.2.3 Cyberspaces

Although there are many definitions of the term cyberspace (see, e.g. Benedikt 1991; Woolley 1992), it was first described by William Gibson[3] as a consensual hallucination. However, among all those definitions, Benedikt (1991) proposes a definition and a model, which is most relevant to this research. Benedikt's proposal is briefly described in this section.

Benedikt (1991) defines cyberspace as a "globally networked, computer-sustained, computer-accessed, and computer-generated, multidimensional, artificial, or 'virtual' reality." He further elaborates:

> ... In this reality, to which every computer is a window, seen, or heard objects are neither physical nor, necessarily, representations of physical objects but are, rather, in form, character and action, made up of data, of pure information. This information derives in part from the operations of the natural, physical world, but for the most part it derives from the immense traffic of information that constitutes human enterprise in science, art, business, and culture.

3. Gibson first used the term in his science fiction *Neuromancer* (Gibson 1986).

> The dimensions, axes, and coordinates of cyberspaces are thus not necessarily the familiar ones of our natural, gravitational environment: through mirroring our expectations of natural spaces and places, they have dimensions impressed with information values appropriate for optimal orientation and navigation in the data accessed. (Benedikt 1991, pp. 122-123)

In his definition, database environments, the WWW, the Internet and many other local networks, and even the information spaces in generative design systems are all cyberspaces. Benedikt proposes a model of the cyberspace and suggests several visualization methods. Among his proposals, the most interesting one is the idea of extrinsic and intrinsic dimensions and the notion of space folding and unfolding based on it.

Extrinsic dimensions record the spatio-temporal location of an object. *Intrinsic* properties of an object (or attributes that are independent of its position), such as the color, shape, weight, size, are considered as intrinsic dimensions. Assume that there is an N-dimensional data space which has *n* extrinsic dimensions and *m* intrinsic dimensions: then

> N = n + m (N > 0, 0 < n < 5)
> (# of) system dimensions = (# of) extrinsic dimensions + (# of) intrinsic dimensions.

A natural way of visualizing this N-dimensional data space is to render it in a 3D space using its extrinsic dimensions and merge data of the intrinsic dimensions. However, it may be desirable to view an object in its larger context. To do so, Benedikt proposes the *unfolding* operation such that "when an object unfolds, its intrinsic dimensions open up, flower, to form a new coordinate system, a new space, from (a selection of) its (previously) intrinsic dimensions." Therefore, an object can be seen with its entirety when all its intrinsic dimensions are unfolded. Figure 2-1 shows an example of unfolding.

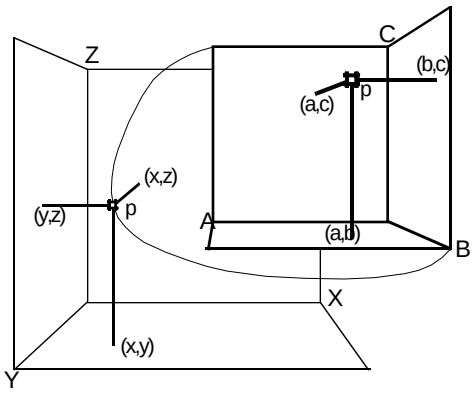

Figure 2-1: An example of unfolding
The intrinsic dimensions of the six-dimensional data object p, located at (x,y,z) in the (extrinsic) dimensional space of XYZ, are unfolded into the space of ABC and have the value given by the location (a,b,c). Reproduced from Benedikt (1991).

There are some information visualization systems which implement Benedikt's ideas; for example, the LEADS project (Ingram and Benford 1995a, 1995b) described in the next section.

2.2.4 Information Visualization Systems

The information visualization systems briefly described below are related to the research area of information navigation. These systems are not designed specifically for visualization purposes, but techniques employed in these systems are relevant to this research.

A4

Hovestadt (1993) proposes a method to structure the data needed in building design, construction and management. The system is called A4. Its data space is multidimensional, where each dimension represents a specific design feature. A4 includes as dimensions the x-, y-, z-coordinates, time, precision, morphology, size, aspect, alternatives, timetag, and user IDs. Each dimension is considered an axis in a multidimensional space modeled as a Cartesian coordinate space. Any design object is viewed as a point in that space whose coordinates are given by the location of the object's features on their respective axes. A strong underlying assumption is that the feature values are independent and can be linearized so that each feature value marks a unique point on the respective feature axis. Two commands, moveTo and Forget, are provided for navigation in the data space.

Fisheye views

Hierarchical Structures are often visualized using 2D tree or network diagrams (see, e.g. Pitkow and Bharat 1994). However, traditional tree or network views become unmanageable when the number of nodes increases. The fisheye view technique (a kind of focus+context scheme) addresses this problem by enabling users to focus on the information of interests while maintaining the relationships between those focused data and the overall context (see, e.g. Brown, Meehan, and Sarkar 1993; Lamping and Rao 1994).

Using a similar technique, Sarkar et al. (1993) demonstrate the rubber sheet metaphor, which also provides the opportunity to display multiple scales in one view within a focus+context visualization scheme. They propose the metaphor of rubber sheet stretching for viewing large and complex layouts within small display areas. As the user stretches an area, a greater level of detail is displayed inside this area. This technique is similar to the fisheye view technique described above. However, the method contains mechanisms to stretch arbitrary (orthogonal or polygonal) regions and multiple foci, which fisheye views do not support.

Galaxy of News

Rennison (1994) introduces a system called Galaxy of News. At the heart of the Galaxy of News is an engine that constructs an associative relation network (potentially multidimensional) that automatically builds implicit links between related articles. Although the information space contains relationships that are multidimensional, not all elements of the relational hierarchy are visible at a single glance; rather, only elements

that are relevant to the user's present view are shown in the pan+zoom visualization scheme with overlapped transparent displays.

HyperSpace

HyperSpace (Wood et al. 1995) is a WWW visualizer developed in the Advanced Interaction Group at University of Birmingham, UK. It organizes the selected areas of WWW according to a set of user-defined criteria, for instance, by displaying related topics adjacent to each other while keeping unrelated topics separated. The information space is displayed three-dimensionally. A WWW page is represented as a sphere; and hyperlinks between pages are represented as links between spheres.

This system is based on a more general information visualization system, Narcissus (Hendley et al. 1995). Unlike most other information visualization systems which use global layout algorithms or apply statistical techniques to produce clusters of related objects (e.g. LEADS and Navigational View Builder in the following text), Narcissus uses the technique of self-organizing systems. It adopts a forced-directed placement method[4] for the spatial layout of objects. The behavior (movement) of an object is determined by two classes of force: the repulsive force between objects and the attractive force between related objects. Once objects are released into the information space, they "migrate through space so that they are spatial[ly] close to those objects with which they are semantically related" (Hendley et al. 1995). Hendley et al. (1995) also suggest that such visualization system can lead users to form an intuitive understanding of the structure and behavior of the domain represented in the information space.

Information Visualizer

The Information Visualizer project at Xerox PARC (Card, Robertson, and Mackinlay 1991; Clarkson 1991) provides 3D *rooms* as multiple virtual workspaces, and interactive objects for different visualization methods. A room is defined by a task-oriented clustering of information. Rooms are connected by *doors*. "Work is distributed throughout a collection of 3D/2D rooms furnished with interactive objects such as walls and floating trees" (Clarkson 1991). Two of these visualization methods are the Spiral Calendar for rapid access to an individual's daily schedule, and the Time Lattice for analyzing the time relationships among the schedules of groups of people (Mackinlay, Robertson, and DeLine 1994). Clarkson (1991) claims that these visualizations are "designed to shift work to your perceptual system, freeing the conscious mind to work on larger problems."

The room metaphor in the Information Visualizer is a very clear example of using the memory palace technique. Staples (1993) suggests another use of the same technique in the GUI (graphical user interface) desktop by rendering the interface as the interior of an office space, where the information is clustered into drawers of a cabinet or a picture on the wall. NetRepreneur (Turtiainen and Auer 1995) is yet another example of using the memory palace technique. The system provides a personal workspace as a virtual city where information is accessed through visiting buildings, such as library, hospital, circus, and so on.

4. This is a method for dimensional reduction (laying out high-dimensional data in a 2D or 3D space) based on optimization. A graph is modeled as a physical system of rings and springs. This system is arranged in some initial layout and let go so that the spring forces on the rings moves the system to a minimal energy state. For further discussion, see Fruchterman and Reingold (1990).

LEADS

LEADS is one of the on-going research projects in the Communication Research Group at the University of Nottingham, UK. It aims to improve the legibility of virtual environments. To achieve this, Ingram and Benford (1995a, 1995b) propose an approach of adapting findings from the environmental design literature, particularly, Lynch's (1960) five features (landmarks, districts, paths, nodes, and edges) of city landscapes. LEADS provides the legibility features as a layer on top of an information visualization system. It identifies districts in a data space as clusters of items that are closely related. Landmarks are positioned at the center of a region of dense districts. LEADS places edges in the space between districts that are significantly large. Finally, nodes and paths are features evolving out of the use of data space. In addition to these features, LEADS places signposts near major nodes. To overcome the absence of a ground-plane in an information space, the system provides an optional "axes object" for orientation in the data space.

One of the visualization systems that LEADS is based on is Q-PIT (Benford and Mariani 1994), which is a database visualization tool developed in the same research group. The design of Q-PIT follows the cyberspace model proposed by Benedikt (1991). The system maps attributes of database information to extrinsic and intrinsic types according to the user specification. Some intrinsic attributes may be visualized using different shapes of 3D objects (e.g. cubes, spheres, cones) or rotation speeds of objects.

Navigational View Builder

Mukherjea and Foley (1995) describe the Navigational View Builder as a tool to assist users visualizing information. They first propose four strategies to structure the information: binding (assigning visual properties, e.g. color, to different data), clustering (grouping files that are closely related), filtering (allowing user control), and hierarchization. The Navigational View Builder provides several visualization methods.[5] The network overview displays information in a 2D graph, where nodes and links are rendered according to different bindings defined by the system or the user. After hierarchizing the information, the system supports table-of-contents, 3D trees, 2D trees, cone trees, or treemaps view to display the data space. "Perspective walls" is another visualization method to view data that have a high number (usually more than 4) of attributes or associations to other data. Examples of visualizing the WWW using the Navigational View Builder are demonstrated in Mukherjea and Foley (1995) as well as in the Georgia Institute of Technology, Graphic Visualization Center website (URL http://www.cc.gatech.edu/gvu/).

Pad and Pad++

Pad (Perlin and Fox 1993) is an infinite two-dimensional information plane. Objects are organized on that plane geographically: every object occupies a well-defined region on the plane's surface. *Portals* are used for navigation; they act like magnifying glasses that can peek into and roam over different parts of the Pad surface. The underlying assumption for search is that all the information is there; to see more detail, you just have to take a closer look. To facilitate the display, two techniques are used: "semantic

5. Some of these visualization methods, such as corn tree and perspective wall, are early research results of Xerox PARC (Mackinlay, Robertson, and Card 1991; Robertson, Mackinlay, and Card 1991).

zooming" and "portal filters." Semantic zooming can be used to control the amounts or types of information to be displayed at certain scales. Portal filters control how information is displayed (e.g. in textual, tabular, chart or graphical format).

The Pad++ system (Bederson and Hollan 1994) is a successor of Pad. It uses a multiscale view to present the user with an overview of a selected set of objects. The objects are history-enriched and stored along with the interaction events that comprise their uses. The display of a "history-enriched object" shows a graphical abstraction of the accrued histories as part of the object itself.

Lieberman (1994) demonstrates a *macroscope* technique that is very similar to the semantic zooming in Pad. However, his technique utilizes a multi-layer translucent display that allows users to receive new information after zooming while keeping track of the surrounding context. These translucent layers are analogous to yellow tracing papers used by architects.

Starfield displays

Starfield displays introduced by Ahlberg and Shneiderman (1994) are 2D scatterplots of a multidimensional database, where every item from the database is represented as a small colored glyph whose position is determined by its ranking along ordinal attributes laid out on multiple axes. These displays use a common approach to visualize high-dimensional objects (i.e. objects with more than three dimensions of interest) by presenting the two primary dimensions in 2D, and then supplying visual cues (such as colors, patterns, and texts) to distinguish additional dimensions.

Tioga

Woodruff et al. (1994) describe a system that allows flight-simulator navigation through a multidimensional data space, and incorporates *wormholes* to allow tunneling between different multidimensional spaces. Wormholes are similar to hyperlinks in a hypertext system. The flight-simulator navigation provides panning and zooming functions. Unlike the normal zooming function (which enlarges a certain portion of the display), the zooming in Tioga shows enhanced detail. Users are allowed to control the range and the elevation (depth) of the zoom window. In addition, Woodruff et al. discuss the handling of multiple browsers to support multiple views.

Worlds within worlds

Feiner and Beshers (1990) propose a worlds-within-worlds metaphor that introduces infinite entries into different dimensions. This metaphor reduces the complexity of a multidimensional space by holding one or more of its independent dimensions constant to decrease the number of dimension to three and then embeds in this 3D world another 3D world that represents three additional dimensions. The position of the embedded world's origin relative to the containing world's coordinate system specifies the value of three of the inner world's variables that are held constant. This process can then be repeated by further recursive nesting of heterogeneous worlds to represent the remaining dimensions. Example applications are demonstrated in the financial visualization domain. This project provides an excellent example of a cyberspace and its unfolding described in Section 2.2.3 of this chapter.

2.2.5 Conclusion

Although information navigation has been a research focus for the past few years, little formal studies have been conducted to examine to what extent navigational behavior in computing environments is affected by human spatial cognition. Among the literature that I have reviewed, only three systems, LEADS reviewed in Section 2.2.4 (Ingram and Benford 1995a, 1995b), HP UVE Help and Access HP (Nabkel and Shafrir 1995) do take account of cognitive maps and environmental design principles. Unfortunately, none of them have provided evidence to support the researchers' assumptions. A recent study on wayfinding in virtual environments by Darken (1996) demonstrates that "real-world wayfinding and environmental design principles are effective in designing virtual worlds which support skilled wayfinding behavior." However, the virtual worlds investigated by Darken (1996) are, unlike the visualizations of most information systems, abstract representations of the physical world in a Virtual Reality (VR) environment, i.e. virtual islands in the virtual ocean; this may confine his results. Nevertheless, his research is a first step in the area of information navigation (Darken 1996). Further studies are necessary to validate such claims because of obvious differences between physical and information spaces (see Section 2.3 below).

The research efforts described above pay more attention to elements on cognitive maps and less on organization structures. Through the reviewed human spatial tasks literature, an interesting concept emerges, which I call levels of abstraction; that is, people may develop multiple cognitive maps (or multiple layers in a cognitive map) each of which represents the same space in a different level of details or based on a different point of view. These cognitive maps (or cognitive map layers) are what Lynch (1960) calls different images, arranged in a series of levels, of a city. In Tversky's (1993) term, they are cognitive collages that "are thematic overlays of multimedia from different points of view" (Tversky 1993, p. 15). Interestingly, this concept has been used in some GIS visualization designs (e.g. Lieberman 1994).

2.3 Physical Space vs. Information Space

In this section, I will discuss the differences and similarities between navigation tasks in physical and information space. The purpose is to examine whether cognitive map theories and environmental design principles are applicable to the design of information spaces.

2.3.1 Differences

Differences between the physical space and information space can be grouped into three areas: contents, structure, and human-space relation.

- **Physical space provides a wide variety of cues; information space has limited variety of cues**.
  The contents of a space constitute the primary distinction between the two types of spaces. In the physical space, people use all their senses (sight, touch, taste, smell, hearing, and so on) so they can remember a café because of the aroma of roasted coffee beans, the art deco style interior design, the live acoustic music performance on every Thursday evening, or the warm sunlight at a corner of the store in a winter day. In the information space, stimuli are usually restricted to visual and auditory ones.

For human sensory input, the physical space is a rich environment because it provides many stimuli. The information space, on the other hand, is a rather homogeneous environment in the same respect. The richness of the physical space enables people to select different landmarks based on their individual navigation needs. In the homogeneous information space, where most objects are usually presented similar to one another, it is difficult to differentiate one thing from another, let alone to identify landmarks. In short, the contents of the information space hinder users' construction of cognitive maps. Or in Lynch's (1960) term, the information space has low legibility.

- **Structure of the physical space is stable; structure of the information space is dynamic**.
  The structure of a space is determined by all objects in the space and all relationships between any two objects. The structure of the physical space changes very little and slowly when considered for our daily navigation needs. People usually don't worry about what the world will look like tomorrow when they go to bed tonight. In the physical space, landmarks, paths and other elements in people's cognitive maps are likely to stay for at least several years. Occasionally, people may realize a shoe store has changed to a book store, or an old building has been torn down. Nevertheless, this kind of surprise rarely happens because most changes are gradual. Devlin (1976) found subjects' cognitive maps evolve as they become acquainted to a new environment. Lynch (1960) observed that people construct their cognitive map of the environment at different levels, for example, from the street level, or at the neighborhood level. One basic criterion for these phenomena is that the environments are stable.

  On the other hand, the structure of the information space changes constantly. Take the WWW for instance: every day, new web pages are added, old ones removed, or have their contents modified. Due to the dynamic nature of the information space, users who are given enough cues to allow them to build up their first cognitive map, may still have difficulties maintaining it. Consequently, the information space has low imageability (Lynch 1960).

- **People navigate "inside" the physical space; but users navigate "outside" the information space.**
  In the physical space, most people navigate at street level, that is, they are submerged in the environment; their navigation relies mainly on spatial relations (in landmark and procedural knowledge), such as next to, to the right, and so on, in reference to nearby landmarks and roads rather than the survey knowledge, such as 2 miles east of CMU (see Section 2.1.1). Furthermore, observations have shown that even the navigation at the same street level, cognitive map of an automobile rider may vary drastically from that of a pedestrian (Downs 1985).

  In the information space, users usually navigate at the overview (or bird's-eye) level, where the structure of the space is present. Consequently, this may encourage users to use their survey knowledge. Navigating at the overview level, users may feel the power of being able to control the environment, which is almost not available in the physical space navigation; however, they may not have the strong sense of direct engagement with the environment, which is common in the physical space navigation. Information space navigation may allow users to navigate at the document level, such as moving through web pages, which seems equivalent to the

street level in the physical space. However, "lost in the hyperspace" has been an issue in this type of information navigation (Edwards and Hardman 1989; Thüring, Haake, and Hannemann 1991; Bederson and Hollan 1994; Dömel 1994; Rivlin et al. 1994; Mukherjea and Foley 1995).

2.3.2 Similarities

There are also similarities between navigation in the physical space and information space: use of sequential wayfinding strategies, visual references, and levels of abstraction.

- **Sequential way-finding strategy**
  A sequential wayfinding strategy memorizes and recalls a sequence of movements; it is a rudimentary human spatial ability (Brown 1932; Hart and Berzok 1982; Somerville and Haake 1985). People remember the path they take to reach a destination, such as the daily route to school or work. Similarly, when navigating in WWW, people remember sequences of locations to reach a certain web-site or utilize history tools provided by WWW browsers.

- **Visual references**
  Prominent visual references in physical spaces are landmarks. Researchers (Devlin 1976; Lynch 1960; Hart and Berzok 1982; Gärling et al. 1985; Golbeck 1985; Mandler 1988) have shown that people use all kinds of landmarks. As mentioned in the previous section, landmarks are important in cognitive maps and serve as references to support the recall of a path to some place. Although landmarks are difficult to identify in information spaces, people usually extract visual references (such as special icons) from the information and use these references later to verify if they are on a correct path. Nabkel and Shafrir (1995) recognize the importance of cognitive maps and utilize the saliency of visual references to design the Access HP web-site.

- **Levels of abstraction**
  Finally, people have different cognitive maps (or cognitive collages, or images), each of which represents the same physical space (city) at a different level of detail or is based on a different viewpoint (Lynch 1960; Tversky 1993). Providing information at different levels of abstraction has long been supported in information spaces, particularly by GIS and database visualization systems.

The similarities between navigation tasks in physical and information spaces suggest that cognitive mapping and environmental design principles may be of use in information space navigation. However, the differences have to be taken into account if one should decide to apply these principles in designing information navigation support.

2.4 Summary

Comparisons between navigation tasks in physical spaces and information spaces show that cognitive mapping theories and environmental design principles may be applicable to the design of navigation supports in the information spaces. However, not all information spaces are created equal. The information spaces mentioned in this chapter can be roughly categorized into two types based on their stability: the ones that have well-defined boundaries (i.e. have a fixed amount of content) and do not change over time such

as hypertext help systems and GIS; and others that do not have clear borderlines because the amount of information they contain changes all the time such as databases and WWW. An information space of the former type is easier to manage because the space is fully "known" in advance so that a definitive map of it can be provided; whereas a space of the latter type may require an imposed structure by the developer of the space, otherwise, navigators of the space may have to construct their own maps through serendipitous discovery. Jul and Furnas (1997) describe another classification based on a structural characteristic: a moderated information space, such as databases and help systems, of which the structure has been coordinated or controlled versus a non-moderated one, such as WWW. These categorizations seem to suggest that information spaces are either "stable and structured/moderated" or "dynamic and unstructured/non-moderated."

The information space within a generative design system may not fit well into the two kinds of categorization described above. Particularly, this space is created and populated by the designer during a design session; i.e. a navigator of the space is also the creator of that space. This is very distinct from the information spaces reviewed in this chapter. An examination into the nature of the information space in generative design systems will be discussed in the following chapter. Nonetheless, supporting navigation in the generative design systems should take account of issues discussed in this chapter.

3. Information Navigation in Generative Design Systems

Imagine a designer who is solving a design problem with the assistance of a generative design system. The system accepts an explicit representation of the requirements to be satisfied, called a *problem*, which the designer can modify interactively and dynamically. The system is able to use these requirements when it assists the designer actively in the generation of solutions satisfying the requirements. The designer first defines the problem (A) in the system. Realizing that the problem is too complicated, she decomposes the problem into five subproblems (A1-A5). She then employs the generative mechanisms of the system to produce solutions to the subproblems and, by combining them in different ways, two alternative solutions (S1 and S2) of the overall problem. At this point, she decides to loosen the constraints on a subproblem (A5') because she is not satisfied with either S1 or S2; this, in turn, automatically leads to an alternative formulation (A') of the top problem (A). Working on the alternative problem, she arrives at another solution (S3). She examines the solutions and decides that she should approach the design problem in yet another way. Thus, she defines a new problem (B), and continues in this manner through an arbitrary number of iterations.

The scenario above describes a design process that is supported by generative design systems. The different problems, subproblems, solutions, and subsolutions created by the designer are illustrated in Figure 3-1, together with the relationships between them. We

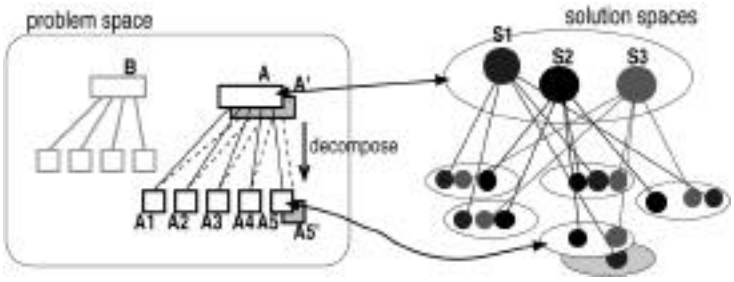

Figure 3-1: Problem space and solution spaces

can easily imagine that, as the design process continues, more solutions and alternative problems will be created in the system. In addition, as the design problems become more complex, the designer may decide to further decompose subproblems into sub-subproblems. At some point during the design process, the designer may want to examine design issues considered in different problem alternatives or to compare different solutions. But how can she find these objects again and with ease?

The similarities between navigation tasks in physical and information spaces (see Section 2.3 in Chapter 2) suggest that cognitive mapping and environmental design principles may be of use in information space navigation. However, the differences have to be taken into account. Differences of contents may be alleviated by visualization techniques, i.e. by additional cues or other methods to allow landmark identifications. The structural differences, however, are not as easy to handle. One approach is to show in some way that there is a coherent structure underlying the space.

Based on the background literature survey, I have made the following assumptions to utilize the navigation metaphor in the context of generative design systems:

1. The cognitive maps theory and environmental design principles are applicable to the design of navigation supports in information spaces.
2. A coherent structure underlying a dynamically changing information space will help navigators to anticipate the kind of changes that will take place or to predict certain behaviors of the environment.
3. Information space visualization should provide visual affordance[1] such that it leads navigators to form an intuitive understanding of the structure and behavior of the space.

Given these assumptions, I have developed an approach to support information navigation in generative design systems. This approach first examines the nature of the information space within a generative design system. The scenario presented in the beginning of this chapter has shown that this space changes constantly, and that the designer not only moves between objects in the space but also creates those objects. Therefore, the approach aims at identifying a model of this ever-changing space in an effort to provide a coherent structure underlying the space. Second, given such a model describing the structure of the space and the kinds of objects within it, a set of operations can be devised to support navigational activities. Finally, since different generative design systems may provide different styles of user interaction, some principles for designing navigation utilities are provided to guide system developers in creating navigation utilities that are tailored to the needs of individual generative design systems. This is in contrast to an approach that would provide one navigation utility to fit all generative design systems.

1. The notion of *affordance* in this dissertation follows the concept described by Norman (1990). It refers to the perceived and actual properties of an object. Affordance provides strong clues to how an object could be used.

3.1 Model of the Design Space

Recall the brief description of generative design systems in Chapter 1. In the research literature related to generative design systems, the set of all possible problem definitions and decompositions of a particular design problem is referred to as the *problem space*, and the set of all possible solutions to a particular problem as the *design space*. Both concepts—problem space and design space—originate from the information processing theory by Newell and Simon (1972), in which a problem-solving activity is considered as the act of searching through the state space of a well-defined problem. Solving a design problem, therefore, is the process of exploring the space of designs, or the design space.

However, the space of designs, from the notion described previously, contains only designs (solutions) of a particular problem definition. Recall the design scenario described in the beginning of this chapter: during a design session, a designer may define a design problem in many different ways. Therefore, the complete space of designs of a design problem should include all design spaces of different definitions of the design problem. Moreover, if a problem is decomposed into many subproblems (also as described in the scenario), the complete space of designs should include all design spaces of different subproblems. To be able to better represent and relate different pieces of information a designer produces when solving a design problem (not just for a particular problem definition), i.e. to be able to describe the information space in a generative design system, I redefine the concept of design space as the information space of generative design systems.

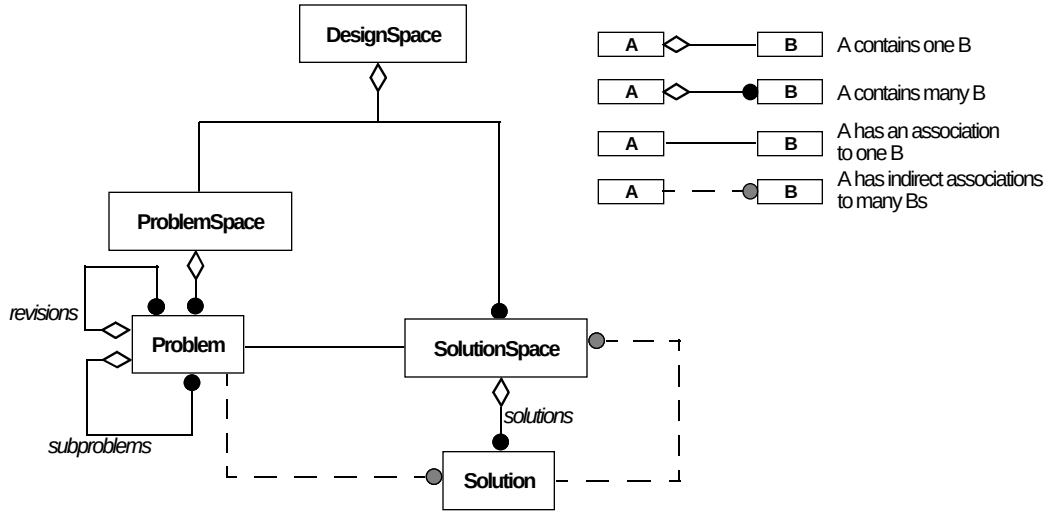

Figure 3-2: Design space object relationship model

3.1.1 Dimensions

The new concept of *design space* in generative design systems is shown in Figure 3-2. A *problem* is a set of design requirements and other specifications that describe desired properties of an artifact to be designed. As we have seen in Figure 3-1, a problem evolves dynamically during the course of a design project as a designer iterates through problem formulation/solution cycles. The set of all possible problem formulations that a designer can define in the context of a given design project constitutes the *problem space* of this project. The designer can decompose a problem into subproblems or revise a problem as needed. The decomposition of a problem forms a *problem hierarchy*, and revisions of a problem constitute a *revision history*. Therefore, the position of a problem in a problem space can be uniquely identified through its positions in the associated problem hierarchy and revision history. Let's denote a problem by $P_{p,q,r}$, where p is the superproblem of which P is a direct subproblem, and q and r are indices given the position of P in a problem hierarchy and revision history, respectively: i.e. P is the r^{th} revision of the q^{th} subproblem of p. A *problem space* is thus the set

$\{ P_{p,q,r} \}$,
where: p iterates through all superproblems,
q iterates through all subproblems of p, and
r iterates through all revisions of the q^{th} subproblem of p.

If a problem is the root of a problem hierarchy, the index p is NULL. This is a special case. Index q identifies a position among sibling problems that are at the same problem decomposition level. It is never NULL because a problem is always located at a valid decomposition level, be it the root level, the leaf level or any level in-between. In addition, a problem is a revision of itself; therefore, the index r will never be NULL.

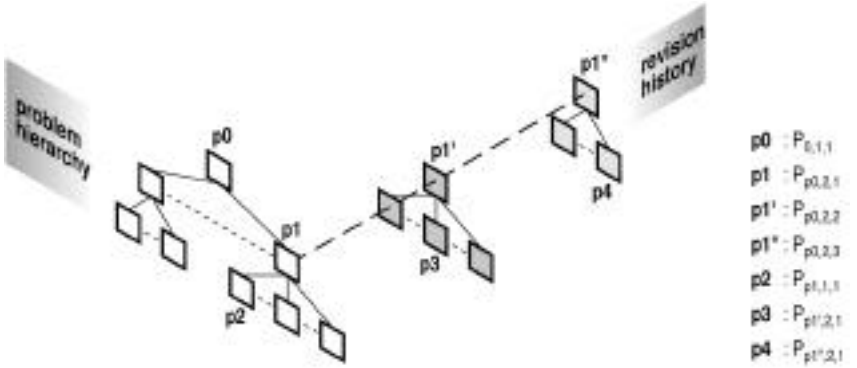

Figure 3-3: An example of problem space
(indices are ordered from left to right)

A *solution space* is a collection of all solutions (partial or complete) to a given problem generated by the designer. In a generative design system, solutions are typically generated

or derived from other solutions through mechanisms provided by the system. This derivational relationship is hierarchical and called a *solution hierarchy*; for instance, it can establish the order of rule applications in shape grammar systems. The position of a solution S in a design space can be identified through its associated problem and position in a solution hierarchy: $S_{(Pp,q,r),s,t}$, where $P_{p,q,r}$ is a problem, s is the parent solution and t gives the position of S in a solution hierarchy. Thus, a solution space for a problem is the set

$\{ S_{(Pp,q,r),s,t} \}$,
where: $P_{p,q,r}$ identifies a version of particular problem at a particular level,
s iterates through all parent solutions, and
t iterates through all derived solutions of s.

A *design state* in the design space represents a partial or complete design solution of a design problem. A *design space* is the set of all problems, subproblems (given by their positions relative to the problem hierarchy), problem revisions (given by their positions in a revision history of a subproblem), and solutions (given by a deriving solution and a position relative to the solution hierarchy).

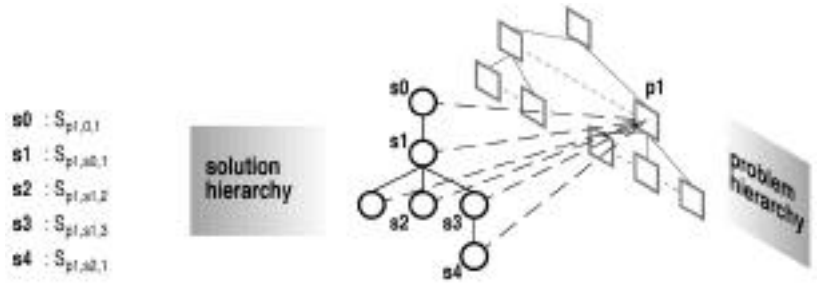

Figure 3-4: An example of solution space
(indices are ordered from left to right)

Taken together, the five indices p, q, r, s, and t establish a structure that is independent of a specific design space. The structure remains stable and is independent of the way in which the space grows: objects can be added along any dimension an arbitrary number of times, but these additions always happen along exactly one of the *dimensions* p, q, r, s, and t. The literature survey (see Chapter 2) reveals that information spaces lack legibility or imageability; and one of Lynch's principles of city design is to improve this quality by providing a clear structure or hierarchy of the spaces in a city (Lynch 1960). Although a design space changes constantly during a design session, the structure established by these five dimensions allows designers to understand how a design space grows and evolves along these predictable dimensions.

It is important to note that these dimensions are not analogous to the dimensions or axes of a Cartesian coordinate space. In design spaces, dimensions can vary only relative to the current value of the preceding dimensions, which puts significant restrictions on the

navigation operations available to designers. On the other hand, these local variables provide a principled way of navigating the space, while being able to keep track of additions or deletions of objects as the design progresses. This dependence between dimensions makes the present model very different from the multidimensional data space described in the A4 system (Hovestadt 1993; see Section 2.2.4 of Chapter 2 for a brief review of this system).

3.1.2 Visualization

An object in the design space is called a *node,* which can be a problem space, problem, solution space, or solution. A node can be uniquely identified in the design space through the dimensions described above. These dimensions are considered *extrinsic* attributes of a node; they can be used, in principle, to identify or visualize its position in the design space. All other attributes (aside from the navigational attributes to be described later) are regarded *intrinsic*. Intrinsic attributes are inherent to a node; they vary depending on the generative design system. Therefore, the visualization of a design space along the five dimensions concerns the extrinsic attributes only. The visualization of intrinsic attributes is best handled by individual generative design systems.

A node can be expanded to display other extrinsic dimensions (see Figure 3-5). This is similar to the cyberspace unfolding concept proposed by Benedikt (1991). Yet unlike cyberspace unfolding, there is no mapping to turn intrinsic attributes into extrinsic ones because expansion is performed only through an extrinsic dimension in the design space.

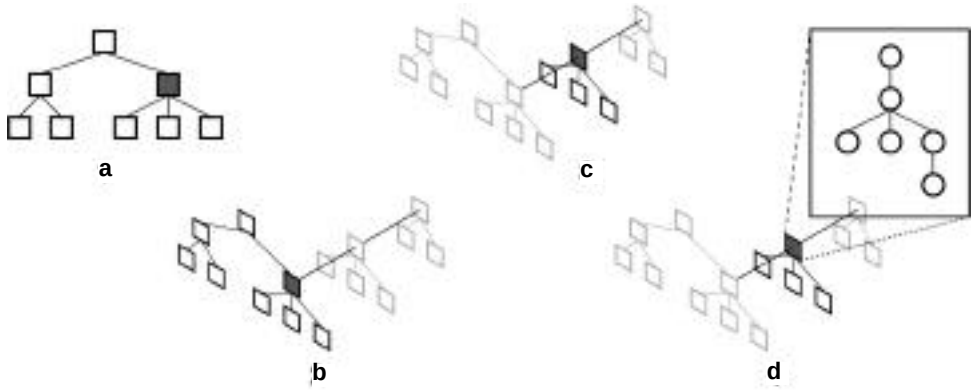

Figure 3-5: An example of node unfolding in a design space
(a) depicts a node (in dark shade) within a problem hierarchy. (b) shows a revision history dimension expanded. (c) shows a move along the revision history dimension. (d) illustrates an expanded solution hierarchy dimension.

A relationship between two nodes is represented as a *link*. Links are not expandable. Both nodes and links may carry navigational attributes, which are extrinsic. The navigational attributes hold information that provides additional visual cues, such as landmarks and paths, to support navigation.

3.1.3 Solution Composition

As illustrated in the scenario at the beginning of this chapter, a designer may decompose a complex problem into more manageable subproblems; thus, a final solution of this complex problem has to be composed from the subsolutions (i.e. solutions of the subproblems). In some generative design systems, the final solution composition may be obtained naturally without additional efforts, while in others, the designer may have to explicitly compose the final solutions from the subsolutions. This is due to the two different problem-solving approaches—top-down and bottom-up—a generative design system may support to handle hierarchically decomposed problems.

A generative design system that applies a strict top-down problem-solving approach requires a designer to solve a problem starting from the most abstract level (top level), then to proceed to subproblems at a lesser abstract level (lower level). Usually, the solutions of an upper level problem supply some contextual information that is needed by the subproblems of that problem in order to continue the exploration of subsolutions. A super/sub-solution relationship is established automatically between certain solution pairs during the design generation process because the subsolution cannot exist without its supersolution. A supersolution may have many alternative subsolutions. In a sense, the supersolution constrains the space of its subsolutions (or subsolution space). Since the supersolution itself is part of the solution space that contains it, we may imagine that a subsolution space is embedded within its supersolution space at the location of the supersolution. Figure 3-6 illustrates this super/sub-solution relationship and embedded

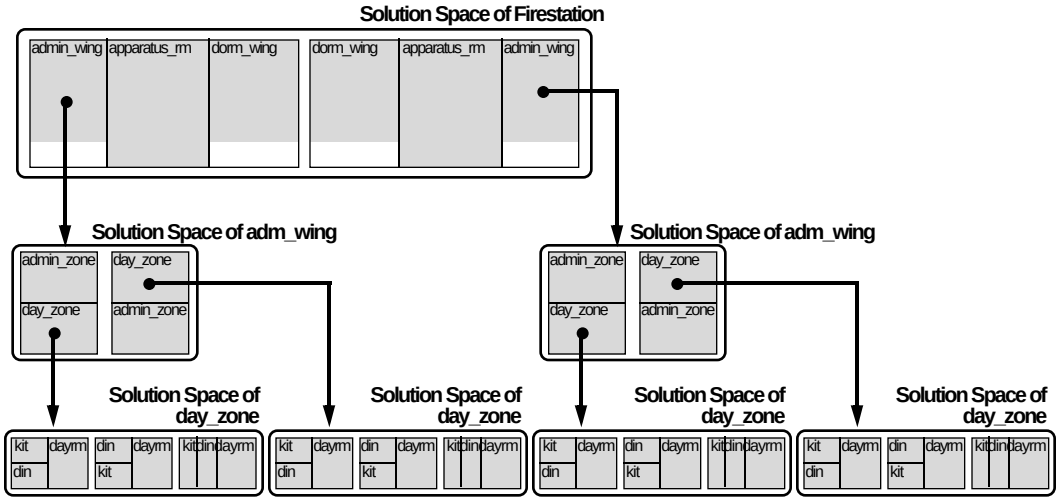

Figure 3-6: Super/sub-solution relationships in a strict top-down design approach
Layouts within each solution space are alternative solutions. Although some solution spaces may appear the same, such as the 2 solution spaces of adm_wing and the 4 of day_zone, they are individual spaces with different contextual information provided by different supersolutions. Each subsolution space can be viewed as embedded in its supersolution space at the location (identified by a black dot) of its supersolution.

solution space concept. The figure depicts solutions (layouts) of a Firestation layout problem generated by a designer using a system for schematic layout design, i.e. SEED-Layout (Flemming and Chien 1995). This Firestation problem is composed of three subproblems adm_wing, apparatus_rm and dorm_wing; the adm_wing, in turn, is composed of adm_zone and day_zone subproblems; and finally day_zone is decomposed into kit, din, and dayrm. In this example, the embedding of a subproblem corresponds with the geometric containment of a subsolution in a part of the higher-level solution. But this geometric relation may not hold in other generative design system: it is a typical feature of SEED-Layout (and possibly other spatial layout systems).

Generative design systems that support a bottom-up approach or a mixture of top-down and bottom-up approach, on the other hand, do not always have the necessary information to create super/sub-solution relationships automatically. Therefore, designers may have to manually identify this relationship between two solutions. Moreover, using a generative design system that applies strict bottom-up approach, a designer may only need to solve problems at the lowest level and devote most of the efforts in composing various alternatives of a final solution. In this approach, a supersolution is created based on a composition of its subsolutions; a different composition of subsolutions should form another supersolution. Figure 3-7 illustrates schematically the differences between these two approaches in arriving at a final solution composition.

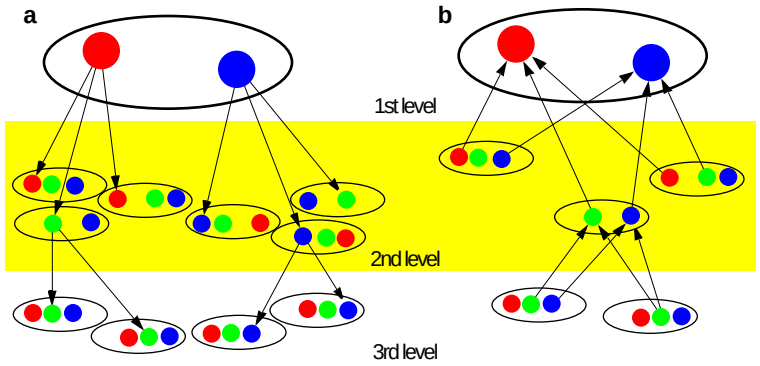

Figure 3-7: Solution composition through strict top-down (a) and bottom-up (b) design approaches Solid circles represent solutions, and ellipses depict solution spaces.

The super/sub-solution relationship is the kind of link that connects across two solution spaces—a solution in one solution space and its subsolution in another solution space. Borrowing a concept from modern Physics, this link is a *wormhole* connecting points that are separated in two parallel spaces. As discussed above, the creation of this relationship can be automatic or manual depending on individual generative design systems. Consequently, it is better to leave the management of super/sub-solution relationships to individual systems. Therefore, the general model of design space presented here does not

incorporate this type of relationship. Nonetheless, it is believed, and will be demonstrated in later chapters, that the model provides great flexibility to allow additional types of relationship, such as the super/sub-solution relationship, to be imposed upon its existing structure.

3.2 Basic Navigation Operations

Definitions of navigation have been briefly discussed in Chapter 1. Recently, the topic of information navigation has gained great interest in the HCI community.[2] Although using the navigation metaphor in information spaces such as databases or hypertext systems has been discussed and demonstrated for over a decade, only recently have researchers started to examine the applicability of this metaphor. As the first step, researchers are pursuing the definition of navigation in information spaces. A number of leading researchers in the field have presented their definitions of navigation, which are heavily influenced by the navigation activities in the physical environments; e.g. navigation is "moving oneself sequentially around an environment, deciding at each step where to go next based on the task and the parts of the environment seen so far" (Jul and Furnas 1997). Furthermore, Furnas (see Jul and Furnas 1997) have distinguished navigation from *querying*—"submitting a description for the object being sought to a search engine which will return relevant content or information." However, given that this research is interested in adopting the navigation metaphor only to help designers accessing desired information easily in generative design systems, navigation in these systems is considered to be a "information seeking" activity; querying is therefore one of many tools available to conduct this activity.

Navigation aids for seeking information in the design space should, at a minimum, show designers at any time the location of an active node in terms of its dimensions, and aid them in revisiting nodes that have been generated before through sequential movements as well as through querying mechanisms. To achieve these two goals, five essential and basic operations are needed:

- **Show location of node.**
  A node in the design space can be identified through up to five dimensions. Showing its location means displaying the node as well as its surroundings along the relevant dimensions. The designer should be able to control the extent of the surroundings to be displayed.
- **Mark node.**
  This operation allows designers to identify a specific node and tag it with a marker. The marker is a shortcut to identifying the location of a node. In addition, it adds a visual attribute to the node, which makes it visually distinct from unmarked nodes. This operation is envisioned to provide similar functions to those provided by landmarks in physical spaces.

2. The fact that the prestigious ACM SIG-CHI annual conference had organized a workshop on "Navigation in Electronic Worlds" in 1997 (one year before this writing) signifies the importance of this research topic.

- **Go to node.**
  Designers can move to a node through a representation of the node displayed on the screen, or by other means provided by the user interface to the design space.
- **Step.**
  This operation allows designers to move forward and backward from a node along any one dimension at a time. For example, a step backward along the p dimension brings the new location to the superproblem of the current node and a step forward along the t dimension brings the location to the next sibling solution of the current node.
- **Find.**
  This operation allows designers to specify search criteria and find nodes that satisfy those criteria.

3.3 Principles in Designing Navigation Support

Since generative design systems may provide different styles of user interaction and since the design tasks supported by these systems may vary, it is naive to assume that one can provide a navigation utility that will work for all generative design systems. The model of design space and basic navigation operations described previously offer a framework to support navigation tasks in generative design systems. Yet the primary task of system developers is to decide how to present or visualize the design space and what kind of interactions designers may use to effectively navigate the space; these issues are not directly addressed by this framework. Nonetheless, principles exist for designing navigation utilities, and can guide system developers in creating navigation utilities that are tailored to the needs of individual generative design systems. Key concepts in developing the principles are landmarks and the cognitive maps to facilitate the understanding of an information space.

The background studies in Chapter 2 have shown that landmarks are the essential spatial knowledge in navigation tasks. Although still under investigation, some empirical evidence (e.g. Watts 1994) has become available to demonstrate that landmarks are important cues for aiding navigation in information spaces as well. Furthermore, studies (Devlin 1976; Watts 1994) have also shown that landmarks identified by individual navigators are preferred and more effective in both physical and information spaces. In addition, cognitive maps of different levels of abstraction (e.g. Lynch 1960) are important in supporting navigation in the design space. This idea provides an intuitive approach to manage large information spaces, which a design space can potentially become, and to alleviate a designer's cognitive loads.

With these concerns in mind, I propose four principles for designing navigation support in generative design systems.

- **Provide supports that allow designers to create their own landmarks.**
  The functionality and importance of landmarks in cognitive maps are described in the previous chapter. A design space navigation tool should at least allow designers to identify their own landmarks. If the tool supports collaborative environment, designers may share landmarks as well as keep personal ones.

- **Provide consistent navigation schemes.**
  The navigation mechanism should be consistent so that designers know where they are and what to expect next at any time. This is important because the content of a design space is constantly changing, and it is almost impossible for designers to form a stable cognitive map. But designers can learn to expect changes. For example, they can learn the generation operations provided in a generative design system and how these operations affect the structure of the design space in this system. With enough practice, designers can anticipate what will happen when a generation operation is issued. Moreover, designers are less likely to feel "lost" when they know what may happen (Edwards and Hardman 1989). In addition, the system should always clearly identify designers' current location in a design space. If more than one navigation level is supported (e.g. bird's-eye/overview level and document level), the same operation should lead to the same result regardless of the navigation level designers are at.
- **Support information display at different levels of abstraction**
  Studies of navigation both in physical and information spaces show the use of cognitive maps at different levels of abstraction. The navigation tool should provide possibility of displaying different views of a set of information through filtering, and should allow designers to identify and store different abstraction settings. This allows designers to control associations from their cognitive maps to abstraction views.
- **Provide search or history facility to reduce designers' cognitive load**
  To minimize designers' memory burden, the navigation tool can provide a history facility to record successful navigation paths. This facilitates a sequential wayfinding strategy and enables designers to retrace a path easily or to go to a desired destination quickly without having to recall information from their long-term memory. In addition, although search (such as database search) is not available in physical space navigation, such a facility can provide a new method for wayfinding in a design space with very minimal memory burden on designers.

4. Design Space Navigation Framework

I have presented, in Chapter 3, a model of the design space in generative design systems and five types of basic navigation operations that sufficiently support information navigation in that space. This chapter documents a software framework—*design space navigation framework*—that encapsulates the model of design space and facilitates the basic navigation operations. The design space navigation framework is a reusable software mini-architecture that provides the infrastructure and mechanisms for supporting information navigation in generative design systems.

The framework is implemented using object-oriented technologies. In order to discuss the design of this framework and its classes without having to describe tedious implementation details (for these details, see Appendix A), I explain the framework through well-known software design patterns. For readers who are not familiar with *software frameworks* and *design patterns*, I first give a summary of these concepts before diving into the discussions about the design space navigation framework. Additionally, I demonstrate the use of the framework in a generative design system and discuss advantages of the framework at the end of this chapter.

4.1 Software Frameworks

A software framework is a reusable software package that provides generic structures and behaviors for a family of software abstractions to be used within a specific domain. If we use the object-oriented terminology, a software framework is a set of cooperating classes (abstract and concrete) that makes up a reusable design for a specific class of software (Gamma et al. 1995, pp. 26-28). A framework captures the design decisions that are common to its application domain. The framework accomplishes this by partitioning the common design into abstract classes and defining their responsibilities and collaborations. These abstract classes are *hot spots* (Pree 1995) or plug-points of the framework; they enable the framework to be adapted and extended to fit varying needs.

Frameworks emphasize design reuse over code reuse. For instance, when using a conventional class/subroutine library, a developer writes the main body of the application and calls the required library code. Conversely, when using a framework, the developer reuses the main body from the framework and writes the code it calls. The key difference between a conventional class library and a framework is that a class library provides a collection of objects that developers must assemble themselves; whereas a framework consists of pre-assembled objects with communications clearly defined.

Applications utilizing software frameworks are easier to build because certain design decisions have been made and encoded in the frameworks. Furthermore, applications using the same framework are easier to maintain and may seem more consistent to their users.

4.2 Software Design Patterns

Pree (1995, p. 61) characterizes software design patterns as "a set of rules describing how to accomplish certain tasks in the realm of software development." More specifically, Riehle and Züllinghoven (1995) define a software design pattern to be "a pattern whose form is described by means of software design constructs, for example objects, classes, inheritance, aggregation and use-relationship."[1] This concept was made popular in the software community by Gamma et al. (1995) as a means to communicate experience in software design and to record the experience in a form that people can use effectively.

There are several ways to document software design patterns (e.g. Gamma et al. 1995; Coad, North, and Mayfield 1995); all of them have evolved from Christopher Alexander and his colleagues' pattern language (Alexander et al. 1977). In general, a pattern has four essential elements:

- a pattern name for identification;
- a problem to describe when to apply the pattern;
- a solution to describe the elements that make up the design, their relationships, responsibilities and collaborations; and
- consequences to show the results and trade-offs of applying the pattern.

Appleton (1997) states "the goal of patterns within the software community is to create a body of literature to help software developers resolve common difficult problems encountered throughout all software engineering and development." Patterns have gradually become a shared language for communicating insight and experience about these problems and their solutions. For complex software frameworks, the design pattern approach helps to describe the design of these frameworks and their individual classes without revealing implementation details.

4.3 Design Space Navigation Framework

The design space navigation framework presented here is based on a reusable software mini-architecture that provides the infrastructure and mechanisms for supporting information navigation in generative design systems. It encapsulates the model of design space and facilitates basic navigation operations presented in Chapter 3. There are two main design goals in developing the framework. One is to identify the hot spots so that it is flexible enough to be adapted to various generative design systems. The other is to

1. Before they give this definition, Riehle and Züllinghoven (1995) qualify the term "pattern" as "the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts", and describe the "form" of a pattern as consisting of "a finite number of visible and distinguishable components and their relationships."

provide a simple interface to facilitate all sorts of navigation operations. These goals are accomplished in the implementation described in this section.

I first explain the design of the design space navigation framework based on the software design patterns employed in the framework. I then describe the facilities provided by the framework.

4.3.1 Framework Design

The design space navigation framework is designed using object-oriented technologies. It encapsulates key concepts identified in the model of design space in terms of the following classes: NAV_DesignSpace (for design space), NAV_ProblemSpace (for problem space), NAV_Problem (for problem), NAV_SolutionSpace (for solution space), and NAV_Solution (for solution).[2] The organization of these classes in the framework is shown in Figure 4-1. Additionally, Figure 4-1 also shows the NAV_Marker class designed to handle

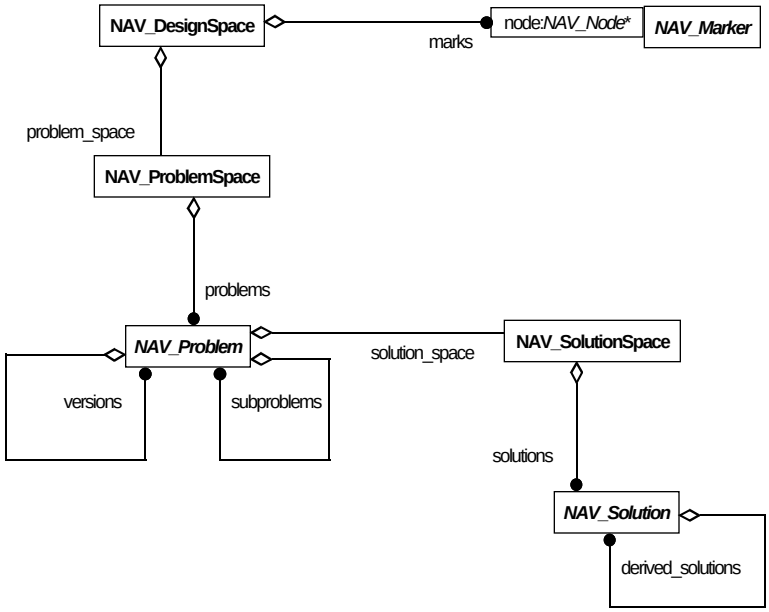

Figure 4-1: Design space navigation framework classes

markers needed for some navigation operations. NAV_Marker, NAV_Problem and NAV_Solution are the hot spots of the framework and are implemented as abstract classes (shown italicized in Figure 4-1) because NAV_Problem and NAV_Solution need to be customized according to the representations of problems and solutions in a specific

2. All classes in the framework are prefixed with "NAV_" to prevent naming conflicts, which may occur when adapting it into generative design systems.

generative design system and because the generative design system may implement many kinds of markers by specializing NAV_Marker.

The overall structure of the design space navigation framework conforms to the 1:N Connection metapattern described in Pree (1995, ch. 4). This metapattern describes the composition of template and hook classes and their corresponding objects in a software framework. A template class contains only the template methods, which define abstract behavior, generic flow of control, or generic relationship between objects. A hook class implements certain aspects of its corresponding template class; it contains hook methods, which are usually the hot spots of a software framework and are elementary components of template methods. In the 1:N Connection metapattern, an object of a template class refers to any number of objects of its hook class. The flexibility of changing objects of the hook class dynamically is the main advantage of this metapattern.

The design space navigation framework contains three hook classes, namely: NAV_Marker, NAV_Problem and NAV_Solution. The primary template class, NAV_DesignSpace, maintains (directly or indirectly) one-to-many reference relationships with these three hook classes. There is no inheritance relationship between the template class and hook classes. This metapattern provides a structure that enables template methods (e.g. NAV_DesignSpace::FindNodes and NAV_DesignSpace::GoToMarker) to iterate over the hook class objects and send certain messages to each of these objects. In addition, the framework offers methods to manage objects of the hook classes, such as methods to add (e.g. NAV_DesignSpace::AddChildProblem and NAV_DesignSpace::MarkNode) and remove (e.g. NAV_DesignSpace::DeleteProblem and NAV_DesignSpace::UnmarkNode) objects from reference lists and methods to gain access (e.g. NAV_DesignSpace::GetMarkers) to each of the grouped hook objects. Figure 4-2 illustrates, by means of an example, how the components of the metapattern correspond to the classes and methods in the design space navigation framework.

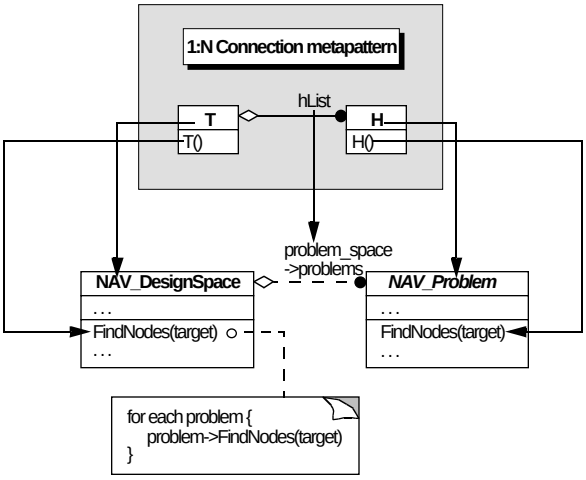

Figure 4-2: 1:N Connection metapattern in the design space navigation framework

The NAV_Marker class encapsulates the structure and behavior of markers. It is designed based on the Bridge pattern[3] described in Gamma et al. (1995, pp. 151-161). NAV_Marker defines the interface to an abstraction that can have many possible implementations. Different implementations are carried out by different concrete subclasses.

The design of NAV_Problem and NAV_Solution classes aims at supporting, respectively, hierarchical structures of problem decomposition and revision hierarchies; and solution derivation hierarchies. The 1:N Recursive Unification metapattern (Pree 1995, ch. 4) is most suitable for this goal. In this metapattern, template methods and hook methods are unified in one class; it allows the construction of a directed tree hierarchy. Particularly, this metapattern allows objects forming the hierarchy to be treated uniformly. For example, in the process of composing a problem revision hierarchy one does not have to decide whether a problem object will be a leaf node (not to be revised any further) or a subroot (to be revised to create new versions of problems). Figure 4-3 illustrates, as an example, how the components of the metapattern correspond to the NAV_Problem class and its method.

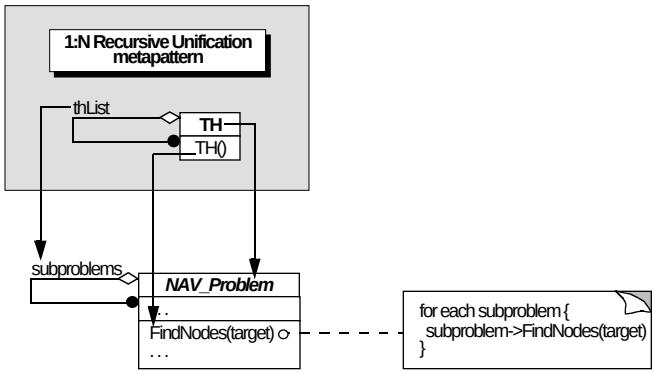

Figure 4-3: Problem composition based on the 1:N Recursive Unification metapattern

Moreover, the NAV_Problem and NAV_Solution classes are designed with the Adapter[4] or Bridge pattern (Gamma et al. 1995, pp. 139-161) in mind so that they can be easily plugged into different generative design systems. Since a generative design system (e.g. SEED-Layout[5] or SEED-Pro[6]) may represent problems (e.g. LayoutProblem in SEED-Layout or Project in SEED-Pro) and solutions (e.g. Layout in SEED-Layout or FunctionUnit Hierarchy in SEED-Pro) differently from the ones in the framework, adapters or bridges[7] are necessary to let different representations of problem or different representations of

3. Bridge pattern enables the decoupling of an abstraction from its implementation so that the two can vary independently.
4. Adapter pattern, also know as wrapper, lets classes that have incompatible interfaces work together by converting the interface of a class into another interface.
5. Refer to Flemming and Chien (1995).
6. Refer to Akın et al. (1995).

solution work together. In a later section, I will demonstrate how to adapt the design space navigation framework to a generative design system (or how to bridge the two) to enhance the navigation support of the generative design system.

To make the framework easy to use, the Facade pattern[8] (Gamma et al. 1995, pp. 185-193) is used to provide a single, simplified interface to the more general facilities of the framework. The facade of the design space navigation framework is encapsulated in the NAV_DesignSpace class. All supports of the framework, such as the organization of objects in a design space and navigation operations, are provided through this class. The complexity of NAV_ProblemSpace and NAV_SolutionSpace is thus hidden from users (i.e. programmers who will adapt the framework to generative design systems). Internally, the framework is decomposed into layers of subsystems. The Facade pattern is again used to define entry points into subsystems. Figure 4-4 shows the facade objects (in light-gray background) in the design space navigation framework and its subsystems, and illustrates how methods are delegated to appropriate objects or subsystems through the facades.

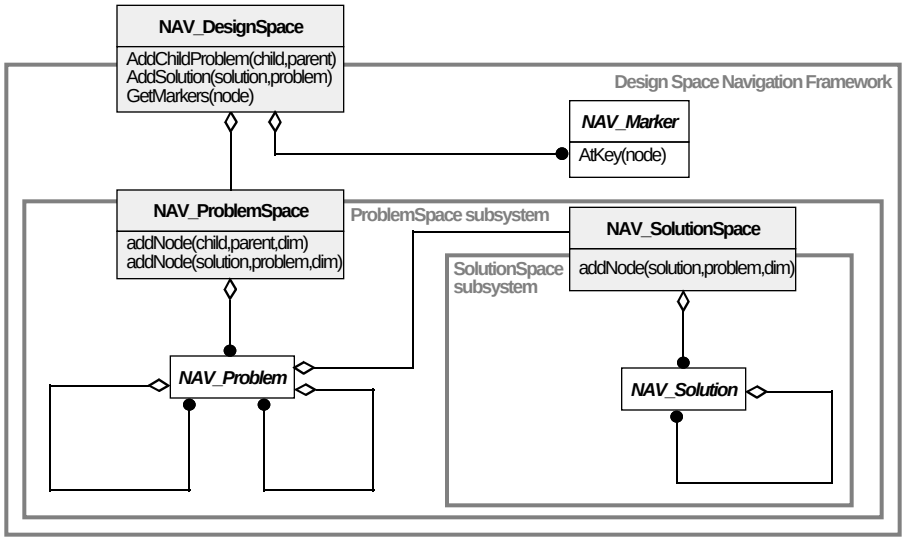

Figure 4-4: Layers of facade in the design space navigation framework

7. I use the Adapter pattern and the Bridge pattern interchangeably here although there is a clear distinction between the two patterns according to Gamma et al. (1995, p. 161). However, I regard these two patterns as the two sides of a coin: from the generative design system's perspective, one uses the Adapter pattern to incorporate the design space navigation framework; yet from the framework's standpoint, one designs the bridge classes to let abstractions and implementations vary independently.
8. Facade pattern aims at providing a unified interface to a set of interfaces in a subsystem. It defines a high-level interface that makes the subsystem easier to use.

The design space navigation framework is implemented utilizing basic building blocks in the ET++ framework (Weinand, Gamma, and Marty 1989). The ET++ framework provides elementary user interface building blocks, basic data structures and high-level application components. The backbone of the ET++ framework is a class hierarchy, in which most classes are derived from a root class Object. This root class defines the protocol of various services, in particular, the service of event notification (or change propagation). The event notification service uses the Observer pattern (Gamma et al. 1995, pp. 293-303) that defines a one-to-many dependency between objects so that when one objects changes state, all its dependents are notified and updated automatically. In addition, ET++ defines abstract data types, often referred to as container classes, for data structures such as linked lists, arrays, hash tables and dictionaries; each container class has a companion iterator class (using the Iterator pattern[9] described in Gamma et al. 1995, pp. 257-271) to support traversal in the class.

All classes in the design space navigation framework are subclasses of the ET_Object class[10] in order to take advantage of the built-in observer mechanism. In addition, the organization and maintenance of NAV_Marker, NAV_Problem and NAV_Solution objects are accomplished using the ET++ container classes and the robust iterators[11] that provide the traversal mechanism in the container classes.

4.3.2 Facilities

The design space navigation framework facilitates the management of object organization as well as the navigation operations in a design space. All facilities are provided through the framework facade, NAV_DesignSpace, class. In this section, I describe these facilities in detail.

Object Management

Among all classes in the framework, NAV_Problem and NAV_Solution are the two classes that interest the users of this framework most because they are the hooks to the representations of problem and solution in a generative design system. Objects of these two classes are organized into hierarchical structures (the design is discussed in the previous section) in the framework. These hierarchical structures are the problem decomposition hierarchy, problem revision hierarchy, and solution derivation hierarchy.

The framework facilitates the construction of a problem decomposition hierarchy through the following interfaces[12]:

- AddProblem: add the problem as a root node in the decomposition hierarchy
- AddChildProblem: add the problem as a subproblem of another problem
- AddSiblingProblem: add the problem as a sibling problem of another problem

---

9. Iterator provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
10. To prevent confusions from this point on, I prefix all classes in the ET++ framework with "ET_".
11. A robust iterator ensures that insertions and removals won't interfere with traversal in an aggregate object. For additional discussions, see Gamma et al. (1995, p. 261).
12. Since all interfaces are provided through the NAV_DesignSpace class, it is omitted in the text, i.e. AddProblem refers to NAV_DesignSpace::AddProblem. For implementation details of these interfaces regarding parameters and return values, please refer to Appendix A.

The AddProblem interface allows, potentially, the creation of multiple decomposition hierarchies by maintaining all root problems in a list. Although this may not be desirable by some generative design systems, the implementation not only fully supports the organization of a single-rooted problem decomposition hierarchy but also provides a future extension to manage multiple hierarchies.

The construction of a problem revision hierarchy or history is facilitated through the AddRevision interface, which adds a new problem as a revision of another problem. This interface supports two types of version management: derivation-based or time-based. Examples of the derivation-based version management are source code version control utilities such as RCS (Ligett 1985). Using this type of version management when an object is copied and modified, this copy is recorded as a derived version of its original. The time-based version management, on the other hand, does not consider where new versions are derived from, but records them in sequence based on the time they arrive. To illustrate how to employ these two version management concepts, let's consider the following scenario: a designer, who had been working on solving a design problem (P1), decided to remove a design constraint from the original problem and created a new problem (P2); however, realizing later that it was a different constraint that should be removed, the designer returned to P1 and created another new problem (P3) with this other constraint removed. Figure 4-5a shows these problem versions organized based on the derivational relationship to form a problem revision hierarchy; whereas Figure 4-5b demonstrates a problem revision history using the time-based version management. Nevertheless, the

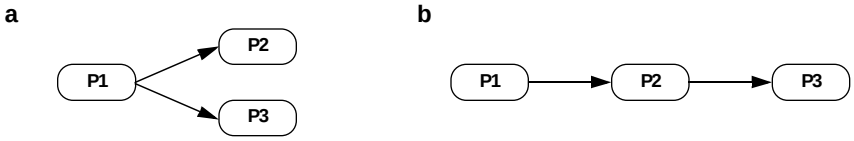

Figure 4-5: Problem revision hierarchy (a) and problem revision history (b)

design space navigation framework cannot enforce the consistent use of one version management type, nor does the framework set preference in the user of one type over the other. Users of the framework are responsible to select the desired type of version management and to ensure consistency.

The framework facilitates the construction of a solution derivation hierarchy through the following interfaces:

- AddSolution: add the solution as a root node in the derivation hierarchy
- AddDerivedSolution: add the solution as a derived solution of another solution
- AddSiblingSolution: add the solution as a sibling (or alternative) solution of another solution

Similar to the AddProblem interface, the AddSolution interface operates without excluding the possibility of managing multiple solution derivation hierarchies that may be associated with the same problem.

Lastly, problems and solutions can be removed from their respective hierarchical organization through the DeleteProblem and DeleteSolution interfaces. When a problem is deleted, its subproblems and derived versions are deleted too; and when a solution is deleted, its derived solutions are deleted as well. Furthermore, to traverse the hierarchical structures of problem decomposition, problem revision and solution derivation the NAV_Problem::MakeChildProblemsIter, NAV_Problem::MakeRevisionsIter and NAV_Solution::MakeDerivedSolutionsIter interfaces are available. These interfaces utilize the robust iterators in the ET++ framework.

Navigation Operations

The design space navigation framework facilitates the five types of navigation operation described in Chapter 3: operations to show the location of current node, to mark a node, to go to a specific node, to step along a specific direction, and to find nodes that satisfy certain criteria.

To assist the display of current nodes[13] in a design space, the framework provides two interfaces to identify current nodes:

- GetCurrentProblem: query for the active problem
- GetCurrentSolution: query for the active solution of the active problem

Additionally, the framework utilizes the observer mechanism to notify interested parties (e.g. objects that are responsible for visualizing certain nodes) when a node becomes active or inactive.

The framework provides marker utilities through the following interfaces:

- MarkNode: attach a marker to a node
- UnmarkNode: remove a marker from a node
- GoToMarker: activate[14] the node marked by a specific marker
- GetMarkers: query for the list of all markers or the list of markers of a specific node

Note that a node can be marked several times with different markers. The interface can only take out one specific marker during each invocation. The interface that handles marker queries provides the means for generative design systems to implement additional navigation operations based on markers instead of just nodes.

13. A current node is also called active node in this chapter. It refers to the problem or solution that is actively displayed in the main working window of a generative design system.
14. In this chapter, "activating a node" refers to "making a node the current (or active) node." Therefore, a node activation may result in having the content (or intrinsic attributes) of the node displayed in the main working window of a generative design system.

Go to operations are facilitated by the following three interfaces:

- GoToNode: activate a node
- SetCurrentProblem: activate a problem
- SetCurrentSolution: activate a solution

The GoToNode interface provides a more general access to a node, which could be a problem or solution; whereas the other two interfaces, SetCurrentProblem and SetCurrentSolution, handle only problems and solutions, respectively. Consequently, all three interfaces send messages to indicate that a node has become active and conversely that the previously active node has become inactive. Therefore, those objects that observe these nodes can react to the status change accordingly.

The framework facilitates the step operations through the interfaces listed below. Again, as a result of each operation, messages will be sent to notify interested parties regarding which nodes become active and inactive. Figure 4-6 illustrates the relevant directions for step operations.

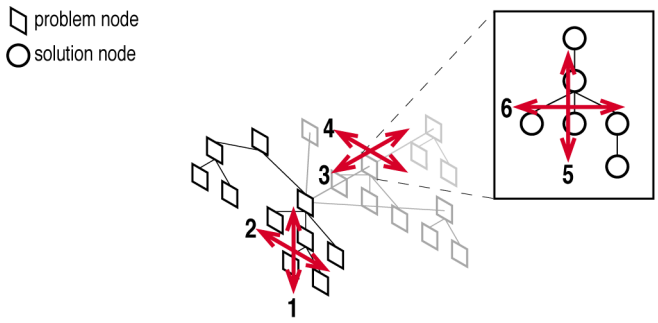

Figure 4-6: Relevant directions for step operations

- ParentProblemNode and FirstChildProblemNode: step along the parent-child problem decomposition direction (see Figure 4-6 direction 1)
- PrevSiblingProblemNode and NextSiblingProblemNode: step along the sibling problem decomposition direction (see Figure 4-6 direction 2)
- PrevRevisionNode and NextRevisionNode: step along the parent-child direction of a problem revision history or hierarchy (see Figure 4-6 direction 3)
- PrevSiblingVersionNode and NextSiblingVersionNode: step along the sibling direction of a problem revision hierarchy (see Figure 4-6 direction 4)
- ParentSolutionNode and FirstDerivedSolutionNode: step along the parent-child direction of a solution derivation hierarchy (see Figure 4-6 direction 5)
- PrevSiblingSolutionNode and NextSiblingSolutionNode: step along the sibling direction of a solution derivation hierarchy (see Figure 4-6 direction 6)

Lastly, the "find nodes" operations are facilitated by the FindNodes interface. To accommodate possibly different search mechanisms, the interface delegates the message, which contains some search criteria, to each node. Each node, in turn, determines whether it itself satisfies the search criteria and reports back. The results from each node are collected and returned. To complete this process, two special interfaces should be implemented once the framework is adapted. They are:

- NAV_Problem::satisfiesTarget: report whether the object itself satisfies the search criteria
- NAV_Solution::satisfiesTarget: report whether the object itself satisfies the search criteria

4.3.3 Summary

Recall the five design space dimensions p, q, r, s, and t discussed in Chapter 3. Dimension p describes the levels of decomposition in a problem decomposition hierarchy, whereas dimension q identifies the breadth of the decomposition, in terms of the number of subproblems, of a particular problem in this hierarchy. Growing the design space along these two dimensions is facilitated by the AddProblem, AddChildProblem, and AddSiblingProblem interfaces; the step-by-step navigation along these dimensions (see Figure 4-6 directions 1 and 2) is supported through the ParentProblemNode, FirstChildProblemNode, PrevSiblingProblemNode, and NextSiblingProblemNode interfaces. The dimension r describes the problem revision history. New revisions can be added using the AddRevision interface and traversal along this dimension (see Figure 4-6 direction 3) is accomplished through the PrevRevisionNode and NextRevisionNode interfaces. Finally, dimensions s and t identify the solution derivation hierarchy. Expanding the design space in these two dimensions is supported through the AddSolution, AddDerivedSolution, and AddSiblingSolution interfaces, while traversing the space along these dimensions (see Figure 4-6 directions 5 and 6) is facilitated by the ParentSolutionNode, FirstDerivedSolutionNode, PrevSiblingSolutionNode, and NextSiblingSolutionNode interfaces.

The design space navigation framework facilitates the growth and traversal of the design space along its five dimensions, and maintains objects, as well as relationships between them, in the space. Developers of generative design systems can utilize its facility without having to worry about managing objects in the design space. In addition, this framework provides a flexible infrastructure that allows generative design systems to offer many types of navigation supports. I will demonstrate the adaptation of the framework to SEED-Layout (Flemming and Chien 1995) in the following section, and illustrate various types of navigation utilities that are built based on the framework.

4.4 Example Application: Supporting Navigation in SEED-Layout

Figure 4-7 shows a screen snapshot of the SEED-Layout navigation utility in order to give an idea of what kind of navigational tools the design space navigation framework supports in a concrete application. The SEED-Layout navigation utility consists of three navigational tools: a 2D tree view to visualize the whole design space (Figure 4-7a, b), a multi-layer view to display the composition of solutions of different problems (Figure 4-7c), and an entity-relationship diagramming tool to allow users to draw the design space according to their views (Figure 4-7d). Each of the three tools provides navigation operations, such as marking a node, go to a node, and stepping to parent, child or sibling solutions. These operations are facilitated by the design space navigation framework; no

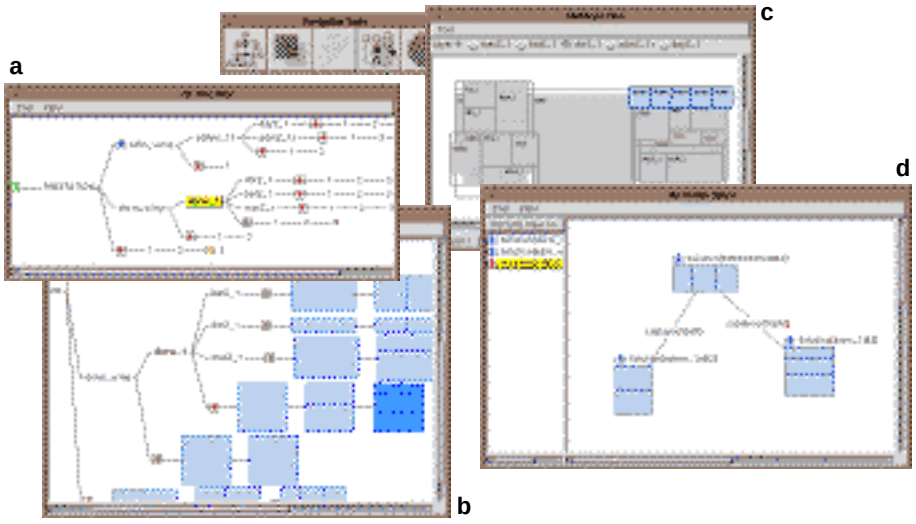

Figure 4-7: SEED-Layout navigation utility
(a) and (b) are tools that display the design space in 2D tree views; (c) shows a multi-layer view of a layout composition; and (c) depicts an entity-relationship diagramming tool.

special efforts by the SEED-Layout developer are required to implement the operations in the navigation utility of SEED-Layout. Other parts of the implementation, particularly the visualization aspect, are not automatically handled, but are supported by components of the navigation framework, for instance the underlying object organization and management (described in previous sections). Readers who are interested in the implementation detail of the SEED-Layout navigation utility are referred to Appendix B.

How did the SEED-Layout developer adapt the design space navigation framework into SEED-Layout? A design problem in SEED-Layout is formulated as a LayoutProblem object; a design solution is represented by a SL_Layout object. The process of adapting the navigation framework to SEED-Layout essentially involves bridging the two different representations, that of the framework and of SEED-Layout, of problems and solutions. Figure 4-8 illustrates the adaptation of the framework through the establishment of connections between SEED-Layout classes (i.e. LayoutProblem and SL_Layout) and the specialized hook classes (i.e. NAV_SLProblemImpl and NAV_SLSolutionImpl). Additionally, a concrete representation, i.e. NAV_SLMarkerImpl (see Figure 4-8), of the NAV_Marker hook class is created to implement additional object behaviors. Once the link between objects of the framework and those of SEED-Layout is established, a representation of the design space based on problems and solutions can be built independently from how their counter parts are organized in SEED-Layout.

The particular SEED-Layout prototype used in this example application supports hierarchically decomposed problems and organizes solutions in derivational hierarchies. These organizations are the same as, respectively, the problem decomposition hierarchy and solution derivation hierarchy organizations in the navigation framework. However, the SEED-Layout prototype did not provide special support for problem revisions. By

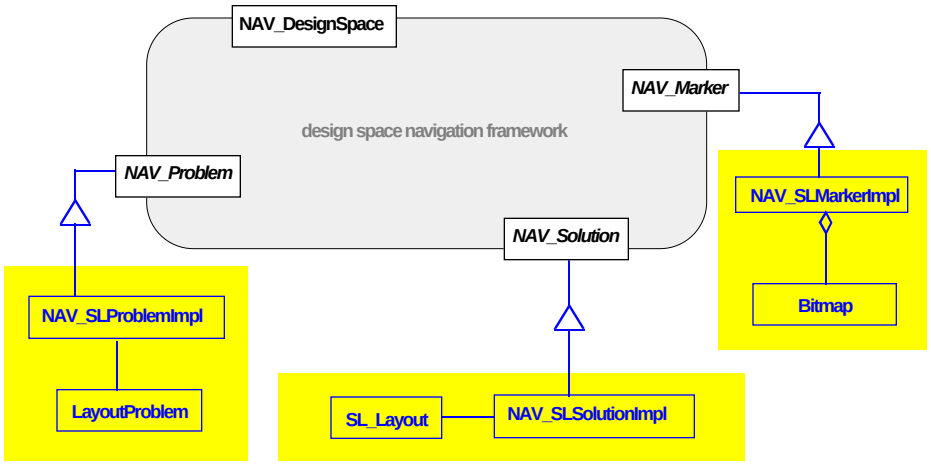

Figure 4-8: Specialization and adaptation of the framework hook classes to SEED-Layout

adapting the navigation framework, SEED-Layout can utilize either the derivation-based or the time-based version management described previously. For the prototype in this example application, the time-based version management scheme was chosen. Furthermore, the super/sub-layout relation in SEED-Layout does not have an equivalent representation in the navigation framework. Nevertheless, this relation can be encapsulated into the hook class, i.e. in this case NAV_SLSolutionImpl. For example, the multi-layout view tool is implemented based on this additional super/sub-layout relation. I will return to this example application in Chapter 6, in which I will elaborate on the user interface designs and examine the usability of the navigation utility.

4.5 Discussion

The major advantage of the design space navigation framework is its generality. I have demonstrated in the previous section how to adapt the framework to an existing generative design system—SEED-Layout. The framework enables a generative design system to provide essential navigation operations through an organized information structure. Moreover, the framework can be extended to handle additional information that is not covered by it. Figure 4-9 illustrates the possibility of plugging objects of different generative design systems through proper encapsulations in order to utilize facilities provided by the framework.

Another advantage of the framework is the single and unified interface through the facade, NAV_DesignSpace, class. Once the hook classes are properly plugged in, all facilities (described in a previous section) are accessed through the facade object. Users (i.e. developers of generative design systems) do not have to worry about other objects. They should note that to make the most of this unified interface (also known as the Facade design pattern in Gamma et al. 1995, pp. 185-193) in the framework, only one facade object is required for each design session. The design space navigation framework, however, is

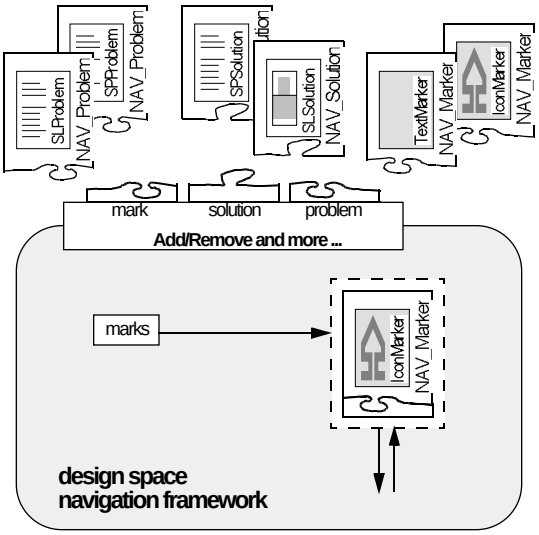

Figure 4-9: Generality of the design space navigation framework

not restricted to only one NAV_DesignSpace object in a session. For a generative design system that supports multiple design sessions, it may utilize this flexibility to maintain multiple design spaces in the system.

5. Software and Usability Engineering

I have presented the comprehensive approach toward supporting information navigation in generative design systems in Chapter 3 and developed the design space navigation software framework in Chapter 4 that embodies the approach. These are necessary elements yet not sufficient to ensure the quality and usability of a navigation support system. We need, in addition, a rigorous software development process. This chapter introduces a software and usability engineering process that takes a hybrid approach to integrate the Object-Oriented Software Engineering methodology in the field of *Software Engineering* with the usability analysis methodology—GOMS—in the field of *Usability Engineering*. The objective of this hybrid approach is to enhance the evaluation phase of the software engineering process so that it covers the full range of the software life-cycle—from specification of the software requirements to usability evaluation of the software product—and ensures the usability of the software system.

5.1 Object-Oriented Software Engineering

As preparation for my discussion of the hybrid software and usability engineering approach stated previously, I first review the particular software engineering method—Object-Oriented Software Engineering (OOSE)—that is employed in this hybrid approach. A representative source will be referred extensively in this section: *Object-Oriented Software Engineering* by Ivar Jacobson and his colleagues (1992).

5.1.1 What is Software Engineering?

Software engineering is a process by which engineering principles are applied in the development of software systems. According to Jacobson and his colleagues (1992), the software system development consists of three distinct phases—analysis, construction, and testing. In the analysis phase, an application-oriented specification is developed that describes what a system offers its users regardless of implementation environments. During the construction phase, the specification is realized through a two-step process: design (identifying building blocks of the system) and implementation (coding these blocks). Finally in the testing phase, the system is checked to make sure that all programs are correctly implemented and that the system performance meets the requirements specified in the analysis phase.

Figure 5-1 illustrates major software development activities in the three phases and the information transfer between activities. System development, according to Jacobson and

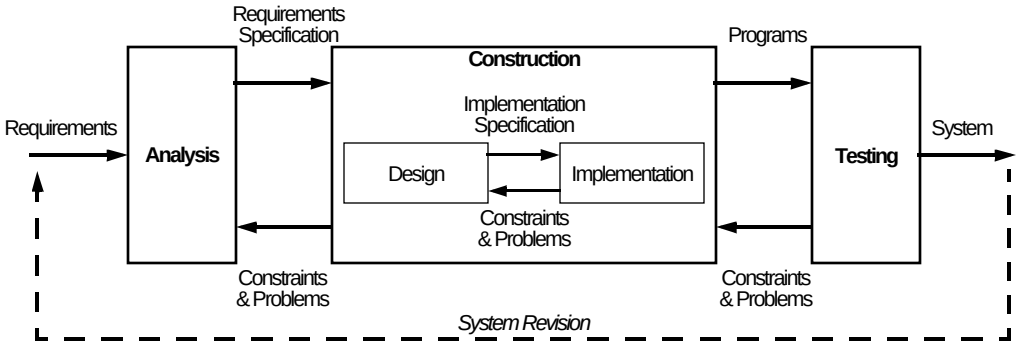

Figure 5-1: Activities in software system development

his colleagues, is a process of change, i.e. a system normally evolves through changes incorporated in new versions. Moreover, software development is usually incremental: new requirements may appear and old ones may change when a version of the system is in operation. Therefore, the reuse of results from earlier development work is highly desirable.

Jacobson and his colleagues argue that in order to significantly increase the productivity in software development, reuse must be a natural ingredient. Moreover, object-orientation offers new techniques that are far better for supporting reuse than traditional techniques. An object-oriented software engineering process, thus, is a software system development process that applies the object-oriented methodology in each of the three phases described earlier. Various object-oriented methods available for different phases of the software engineering process are discussed in the following section.

5.1.2 Object-Oriented Methods

Several object-oriented software system development methods have been proposed. According to Jacobson and his colleagues, the number of methods will increase as a result of the recent high interest in object-oriented techniques. Although some methods used in the computing industry are not available to the general public, Jacobson and his colleagues review the list of publicly available object-oriented methods that have been given generous attention in computer science textbooks.

Object-Oriented Design (OOD) by Grady Booch (1983, 1991), a pioneer in this field, provides a rich set of diagramming techniques to describe the software system. Diagrams form a static description of the system, but describe also dynamic interactions between objects in the system. Object-Oriented Systems Analysis (OOSA) by Shlaer and Mellor (1988) consists essentially of information analysis based on data modeling. This method focuses on describing relationships between objects. Object Modeling Technique (OMT) by Rumbaugh et al. (1991) is based on entity-relationship modeling with extensions to modeling classes, inheritance and behavior. It is one of the few object-oriented methods that make serious attempt to support both analysis and construction phases in software development. Object-Oriented Analysis (OOA) by Coad and Yourdon (1991) is a step-by-step method for developing object-oriented system models. This method focuses on the analysis aspect, but does not offer support to capture systematically all the dynamic roles

played by objects. Hierarchical Object-Oriented Design (HOOD), initially developed for the European Space Agency by a consortium (see HOOD 1989), elaborates a model in a stepwise refinement that subsequently may be implemented directly in a target language. It is intended to support an abstract view of the design and implementation activities. Object-Oriented Structured Design (OOSD) by Wasserman, Pircher, and Muller (1990) is an extension of the structure charts used in structured design and Booch notation. Responsibility-Driven Design by Wirfs-Brock, Wilkerson, and Wiener (1990) is a design method that uses informal techniques and guidelines to develop an appropriate design. Finally, Object-Oriented Software Engineering (OOSE) by Jacobson and his colleagues employs *use cases* as the guiding principle in all phases (i.e. analysis, construction and testing) of the software development process.

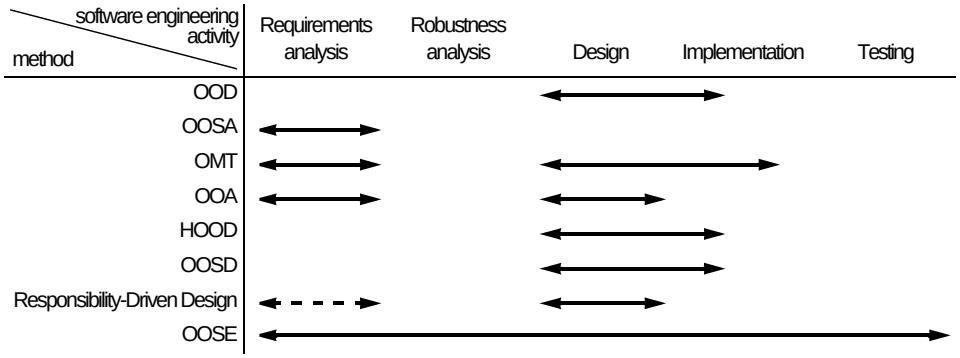

Figure 5-2: The activities of software engineering related to some object-oriented methods (Adopted from Jacobson et al. 1992)

Figure 5-2 summarizes the software engineering activities supported by the object-oriented methods described previously. Among these methods, only OOSE addresses wide ranges of activities in software system development. This method is further discussed later.

5.1.3 Use Case-Driven Approach

Software system development is a complex task. The OOSE method by Jacobson and his colleagues attempts to handle the complexity in an organized way by working with models, each of which focuses on a certain aspect of the system. Based on the three activity phases in the system develope process, OOSE contains five models: requirements model (to capture the functional requirements), analysis model (to give the system a robust and changeable object structure), design model (to adopt and refine the object structure to the implementation environment), implementation model (to implement the system), and test model (to verify the system). System development, thus, consists of the development of different models of a software system.

Jacobson and his colleagues introduce the *use case model* to form the basis of the above five models in the OOSE process. The use case model is a part of the requirements model,

which also includes a problem domain model (or domain object model). It uses *actors* and *use cases* to define what exists outside the software system (i.e. actors) and what should be performed by the system (i.e. use cases). The actors represent everything that needs to exchange information with the system. They can be human users or other software systems. However, there is a clear distinction between actors and users—an actor represents only a certain role that a user can play. A use case is a specific way of using the system by an actor through a sequence of interactions to achieve a desired outcome. The use case-driven approach in OOSE utilizes the use case model as the underlying requirements specification that all other models are geared to satisfy (see Figure 5-3).

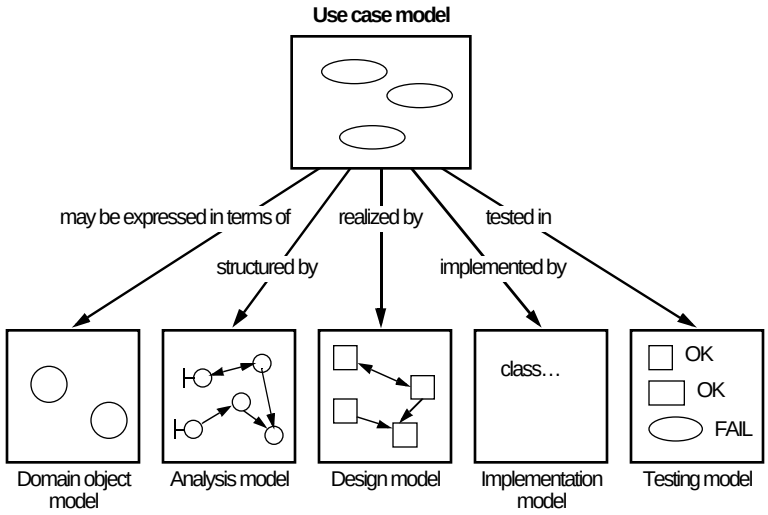

Figure 5-3: The use case model is used when developing all other models (Redrawn from Jacobson et al. 1992)

The use case-driven approach emphasizes system development based on user requirements. The use case model specifies the system functionality from a user's perspective. However, Jacobson and his colleagues have relied on informal techniques and guidelines to develop the use case model, and this lack of standards in establishing the use case model has become a weak point in the OOSE process. Researchers and practitioners in this field have proposed and demonstrated several ways to remedy this weakness (e.g. Regnell, Andersson, and Bergstrand 1996; Cockburn 1997). Among these, a goal-oriented approach to develop and structure use cases (Cockburn 1997) is, in my opinion, the most natural and useful way. In particular, task performance by users is goal-driven (Newell and Simon 1972), therefore formulating system requirements from the user perspective should naturally be based on goals a user wishes to achieve.

From the HCI viewpoint, the process to develop the use cases is similar to task analysis. Both focus on the users and the tasks they perform in a software system. This offers a promising starting point to consider the integration of many HCI or usability evaluation

methods into the OOSE process. In the later sections, I will demonstrate an approach that extends OOSE to incorporate usability analysis techniques.

5.2 Usability Engineering

This section reviews the concept of usability engineering and the GOMS family of usability analysis techniques that is used in the hybrid software and usability engineering process proposed here. Two representative sources will be referred extensively in this section: *Usability Engineering* by Jakob Nielsen (1993), and *The GOMS Family of Analysis Techniques* by Bonnie E. John and David E. Kieras (1994).

5.2.1 What is Usability?

Nielsen (1993) uses the term "usability" to refer to a specific concern within the overall concept of "user friendliness." However, the term "user friendly" has been considered inappropriate by user interface professionals in recent years, partly due to the "warm, fuzzy feeling" the term seems to evoke. Nielsen introduces the term "system acceptability" to examine whether a computer system is good enough to satisfy all the needs and requirements of the users and other potential stakeholders. Figure 5-4 illustrates different considerations within the issue of system acceptability.

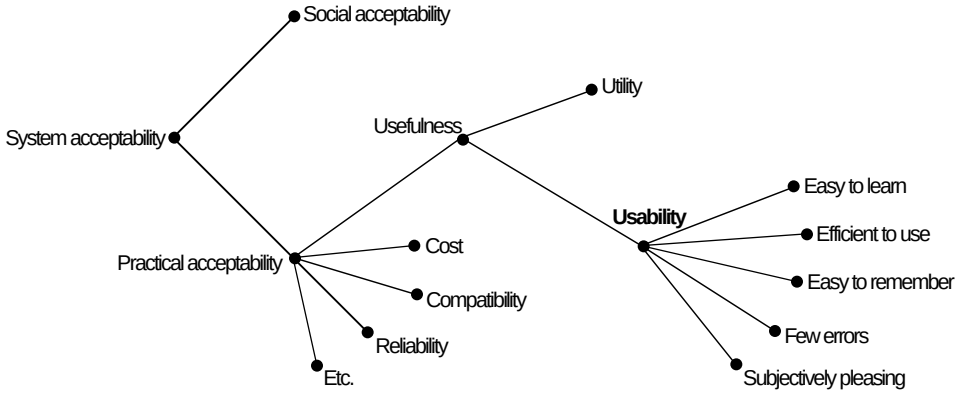

Figure 5-4: A model of the attributes of system acceptability
(Redrawn from Nielsen 1993)

Nielsen argues that we should first consider the social acceptability of a computer system before examining its practical acceptability. Practical acceptability concerns issues such as cost, support, reliability, compatibility with existing systems, and so on, as well as usefulness. Usefulness deals with the question of whether a system can be used to accomplish desired goals, and it involves two issues—utility and usability. Utility focuses on the functionality of a system, and usability examines how well users can use that functionality.

Usability, as Nielsen has stressed, is not a single, one-dimensional property of a user interface. It has five component attributes: learnability, efficiency, memorability, errors, and satisfaction. A system should be easy to learn so that users can quickly start using the system to perform tasks. The system should be efficient to use so that users may achieve a high level of productivity once they have learned the system. The system should be easy to remember so that casual users do not have to relearn the system every time. The system should prevent users from making errors and allow easy error-recovery if errors do occur. Finally, the system should be pleasant to use so that users enjoy working with it.

According to Nielsen, each of these five attributes is a measurable component of usability. By defining usability in terms of these measures, usability can be systematically approached, improved, and evaluated through an engineering discipline, i.e. usability engineering.

5.2.2 Usability Assessment Methods

Methods to evaluate usability can be categorized, in terms of their approaches, into two types—usability analysis and usability testing. Usability analysis evaluates a system based on knowledge derived from physical, psychological, or sociological theories, theories of design, or usability heuristics. Usability testing refers to the use of empirical investigation conducted with users of the system.

Usability testing provides direct information about how people use a system and uncovers problems with the concrete interface being tested. Observation, questionnaires, interviews, focus groups, logging actual use, performance measures, and thinking aloud are usability methods of this type (see Nielsen 1993). Since real users are involved in these methods, usability testing can be time-consuming and costly. Additionally, reliability[1] and validity[2] are two critical issues need to be considered in this type of usability method.

On the other hand, usability analysis does not require user involvement. The keystroke-level model, GOMS model, Natural GOMS Language (see John and Kieras 1994), heuristic evaluation (Nielsen 1993), and cognitive walkthroughs (Lewis et al. 1990; Wharton et al. 1992) are usability methods of this type. Particularly, the first three methods are based on the conceptual framework of human information processing (Newell and Simon 1972) and aim at predicting the usability of a system before it has been tested; therefore, these methods provide estimates of the trade-offs between different interface designs without the need to build them. These three usability methods are part of the GOMS family of usability analysis techniques, which I will elaborate in the section below.

5.2.3 The GOMS Family of Usability Analysis Techniques[3]

John and Kieras (1994) state the general GOMS concept as follows:

> It is useful to analyze knowledge of how to do a task in terms of the components of goals, operators, methods, and selection rules.

---

1. Reliability refers to whether one would get the same result if the test were to be repeated.
2. Validity refers to whether the result actually reflects the usability issues one wants to test.
3. Summary of the GOMS family of analysis techniques in this section is primarily based on John and Kieras (1994). Individual references of each technique are: Card, Moran, and Newell (1980b, 1983) for KLM; Card, Moran, and Newell (1980a, 1983) for CMN-GOMS; Kieras (1988, 1994) for NGOMSL; and John (1990) and Gray, John, and Atwood (1993) for CPM-GOMS.

Goals are what a user has to accomplish; they are often decomposed into subgoals, and all of the subgoals must be accomplished in order to achieve the overall goal. Operators are actions performed by the user in service of a goal; they can be perceptual, cognitive, or motor acts, or a composite of these. Methods describe procedures (sequences of operators and subgoals invocations) for accomplishing a goal. Selection rules determine which method to use when there is more than one method available to the user to accomplish a goal. This general GOMS concept has spawned a family of task analysis and modeling techniques.

The GOMS family includes KLM (Keystroke-Level Model), CMN-GOMS (the GOMS model presented in *C*ard, *M*oran, and *N*ewell 1983), NGOMSL (*N*atural GOMS *L*anguage), and CPM-GOMS (the acronym stands for both *C*ognitive-*P*erceptual-*M*otor analysis of activity and *C*ritical *P*ath *M*ethod) task analysis techniques. KLM is a simplest GOMS technique; it includes six types of operators and five heuristic rules.[4] CMN-GOMS describes a task in terms of a hierarchical goal structure and set of methods in pseudo-code-like notation that consists of a series of steps executed in a sequential order. NGOMSL provides a well-defined, structured language, suitable for practical application and contains explicit procedure for constructing GOMS models. Finally, CPM-GOMS is based on the parallel multi-processor stage model of human information processing; specifically, it uses a schedule chart[5] to represent the operators and dependencies between operators.

John and Kieras have provided an extensive comparison of these techniques. They have summarized the information required for a system developer to choose one of the techniques in Table 5-1. Two characteristics are important to selecting a GOMS analysis

Table 5-1: GOMS techniques available for different combinations of task type and the type of design information desired
(From John and Kieras 1994) Note the that term "design" refers to user interface design.

Task Type / Information Type	Routine Cognitive Skill			
	Passive System		Active System	
	Sequential	Parallel	Sequential	Parallel
Functionality Coverage	Any GOMS	Any GOMS	Any GOMS	Any GOMS
Functionality Consistency	NGOMSL		NGOMSL	
Operator Sequence	CMN-GOMS NGOMSL	CPM-GOMS	*see John and Kieras (1994)*	CPM-GOMS
Execution Time	KLM CMN-GOMS NGOMSL	CPM-GOMS	*see John and Kieras (1994)*	CPM-GOMS
Procedure Learning Time	NGOMSL		NGOMSL	
Error Recovery Support	Any GOMS	Any GOMS	Any GOMS	Any GOMS

4. In KLM, each operator has an estimated execution time; and the heuristic rules are used for placing mental operators to account for mental preparation time during a task that requires several physical operators.
5. PERT chart familiar to project managers.

technique: the types of task the users will be engaged in, and the types of information gained by applying the technique. Note that GOMS techniques apply only to routine cognitive skills, when a user knows exactly what to do during a task and simply has to recognize the task situation and execute the appropriate actions. However, even when a task may be non-routine and primarily problem-solving ones (i.e. the user has to discover the appropriate operators, methods or selection rules), GOMS analysis can be used to improve the user interface design so that the user can effectively work on the non-routine and problem-solving parts of the tasks. Furthermore, GOMS analysis techniques are suitable for examining both passive and active computer systems[6], as well as investigating sequential and parallel activities (i.e. perceptual, cognitive, and motor operators) during a task.

The GOMS family of usability analysis techniques, as discussed in John and Kieras, provides many indicators in guiding the user interface designs. These techniques predict system performance, which can be used to determine where the user interface design effort should be focused. Moreover, since GOMS analyses can make performance predictions without a running system, they can be used early in the system development process to evaluate different ideas before the systems or even prototypes are implemented. Since GOMS techniques can make quantitative predictions of performance, they can be used to do sensitivity and parametric analyses[7]. Finally, GOMS can support the design of documentation and on-line help systems so that they can explicitly present the methods and selection rules that users need in accomplishing their goals.

However, John and Kieras have cautioned that there is a substantial start-up cost to apply GOMS techniques, particularly, in building the first GOMS models. This has been a primary reason to prevent system developers from using the GOMS techniques during the system development process. But my experience in practicing the use case-driven approach of OOSE and several of the GOMS techniques convinced me that the construction of GOMS models can be greatly facilitated through a hybrid of software and usability engineering process. This process is elaborated in the following section.

5.3 Use Case Driven Approach Meets GOMS Analysis

I mentioned previously that GOMS techniques complement the OOSE process, particularly in the testing phase. Combining these two methodologies is based on the similarity between them. My experience of practicing the OOSE use case-driven approach as well as literature studies (e.g. see Regnell, Andersson, and Bergstrand 1996; Cockburn 1997) have shown that this approach is most successful when use cases are developed based on user goals or tasks (a task describes a goal to be achieved by a user). Similarly, GOMS models user tasks based on users' goals.

---

6. When using a passive computer system, the user has control over the pace and timing of task events, and the computer merely waits for inputs from the user. Whereas in an active computer system, the computer may produces events that require the user to abandon the current goal and set up a new goal to accomplish.
7. The sensitivity analysis examines how sensitive the predictions are to the assumptions; the parametric analysis studies how the predictions vary as a function of some parameters.

In this section, I will illustrate the integration of the two methodologies through examples. Although examples of GOMS models will be based on the CMN-GOMS technique described in Section 5.2.3, it does not mean that only CMN-GOMS is suitable for this integration; rather all GOMS family techniques are suitable. I selected CMN-GOMS because the KLM technique is too simplified in modeling tasks in software systems with navigation support and because GOMS models using CMN-GOMS, among the rest of the GOMS techniques, are easier to understand and codify.

5.3.1 Use Case Description

As stated earlier, a use case is a specific way of using the system by performing some part of the system's functionality. Each use case is documented with a descriptive statement called *use case description*. Recall that in Section 3.2 of Chapter 3, I have identified a set of basic navigation operations. One of them states that the navigation support should show designers the location of an active node. To describe this functionality in terms of use case, we need first to identify the designer as one kind of actor. Based on some user interface design assumptions, we can then describe the use case *Show Location of Active Node* as follows:

> *Show Location of Active Node* starts when *Designer* requests display of a design space view. The system displays a design space view with the active node highlighted.

In practice, a use case description is usually more elaborate and organized in formats that are easier to read instead of the block paragraph shown above. For instance, in the *SEED-Layout Requirement Analysis* (SEED 1998), each use case is formulated in three parts: a brief description that summarizes the use case in a short sentence; a flow of events that describes the course of events during the performance of the use case; and a comments section for special considerations in the software development process regarding this use case. Using this format, the use case *Show Location of Active Node* can be revised as follows:

> *Show Location of Active Node*
>
> Brief Description:
>
> The system displays a design space view and highlights the active node.
>
> Flow of Events:
>
> 1.The designer requests display of a design space view.
> 2.The system displays a design space view with the active node highlighted.
>
> Comments:
>
> A node represents a *problem* or a *solution*. There may be more than one active node in a design space, i.e. an active problem and its associated active solution. A design space view does not necessarily display a complete view of all nodes populating the design space; it may be a partial view, e.g. a view of the problem decomposition hierarchy only. The highlighting mechanism may employ visual cues in addition to the color change method (e.g. reverse video display) commonly used.

Jacobson and his colleagues stress that use cases are identified through the actors. They state that in order to identify a use case, system developers can read the requirements specification from an actor's perspective and carry on discussions with those who will act in this role. Particularly, they suggest that developers ask the question "What are the main *tasks* of each actor?" during the process of developing use cases.

5.3.2 GOMS Model

I have briefly mentioned that the process to establish use cases in OOSE is similar to the task analysis process described in usability engineering. Among many task analysis techniques, the GOMS family of analysis techniques is most powerful in terms of its ability to predict user performance and to evaluate different user interface designs without the need to build them. The GOMS analysis begins with constructing a GOMS model of the task to be studied. A GOMS model, as described earlier in Section 5.2.3, encodes user interactions to accomplish a certain task in terms of goals, operators, methods and selection rules. Figure 5-5 illustrates a GOMS model with annotations indicating how statements in the model relate to goals, operators, methods and selection rules. This sample model encodes the task of locating an active node in a design space view. Additionally, the GOMS model shows that there are two alternative methods to view the design space: through a 2D tree view or a multi-layer view. Lastly, the model shows the keystroke-level operations to accomplish a goal. If a designer should choose to view the design space using the multi-layout view, the predicted task performance time could be estimated by adding the times needed to execute all related operators (see Figure 5-5).

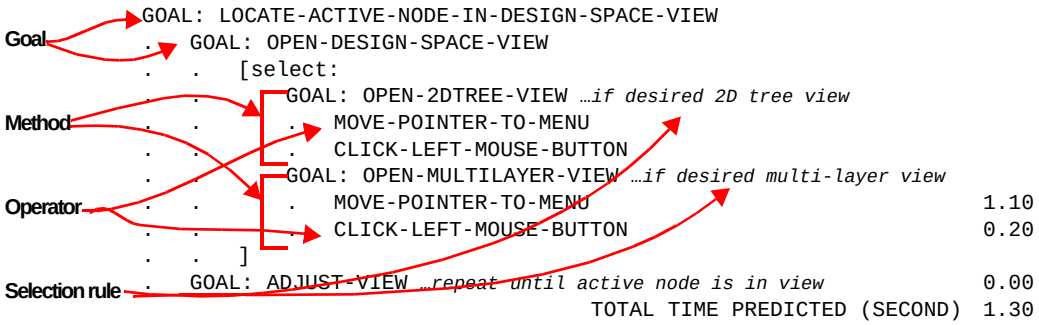

Figure 5-5: A sample GOMS model with time prediction
The time prediction is based on the assumption that the designer can see the active node in the multi-layer view without having to adjust (zoom or pan) the display area. Execution times for the mental and physical operators are taken from Card, Moran and Newell (1983).

Constructing a GOMS model is usually time-consuming, especially when the task analysis process is performed as an extra effort aside from the software engineering process. Given that use cases capture the system requirements best when they are developed based on user tasks, they can form the basis for building GOMS models. In the following section, I will illustrate how to take advantage of a use case description and expand it into a GOMS model.

5.3.3 From Use Case To GOMS Model

Let us review the use case *Show Location of Active Node* demonstrated in Section 5.3.1. The complete course of events to perform this use case is described in its flow of events section as follows:

Show Location of Active Node

Flow of Events:

1. The designer requests display of a design space view.
2. The system displays a design space view with the active node highlighted.

If we consider the two events in terms of user goals during the task, we can define the designer's top level goal:

GOAL: LOCATE-ACTIVE-NODE-IN-DESIGN-SPACE-VIEW

Naturally, if a design space view is available on the screen, the designer can go to that view directly and proceed to look for the active node. Conversely, the designer has to bring up the design space view first if it is not available on the screen. In either of these cases, the designer may have to adjust the display of the view in order to see the active node. Therefore, we can consider these two events as two subgoals to be achieved in order to accomplish the top level goal. Using GOMS notation, we write:

GOAL: LOCATE-ACTIVE-NODE-IN-DESIGN-SPACE-VIEW
. **GOAL:** OPEN-DESIGN-SPACE-VIEW *…if not available on screen*
. **GOAL:** ADJUST-VIEW *…repeat if active node not in view*

The indentation above indicates, for instance, that **GOAL:** ADJUST-VIEW is a subgoal of **GOAL:** LOCATE-ACTIVE-NODE-IN-DESIGN-SPACE-VIEW; and the text in italics says that this subgoal is to be invoked repeatedly until the active node appears in view. For comparison, Figure 5-6 illustrates the mapping between the use case event flow and the GOMS model goal hierarchy discussed above.

Show Location of Active Node

Flow of Events	GOMS Model
1. The designer requests display of a design space view. 2. The system displays a design space view with the active node highlighted.	GOAL:LOCATE-ACTIVE-NODE-IN-DESIGN-SPACE-VIEW . GOAL: OPEN-DESIGN-SPACE-VIEW . GOAL: ADJUST-VIEW *…repeat if active node not in view*

Figure 5-6: Mapping between use case flow of events and GOMS model goal statements

The completion of GOMS models, however, requires detailed designs of the user interface in order to encode operators, methods, and selection rules. Consequently, the user interface can be evaluated by examining the appropriateness (from the cognitive aspect) of GOMS models and by analyzing data collected from the user testing against GOMS models. For example, as the system development process continues, if a user interface design decision is made to offer the designer two alternative ways to view the design

space (i.e. a 2D tree view and a multi-layer view), the subgoal can be expanded to model the choice between two methods:

GOAL: OPEN-DESIGN-SPACE-VIEW
. [select: **GOAL:** OPEN-2DTREE-VIEW …*if desired 2D tree view*
. **GOAL:** OPEN-MULTILAYER-VIEW …*if desired multi-layer view*]

The GOMS model can be continuously refined along the software engineering process with little or no extra effort. As more and more user interface design decisions are made (with or without a prototype or actual implementation), the GOMS model will have enough detail to predict the time needed to perform the particular task. For example, Figure 5-5 shows a further refined GOMS model of the *Show Location of Active Node* task and illustrates a time prediction for the case when the designer chooses a particular method to view the design space.

5.4 Conclusion

I have demonstrated a hybrid approach of software and usability engineering. By integrating the GOMS analysis into the OOSE process, I hope to make software system developers more aware of usability issues as well as to enable usability engineers to involve in the software development process as early as the analysis phase. Consequently, this may improve the quality of software systems as well as ensure the usability of the systems. This software and usability engineering process has been applied to develop the navigation utility of a generative design system (to be discussed in Chapter 6). The specification and design documents that result from the software and usability engineering process described in this chapter are available in Appendix C.

6. Navigation in SEED-Layout: A Case Study

This chapter presents a case study of supporting information navigation in a generative design system. The purpose of the case study is twofold: first, to provide a proof-of-concept to affirm the approach to support information navigation (discussed in Chapter 3) as well as the design space navigation software framework (see Chapter 4) that embodies the approach; second, to examine assumptions underlying the approach and to highlight aspects that the system developers may need to consider when developing a navigation support.

In the following sections, I first introduce the generative design system—SEED-Layout—that is the subject of this case study and its navigation utility, which consists of existing and newly developed navigation tools. Subsequently, I describe usability studies conducted on these navigation tools. These usability studies are pilot studies. They seek to gain insights to highlight issues that future studies in information navigation (or continuing studies) may focus on. The usability studies include an empirical study to understand the effect of these navigation tools, and an assessment of a marker utility[1] through GOMS analysis (see Chapter 5) and user testing. Finally, I present the lessons learned from this case study.

6.1 SEED-Layout

SEED-Layout (Flemming and Chien 1995) is a generative design system that supports schematic layout planning including the rapid generation of layout alternatives based on explicitly stated requirements. Designers who use SEED-Layout may modify problem specifications as their perception of the design problem evolves, control the layout generation process, and examine and evaluate design solutions. The generative operations offered in SEED-Layout allow designers to derived layout solutions from layout solutions. For a complex design problem, SEED-Layout provides supports for designers to start with more abstract problem specifications and then successively refine them.

1. A utility that is designed based on the concept of landmark and bookmark to provide prominent visual cues in a design space and quick accesses to marked nodes. Further discussion is available in the coming section.

6.1.1 Design Space

In SEED-Layout, a layout solution (or *layout*) is composed of *design units* and *walls* that separate design units. A design unit is a part of the spatial or physical structure of a building with an identifiable spatial boundary. Figure 6-1 illustrates a layout solution consisting of five design units (light-color rectangles) and many walls (dark-color lines).

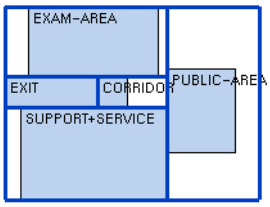

Figure 6-1: An example layout solution in SEED-Layout

A layout design problem consists of *context* information and *functional units* which encapsulate functional requirements of the problem. A functional unit can contain other functional units that are its *constituent units* (or constituents for short); each of the constituent units may, in turn, have constituents; thus the functional units taken together form a *constituent hierarchy*. Figure 6-2 illustrates a constituent hierarchy depicting the functional requirements of a clinic that contains a public area, examining area, support and service area, corridor area and an exit, some of which contain rooms inside their boundaries. A context may be associated with any functional unit in this hierarchy. To

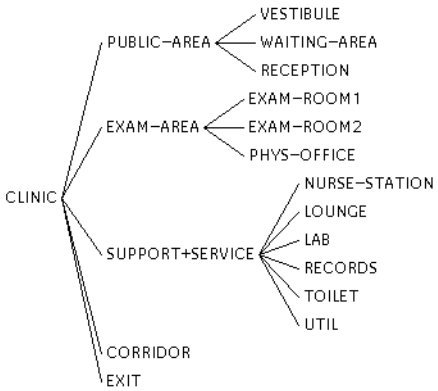

Figure 6-2: An example constituent hierarchy in SEED-Layout

solve a layout design problem, SEED-Layout expects that at least a context is associated with the top unit in its constituent hierarchy (e.g. CLINIC in Figure 6-2). Given a valid layout problem, SEED-Layout supports a top-down design process in an orderly fashion.

Figure 6-3 illustrates the top-down design process. It begins with associating the top functional unit with a design unit whose dimensional attributes reflect the context restrictions; this design unit is the enclosing rectangle for the entire layout. The constituents can then be allocated within this rectangle. Each design unit is associated with a constituent and defines, in turn, a context and enclosing rectangle for the allocation of the constituents of this constituent; it thus completes the definition of a layout problem at the next level. The overall problem can therefore be solved by solving the layout problems defined at each level by a constituent and its context.

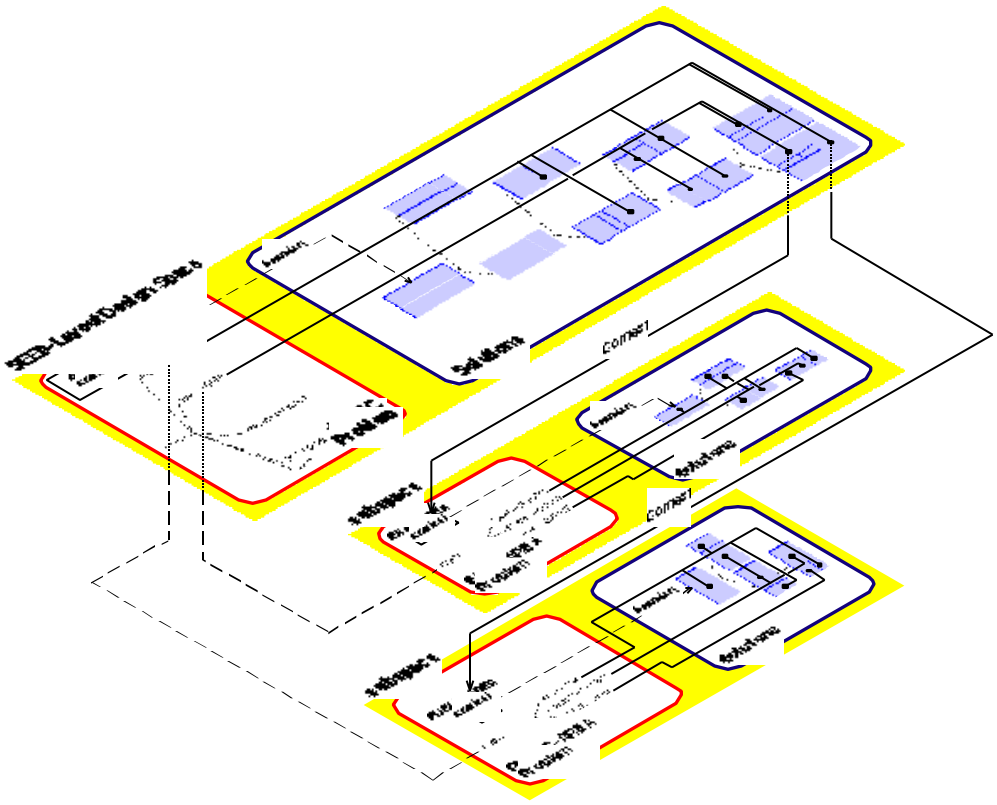

Figure 6-3: An example design space (result of a top-down design process) in SEED-Layout
Functional units with the same name, i.e. PUBLIC-AREA and EXAM-AREA, refer to an identical unit.

As illustrated in Figure 6-3, a functional unit becomes a layout problem when it is associated with a specific allocation context (depicted by a solid line, labelled **Context**, connecting a design unit and a context). The hierarchical decomposition of a functional

unit into constituents can be interpreted directly as the decomposition of a layout problem into subproblems, although the creation of subproblems happens dynamically during the design process. It should be noted that a constituent can be associated with different context objects and can thus form alternative subproblems.

The objects, such as layout problems and solutions, and relationship between objects described so far are appropriately represented by different types of nodes, such as problem and solution, and links in the design space model discussed in Chapter 3. Therefore, the model can be utilized to represent the SEED-Layout design space and to describe relationships between different layout problems (alternatives or revisions and decomposition hierarchy), between different layout solutions (derivational hierarchy), and between layout problems and layout solutions (problem-solution relation). Given the need to provide additional navigation support in SEED-Layout, three navigation tools were developed by adopting the design space navigation framework discussed in Chapter 4 (in particular, Section 4.4 which describes technical details of the adopting process).

In the following sections, I will explain the built-in navigation support in SEED-Layout as well as the three new navigation tools.

6.1.2 Built-in Navigation Support

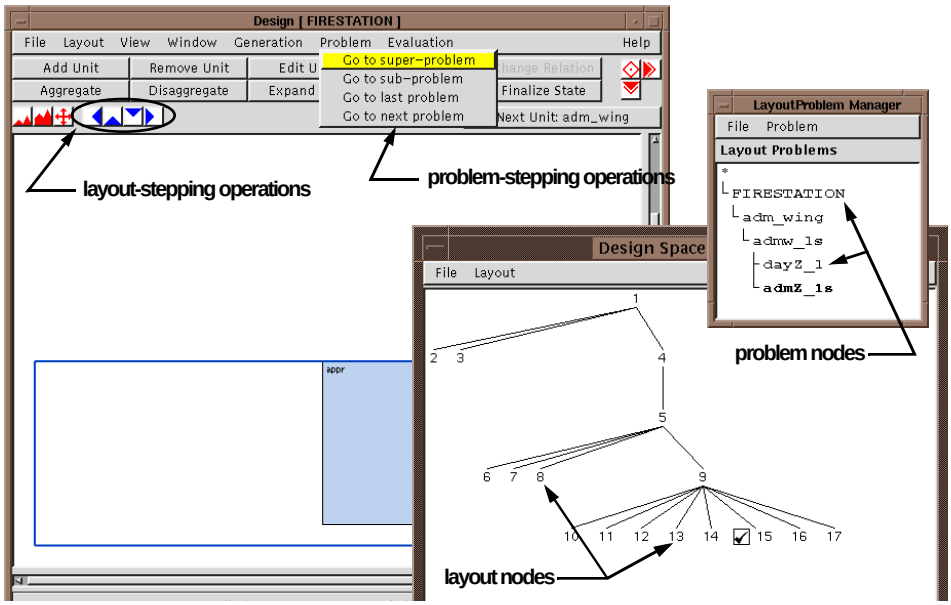

Figure 6-4: Built-in navigation supports in SEED-Layout

Navigation support is available through existing operations in SEED-Layout. To go to different problems, the designer can use the problem-stepping operations through the

Design Window or use the direct go-to-problem operations through the LayoutProblem Manager Window. To explore different layout solutions, the designer can use the layout-stepping operations through the Design Window or use the direct go-to-layout operations through the Design Space Window. A detailed description of these operations with step-by-step instructions is available in Appendix D. Figure 6-4 shows a snapshot of the built-in navigation supports through the three windows stated above.

6.1.3 2D Tree View Tool

The 2D Tree View Tool is designed to provide a global view of the design space. It visualizes the complete design space through two display modes—enhanced and simple modes. The enhanced mode displays a layout with a graphical icon to indicate the spatial configuration in that layout (e.g. Figure 6-5a). Alternatively, the simple mode shows the design space in a simplified fashion where layouts are identified by numbers (e.g. Figure 6-5b). The designer can directly go to any problem or layout in the window or mark problems or layouts for later references. Detailed descriptions of these navigation operations are available in Appendix F.

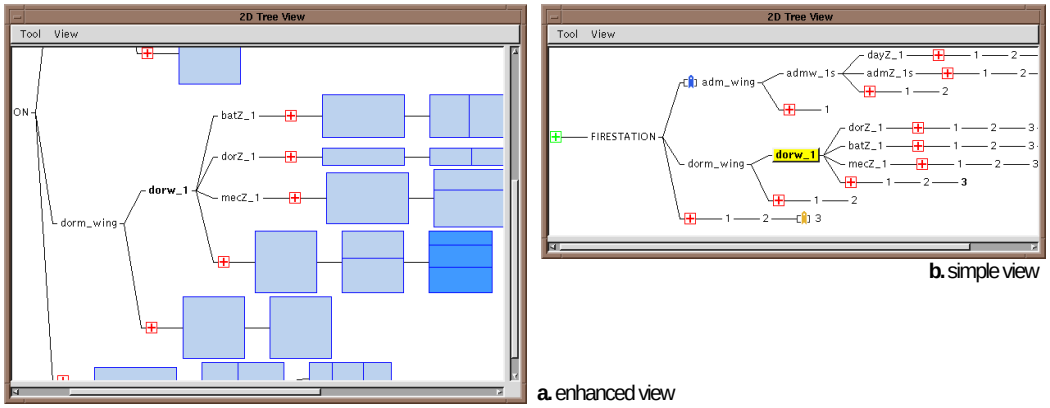

Figure 6-5: Enhanced and simple 2D tree views

In addition to being able to toggle between enhanced and simple display mode and the navigation operations listed above, the designer is provided with filtering commands to show or hide layout solutions and to show or hide problem alternatives or revisions. For example, if the "Hide Solutions" command is selected (from the "View" menu), the window displays only the problem decomposition hierarchy and alternative/revision history but hides layout solutions. Furthermore, the designer can change display scales through zoom-in and zoom-out commands. These view manipulation and information filtering commands assist designers to manage the potentially boundless and ever-growing design space.

6.1.4 Entity-Relationship Diagramming Tool

The Entity-Relationship Diagramming Tool is based on the understanding that individuals form their own mental images of an environment (see discussions in Chapter 2); it thus aims at allowing designers to build their own maps of a design space. Therefore, the particular window that implements this tool is named My Design Space. It operates on marked layouts and problems, and allows a designer to create relationships, in addition to the ones described in the design space model (see Chapter 3), between these marked nodes and make annotations. Figure 6-6 shows a snapshot of the My Design Space window, which displays a diagram of three marked layouts and annotations of sublayout relationships between these layouts.

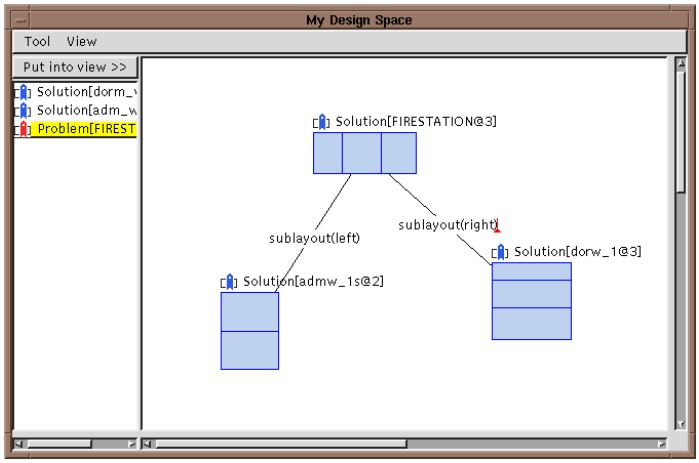

Figure 6-6: My Design Space: the entity-relationship diagramming tool

This tool is designed to work in conjunction with the 2D Tree View Tool that provides commands to select and mark desired nodes (see Appendix E for detailed descriptions of how to use the Entity-Relationship Diagramming Tool together with the 2D Tree View Tool). Once a node is marked, its marker is shown in the list of markers located on the left panel of the My Design Space window (see Figure 6-6). The designer can put markers from the list into the main view (right panel) and make notes. Moreover, the list of markers offers a functionality similar to bookmarks that allow the designer to quickly activate a marked node without having to locate it back in the 2D Tree View Tool.

6.1.5 Multi-layer Display Tool

The Multi-layer Display Tool is designed to provide an interface that mimics the layers of yellow tracing paper used by architects. It arranges layouts into different layers according to the decomposition hierarchy of their associated problems. Designers can, without losing track of the overall layout composition, change their focus to different problems and explore layout alternatives. The tool supports navigation operations, but does not

provide designers with a visualization of the design space. Figure 6-7 shows an example snapshot of the tool.

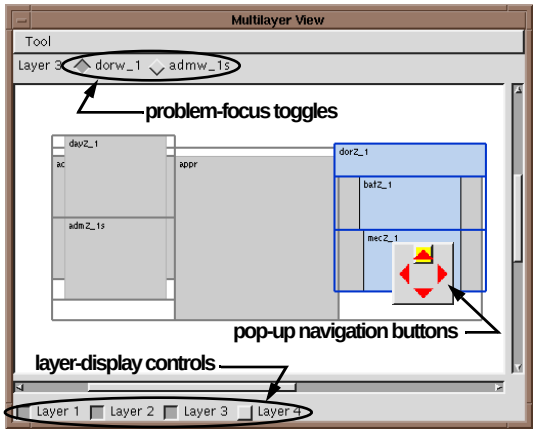

Figure 6-7: Multi-layer display tool

This tool maintains the super/sub-layout relationships that are imposed on the structure established by the design space model (see the discussion in Section 3.1.3 of Chapter 3). Since a superlayout or sublayout is a solution of the superproblem or subproblem respectively, stepping to a superlayout or sublayout will also activate the appropriate problem. Aside from this super/sub-layout stepping operations, the designer can step through alternative layout solutions of the same problem using pop-up navigation buttons. The designer can change focus to different layouts (solutions of different problems) on the same layer using problem-focus toggles. In addition, the tool offers controls to show or hide individual layers through the layer-display controls. A detailed description of the operations mentioned above, with step-by-step instructions, is available in Appendix G.

6.2 Empirical Study

The design and implementation of the new navigation tools described in previous sections have taken the approach discussed in Chapter 3. Recall from Chapter 3 that there are three assumptions underlying this approach of supporting information navigation in generative systems. They are:

1. The cognitive maps theory and environmental design principles are applicable to the design of navigation supports in information spaces.
2. A coherent structure underlying an information space will help designers to anticipate the kind of changes that will take place.

3. An effective visualization of an information space can lead designers to form an intuitive understanding of the structure and behavior of the space.

The second assumption is addressed by the development of the design space model and the design space navigation software framework which has been adopted in creating new navigation tools in SEED-Layout. This empirical study aims to initiate the investigation of assumptions 1 and 3 stated above. Specifically, this study hopes to gain insights in issues, such as the impact of using the landmark concept (the key element in the cognitive maps theory and environmental design principles) in navigation supports; the effect of personal navigation aids, such as build-your-own-map kind of tools, for individuals; and the need of having a complete map of the design space in navigation supports.

The study uses four SEED-Layout user interface configurations, each of which provides a different type of navigation support. Participants of the study perform navigation tasks using one of the user interface configurations. Results of this study are expected to shed light on the following two points:

- Customizable navigation tools may improve subjective user satisfaction of the navigation experience.
- The visualization of a full design space is desirable for basic navigation supports.

6.2.1 Method

The basic form of the study is to measure performance on navigation tasks of each participant in several navigation support treatments in SEED-Layout. Each treatment addresses one or more of the issues described in the previous section. Therefore, the independent variable is the user interface implementation that provides a particular type of navigation support. The dependent variable is performance and behavior on specific navigation tasks. Four treatments are used in the study:

- Control Treatment: existing navigation assistance in SEED-Layout through multiple windows.
- Map-and-Notepad Treatment: a simple view of the design space with a build-your-own-map tool.
- Map Treatment: a comprehensive view of the design space.
- View Treatment: a multi-layer view of layout solutions.

Each of these treatments will be described in detail later.

Data collection for navigation tasks was done using execution timings, keystroke analysis, observation notes, and post-task interviews. As a participant would locate targets within each environment, the time was noted to allow analysis of the overall time it took to complete the task and also the time to complete particular sub-tasks within the treatment. While traversing the design space in SEED-Layout, the participant's mouse pointer activities, including the window in focus, were logged. This data was used to analyze a variety of variables associated with the tasks. The post-task interview was conducted by the experimenter to collect additional information about the participant's experience with the system and relevant comments. Post-experiment analysis could then be done by correlating the keystroke log and the participant's actions or relevant comments noted by the experimenter.

Participants

All participants were given a three-lesson tutorial to learn fundamentals of SEED-Layout prior to the actual tests. The tutorial sessions were conducted using a user interface configuration that is different from the four treatments in this study.[2] For the actual tests, the participants were divided into three groups and all tested on Control Treatment. Participants in the first group were also tested on Map-and-Notepad Treatment; the second group on Map Treatment; and third group on View Treatment. Seven participants, three female and four male, took part in the study. Time was the major reason that limited the total number of participants because the total time taken to complete all tutorials and tests per participant was approximately four hours. All participants were architecture graduate students, who had had at least four years of design training. Since SEED-Layout is designed to be used by designers in architectural design offices, the participants' design background was desired so that they represented potential users.

A primary concern in the experimental design was the problem associated with carryover between tests within participants. That is, lessons learned on one treatment can be applied to a subsequent treatment, thus altering perceived behavior. Wilson and Corlett (1991) suggest an alternative solution to this problem: the tests could be limited in time duration as to minimize the amount of learning that takes place. In the case of this study, each test was held on a separate day and was limited to 45 minutes since the length of time could be associated, by participants, to the degree of difficulty of the task and thus could discourage participants. This, in conjunction with the complexity of the design problem assigned to the task, minimized the cross-talk between treatments. In addition, the order of testing for the treatments was assigned at random.

User Interfaces and Tasks

It was already noted that there are four SEED-Layout user interface configurations for the purpose of this study; one for each treatment:

- Control Treatment
  Navigation supports are provided through the built-in operations in the existing SEED-Layout prototype (see Figure 6-4).
- Map-and-Notepad Treatment
  Navigation supports are provided through a limited version of the 2D Tree View Tool (map) showing the design space only in simple view mode (see Figure 6-5b) and the Entity-Relationship Diagramming Tool (notepad, see Figure 6-6). In addition, the Design window (also used in Control Treatment) is available to display layouts, but no navigational functions are accessible through this window.
- Map Treatment
  Navigation supports are provided through the 2D Tree View Tool (map) with its full functionality, i.e. both simple and enhanced display modes are available (see Figure 6-5a and Figure 6-5b) vs. the limited version (simple view mode only) in Map-and-Notepad Treatment. In addition, the Design window (also used in Control Treatment)

2. Although different, the environment used in these tutorial sessions is similar to the Control Treatment environment. The possible effect of this similarity is discussed in Section 6.2.3.

is available to display layouts, but no navigational functions are accessible through this window.

- View Treatment
  Navigation supports are provided only through the Multi-layer Display Tool (view, see Figure 6-7) that shows layout compositions at all times, but does not provide a map (either partial or complete) of the design space.

The design problem, FIRESTATION, was used in all tasks for all participants. This problem contains functional components that are hierarchically structured as shown in Figure 6-8. Prior to each test, a setup routine was executed to restore a design session in which a designer had created about 150 nodes (consisting of problems and layouts) in the SEED-Layout design space. Participants performed tasks of navigating in the space to locate eight target layouts that form a complete solution of the FIRESTATION problem. These target layouts are shown in Figure 6-9.

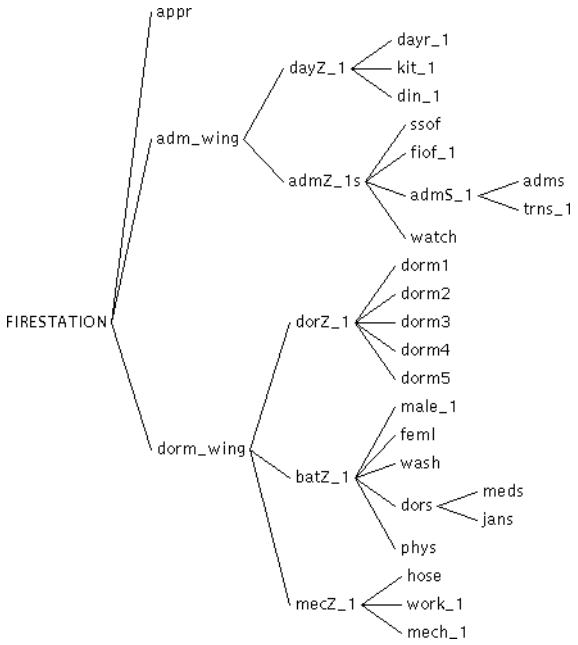

Figure 6-8: Constituent hierarchy of the FIRESTATION problem

Procedure

All participants were new users of SEED-Layout (i.e. none of them had used SEED-Layout prior to this study). They were given a tutorial session and a design exercise to learn the fundamentals of SEED-Layout. In the tutorial session, which takes from one and one half hours to two hours, the participant went through a three-lesson tutorial document with

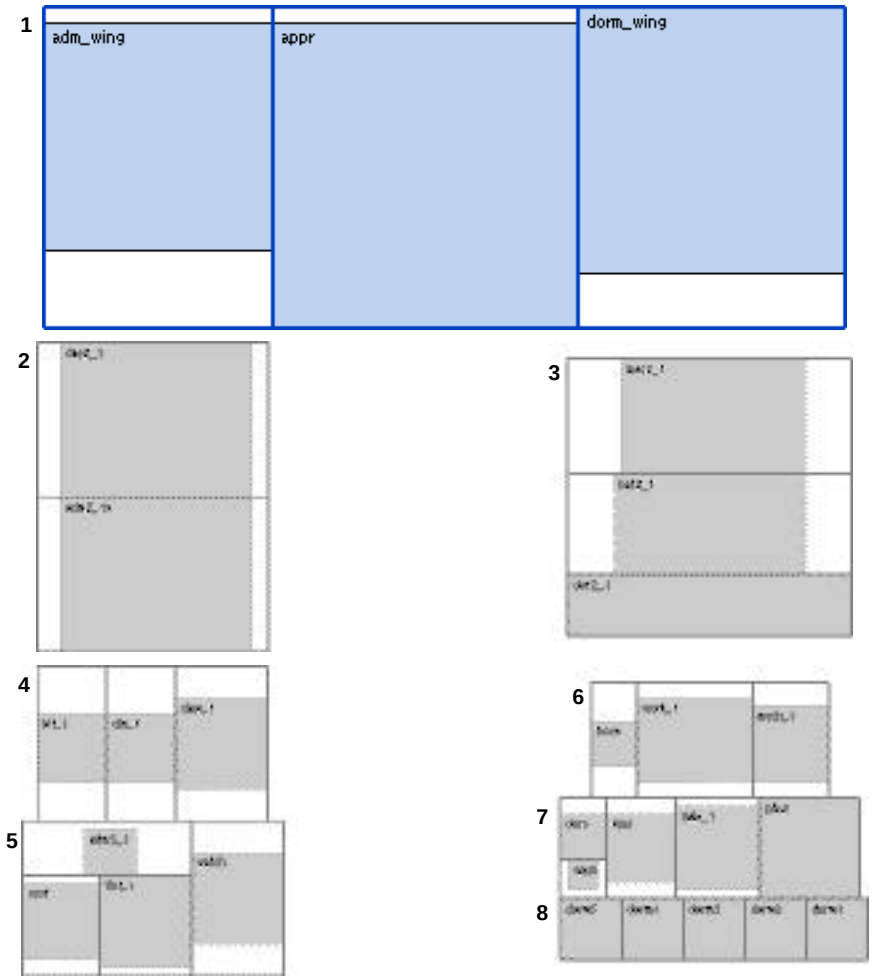

Figure 6-9: Target layouts

step-by-step instructions. The tutorial was designed to convey the following concepts, in addition to basic interactions in the user interface, in SEED-Layout:

- Layout generation: the derivational relation between layouts and the formation of the layout derivation hierarchy
- Problem decomposition: the creation of subproblems, the relation between problems and the structure of the problem decomposition hierarchy
- Layout composition: the relation between layouts of different problems and the composition of layouts into a complete design solution

After the completion of the tutorial session, the participant was given a design task to solve a layout problem and create specific layout solutions. The purpose of this task was to ensure that the participant has attained basic understandings of the concepts described

above. All participants had completed the tutorial and the design task before beginning the actual tests of this study.

Furthermore, these tutorial sessions were conducted using an environment that is similar to the Control Treatment environment. This is a weakness in the design of this empirical study. The possible effect of this similarity is discussed in Section 6.2.3.

The study measured participants' performances in search of a particular layout solution of a design problem. There are four test treatments (described in the previous section). Each task used one of these treatments. The task descriptions and instructions for each treatment are included in Appendix D through Appendix G.

All tasks employed the same design problem (described in the previous section). Participants were required to locate eight target layouts shown in Figure 6-9 in all tasks. The participant started on a state that shows a layout of a problem, which is a subproblem of the root problem and an alternative of another subproblem, and proceeded to navigate through different problems and solutions to locate each of the eight target layouts, which were shown as hints on the task description sheet. The target layouts were not numbered, and no specific order was required for the participant to search. These search times were recorded. A task was considered complete once the last target layout had been located. The overall amount of time spent in the environment was also recorded.

Participants were given 45 minutes to complete the task. However, the task could be discontinued at the participant's request.[3] Nevertheless, participants were encouraged to ask for helps from the experimenter when they felt unable to make any progress toward task completion.

During the task execution, the participant's mouse actions and the window in focus were recorded automatically. Meanwhile, the experimenter observed and noted the participant's actions, answered questions and provided hints when necessary.

Following each task, participants were interviewed about their experiences. Questions asked by the experimenter in the interview were:

- What do you feel was most difficult in performing this task? and Why?
- Does the user interface provide enough help to assist you? if not, What's wrong?
- Can you suggest changes/additions to the user interface to make the task easier?

All participants performed two tasks (one in Control Treatment and the other in one of other three treatments). After the second task, they were asked, in addition to the previous three questions, the following two questions.

- Can you compare this experience with your experience of the last session?
- What are the advantages/disadvantages of the two user interfaces?

6.2.2 Result

The data analysis contains two parts: a quantitative data analysis based on the keystroke events collected during each task and a summary of interviews of subjective evaluation after the task. Taken together, these two sets of data presented in this section will shed

3. All participants completed tasks in the allotted time.

some light on possible answers to issues regarding the effect of personal navigation aids and the need of having a complete map of the design space in navigation supports. The processed data is shown in Appendix H.

Total Time

The observed total time for any treatment was taken to be the elapsed time from the first mouse event to the successful completion of locating the last of the eight target layouts by the participant. Total time should coarsely measure the efficiency of search during any test. It is therefore expected that total time for the map treatments (i.e. Map-and-Notepad Treatment and Map Treatment) will be lower than View Treatment; and Control Treatment is expect to have the highest time of all.

Table 6-1 shows the total time measure of each participant in all four treatments. Each column records times in minutes and seconds (e.g. 2 minutes and 5 seconds are shown as 2′05″) according to the following categorization:

- **T(A1)**: total time in Control Treatment
- **T(A2)**: total time in Map-and-Notepad Treatment
- **T(A3)**: total time in Map Treatment
- **T(A4)**: total time in View Treatment

Data in the **T(W)** and **T(M)** columns are discussed in later sections.

Table 6-1: Time measures for tasks in all treatments

	Control		*Map-and-Notepad*		*Map*	*View*
Participant	**T(A1)**	**T(W)**	**T(A2)**	**T(M)**	**T(A3)**	**T(A4)**
B2	23'09"	4'29"	12'45"	6'49"		
B8	11'15"	0'00"	8'07"	3'29"		
C3	20'18"	9'02"			28'01"	
C5	20'24"	0'00"			[a] 14'42"	
C6	24'29"	0'42"			21'27"	
D1	16'30"	0'18"				11'52"
D7	25'07"	6'12"				11'47"

a. Data based on an incomplete keystroke log file. The log was interrupted due to a system failure while the participant attempted to corrected an incorrectly marked node.

The total time data is charted in Figure 6-10. The individual times for each treatment within each participant are shown vertically.

Due to the fact that relatively few navigational operations were introduced in each treatment (four operations were introduced in Control Treatment; four in Map-and-Notepad Treatment; two in Map Treatment; and four in View Treatment) and all these

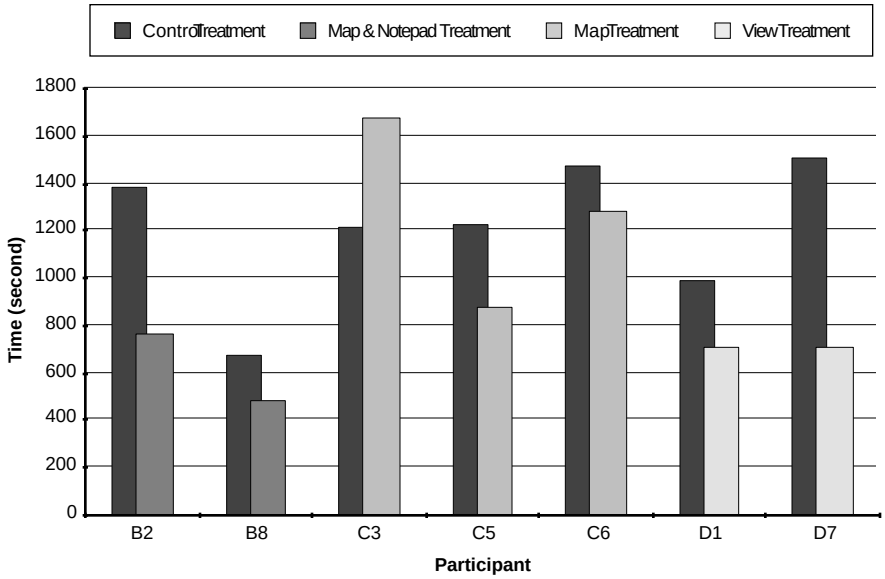

Figure 6-10: Total time

operations were easy to learn, and no special time was allotted for familiarizing the user interface during all tasks. However, it was observed that the majority of participants spent their first few seconds or minutes to get comfortable with the user interface. Since it is difficult to pinpoint the beginning and the end of this familiarization period, the total time may consist of the actual search time and the learning time (i.e. the familiarization period). Suggestions to address this issue are discussed in Section 6.2.3.

Total Wandering Time

Participants, during tests, were observed "wandering" among problems and layouts and appeared *lost* or *confused*[4]. In Control Treatment, this type of behavior was frequently observed and captured in participants' keystroke data. In the keystroke data, a wandering period is identified by a sequence of operations that does not bring the search state closer to a target layout; the wandering time was taken to be the elapsed time from the first operation of this sequence to the last (see Appendix H for examples of the keystroke analysis). Total wandering time takes the sum of all wandering times during a test.

This measure provides indications of a participant's understanding of the structure of a SEED-Layout design space. Participants who had weak knowledge of the structure were expected to spend more time wandering. Data of this measure is shown in the **T(W)** column of Table 6-1. Due to the nature of the interaction styles in other treatments, data of this measure cannot be derived from the keystroke data of other treatments. However, another

4. "Lost" and "confused" were referred from time to time by many participants in their comments during the test or in their answers in the post-task interviews.

measure of operation count, to be discussed later, will provide additional data for the same considerations for all treatments.

Map-building Time

In Map-and-Notepad Treatment, participants used the Entity-Relationship Diagramming Tool (notepad) that allows them to organize selected pieces of information as they wish (see Figure 6-11). This tool is not critical in supporting the kind of search tasks in this

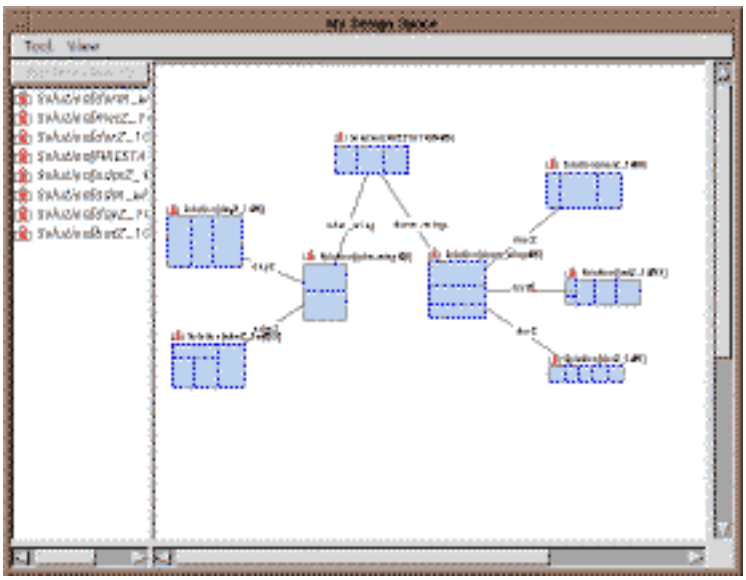

Figure 6-11: Sample snapshot of a *personal map* in the My Design Space window

study. However, participants in this treatment were asked, after completing the task, to use the tool to organize those target layouts they had located and to annotate the relations between layouts. The observed map-building time was taken to be the elapsed time from the first selection of a target layout to be put into the working area in the tool to the last annotation that completes the description of relationships between all target layouts. Data of this measure is shown in the **T(M)** column of Table 6-1.

Operation Count

The operation count for any treatment records the number of navigation operations a participant used to perform a sub-task (i.e. finding of a target layout). Navigation operations are operations used by participants to perform the following functions:

- invoking auxiliary windows;
- adjusting viewable content of a window (excluding resizing window and scrolling);

- going to (activating) a problem; and
- going to (activating) a layout.

Operation count provides indications in two measures: (1) the efficiency of the navigation support, and (2) the efficiency of the participant's search during any test. For the first measure, an efficient navigation support should allow participants to locate a target layout in fewer number of steps regardless of how many alternative layout solutions there are. Therefore, it is expected that the operation count for Map Treatment will be lower than the Control and Map-and-Notepad treatments because that although all three treatments support direct go-to operations, only the Map Treatment environment provides additional visual information that allows participants to narrow down the number of candidate target layouts before performing the navigation operations. Furthermore, View Treatment is expected to have the highest count of all because it does not provide direct go-to operations and participants have to find a target layout through a sequential step-by-step process. Figure 6-12 and Figure 6-13 compare operation count for locating target layout #5, which has 34 other alternatives, and target layout #8, which has 4 other alternative layouts, in all treatments.

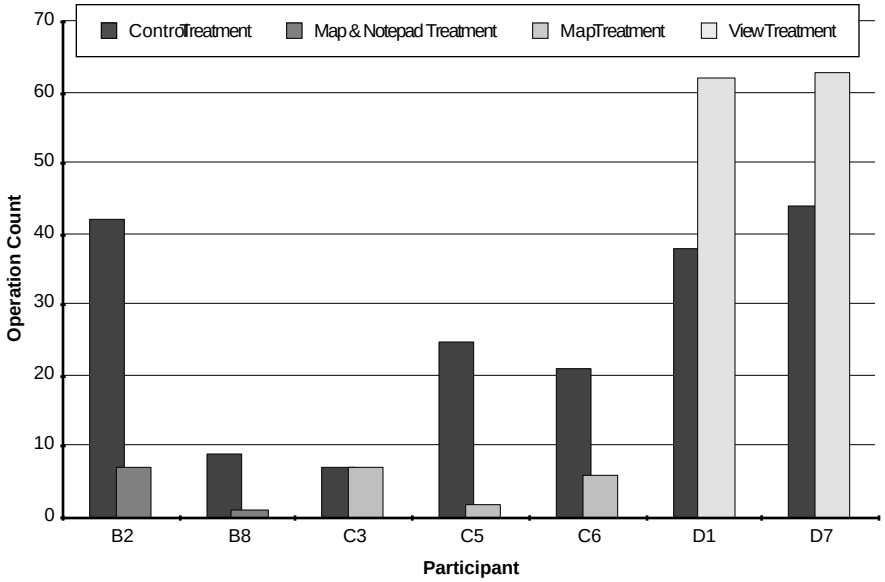

Figure 6-12: Operation count for locating the target layout of admZ_1s (#5)

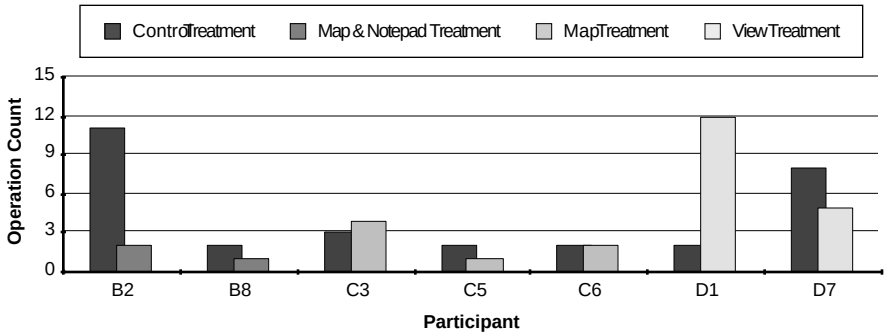

Figure 6-13: Operation count for locating the target layout of dorZ_1 (#8)

Since the maximum operation count[5] of a sub-task in each treatment is attainable for the second measure (i.e. the efficiency of the participant's search), a higher count represents an increase in search by participants during any test. This measure is illustrated in Figure 6-14 based on the data in the **C3** row of Table 6-2 and the estimated maximum count of each sub-task using the Control Treatment environment. The operation counts for each sub-task are shown vertically; horizontally, each number identifies a target layout. This measure can be considered together with the total wandering time measure (described previously) to express a participant's understanding of the structure of a SEED-Layout design space. Participants who had weak knowledge of the structure were expected to spend more time wandering and to execute extra operations.

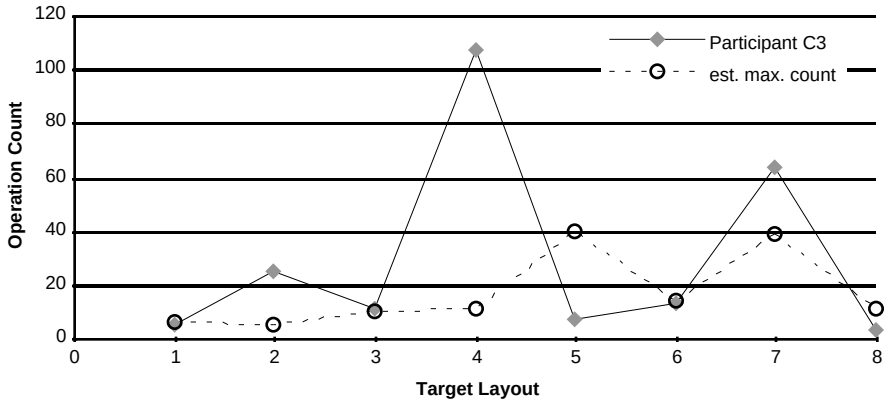

Figure 6-14: Operation count comparison
(data from participant C3 and estimated maximum count of Control Treatment)

5. Assuming the worst case situation where the participant had to check each and every alternative layouts of a problem to locate the target layout, the maximum operation count is equal to the number of alternative layouts.

Table 6-2 through Table 6-5 show the number of operations each participant used to locate target layouts. Each column records operation counts according to the following categorization:

- **C(1)**: the number of operations used to the finding of the target layout of **FIRESTATION** (#1) from the finding of another layout prior to this one
- **C(2)**: the number of operations used to the finding of the target layout of **adm_wing** (#2) from the finding of another layout prior to this one
- **C(3)**: the number of operations used to the finding of the target layout of **dorm_wing** (#3) from the finding of another layout prior to this one
- **C(4)**: the number of operations used to the finding of the target layout of **dayZ_1** (#4) from the finding of another layout prior to this one
- **C(5)**: the number of operations used to the finding of the target layout of **admZ_1s** (#5) from the finding of another layout prior to this one
- **C(6)**: the number of operations used to the finding of the target layout of **mecZ_1** (#6) from the finding of another layout prior to this one
- **C(7)**: the number of operations used to the finding of the target layout of **batZ_1** (#7) from the finding of another layout prior to this one
- **C(8)**: the number of operations used to the finding of the target layout of **dorZ_1** (#8) from the finding of another layout prior to this one

As I have described previously, participants often tried some operations to explore the user interface before the first finding of a target layout. Therefore, the operation count during this period may be larger than the actual count to locate the first target. Since it is difficult to separate the familiarization period from searching, these operation count records are underlined in Table 6-2 through Table 6-5. In addition, columns enclosed by shaded areas contain data that are used to create comparison charts in Figure 6-12 and Figure 6-13.

Table 6-2: Operation count for each sub-task in Control Treatment

Participant	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	C(8)
B2	8	2	5	11	42	5	31	11
B8	3	1	6	12	9	4	5	2
C3	5	25	11	107	7	13	63	3
C5	2	12	6	10	25	4	8	2
C6	23	5	6	9	21	4	4	2
D1	9	9	4	7	38	5	8	2
D7	14	10	11	11	44	13	16	8

Table 6-3: Operation count for each sub-task in Map Treatment

Participant	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	C(8)
B2	3	3	3	4	7	5	8	2
B8	2	2	4	1	1	1	3	1

Table 6-4: Operation count for each sub-task in Map-and-Notepad Treatment

Participant	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	C(8)
C3	5	17	6	4	7	1	1	4
C5[a]	12	1	1	1	2	1	1	1
C6	7	2	4	5	6	2	4	2

a. Data based on an incomplete keystroke log file. The log was interrupted due to a system failure while the participant attempted to corrected an incorrectly marked node.

Table 6-5: Operation count for each sub-task in View Treatment

Participant	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	C(8)
D1	60	2	9	6	62	8	22	12
D7	6	10	9	9	63	8	26	5

Subjective Evaluation

The subjective evaluation was compiled from participants' comments in the post-task interviews. Participants were asked to focus their comments in addressing the five questions described in the "Procedure" section earlier. A summary of the subjective evaluations is presented in Table 6-6. The **Advantage** column shows features that provide sufficient support for the navigation task in each treatment; the **Disadvantage** column lists features that hinder the navigation task; and the **WishList** column shows possible improvements that participants suggested for each treatment.

6.2.3 Discussion

This empirical study is limited in that it examines only one specific type of navigation task that may take place in the SEED-Layout environment. Furthermore, the tutorial sessions completed by each participant utilized a user interface similar to the one in the Control Treatment environment. Therefore it might have affected participants' capabilities to fully exploit the new navigation tools. In addition, participants might show more improvement using new tools (e.g. the 2D Tree View Tool) that provide navigation operations through a style of user interaction similar to the style in the tutorial interface than participants using those tools that are not (e.g. the Multi-layer View Tool). Future studies may address this

Table 6-6: Subjective evaluation summary

Treatment	*Advantage*	*Disadvantage*	*Wish List*
Control	The Layout Problem Manager window (LPM), Design Space window (DSW), and Constituent Hierarchy window (CHW) together provide all the necessary information to complete the task.	Necessary information is distributed in multiple windows. Difficult to understand and differentiate information in the three different hierarchical structures in LPM, DSW and CHW. The concept of alternative problems and their presentation in LPM is confusing.	Being able to identify which subproblem to activate through the layout displayed in Design window (DW). Being able to view sublayouts in DW.
Map-and-Notepad	Necessary information is organized in 2D Tree View window (2DTV). Markers in 2DTV help to verify progresses during the navigation task. My Design Space window (myDS) allows users to construct a diagram that reflects the final layout composition.	The organization of problems and layouts in 2DTV is not easy to understand. The screen is not large enough to display all information legibly at one time in 2DTV. The concept of alternative problems and their presentation in 2DTV is confusing.	Automate the process of drawing super/sublayout relations in myDS.
Map	Necessary information is organized in 2D Tree View window (2DTV). In 2DTV, presenting layouts in graphical icons rather than numbers allows users to search visually. In 2DTV, being able to toggle the display of layouts between graphical icons and numbers is helpful.	The organization of problems and layouts in 2DTV is not easy to understand. The screen is not large enough to display all information legibly at one time in 2DTV. The concept of alternative problems and their presentation in 2DTV is confusing.	Provide filtering utility in 2DTV to manage the overwhelming amount of information and to enable users to focus on desired subset of information. Display the super/sublayout relationship in 2DTV.
View	The multi-layer display shows the layout composition result at all time. The interface is easy to work with; it's more like working on a drawing board.	It may be difficult to navigate through layouts without seeing how many alternatives there are and how they relate to each other.	Provide a option to view the organization of layouts (i.e. a view of solution space).

issue in two ways: to select a different tutorial interface or to eliminate tutorial sessions. For example, a continuation of this pilot study can use another SEED-Layout prototype, which offers different appearance and interaction style, for the tutorials (see Figure 6-15 vs. Figure 6-4). The other method is to select participants from experienced users of the system and thereby eliminates the need for tutorial sessions.

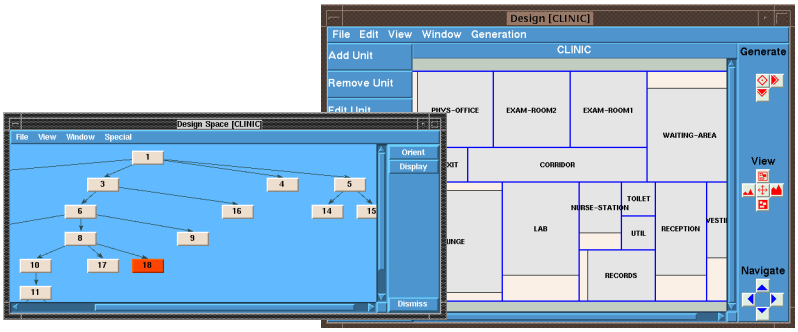

Figure 6-15: Another SEED-Layout prototype

The other issue regarding a weakness in the design of this pilot experiment is that the total search time measure may have been inflated by the learning time taken by individual participants to familiarize the user interface. A continuation of this study may address this issue by designating a learning period prior to the actual tests. During this learning period, the participant may learn and practice navigation operations using a different design problem. Lastly, the navigation tasks themselves were not the most typical for users of SEED-Layout. But the experiment could not address the more normal tasks because it required experienced users working in realistic problems, which were not available for this thesis. The implications are discussed in Section 6.4.

With respect to the landmark concept, two treatments (i.e. Map-and-Notepad Treatment and Map Treatment) provide a marker utility based on this concept. The subjective evaluations indicate that markers are helpful in checking progress during the search task. To limit the complexity of the search task, participants were instructed to use the marker utility when a target layout was found, which can only happen at the end of a search process. Therefore, no data were available to examine the impact of markers in this study. To further understand the effect of markers in SEED-Layout, additional assessments were conducted and the results are described in the next section. In addition, a further elaboration of lessons learned from this empirical study, together with results from other studies, is presented in Section 6.4 at the end of this chapter.

6.3 Marker Utility Assessment

The empirical study described in the previous section does not show (aside from the subjective evaluations by participants) the effect of markers in supporting navigation activities. Markers in SEED-Layout are designed to provide prominent visual cues and to enable quick access to desired and previously visited information. They are expected to play a role that is similar to landmarks in the physical world or bookmarks in the conventional paper-based books. Although some studies, such as Watts (1994) and Darken (1996), have shown empirical evidence that landmarks are important to electronic-world navigation, it is a relatively new and still on-going research area in the HCI community.[6] Therefore, instead of relying on the limited empirical evidence to assume possible advantages of markers in SEED-Layout, in this section, two methods—GOMS analysis and user testing—are used to assess the marker utility.

In a SEED-Layout design session, a designer can explore alternative layout solutions, modify and refine problem specifications, *revisit* promising solutions, and perform many other operations in pursuit of a final layout solution for a design problem. The marker utility is expected to provide assistance when a piece of information, such as a promising layout solution, is to be revisited for reasons such as reevaluation or comparison to other solutions. The two assessment methods in this section are based on the scenario described above: they compare the times a designer may take to visit (first visit) and revisit (second visit) a layout solution in different treatments—without the marker utility or with variations of the utility—of the SEED-Layout environment.

6. The CHI '97 conference, for the first time, organized a workshop on "Navigation in Electronic Worlds." Several position papers, such as Jul (1997), discussed applying the concept of landmarks to improve navigability of electronic worlds.

6.3.1 GOMS Analysis

GOMS analysis generally refers to the use of a usability analysis technique from the GOMS family to evaluate the usability of a software system (see discussions in Chapter 5). The particular GOMS technique employed in this section is based on the GOMS model presented by Card, Moran and Newell (1983), or CPM-GOMS. GOMS analysis may provide many indicators in guiding user interface designs. Although some may consider it costly to use the GOMS analysis for small-scale usability evaluations like the present one, I would like to stress that very little extra effort is required for this study because, as I have illustrated in Chapter 5, the GOMS models are obtained as a result of the software and usability engineering process used to develop the navigation support in SEED-Layout.[7]

This analysis examines the user performance in two tasks:

1. to locate target layouts, and
2. to revisit the target layouts.

The first task is based on the experiment setting described in Section 6.2.1, in which the designer attempts to locate and mark (if the marker utility is available) target layouts (see Figure 6-9) that are partial solutions of the FIRESTATION design problem. The second task is a continuation from the first one during the same design session. The designer is assumed to have completed the first task before starting the second one. Therefore, during the task, the designer can utilize the markers (if available) to revisit target layouts.

A general GOMS model of the two tasks can be formulated as follows:

goal: locate-all-target-layouts
. **goal:** locate-a-target-layout — *…repeat until no more target layouts*
. . **goal:** acquire-target-layout — *…read from task description*
. . **goal:** go-to-target-layout
. . . **[selection goal:** use-step/goto-methods — *…if no markers available*
. . . **goal:** use-marker-methods**]** — *…if markers are available*
. . . verify-target-layout

The use of a marker utility is encoded as selection rules in the GOMS model above: when markers are not available, i.e. in the case of performing the first task or the case of a system not supporting markers, the designer sets the goal to the use-step/goto-methods to locate a target layout using typical "step to node" and "go to node" operations; conversely, when markers are available, i.e. in the case of performing the second task with markers support, the designer sets the goal to the use-marker-methods to revisit a target layout through marker commands.

The detail GOMS analysis examines user performance in these two tasks in three different SEED-Layout user interface treatments: Control Treatment, Map-and-Notepad Treatment, and Map Treatment. These treatments are the same as the ones used in the empirical study discussed previously (for descriptions of these treatments, see "User Interfaces and Tasks" in Section 6.2.1).

Observing the GOMS model of the overall task to locate all target layouts, since the top-level goal locate-all-target-layouts is achieved by completing all sub-tasks of locate-a-

7. Many GOMS models developed along this process are available in Appendix C.

target-layout, the detail analysis will focus on the sub-task go-to-target-layout only. This allows a much more thorough analysis and eliminates redundant analyses of the same type of sub-task. The particular scenario of the sub-task to be examined uses the FIRESTATION design problem and system settings described in Section 6.2.1. The designer is assumed to have located or revisited the target layout of the mecZ_1 subproblem and is proceeding to locate or revisit the target layout of the batZ_1 subproblem (see target layout #7 in Figure 6-9). The detail GOMS analyses also provide time estimates for tasks performed in these different interface treatments; this portion of the analyses is available in Appendix I.

Control Treatment

The process of locating the target layout of batZ_1 begins soon after the designer has located the target layout of mecZ_1. In the Control Treatment environment, the designer first has to activate the batZ_1 subproblem. Once the subproblem is active, its associated solution space is displayed; this allows the designer to examine the space and select candidate target layouts. Finally, the designer examines these candidates one-by-one using the step-to-sibling-layout operations until the target layout is located and verified.

The GOMS model of locating the target layout of batZ_1 in the Control Treatment environment is given below:

goal: go-to-target-layout(batZ_1)
. **goal:** use-step/goto-methods *…no markers available*
. . **goal:** go-to-target-problem(batZ_1)
. . **goal:** examine-solution-space
. . . **goal:** adjust-view *…repeat until seeing all nodes*
. . **goal:** go-to-a-layout *…repeat until target layout*
. . . **[selection goal:** use-goto-method
. . . **goal:** use-step-method
. . . compare-layout
. verify-target-layout

This GOMS model can be further expanded to include operation-level behaviors. For instance, the go-to-target-problem(batZ_1) sub-task involves the same operations as those of the use-goto-methods sub-task within the selection rules of go-to-a-layout. A more general purpose GOMS model to encode the direct go-to a problem or go-to a layout operation can be formulated as follows:

goal: go-to-a-node
. **goal:** select-node
. . move-pointer-to-node
. . click-left-mouse-button
. . verify-selection
. **goal:** issue-go-to-command
. . press-right-mouse-button
. . move-mouse-to-goto
. . verify-highlight
. . release-mouse-button
. verify-display

The adjust-view and use-step-method (to go to a layout) sub-task can also be elaborated. Recall the discussion in Chapter 5: these operation-level GOMS models can be maintained

throughout the software and usability engineering process. They can be reused and need not be recreated for each usability evaluation. Many operation-level GOMS models of sub-tasks used in this section, such as adjust-view and step-to-parent/child/sibling-solution (an equivalent task of use-step-method), are recorded in the system specification and design documentation in Appendix C. Combining these GOMS models, one can estimate how much time a designer will take to perform this task of locating the target layout of batZ_1 (assuming that the designer does not make any mistake during the process).

For the task of revisiting the target layout of batZ_1, the GOMS model is changed slightly because the designer may remember the location of the target layout from the previous visit. Having revisited the target layout of mecZ_1, the designer first activates the batZ_1 subproblem. Once the subproblem is active, its associated solution space is displayed; the designer recalls the location of the target layout and activates it.

goal: go-to-target-layout(batZ_1)
. **goal:** use-step/goto-methods *…no markers available*
. . **goal:** go-to-target-problem(batZ_1)
. . recall-target-layout-location
. . **goal:** adjust-view *…repeat until location in view*
. . **goal:** go-to-a-layout
. . . **[selection goal:** use-goto-method
. . . **goal:** use-step-method**]**
. . . compare-layout
. verify-target-layout

Simply by comparing steps in the above GOMS model and in the go-to-target-layout model on the preceding page, one can predict that revisiting a target layout in Control Treatment requires less amount of time than locating the layout for the first time. Nevertheless, the total time predicted for locating the target layout of batZ_1 is 47.25 seconds, and that for revisiting is 21.10 seconds (see Appendix I: Table I-2 and Table I-6). Here one can see the effect of learning; by being able to remember and recall the location of a target layout, the revisit task time is reduced to only 44.7% of the first visit time.

Map-and-Notepad Treatment

In Map-and-Notepad Treatment, a designer, after locating and marking mecZ_1 in the 2D tree view, can examine the solution space of batZ_1 in the same view to determine candidate target layouts. To visit these candidates, the designer uses the direct go-to command to activate the layouts one-by-one.

goal: go-to-target-layout(batZ_1)
. **goal:** use-step/goto-methods *…no markers available*
. . **goal:** examine-solution-space
. . . **goal:** adjust-view *…repeat until seeing all nodes*
. . **goal:** go-to-a-layout *…repeat until target layout*
. . . **goal:** use-goto-method
. . . compare-layout
. verify-target-layout

The GOMS model for this task is very similar to the one in Control Treatment (i.e. the go-to-target-layout model on the preceding page). However, a designer using this treatment can examine the solution space of a problem without having to activate that problem

(which is necessary in Control Treatment). Another difference between the two treatments is that Map-and-Notepad Treatment does not provide stepping operations. The predicted execution time for locating the target layout of batZ_1 in this treatment is 53.10 second (see Appendix I: Table I-7). Furthermore, in order to take advantage of the markers for future activities, the designer should mark the target layout, which is estimated to need additional 7.95 seconds (see Appendix I: Table I-8).

Once a layout is located and marked, the list of markers is available in the Entity-Relationship Diagramming Tool, i.e. the notepad (the My Design Space window), so that a designer can revisit the target layout of batZ_1 simply by activating the marker (see Figure 6-6 for an example of the marker list).

goal: go-to-target-layout(batZ_1)
. **goal:** use-marker-methods *…markers available*
. . **[selection goal:** go-to-marked-layout *…if no marker list*
. . **goal:** go-to-marker *…if marker list available*
. . compare-layout
. verify-target-layout

This GOMS model shows a much lower number of cognitive processes involved in preforming the revisiting task with aids of the marker list. The estimated execution time for this task is 9.35 seconds (see Appendix I: Table I-9), which is only 17.6% of the estimated time for performing the first visit task. Taking the overhead of marking layouts into account, the revisit task still takes only 32.6% of the first visit time.

Map Treatment

In Map Treatment, a designer is expected to use the enhanced 2D tree view because it displays layouts graphically so that the designer can see their schematic configurations without having to activate them. After having located and marked mecZ_1 in the 2D tree view, the designer examines the solution space of batZ_1 in the same view to determine candidate target layouts. A candidate target layout shows a similar spatial configuration as the target layout. To visit these candidates, the designer can use the direct go to command to activate the layouts one-by-one until the target layout is found.

The interactions described above can be modeled as shown below. This GOMS model is identical to the one for locating the target layout in Map-and-Notepad Treatment (shown near the bottom of the preceding page) because in both cases, the same interface tool—2D Tree View—is used. They differ in that only the simple view is available in Map-and-Notepad Treatment, whereas the enhanced view is used in Map Treatment. Since the enhanced view allows a designer to narrow down the number of candidate target layouts, fewer repetitions of the go-to-a-layout sub-task should be needed. The estimated time to complete the task is 40.35 seconds (see Appendix I: Table I-10). Additional 7.95 seconds should be considered for marking the target layout.

goal: go-to-target-layout(batZ_1)
. **goal:** use-step/goto-methods *…no markers available*
. . **goal:** examine-solution-space
. . . **goal:** adjust-view *…repeat until seeing all nodes*
. . **goal:** go-to-a-layout *…repeat until target layout*
. . . **goal:** use-goto-method
. . . compare-layout
. verify-target-layout

Map Treatment provides a marker utility, but without the marker list support as shown in Map-and-Notepad Treatment. Once markers are available, a designer is expected to use the simple 2D tree view to find marked layouts because the display, compared with the enhanced view, requires less screen real estate.

goal: go-to-target-layout(batZ_1)
. **goal:** use-marker-methods …*markers available*
. . **[selection goal:** go-to-marked-layout …*if no marker list*
. . . . **goal:** adjust-view …*repeat until seeing marked node*
. . . . **goal:** go-to-marked-node
. . **goal:** go-to-marker**]** …*if marker list available*
. . compare-layout
. verify-target-layout

Comparing the GOMS model for this task with the one shown near the bottom of the preceding page, one can see very few differences in the number of cognitive processes required to perform these tasks. However, one can predict that the revisiting task requires less time to perform because the designer does not need to adjust the view as often and makes only one move to activate the target layout.

The predicted time to execute this task is 9.35 seconds, which is only 23.2% of the time it takes to accomplish the first visit task (see Appendix I: Table I-11). However, this GOMS model assumes that the designer has changed the 2D tree view display from the enhanced mode (used in the locating target layout task) to the simple display, which is estimated to require additional 5.10 seconds (see Appendix I: Table I-12); but it will only have to be performed once. Taking into account of the overhead of marking layouts and changing the display, the revisit task still takes only 55.5% of the first visit time to complete. Since the overhead time only needs to be considered once during the task of revisiting all target layouts, the estimated time for revisit a target layout in this treatment may require less than 50% of the first visit task performance time.

Summary

The GOMS analyses for the particular task of visiting the target layout of the batZ_1 subproblem predicts that having marker support reduces the time a designer takes to revisit the target layout. The task performance time could be further reduced if the marker support provides, in addition to visual cues on the display, a marker management utility such as the marker list in the Map-and-Notepad Treatment environment. Although the solution space of the batZ_1 subproblem is not a complex space (i.e. it has only 33 alternative solutions and only a single solution branch is explored), the time-saving prediction may hold true, particularly in the case with marker list support, when solution spaces are larger and more complex. In order to examine this prediction, user tests were conduced with two designers.

6.3.2 User Testing

Similar to the empirical study discussed in Section 6.2, the basic form of this user study is to measure performance on navigation tasks, particularly the task of revisiting previously visited nodes, of each participant in several navigation support treatments in SEED-Layout. The independent variable is the user interface implementation that provides a particular type of navigation support. The dependent variable is performance and behavior on specific navigation tasks. Three treatments are used in the study: Control

Treatment, Map-and-Notepad Treatment, and Map Treatment (see "User Interfaces and Tasks" in Section 6.2.1 for detailed descriptions of these treatments). Control Treatment is an environment that does not offer a marker utility; Map-and-Notepad Treatment offers marker utility with a marker list that allows designers to go to a marked node directly from its associated markers in that list; and Map Treatment provides visual icons associated with marked nodes without the additional marker list.

Data collection for navigation tasks was done using execution times, keystroke analysis, and observation notes. As the participant locates targets within each environment, the time was noted to allow analysis of the overall time to complete the task and the time to complete particular sub-tasks within the treatment. While traversing the design space in SEED-Layout, the participant's mouse pointer activities, including the window in focus, were logged. This data was used to analyze a variety of variables associated with the tasks. Post-experiment analysis could then be done by correlating the keystroke log and the participant's actions or relevant comments noted by the experimenter.

Participants and Tasks

Two participants from the previous empirical study (B2 and C6) were invited to take part in this study. One participant was from the Map-and-Notepad Treatment group in the previous study and the other from the Map Treatment group. Since the tasks in this study are considered routine tasks to be performed by designers regularly when using SEED-Layout, no new participants (i.e. new users of the system) were added.

This study uses the FIRESTATION design problem (same as the problem used in the previous study) in all tasks for all participants. Prior to each test, a setup routine was executed to restore a design session. Participants performed tasks of navigating in the space to locate and mark (if the interface treatment provides the marker support) eight target layouts that form a complete solution of the FIRESTATION problem. After successfully locating all target layouts, participants took a 5-10 minutes intermission. The purpose of the intermission is to simulate a work break taken by designers. After the intermission, participants were asked to revisit all of the target layouts.

Procedure

The two participants were considered novice users of SEED-Layout because of their experience in the previous study. The study measured participants' performances in search of a particular layout solution of a design problem and in revisiting the solution at a later time. There are three test treatments, i.e. Control Treatment, Map-and-Notepad Treatment, and Map Treatment. Each task used one of these treatments. The task descriptions are included in Appendix J.

All tasks employed the same design problem. Participants, in all tests, were required to perform two tasks in the following order:

1. to locate eight target layouts (shown in Figure 6-9), and
2. to revisit these target layouts after a break for at least 5 minutes.

The participant started on an initial state that shows a layout of a problem, which was a subproblem of the root problem and an alternative of another subproblem, and proceeded to navigate through different problems and solutions to locate each of the eight target layouts which were shown as hints on the task description sheet. The target layouts were

not numbered and no specific order was required for the search. The search times were recorded. The first task was considered complete once the last target layout had been located.

The participant were asked to take a break once the first task had been completed. During this time the experimenter reset the active state to the initial state described previously. However, the appearance of the work environment, particularly the position of windows on the computer screen and their sizes and viewing contents (except the window that displays the active state), were preserved. The length of the break was kept at least 5 minutes.

After the break, the participant started the second task to revisit each of the eight target layouts. Depending on the interface treatment, the participant would either navigate through different problems and solutions again or use visual markers to access particular problems and solutions. Again, the target layouts were not numbered and no specific order was required for the revisits; the task was considered completed when the last target layout was revisited.

Participants were given 30 minutes to complete each task. However, the task could be discontinued at the participant's request.[8] Nevertheless, participants were encouraged to ask for help from the experimenter when they felt unable to make any progress toward task completion. During the task execution, the participant's mouse actions and the window in focus were recorded automatically. Meanwhile, the experimenter observed and noted the participant's actions, answered questions and provided hints when necessary.

Result

A quantitative data analysis based on the keystroke events recorded during each task was performed. The data presented in this section supports the prediction from the GOMS analysis in Section 6.3.1: using an interface with a marker utility may reduce the information revisit time.

The observed search time for any treatment was considered equal to the elapsed time from the finding of a target layout to the finding of a second target layout. Search time should coarsely measure the efficiency of search during any test. It is expected to be influenced by the complexity of the solution spaces and the availability of markers. Table 6-7 and Table 6-8 show the search time data of the two participants performing all tasks in different treatments.Columns **T(1)** through **T(8)** in these tables record the elapsed time used to find a target layout (ordered according to the number shown in Figure 6-9) from the finding of another target layout prior to this one; for instance, **T(1)** records the time it takes to find target layout #1 and **T(3)** for #3. These times are recorded in minutes and seconds (e.g. 1 minute and 14 seconds are shown as 1'14").

Since the focus of this study is the effect of markers, which are used in the revisit task, Figure 6-16 and Figure 6-17 plot the performance in revisit tasks for different interface treatments in relation to the performance in first visit tasks. The performance ratios of finding each target layout (the performance in revisit tasks as a percentage of the performance in first visit tasks) are shown vertically. Horizontally, each number identifies

8. All participants completed tasks in the allotted time.

Table 6-7: Time measures for all tasks performed by participant B2

Treatment	Task	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	T(7)	T(8)
Control	First Visit	1'26"	0'34"	0'56"	0'37"	1'15"	0'30"	0'37"	0'14"
	Revisit	0'15"	0'18"	0'30"	0'25"	0'17"	0'08"	0'19"	0'12"
Map and Notepad	First Visit	1'36"	5'48"	0'59"	5'31"	1'07"	0'51"	0'39"	0'35"
	Revisit	0'15"	0'20"	0'13"	0'11"	0'10"	0'09"	0'08"	0'08"

Table 6-8: Time measures for all tasks performed by participant C6

Treatment	Task	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	T(7)	T(8)
Control	First Visit	0'26"	1'52"	0'57"	2'53"	0'25"	0'28"	0'30"	0'26"
	Revisit	0'18"	0'38"	0'15"	0'28"	0'32"	0'30"	0'24"	0'14"
Map	First Visit	7'22"	2'55"	2'19"	2'12"	1'55"	1'14"	1'07"	1'31"
	Revisit	0'30"	0'31"	2'01"	0'29"	0'16"	2'22"	0'09"	0'09"

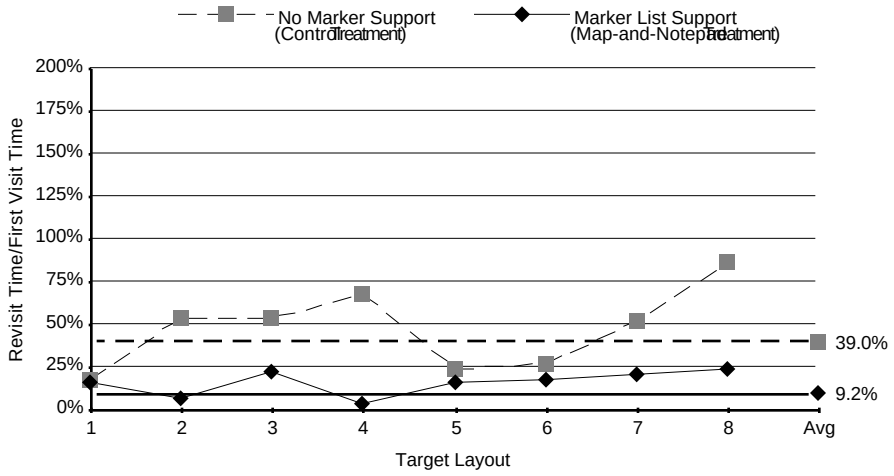

Figure 6-16: Performance of revisit tasks by participant B2

a target layout as numbered in Figure 6-9; the last data point (labeled **Avg**) presents the average performance ratio in the particular interface treatment.

6.3.3 Discussion

Results from both the GOMS analysis and user testing indicate that designers may need less time to revisit a previously visited piece of information regardless of whether a marker utility is available. In the Control Treatment environment, the GOMS analysis predicts the revisit of target layout #7 requires only 44.7% of the first visit time (i.e. a

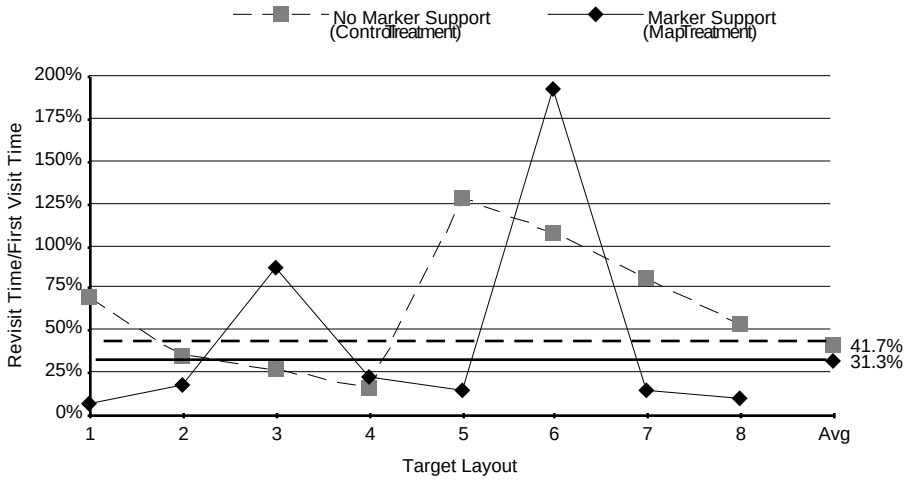

Figure 6-17: Performance of revisit tasks by participant C6

55.3% decrease). The results from the user testing present a similar trend: the two participants show a 48.6% and a 20.0% decrease in time used to revisit this particular target layout compared to their first visits. On average, data from the user testing indicates that revisit times used by the two participants take only 39.0% and 41.7% of their first visit times (i.e. a 61.0% and 58.3% decrease, respectively).

With a marker utility, the GOMS analysis predicts a greater decrease in time needed to revisit target layout #7 when compared to the first visit time. In addition, it predicts the marker utility with a marker list (Map-and-Notepad Treatment) to have the greatest decrease in revisit time, among the three interface treatments. This trend is also observed in the results of the user testing. On average, the two participants spent only 9.2% and 31.3% of the first visit time for their revisit task (a 91.8% and 68.7% decrease, respectively), and the largest decrease occurred in the Map-and-Notepad Treatment environment.

6.4 Lessons Learned

In this case study, I have described a set of navigation tools in SEED-Layout. These navigation tools introduce several novel supports to facilitate navigation in generative design systems, such as the marker utility and the multi-layer display. These tools are developed based on the design space model described in Chapter 3, and they implement the basic navigation operations presented in that chapter. The implementation of these novel tools demonstrates the applicability of the design space model and verifies the validity of this model. In addition, I have presented results from several usability studies of these tools. Although the usability studies—the empirical study and the marker assessment—do not aim at obtaining conclusive result regarding navigational activities in SEED-Layout or the effectiveness of the new navigation tools, the results show a positive trend for the usability of these tools to support the specific type of navigation tasks

investigated. Particularly, several user interface design considerations are suggested by the result of this study. These considerations are discussed in this section.

In the following sections, I will refer to the type of navigation tasks discussed in this study as *target-centered navigation*, i.e. the process of locating target layout solutions that a designer may have in mind. The lessons learned from this case study may only be applicable to supporting the target-centered navigation activities. However, these lessons should not be overlooked when supporting other types of navigation tasks in generative design systems.

6.4.1 Lost in Design Space

It has been observed by the experimenter and mentioned by some participants in the empirical study that they felt lost or were confused about where they were when performing the navigation task. This "lost in design space" phenomenon is reflected in the keystroke data recorded. Particularly, the measure of total wandering time in the Control Treatment environment shows that some participants spent up to 44.5% of the total task time moving seemingly aimlessly between problems and solutions. Similarly, the operation count measure in Control Treatment and View Treatment exhibits that some participants took many operations to reach a target layout when others used very few operations; for example, the participant C3 used over 100 operations to locate target layout #4, whereas the other participants took only 12 or less operations to find the same target layout (see data in the column **C(4)** in Table 6-2).

I have speculated in previous sections that this phenomenon may result from the participant's weak understanding of the structure of design space. The two participants who used the largest number of operations in Control Treatment are also those two who logged longest total wandering time in the same treatment. A review of their tutorial session shows that both of them had finished portions of the tutorial unsupervised, whereas other participants had supervised tutorial sessions throughout. The tutorial session being the only source for participants to learn the structure of design space seems critical in this case. Therefore, although the lack of understanding of the structure of design space may be remedied by providing a global map of the design space, e.g. the 2D Tree View Tool, designing and developing adequate tutorial materials should not be overlooked.

6.4.2 Markers

The marker utility available in the 2D Tree View Tool and Entity-Relationship Diagramming Tool is designed to provide prominent visual features in a space that is populated with many objects; these visual features are similar to landmarks. Additionally, the utility also works like bookmarks to allow users of the system to quickly go to a marked object. The marker utility assessment has indicated that this utility may help to reduce participants' search time when performing target-centered navigation tasks. Moreover, subjective evaluations by participants in the empirical study show that markers are helpful in visually checking progress during this type of task.

Markers can offer more functionality in addition to playing the role of landmarks or bookmarks. Utilities can be developed to operate on markers, e.g. to achieve information filtering or additional information management. The Entity-Relationship Diagramming Tools is an example of such a utility. It allows designers to manipulate markers and

augment them with additional and even personal information that originally is not available in the design system.

6.4.3 Information Integration

A complex design space can be presented through multiple windows, such as the windows used in the SEED-Layout built-in navigation support, each of which shows a portion of the space from a specific viewpoint (see Figure 6-4). It is however generally desirable to have all information integrated in one window. This integration generally reduces the search time and increases participants' subjective satisfactions while performing target-centered navigation tasks. From the functional standpoint, a single window interface does not necessarily reduce the number of operations participants need to reach a target layout. However, the result of the empirical study shows that participants in the map treatments (i.e. Map-and-Notepad Treatment and Map Treatment), where there is one window that displays the complete design space, spent less time wandering (or executed less redundant operations) than in Control Treatment. I would speculate that the single window view, which can present additional relationships between objects that otherwise reside in different windows, improves participants' understanding of the design space and thus reduces their cognitive load during the test (for example, it leads to fewer retrievals from long-term memory).

However, the integrated information can become too much to be presented in a single window. A window with a huge amount of information may hinder the navigation process. This was observed, in the empirical study, in the Map Treatment and Map-and-Notepad Treatment environments. The Map Treatment environment used a graphical presentation of layouts so that participants could quickly determine likely candidates of a target layout without having to go to the layout (i.e. open to view in another window), whereas in the Map-and-Notepad Treatment environment, layouts were shown in numbers. Figure 6-5 shows a comparison of these two types of presentation of the same design space. The enhanced view (Figure 6-5a), when compared to the simple view (Figure 6-5b), requires more screen real estate; therefore, the advantage of graphical layout presentation was offset by frequent scrolling.

The integration of information is particularly relevant for novice users; expert users, on the other hand, may understand the structure of the information well so that the visualization of the space matters less during a target-centered navigation task. Furthermore, to make such support effective, auxiliary utilities to allow information hiding and filtering are necessary.

6.4.4 Navigation as Side-effect

In the empirical study, results from View Treatment show that with sufficient understanding of the structure of a design space, participants can efficiently search for targets without seeing the space. More importantly, participants commented that they were able to experience the system as a design tool rather than a complex environment that contains design solutions. That is, the multi-layer display in this treatment allowed participants to focus on the composition of layouts all the time. Thus, the act of navigation became a side-effect of the design activity. I would again speculate that View Treatment (i.e. the Multi-layer Display Tool) was able to achieve this by supporting the navigation at *street-level*[9] (or content-level) so that participants could focus on the contents rather than the external structure that relates these contents. After all, navigation support is to

improve the usability of a generative design system. Therefore, navigation support should be seamless so that users of design systems can fully focus on their real task—design.

6.4.5 Usability

Nielsen (1993) presents five usability measures: easy to learn, efficient to use, easy to remember, few errors, and subjectively pleasing (see Chapter 5 for a brief discussion). This case study has provided limited evidence indicating that the new navigation tools in SEED-Layout have high usability according to these measures. First of all, they seem easy to learn. In the empirical study, all participants had learned the tool used in the corresponding user interface treatment within a few minutes, and each time, were able to use the tool without having to refer to its user manual. Compared to the built-in navigation support in SEED-Layout, these tools appear relatively efficient to use particularly for the target-centered navigation tasks. This is indicated by the total time measure in the empirical study; almost all participants (except one) were able to accomplish their tasks faster using these tools. Furthermore, the tools seem easy to remember. This is observed when two of the participants in the empirical study were invited to the marker utility assessment study; these participants took only seconds to reacquaint themselves with the user interfaces. However, no conclusion can be drawn to whether these navigation tools effectively prevent designers from making errors since there was no attempt to record errors throughout the usability studies. Finally, these tools seem subjectively pleasing. This is indicated through the individual participant's comments in the post-task interviews. In summary, the 2D Tree View Tool, Entity-Relationship Diagramming Tool, and Multi-layer Tool, compared to the SEED-Layout built-in navigation support, are observed to have higher usability when supporting target-centered navigation tasks.

9. Navigation at street-level within a physical environment is common in our daily wayfinding tasks. Information navigation, however, usually puts the navigator external to the space. This is one of the major differences between a physical environment and an information space. Additional discussions about this is available in Chapter 2.

7. Conclusion

7.1 Summary

Generative design systems promise to complement current CAD tools to assist designers in the early design phases. To date, most research in generative design systems has primarily focused on the generative mechanisms or algorithms. Moreover, the majority of these systems have blackbox-like environments, where designers have little control and understanding of the mechanisms by which solutions are generated. Little attention has been paid to the usability of these systems. As the research progresses, generative design systems, such as SEED-Pro (Akın et al. 1995), SEED-Layout (Flemming and Chien 1995) and SEED-Config (Woodbury and Chang 1995), have become more comprehensive so that they may be suitable for practical use. In order to bring these systems into architectural practice, however, the issue of usability cannot be ignored. Since generative design systems can help designers to generate many design solutions rapidly and to explore different design problem formulations, designers particularly can be overwhelmed by the amount of information they may create during the design process.

This research addresses this issue with emphasis on supporting information navigation in generative design systems. Chapter 2 examines the appropriateness of employing the navigation metaphor in terms of the different applications of this metaphor in many software systems, the nature of navigation behavior, and the similarity and difference between physical environments and information spaces; the goal is to understand potential weaknesses in applying such a metaphor. Chapter 3 and Chapter 4 present a comprehensive approach to provide the navigation support through a design space model that establishes the underlying structure of this dynamic space; a set of basic operations that support navigation activities; and finally a design space navigation framework that encapsulates the model and operations in a software environment. Chapter 5 recommends a unique software and usability engineering process, i.e. the integration of OOSE (Jacobson et al. 1992) technology and the GOMS family of usability analysis (John and Kieras 1994), to ensure the quality and usability of a navigation support. Chapter 6 examines a validation of the approach through the development and evaluation of three proof-of-concept prototype navigation tools (developed using the navigation framework). In summary, this dissertation has demonstrated why the navigation support is necessary, how to provide the support, and what type of user interaction it can offer.

7.2 Contributions

This dissertation makes four major contributions:

- **Identification of requirements for navigation support in generative design systems**
  The premise of this research is that navigation support is necessary in generative design systems in order to improve their overall usability. This research has augmented the premise by defining user requirements—a set of basic navigation operations—for navigation support. Since there exist very few other studies in a similar subject area, the user requirements may be used by future generative system developers to design the human-computer interaction and to evaluate the functionality of new generative design systems.
- **Formulation of a design space model for generative design systems**
  A design space model has been developed to capture all types of information and relationships between different pieces of information in the design space. Although there have been many studies of navigation and wayfinding in computer environments, none has examined an information space with a structure as unique as that of a design space. The design space model can provide guidelines for designing future generative design systems.
- **Development of design space navigation software framework**
  This research takes a pro-active approach toward providing navigation support in generative design systems. That is, in addition to introducing the design space model and basic navigation operations, it proposes a software framework to encapsulate the model and operations using object-oriented technology. Generative design systems can easily adopt the framework to create their own navigation utilities.
- **Implementation and examination of multiple navigation supports**
  To demonstrate the advantage and usefulness of this design space navigation framework, three navigation tools have been implemented. These tools offer a wide variety of user interactions and are very different in their appearances; however, they are founded on a single design space. Moreover, limited pilot usability studies have been conducted to examine these navigation as well as user interface design issues, such as the "lost in design space" phenomenon and the marker utility, that are pertinent to the design of navigation support.

In addition to these major contributions, two areas may benefit from this research. First, a survey of different navigational aids in computing environments has been presented in the background study of this research. It summarizes the navigational aids available and different types of navigation processes or strategies supported by these aids. Since navigation in information space is a new research area, this survey offers basic understanding of related past research efforts. Second, this research has introduced a software and usability engineering process that integrates object-oriented software engineering techniques (Jacobson et al. 1992) with human-computer interaction methodologies (Card, Moran, and Newell 1983; Nielsen 1993). It has been applied throughout the prototype system development process. Such integration brings HCI concerns and formal usability methods into the software development process. The advantage of such integration has been demonstrated particularly in evaluating the usability of the marker utility of different navigation tools.

7.3 Future Directions

This research is the first effort of its kind in generative design systems research. There are four research directions in which to extend the work begun in this dissertation: extension of the design space model, studies of design space visualization, studies of navigational behavior in generative design systems, and studies of information navigation in general.

- **Extension of design space model to capture additional design information**
  In generative design systems, solutions are generated based on certain generative mechanisms. A designer may change a problem formulation or decompose it into subproblems; these changes are made based on the designer's reasoning and decisions. These generative mechanisms and design decisions provide additional information that may help designers to understand the system, to remind themselves of previous decisions, or to review other designers' work. These particular types of information are similar in that they are the driving force to create new nodes in a design space. Although the design space model does not support the maintenance of these types of information, it does provide a place holder for them. That is, they can be considered as intrinsic attributes of a link. From this perspective, the design space model can be extended to incorporate additional design information that is useful for designers.

- **Design space visualization**
  The case study in this dissertation has indicated that information integration, i.e. visualizing information in the design space in one view, is desirable. In the case study, two visualizations of a design space have been presented. However, they are limited to a two-dimensional presentation of a potentially multi-dimensional design space (given that there are five dimensions at each problem node and two at each solution node). Further studies are necessary to discover visualization methods that can efficiently present a design space regardless of how the space may grow.

- **Navigational behavior in generative design systems**
  Behavioral studies are important for identifying and characterizing behavior patterns evident in human-computer interactions. As demonstrated in the case study of this dissertation, such behavioral studies provide important insights into the navigation support that designers need in their interaction with a generative design system. The case study has examined only one type of navigation tasks. Further research is needed to examine different types of navigation tasks and to evaluate the design of navigation support.

- **Further investigations of navigation in information spaces**
  Information navigation has become an important research topic since the introduction of WWW. A recent workshop has compiled a list of research issues (Jul and Furnas 1997). This dissertation has touched on some issues, such as surveying existing navigational aids, identifying a design process for supporting information navigation, and developing a model describing the semantic structure and visualization structure of the information space. In addition, recent publications have discussed issues such as characterizing the navigation process; integrated taxonomy, in terms of the kinds of information spaces, navigation related tasks, and navigational processes and strategies; residue or scent (or trace, hints, cues) effects; and social navigation. For example, the issue of residue or scent effects (Furnas 1997; Pirolli 1997) has invited

studies such as landmarking in information spaces. Furthermore, the issue of characterizing the navigation process has prompted investigations into navigational behaviors in physical environments and information spaces. These issues can be examined within the scope of navigation in generative design systems. Any findings will contribute to the general information navigation research.

8. Bibliography

Ahlberg, C., and B. Shneiderman. (1994). Visual information seeking: tight coupling of dynamic query filters with starfield displays. *CHI '94 Conference on Human Factors in Computing Systems: Celebrating Interdependence* (Boston, MA), 313-317.

Akın, Ö. (1986). *Psychology of Architectural Design*. London: Pion Limited.

Akın, Ö., C. Baykan, and D. R. Rao. (1987). Structure of a directory space: a case study with UNIX operating system. *International Journal of Man-Machine Studies* **26**, 361-382.

Akın, Ö., B. Dave, and S. Pithavadian. (1992). Heuristic generation of layouts (HeGel) based on a paradigm for problem structuring. *Environment and Planning B: Planning and Design* **19**, 33-59.

Akın, Ö., R. Sen, M. Donia, and Y. Zhang. (1995). SEED-Pro: computer-assisted architectural programming in SEED. *Journal of Architectural Engineering* **1**, 4 (December), 153-161.

Akscyn, R. M., D. L. McCracken, and E. A. Yoder. (1988). KMS: a distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM* **31**, 7 (July), 820-835.

Alexander, C. (1964). *Notes on the Synthesis of Form*. Cambridge, MA: Harvard University Press.

Alexander, C. (1979). *The Timeless Way of Building*. New York, NY: Oxford University Press.

Alexander, C., S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York, NY: Oxford University Press.

Appleton, B. (1998). *Patterns and Software: Essential Concepts and Terminology* [WWW document]. URL http://www.enteract.com/~bradapp/docs/patterns-intro.html.

Asimov, M. (1962). *Introduction to Design*. Englewood Cliffs, NJ: Prentice-Hall.

Balasubramanian, V. (1994). *State of the Art Review on Hypermedia Issues and Applications* [WWW document]. URL http://www.isg.sfu.ca/~duchier/misc/hypertext_review/index.html.

Beck, K., and W. Cunningham. (1987). Using Pattern Languages for Object-Oriented Programs. Technical Report CR-87-43. Computer Research Laboratory, Tektronix, Inc.

Beck, K., and R. Johnson. (1994). Patterns generate architectures. *Lecture Notes in Computer Science: ECOOP'94*. 139-149.

Bederson, B. B., and J. D. Hollan. (1994). Pad++: a zooming graphical interface for exploring alternate interface physics. *ACM Symposium on User Interface Software and Technology (UIST '94)* (Marina del Rey, California), 17-26.

Benedikt, M. (1991). Cyberspace: some proposals. In M. Benedikt (Ed.), *Cyberspace: First Step*. Cambridge, MA: The MIT Press. 119-224.

Benford, S., and J. Mariani. (1994). Virtual environments for data sharing and visualisation–populated information terrains. *Interactive Database Systems (IDS '94)* (Lancaster UK, July).

Benyon D., and D. Murray (1993). Adaptive systems: from intelligent tutoring to autonomous agents. *Knowledge Based Systems* **6**, 4 (December), 197-219.

Bier, E. A., N. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. (1993). Toolglass and Magic Lenses: the see-through interface. *SIGGRAPH '93* (Anaheim, CA), 73-80.

Booch, G. (1983). *Software Engineering with Ada.* Menlo Park, CA: Benjamin/Cummings.

Booch, G. (1991). *Object Oriented Design with Applications.* Redwood City, CA: Benjamin/ Cummings.

Brown, M. H., J. R. Meehan, and M. Sarkar. (1993). Browsing graphs using a fisheye view. *INTERCHI '93 Conference on Human Factors in Computing Systems: Bridges Between Worlds* (Amsterdam, The Netherlands), 516.

Brown, W. (1932). Spatial integrations in a human maze. *University of California Publications in Psychology* **V**, 5, 123-134.

Card, S. K., T. P. Moran, and A. Newell. (1980a). Computer text-editing: an information processing analysis of a routine cognitive skill. *Cognitive Psychology* **12**, 1, 32-74.

Card, S. K., T. P. Moran, and A. Newell. (1980b). The keystroke-level model for user-performance time with interactive systems. *Communications of the ACM* **23**, 7, 396-410.

Card, S. K., T. P. Moran, and A. Newell. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Card, S. K., G. G. Robertson, and J. D. Mackinlay. (1991). The information visualizer, an information workspace. *CHI '91 Conference on Human Factors in Computing Systems: Reaching Through Technology* (New Orleans, Louisiana), 181-188.

Carruthers, M. J. (1990). *The Book of Memory: A Study of Memory in Medieval Culture.* Cambridge: Cambridge University Press.

Chimera, R. (1991). Value Bars: An Information Visualization and Navigation Tool for Multi-attribute Listings and Tables. Technical Report CAR-TR-589/CS-TR-2773, Human-Computer Interaction Laboratory, University of Maryland, College Park, MD 20742-3255.

Clarkson, M. A. (1991). An easier interface. *Byte*, 2 (February), 277-282.

Coad, P., D. North, and M. Mayfield. (1995). *Object Models: Strategies, Patterns, and Applications.* Englewood Cliffs, NJ: Yourdon Press.

Coad, P., and E. Yourdon (1991). *Object-Oriented Analysis, (2nd ed.)*. Englewood Cliffs, NJ: Yourdon Press.

Cockburn, A. (1997). Using goal-based use cases. *Journal of Object-Oriented Programming* **10**, 7, 56-62.

Coyne, R. D. (1995). *Designing Information Technology in the Postmodern Age*. Cambridge, MA: MIT Press.

Coyne, R. D., M. A. Rosenman, A. D. Radford, M. Balachandran, and J. S. Gero. (1990). *Knowledge-Based Design Systems*. Reading, MA: Addison-Wesley.

Coyne, R. F., U. Flemming, P. Piela, and R. Woodbury. (1993). Behavior modeling in design system development. In U. Flemming and S. Van Wyk (Eds.), *CADD Futures '93*. Amsterdam: Elsevier Science Publishers. 335-354.

Dahlbäck, N. (1998). *On Spaces and Navigation In and Out of the Computer* [Compressed file]. URL http://sics.se/humle/projects/persona/web/littsurvey/ch2.zip.

Dahlbäck, N., K. Höök, and M. Sjölinder. (1996). Spatial cognition in the mind and in the world. *Eighteenth Annual Meeting of the Cognitive Science Society* (San Diego, CA, July).

Darken, R. P. (1996). Wayfinding in Large-Scale Virtual World. Unpublished Ph.D. Dissertation, The George Washington University, Washington, D.C., USA.

De Hoop, S., P. von Oosterom, and M. Molenaar. (1993). Topological querying of multiple map layers. *COSIT '93* (Marciana Marina, Elba Island, Italy, September), 139-157.

Devlin, A. S. (1976). The "small town" cognitive map: adjusting to a new environment. In G. T. Moore and R. G. Golledge (Eds.), *Environmental Knowing*. Stroudsburg, PA: Dowden, Hutchinson and Ross, Inc. 58-66.

Doerry, E., S. Douglas, T. Kirkpatrick, and M. Westerfield. (1997). Task-centered navigation in a web-accessible dataspace. *AACE/WebNet'97* (Toronto, Canada, November 1).

Dömel, P. (1994). Webmap–a graphical hypertext navigation tool [WWW document]. *The Second International WWW Conference '94: Mosaic and the Web* (Chicago USA, October). URL http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/doemel/www-fall94.html.

Downs, R. M. (1985). The representation of space: its development in children and in cartography. In R. Cohen (Ed.), *The Development of Spatial Cognition*. Hillsdale, NJ: Lawrence Erlbaum Associates. 323-345.

Eastman, C. M. (Ed.). (1975). *Spatial Synthesis in Computer-Aided Building Design.* London, England: Applied Science Publishers Ltd.

Edwards, D. M., and L. Hardman. (1989). 'Lost In Hyperspace': cognitive mapping and navigation in a hypertext environment. In R. McAleese (Ed.), *Hypertext: Theory into Practice* . Norwood, NY: Ablex. 105-125.

Feiner, S., and C. Beshers. (1990). Worlds within worlds: metaphors for exploring n-dimensional virtual worlds. *Third Annual Symposium on User Interface Software and Technology (UIST '90)* (Snowbird, Utah, USA), 76-83.

Flemming, U., and S.-F. Chien. (1995). Schematic layout design in SEED environment. *Journal of Architectural Engineering* **1**, 4 (December), 162-169.

Flemming, U., and R. Woodbury. (1995). Software environment to support early phases in building design (SEED): overview. *Journal of Architectural Engineering* **1**, 4 (December), 147-152.

Fruchterman, T., and E. Reingold. (1990). Graph Drawing by Force-directed Placement. Technical Report UIUCDCS-R-90-1609, Department of Computer Science, University of Illinois at Urbana-Champaign.

Furnas, G. W. (1997). Effective view-navigation. *CHI '97 Conference on Human Factors in Computing Systems: Looking to the Future* (Atlanta, GA), 367-374.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley.

Gibson, W. (1986). *Neuromancer.* West Bloomfield, MI: Phantasia Press.

Gärling, T., A. Böök, and E. Lindberg. (1985). Adults' memory representations of the spatial properties of their everyday physical environment. In R. Cohen (Ed.), *The Development of Spatial Cognition*. Hillsdale, NJ: Lawrence Erlbaum Associates. 141-184.

Golbeck, S. L. (1985). Spatial cognition as a function of environmental characteristics. In R. Cohen (Ed.), *The Development of Spatial Cognition* . Hillsdale, NJ: Lawrence Erlbaum Associates. 225-255.

Gray, W. D., B. E. John, and M. E. Atwood. (1993). Project Ernestine: validating a GOMS analysis for predicting and explaining real-world task performance. *Human-Computer Interaction* **8**, 3, 237-309.

Harada, M. (1997). Discrete/Continuous Design Exploration by Direct Manipulation. Unpublished Ph.D. Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, USA.

Hart, R., and M. Berzok. (1982). Children's strategies for mapping the geographic-scale environment. In M. Potegal (Ed.), *Spatial Abilities: Development of Physiological Foundations*. New York, NY: Academic Press. 147-169.

Hendley, R. J., N. S. Drew, A. M. Wood, and R. Beale. (1995). Narcissus: visualising information. *IEEE Symposium on Information Visualisation (InfoVis '95)* (Atlanta Georgia USA, October 31), 90-96.

Hix, D., and H. R. Hartson. (1993). *Developing User Interfaces: Ensuring Usability through Product and Process*. New York, NY: John Wiley & Sons.

HOOD (1989). HOOD user manual, issue 3.0. WME/89-353/JB. HOOD Working Group, European Space Agency.

Hovestadt, L. (1993). A4 digital building: extensive computer support for building design, construction, and management. In U. Flemming and S. Van Wyk (Eds.), *CADD Futures '93*. Amsterdam: Elsevier Science Publishers. 405-421.

Ingram, R., and S. Benford. (1995a). Improving the legibility of virtual environments. *Second Enrographics Workshop on Virtual Environments* (Monte Carlo, January 31-February 1).

Ingram, R., and S. Benford. (1995b). Legibility enhancement for information visualisation. *Visualization 1995* (Atlanta Georgia USA, October 30-November 3).

Jacobson, I., M. Christerson, P. Jonesson, and G. Övergaard. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading, MA: Addison-Wesley.

Jog, B. (1993). Integration of computer applications in the practice of architecture. In F. Morgan and R. W. Pohlman (Eds.), *Education and Practice: The Critical Interface*. The Association for Computer-Aided Design in Architecture. 89-97.

John, B. E. (1990). Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. *CHI '90 Conference on Human Factors in Computing Systems: Empowering People* (Seattle, WA), 107-115.

John, B. E., and D. E. Kieras. (1994). The GOMS Family of Analysis Techniques: Tools for Design and Evaluation. Technical Report CMU-CS-94-181/CMU-HCII-94-106. School of Computer Science, Carnegie Mellon University.

Jul, S. (1997). Landmarking for navigability [WWW document]. *CHI '97 Workshop on Navigation in Electronic Worlds* (Atlanta, GA, March 23-24). URL http://www-personal.umich.edu/~sjul/papers/chi97_nav.html.

Jul, S., and G. W. Furnas. (1997). Navigation in electronic worlds: a CHI 97 workshop. *SIGCHI Bulletin* **29**, 4 (October).

Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. New York, NY: North-Holland. 135-157.

Kieras, D. E. (1994). GOMS modeling of user interfaces using NGOMSL. Technical Notes, *CHI '94 Conference on Human Factors in Computing Systems: Celebrating Interdependence* (Boston, MA).

Lakoff, G., and M. Johnson. (1980). *Metaphors We Live By*. Chicago: The University of Chicago Press.

Lamping, J., and R. Rao. (1994). Laying out and visualizing large trees using a hyperbolic space. *ACM Symposium on User Interface Software and Technology (UIST '94)* (Marina del Rey, California), 13-14.

Langdon, G. M. (1997). *Architectural CADD Rankings, Ratings, and Reviews* [WWW document]. URL http://www.architecturalcadd.com/reviewsa.html.

Leslie, M. (1992). Computer-aided design in practice: a closer analogue to reality. In F. Penz (Ed.), *Computers in Architecture: Tools for Design*. London, England: Longman Group UK Ltd. 85-95.

Lewis, C., P. Polson, C. Wharton, and J. Rieman. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. *CHI '90 Conference on Human Factors in Computing Systems: Empowering People* (Seattle, WA), 235-242.

Lieberman, H. (1994). Powers of ten thousand: navigating in large information spaces. *ACM Symposium on User Interface Software and Technology (UIST '94)* (Marina del Rey, California), 15-16.

Ligett, D. (1985). RCS Revision Control System on UNIX. Technical Report TR-85-20. Wang Institute of Graduate Studies, Tyngsboro, MA.

Lynch, K. (1960). *The Image of the City*. Cambridge, MA: The Technology Press & Harvard University Press.

Mackinlay, J. D., G. G. Robertson, and S. K. Card. (1991). The perspective wall: detail and context smoothly integrated. *CHI '91 Conference on Human Factors in Computing Systems: Reaching Through Technology* (New Orleans, Louisiana), 173-180.

Mackinlay, J. D., G. G. Robertson, and R. DeLine. (1994). Developing Calendar Visualizers for the Information Visualizer. *ACM Symposium on User Interface Software and Technology (UIST '94)* (Marina del Rey, California), 109-118.

Mandler, J. M. (1988). The development of spatial cognition: on typological and Euclidean representation. In J. Stiles-Davis, M. Kritchevsky, and U. Bellugi (Eds.), *Spatial Cognition: Brain Bases and Development*. Hillsdale, NJ: Lawrence Erlbaum Associates. 423-432.

Mantei, M. M. (1982). Disorientation Behavior in Person-Computer Interaction. Unpublished Ph.D. Dissertation, University of Southern California, Los Angeles, CA, USA.

Merriam-Webster, Inc. (1998). *Merriam-Webster Dictionary* [WWW document]. URL http://www.m-w.com/dictionary.

Mitchell, W. J. (1977). *Computer-Aided Architectural Design.* New York, NY: Van Nostrand Reinhold Company.

Mukherjea, S., and J. D. Foley. (1995). Visualizing the World-Wide Web with the Navigational View Builder. *Computer Networks and ISDN System, Special Issue on the Third International Conference on the World-Wide Web '95* (Darmstadt, Germany, April 10-13).

Nabkel, J., and E. Shafrir. (1995). Blazing the trail: design considerations for interactive information pioneers. *SIGCHI Bulletin* **27**, 1, 45-54.

Newell, A., and H. A. Simon. (1972). *Human Problem Solving*. Englewood Cliffs: Prentice-Hall.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. *CHI '92 Conference on Human Factors in Computing Systems: Striking a Balance* (Monterey, CA), 373-380.

Nielsen, J. (1993). *Usability Engineering*. Cambridge, MA: AP Professional.

Nielsen, J. (1995). *Multimedia and Hypertext: The Internet and Beyond.* Boston, MA: AP Professional.

Norman, D. A. (1990). *The Design of Everyday Things*. New York, NY: Doubleday.

Norman, D. A. (1993). *Things that Make Us Smart: Defending Human Attributes in the Age of the Machine*. Reading, MA: Addison-Wesley.

Passini, R. (1984). *Wayfinding in Architecture*. New York: Van Nostrand Reinhold Company Inc.

Perlin, K., and D. Fox. (1993). Pad: an alternative approach to the computer interface. *SIGGRAPH '93* (Anaheim, CA), 57-64.

Piaget, J., and B. Inhelder. (1956). *The Child's Conception of Space*. Translated by F. J. Langdon and J. L. Lunzer. London: Routledge & Paul.

Pick, H. L., and J. J. Rieser. (1982). Children's cognitive mapping. In M. Potegal (Ed.), *Spatial Abilities: Development of Physiological Foundations*. New York, NY: Academic Press. 107-128.

Pirolli, P. (1997). Computational models of information scent-following in a very large browsable text collection. *CHI '97 Conference on Human Factors in Computing Systems: Looking to the Future* (Atlanta, GA), 3-10.

Pitkow, J. E., and K. A. Bharat. (1994). WebViz: a tools for World-Wide Web access log analysis [PostScript file]. *The First International WWW Conference.* (Geneva, May). URL http://www1.cern.ch/PapersWWW94/pitkow-webvis.ps.

Plaisant, C., D. Carr, and B. Shneiderman. (1994). Image Browsers: Taxonomy, Guidelines, and Informal Specifications. Technical Report CAR-TR-712/CS-TR-3282/ISR-TR-94-39, Human Computer Interaction Laboratory, University of Maryland, College Park, MD 20742.

Pree, W. (1995). *Design Patterns for Object-Oriented Software Development.* Reading, MA: Addison-Wesley.

Rao, R., S. K. Card, H. D. Jellinek, J. D. Mackinlay, and G. G. Robertson. (1992). The Information Grid: a framework for information retrieval and retrieval-centered applications. *ACM Symposium on User Interface Software and Technology (UIST '92)* (Monterey, California), 23-32.

Regnell, B., M. Andersson, and J. Bergstrand. (1996). A hierarchical use case model with graphical representation. *IEEE Symposium and Workshop on Engineering of Computer-Based Systems* (Friedrichshafen, Germany), 270-277.

Rennison, E. (1994). Galaxy of news: an approach to visualizing and understanding expansive news landscape. *ACM Symposium on User Interface Software and Technology (UIST '94)* (Marina del Rey, California), 3-12.

Riehle, D., and H. Züllinghoven. (1995). A pattern language for tool construction and integration based on the tools and materials metaphor. In J. Coplien and D. C. Schmidt (Eds.), *Pattern Language of Program Design.* Reading, MA: Addison-Wesley. 9-42.

Rivlin, E., R. Botafogo, and B. Shneiderman. (1994). Navigating in hyperspace: design a structure-based toolbox. *Communications of the ACM* **37**, 2, 87-96.

Roberts, D., and R. Johnson. (1998). *Evolving Frameworks: A Pattern Language For Developing Object-oriented Frameworks* [WWW document]. URL http://st-www.cs.uiuc.edu/users/droberts/evolve.html.

Robertson, G. G., J. D. Mackinlay, and S. K. Card. (1991). Cone trees: animated 3D visualizations of hierarchical information. *CHI '91 Conference on Human Factors in Computing Systems: Reaching Through Technology* (New Orleans, Louisiana), 189-194.

Rowe, P. G. (1987). *Design Thinking*. Cambridge, MA: The MIT Press.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. (1991). *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall.

Sarkar, M., S. S. Snibbe, O. J. Tversky, and S. P. Reiss. (1993). Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. *ACM Symposium on User Interface Software and Technology (UIST '93)* (Atlanta, Georgia), 81-91.

SEED (1998). *SEED-Layout Requirement Analysis* [WWW document]. URL http://seed.edrc.cmu.edu/SL/SL-start.book.html.

Shlaer, S., and S. J. Mellor. (1988). *Object-Oriented System Analysis: Modeling the World in Data.* Englewood Cliffs, NJ: Yourdon Press.

Shu, E. H., R. Neeman, and G. M. Langdon. (1994). *CADD and the Small Firm '94. (6th ed.).* Boston, MA: The Boston Society of Architects.

Simon, H. A. (1981). *The Sciences of the Artificial. (2nd ed.).* Cambridge, MA: The MIT Press.

Somerville, S. C., and R. J. Haake. (1985). The logical search skills of infants and young children. In H. M. Wellman (Ed.), *Children's Searching: the Development of Search Skill and Spatial Representation*. Hillsdale, NJ: Lawrence Erlbaum Associates. 73-104.

Staples, L. (1993). Representation in virtual spaces: visual convention in the graphical user interface. *INTERCHI '93 Conference on Human Factors in Computing Systems: Bridges Between Worlds* (Amsterdam, The Netherlands), 348-354.

Thorndyke, P. W., and S. E. Goldin. (1983). Spatial learning and reasoning skill. In H. L. Pick and L. P. Acredolo (Eds.), *Spatial Orientation: Theory, Research, and Application*. New York: Plenum Press. 195-217.

Thüring, M., J. M. Haake, and J. Hannemann. (1991). What's Eliza Doing in the Chinese Room?: Incoherent Hyperdocuments–and How to Avoid Them. Technical Report Arbeitspapiere der GMD 533, Gesellschaft für Mathematik und Datenverarbeitung MGH.

Turtiainen, K.-P., and A. Auer. (1995). NetRepreneur [WWW document]. *Third International World-Wide Web Conference.* (Darmstadt Germany, April 10-14). URL http://www.igd.fhg.de/www/www95/proceedings/posters/17/index.html.

Tversky, B. (1993). Cognitive maps, cognitive collages, and spatial mental models. *COSIT '93* (Marciana Marina, Elba Island, Italy, September), 14-24.

Tzonis, A., and I. White (Eds.). (1994). *Automation Based Creative Design: Research and Perspectives*. Amsterdam, The Netherlands: Elsevier Science B.V.

Vicente, K. J., and R. C. Williges. (1988). Accommodating individual differences in searching a hierarchical file system. *International Journal of Man-Machine Studies* **29**, 647-668.

Wasserman, A. I., P. A. Pircher, and R. J. Muller. (1990). The object-oriented structured design notation for software design representation. *IEEE Computer* **23**, 3, 50-63.

Weinand, A., E. Gamma, and R. Marty (1989). Design and implementation of ET++, a seamless object-oriented application framework. *Structured Programming* **10**, 2, 63-87.

Wharton, C., J. Bradford, R. Jeffries, and M. Franzke. (1992). Applying cognitive walkthroughs to more complex user interfaces: experiences, issues, and recommendations. *CHI '92 Conference on Human Factors in Computing Systems: Striking a Balance* (Monterey, CA), 381-388.

Wilson, J. R., and E. N. Corlett (Eds.). (1991). *Evaluation of Human Work: A Practical Ergonomics Methodology*. London, UK: Taylor & Francis.

Wirfs-Brock, R., B. Wilkerson, and L. Wiener. (1990). *Designing Object-Oriented Software.* Englewood Cliffs, NJ: Prentice Hall.

Wood, A. M., N. S. Drew, R. Beale, and R. J. Hendley. (1995). HyperSpace: web browsing with visualisation. *Third International World-Wide Web Conference Poster Proceedings* (Darmstadt Germany, April 10-14), 21-25.

Woodbury, R., and T.-W. Chang. (1995). Massing and enclosure design with SEED-Config. *Journal of Architectural Engineering* **1**, 4 (December), 170-178.

Woodruff, A., P. Wisnovsky, C. Taylor, M. Stonebraker, C. Paxson, J. Chen, and A. Aiken. (1994). Zooming and Tunneling in Tioga: Supporting Navigation in Multidimensional Space. Technical Report UCB 94/48, University of Berkeley.

Woolley, B. (1992). *Virtual Worlds: A Journey in Hype and Hyperreality*. Cambridge, MA: Blackwell.

Yates, F. A. (1966). *The Art of Memory*. Chicago: The University of Chicago Press.

Zarmer, C. L., and C. Chew. (1992). Frameworks for interactive, extensible, information-intensive applications. *ACM Symposium on User Interface Software and Technology (UIST '92)* (Monterey California), 33-41.