CoDeIn

A Knowledge-Based Framework for the Description and Evaluation of Reality-Based Interaction

A dissertation

submitted by

Georgios Christou

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

Tufts University

May 2007

Adviser: Dr. Robert J. K. Jacob

Abstract

This thesis presents Cognitive Description and Evaluation of Interaction (CoDeIn), a framework that allows one to describe, compare, and estimate completion times for tasks that are designed in a variety of interaction styles. Task time estimates based upon evaluations performed with CoDeIn are consistently more accurate than evaluations carried out using existing model-based evaluation methods. This accuracy arises from several sources, including separating and distinguishing between different forms of knowledge necessary for performing the task. This distinction allows -- as not provided by other methods -- the ability to model non-expert task performance by estimating knowledge and its effect upon task completion time.

The accuracy of the CoDeIn methodology is supported by several experiments that compare the predictions produced by CoDeIn to actual measurements and predictions from the GOMSL methodology. To utilize GOMSL, several sub-models must be created, including a model of grasping and a model for iterative shuffling of physical objects.

I conclude that CoDeIn is in these circumstances more accurate and more expressive than GOMSL, due to the ability to account for and manipulate representations of knowledge.

Acknowledgements

I wish to take this opportunity to thank all the people that made this work possible.

I would like to thank Professor Robert J. K. Jacob for his support, encouragement, and belief in me when I decided to work and write this thesis remotely. His guidance and inspiration are what made this work possible, and for that I am forever grateful. I am honored to be one of his students.

I would also like to thank Professor Frank E. Ritter of Penn State University. His help instrumental in clarifying the goals and arguments, as well as defining and analyzing the experiments presented in this work. I thank him for all the hours that he put in reading and commenting many preliminary drafts of this thesis.

Special thanks to Professor Alva L. Couch. His help in the last phase of the writing of this thesis was more than one could hope. I thank him for the many hours of discussion he devoted to help me write precisely and accurately.

A huge debt of gratitude is owed to my father and mother in Cyprus who have constantly pushed me to strive for the best, and for not ever letting me quit. Without them, I would not be who I am today.

But the greatest gratitude is owed to my wife, Nafsika, for her love and support, for her never-ending faith in me, and for her understanding (for not being always there) during the preparation and completion of this thesis.

This work was partly supported by National Science Foundation grant IIS-0414389. We gratefully acknowledge their support.

Table of Contents

List of Tables

List of Figures

CoDeIn

A Knowledge-Based Framework for the Description and Evaluation of Reality-Based Interaction

A dissertation

submitted by

Georgios Christou

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

Tufts University

May 2007

Adviser: Dr. Robert J. K. Jacob

Chapter 1 Introduction

Today a revolution in the field of interaction design is in process. The computer has left its traditional setting, the office, and has entered daily life, by becoming smaller and more ubiquitous. Cell phones and Personal Digital Assistants have become ubiquitous. Even children, younger and younger, are becoming masters of this technology (Marcus, 2006). This revolution is partly the product of a generation of interaction technologies that were brought together by Hutchins, Hollan and Norman (1986) and Shneiderman (1983). Shneiderman called the various technologies together Direct Manipulation (DM), and today they have also acquired the name WIMP, short for Windows, Icons, Menus, Pointer, because of the interaction styles' based on them. This generation of technology made computer use easier and more widespread than before, and has become the nearly universal state of practice in user interface design.

While Shneiderman recognized that the technologies could be brought together under a unified banner (Shneiderman, 1983), Hutchins, Hollan and Norman presented why direct manipulation user interfaces were easier to learn, were more user-friendly, and took less time for one to become an expert in using them (Hutchins et al., 1986). They compared the then fairly recent direct-manipulation style to command-line interfaces, and claimed that because abstractions are depicted visually, and because manipulation of these abstractions is also done visually, the user feels more comfortable in understanding what is being manipulated, and how (Hutchins et al., 1986).

Hutchins et al. (1986) suggested that because the results are shown immediately as graphical representations and not in textual form, users feel a sense of directness in this interaction style.

Today though, we see research that looks toward a new generation of Non-WIMP interaction styles (Jacob, Deligiannidis, & Morrison, 1999; Van Dam, 1997). Non-WIMP interaction styles are based upon new input and output devices that leverage the user's existing knowledge and skills to accomplish tasks (Jacob, 2004). Examples come from areas such as virtual reality (Foley, 1987), augmented reality (Azuma, 1997), ubiquitous computing (Weiser, 1991), tangible user interfaces (Ishii & Ullmer, 1997), affective computing (Picard, 2000), pervasive and handheld interaction (Saha & Mukherjee, 2003; Satyanarayanan, 2001), lightweight, tacit, or passive interaction (Nielsen, 1993), perceptual interfaces (Turk & Robertson, 2000), context-aware interfaces (Schilit, Adams, & Want, 1994), and speech and multi-modal interfaces (Waibel, Tue, Duchnowski, & Manke, 1996). We call this new class of interaction styles Reality-Based Interaction (RBI) styles (Jacob, 2004).

Research on this front, though, is less unified than that for the previous generation of interaction styles. There is yet no common thread to connect these new interaction styles, which makes it difficult to compare designs employing two different interaction styles. Thus, when a design might be suited to a particular interaction style, there is no way that one can choose which style that should be, other than basing decisions on intuition and experience.

The general goal of the research described in this thesis, is to create a framework that allows the description of tasks and then categorize them, according to their easiness of performance. A task, for the purpose of this thesis, is a collection of actions put in chronological order, whose purport is to change the state of the world. This thesis is restricted to the study of tasks in user interfaces.

The framework that is presented in this thesis, named Cognitive Description and Evaluation of Interaction (CoDeIn), is philosophically based upon pragmatic solipsism (Diaper & Stanton, 2004a), which states that the existence of the world is an assumption, so that anything that can be known about the world is a model. This creates the necessity for tools that support the creation of such models, so that understanding about the world can come about.

CoDeIn is one such tool. CoDeIn allows task modeling, based on the knowledge that is required for the performance of the constituent actions of the task. Obviously, CoDeIn is not the only tool that allows the creation of task models (see Chapter 2). The need for using CoDeIn to model tasks will become apparent in the discussions that follow throughout this thesis.

Through task modeling, CoDeIn tries to predict the completion time of the modeled task. But CoDeIn also provides a description of the task through which allows the evaluator to find places where errors may occur during the task's performance. This places CoDeIn in the class of predictive task modeling methods used for the evaluation of user interfaces. The primary goal of CoDeIn, though, is to predict the aforementioned measure in the set of Reality-Based

Interaction (RBI) styles (Jacob, 2004; , 2006). The framework is interaction-style-independent, so that it may cater to all interaction styles that fit under the RBI umbrella. The framework uses the knowledge required for the performance of a task to describe that task. This is done through a diagrammatic notation, which is presented in subsequent chapters of the thesis. The reasoning that guided the creation of this framework was to separate from the user's intention to perform the task from the analysis, description, and evaluation of the task. Rather, the framework was designed to allow the task to be described without regard to a user's desire to perform the task. This mindset brings the framework closer to Diaper and Stanton's suggestion that "the whole concept of goals should be abandoned as unnecessary and thus one of the major sources of confusion about task analysis would be removed" (Diaper & Stanton, 2004b).

Towards this end, Christou and Jacob presented an idea about how seemingly different interaction styles could be put on an equal footing by using task knowledge for evaluation (Christou & Jacob, 2003). The idea was based on the principle of comparing interaction styles based on what knowledge the user would need to possess to use these interaction styles effectively. The intuition behind this idea was that if the user needs to know fewer things in order to use an interaction style effectively, then that interaction style would be "better", in the sense that the user would have to expend less effort in order to learn how to perform a task. Also, if the interaction is natural, then users should not feel that they are using a tool which will give them an answer; it would feel like an extension of the user's intention and natural tools, following Norman's idea of the invisible computer (Norman, 1999b).

1.1 Thesis Overview

The argument of this thesis is that the presented framework allows the analysis of tasks, designed in different interaction styles, and yields quantitative evaluations and comparisons of these tasks that consistently outperform evaluations that are done using existing evaluation methods. The evaluation process is based on the fact that the framework provides structures that specify what is meant by "good" or "bad" interfaces and for specifying users according to their knowledge of task performance in some interface.

The chapters in this thesis are organized as follows. Chapters 2, 3, and 4 will present a review of relevant literature. Chapter 2 presents a discussion on interaction styles in general, followed by more discussion of the direct manipulation interaction style, along with a discussion and definition of RBIs. Because this work is based heavily on Tangible User Interfaces, these will be presented in some detail.

Chapter 3 presents Task Analysis, and presents and contrasts several description and evaluation frameworks and theories to show relevancy to this work. Theories such as Goals, Operators, Methods, Selection rules (GOMS) (Card, Moran, & Newell, 1983) and Task Knowledge Structures (TKS) (P. Johnson, Johnson, Waddington, & Shouls, 1988) are reviewed to show how my work differs and builds upon them. Also cognitive architectures will be reviewed, such as ACT-R (Anderson & Lebiere, 1998) and Soar (Laird, Newell, &

Rosenbloom, 1987), to show how these relate to the description and evaluation of RBIs, as presented in this thesis.

Chapter 4 provides an overview of the theory of affordances and constraints as was introduced in the Human-Computer Interaction field (Andre, Belz, McCreary, & Hartson, 2000; Gaver, 1991; Gibson, 1977; , 1979; Hartson, 2003; Norman, 1999a; , 2002; Overbeeke & Wensveen, 2003; Steedman, 2002).

Chapter 5 presents the definition of a “good” and “bad” interface. Interfaces are evaluated on many levels throughout the HCI literature and this chapter presents yet another way of defining which interfaces are “good” and which are “bad”, based on the knowledge required to use them. The thesis takes the view that a good interfaces may be defined using the amount of knowledge-in-the-head and knowledge-in-the-world, vs. task knowledge required by the interface for users to perform a task. The goal of this chapter is to explicate this procedure, and provide examples of how interfaces may be categorized using these three knowledge sets. It also describes how the theory of affordances and constraints is applied in the described structures of the CoDeIn framework.

Chapter 6 presents the definition of CoDeIn. The nomenclature used in the framework, as well as the constituents of the framework are presented, along with a long example, to show how CoDeIn describes interfaces in different interaction styles. Chapter 6 closes the first part of the thesis that summarizes the theoretical foundations of the framework.

The second part of this thesis presents the description and evaluation power of the framework. Chapter 7 presents an experiment that provides evidence for the framework's capability in calculating the completion time of a task, and compares the framework's performance to the performance of GOMSL (Kieras, 1999), a widely used model-based evaluation method.

Chapter 8 presents another experiment that provides evidence to support the supposition that CoDeIn can model the performance of users ranging from novice to expert, and not just expert users as other model-based evaluation methods assume. Evidence is also provided to show that CoDeIn's predictions about users that gradually become more versed in the experimental task conform to other established methods of performance prediction, such as the Power Law of Practice (Card et al., 1983).

Finally, chapter 9 provides the conclusions of this dissertation and directions for future work. It summarizes the advantages and disadvantages of the framework and provides directions for further validation of this work.

1.2 Background

The goal of the thesis is to develop a cohesive framework for describing human-computer interaction that may be used as a descriptive and evaluative theoretical tool. This tool will allow the description and evaluation of tasks in a user interface that is created in any interaction style. Through a diagrammatic methodology it will allow the calculation of the task's completion time by any level of user, from novice to expert, as well as the identification of steps in the task that may be more difficult for users to perform than others. Finally, the framework will provide a way to depict differences in task performance when a task is designed under different interaction styles.

The work presented in this thesis is multi-disciplinary in the fields of Human-Computer Interaction (HCI), Cognitive Science, Programming Language Theory, and Kinesiology. HCI is a relatively new field that was created out of the need to create better interfaces for users, and is a subset of Human Factors and Ergonomics. It is a multi-disciplinary field that includes such disciplines as cognitive psychology, human factors design, graphics design, and ergonomics. The goal of the field is to provide an understanding of how humans can communicate better with computers. The Association of Computing Machinery (ACM) Special Interest Group on Computer-Human Interaction (SIGCHI) Curriculum for HCI defines the field as '… a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.'

HCI has only recently become a major focus of research, because only recently have computers gained the capabilities that can support modes of interaction other than the command line. But HCI includes more than just research on how communication between humans and computers can become better. HCI research is also oriented towards understanding why and how the interaction between the computer and the user becomes easier and more efficient.

Cognitive Science is another recently created field that studies the normal operation of the human mind. However, Psychology is the field that is usually associated with the study of the human mind, although this study occurs through the study of human behavior ("Psychology"). Cognitive Science, on the other hand, tries to understand the details of how the human mind works not just through the study of human behavior, but by combining findings from different sciences. The Stanford Encyclopedia of Philosophy ("Stanford Encyclopedia of Philosophy: Cognitive Science") defines Cognitive Science as "… the interdisciplinary study of mind and intelligence, embracing philosophy, psychology, artificial intelligence, neuroscience, linguistics, and anthropology. The central hypothesis of cognitive science is that thinking can best be understood in terms of representational structures in the mind and computational procedures that operate on those structures." Cognitive Science employs computer science and other sciences to understand the process of thinking, and not just traditional psychology. This field provides several theories of how people gain knowledge, and how they gradually become experts. It also provides theories for the description and specification of knowledge, and how and in what forms this knowledge may exist in the world or the environment of a task. These theories have a direct impact on this thesis.

Another field that has impacted the work in this thesis is the theory of Formal Grammars. Formal Grammars have been used to represent many computational problems in many fields, from the Theory of Computation (Sipser, 2006), to Linguistics (Harrison, 1978), to Programming Languages (Sebesta, 2006), to HCI (Payne & Green, 1986). Statecharts (Harel, 1987) and Augmented Transition Networks (ATNs) (Woods, 1970) have influenced the creation of a diagrammatic notation for the design of task models. ATNs are graph theoretic structures that are used in defining formal languages (Woods, 1970). Statecharts are extensions to conventional state diagrams that allow representations of hierarchy, communication and concurrency. Statecharts also allow viewing descriptions at several levels of detail, which is part of what inspired the creation of the diagrammatic notation presented in this thesis.

Kinesiology is the scientific study of human movement. While kinesiology mostly studies movement to understand its relation to health through scientific studies of human motion, this field provides models that can predict limb and body movement times very precisely. Kinesiology studies movement and function of the human body both at the behavioral level, as well as at the neurophysiological level. Work in this thesis studies human motion as it is performed for the completion of tasks in RBIs. Because RBIs allow the use real-world actions, such as pointing and grasping, models developed to predict the completion time of these motions become particularly relevant during the prediction of completion times of task performance.

1.3 Motivation

The motivation for this thesis came from the study of Reality-Based Interaction (RBI) styles. So far, next generation interaction styles seem disconnected and more difficult to define than any previous interaction style. The diversity covered by Reality-Based Interaction styles is so great that there does not seem to be a unifying principle or a common theoretical basis with which to connect, for example, VR with TUI, even though both share many characteristics. VR provides a simulated world where users may act, and TUI provides physical manifestations of data in the real world. In both cases though, users act on physical (or virtualizations of physical) artifacts to perform a task in a world. But even though users may perform the same actions in both interaction styles, there is only one explicit attempt (Jacob, 2004; , 2006) to find such cases and explain the phenomena that govern them.

This attempt has identified some threads that may allow for the connection of these seemingly completely distinct interaction styles. First, all RBI styles are based on the fact that interaction should occur more naturally, using as a model the way interaction happens in the real world. Second, users seem to be able to use interfaces that are created in RBI styles more easily and without knowing much about how to work with a computer, than contemporary and previous generation interaction styles. Third, users seem to feel less afraid of the machine if the interaction style used to communicate provides actions that simulate real world actions (Shneiderman, 1998).

Figure 1.1 summarizes the specific parts from each discipline that have been used in the creation of the theoretical basis of the CoDeIn framework. Cognitive Science has offered Cognitive Task Analysis (Chipman, Schraagen, & Shalin, 2000a), the study of the theory of knowledge transfer (Singley & Anderson, 1989), the theory of automatization (Logan, 1988), and the Model Human Processor (Card et al., 1983). These are used to provide a theoretical substratum for the use of task knowledge in building task models. Kinesiology offered theories of arm movement (Jones & Lederman, 2006; C. L. Mackenzie & Iberall, 1994), used to predict the completion time of pointing and grasping actions, common in RBI. Formal Grammars provided Statecharts (Harel, 1987) and ATNs (Woods, 1970), used in the creation of the diagrammatic notation for creating task models. Last, Human-Computer Interaction provided the context of the problem. Existing evaluation methods in this field need to be extended to be able to evaluate RBI interfaces (Christou, 2007).

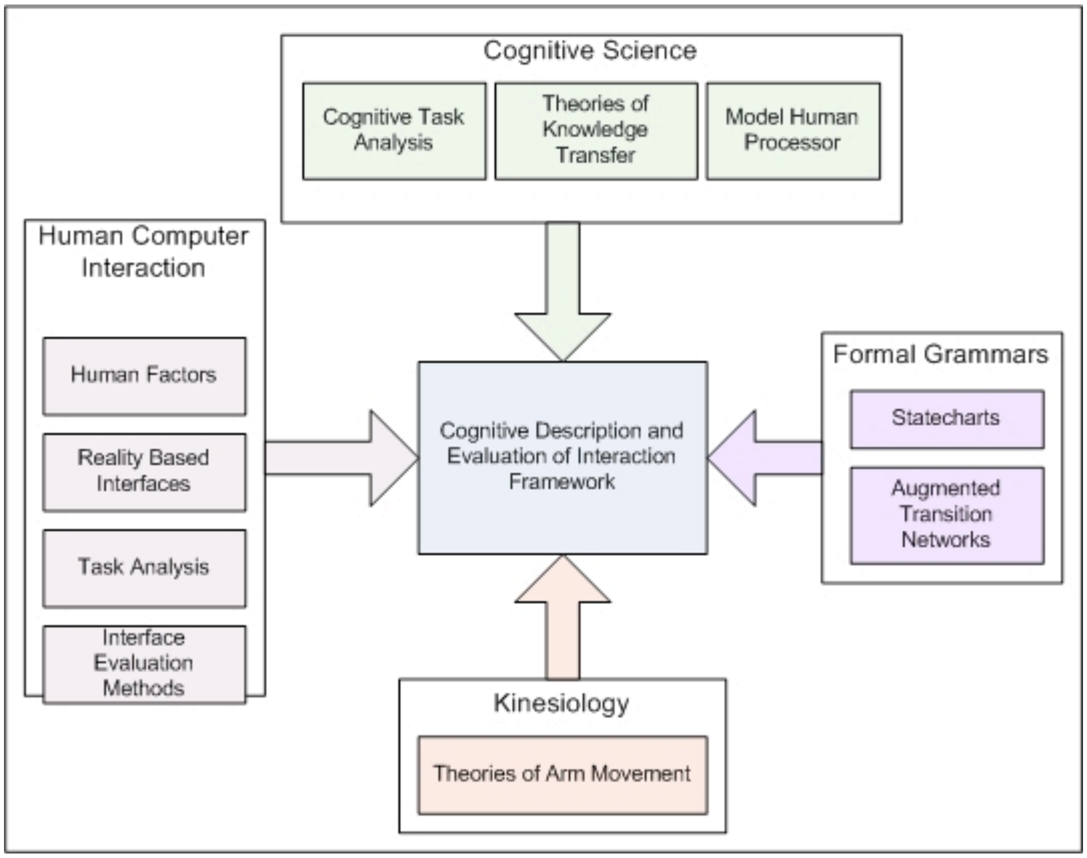

Cognitive Science
Cognitive Task Analysis
Theories of Knowledge Transfer
Model Human Processor
Human Computer Interaction
Human Factors
Reality Based Interfaces
Task Analysis
Interface Evaluation Methods
Cognitive Description and Evaluation of Interaction Framework
Formal Grammars
Statecharts
Augmented Transition Networks
Kinesiology
Theories of Arm Movement

Figure 1.1 Discipline Diagram

1.4 Problem Refinement

The questions that I address in this thesis are the following:

- How can the informal understanding of the nature of a good interface be formalized?
- If we define the nature of a good user interface, can we also define user categories in a formal way?

- Using the formal definition of interface and user, can we perform interface evaluation that provides quantitative data about task completion times and error-rates?

A major hurdle to overcome is the non-existence of a common vocabulary of concepts that allows the description of different parts of the interfaces designed in RBIs. This lack of vocabulary makes it extremely difficult to discuss the advantages and disadvantages of each RBI and to compare parts of interfaces that perform the same function.

CoDeIn provides a solution to this problem. The way that different interface components are named in the CoDeIn framework is designed so that the names are interaction-style-independent. Thus, it is easy to describe any structure in any interaction style, without referral to any interaction-style-specific structures, making the comparison and evaluation of tasks designed in different interaction styles easier.

1.5 Contributions

This thesis tackles a small part of the larger problem of task evaluation. Up to this point, user interfaces have been using a few allowable actions, which allowed a relatively small number of combinations between them. However, in RBIs this is not entirely true. RBIs mimic the real world, and this allows a large number of actions and their combinations, making the evaluation and comparison of RBIs very difficult. Also, the large differences between users' experiences make pinpointing the exact type of user who would use Reality-Based interfaces to also be difficult.

Because RBIs use existing knowledge upon which to base their allowable interactions, the problem of knowledge transfer becomes extremely relevant. Also, because people have differing experiences with different things, some interactions may be more familiar to some people and less familiar to others. Thus, the difference between novice and expert is a second problem that becomes very relevant and may be wider in RBI than in WIMP.

The presented framework deals with the prerequisites and effects of actions done by the user on the environment for the completion of a task. One may therefore presume that the frame problem ("Stanford Encyclopedia of Philosophy: The Frame Problem") is relevant. The frame problem refers to the fact that if an action in the world has specific consequences, the representation set of these consequences should also include all the non-consequences, or the things that remain constant after the action's results in the world. The solution taken in this thesis for the frame problem is to treat each task as a closed world entity, where the knowledge and actions required for the performance of the task are the only things that exist. Therefore, there is no need to represent any of the non-effects, that is, things that do not change in the larger context of the environment in which the task is executed.

As a solution to the questions mentioned in the previous section, I have created a framework that can formally define the user and the task, and based on these definitions, predict task completion times. Furthermore, this framework allows quantitative comparisons to be made between different task descriptions and different classes of users. The proposed framework has been tested in a variety of situations on different interaction styles, to create a realistic view of its capabilities.

The design of the framework proposes a shift in the mentality from goal-based evaluation to task-based evaluation, as per the suggestion by Diaper and Stanton (2004b), who propose that “the whole concept of goals should be abandoned as unnecessary and thus one of the major sources of confusion about task analysis would be removed”.

The contribution of this thesis can be summarized in the points below:

- I have provided an extension to GOMSL to allow the modeling of grasping, by implementing a grasping operator.
- I have explored how Fitts’ Law may be used to predict the completion time of grasping actions. I found that grasping actions may be described by a linear model, rather than a log model such as Fitts’ Law.
- I have proposed a new way of specifying user performance according to their knowledge about the performance of tasks in an interface.
- I have presented a measure for describing the expertise of users as a ratio of the rate of non-expert user performance and the rate of expert user performance of a particular task.
- I have created a new task analysis approach that allows the description, comparison, and prediction of the completion time of tasks designed in different interaction styles. This approach has been developed and tested, and found to be more accurate in the context of new interaction styles or devices than GOMSL.

Chapter 2 Previous Work – Interaction Styles

This chapter presents and defines various interaction styles and defines the term Reality-Based Interaction (Jacob, 2004; , 2006), a central concept in this thesis. The chapter concludes with the presentation and definition of various interaction styles.

2.1 Interaction

Interaction (as defined in the context of this thesis) is the set of the actions of users on a user interface to fulfill some task. This thesis is mostly concerned with two kinds of interaction styles that are most prevalent today: Direct Manipulation Interaction (DM) (Hutchins et al., 1986; Shneiderman, 1983) and Reality Based Interaction (RBI) (Jacob, 2004; , 2006).

The DM interaction style, also called the Windows, Icons, Menus, Pointer (WIMP) interaction style (Shneiderman, 1998), is most prevalent today in commercial applications; the keyboard and mouse necessary interaction devices for this interaction style. Non-WIMP (Jacob, 1995; Jacob et al., 1999) interaction styles or RBI (Jacob, 2004; , 2006) encompass many new ways of interacting with a computer, including tangible user interfaces (TUIs) (Ishii & Ullmer, 1997), virtual reality (VR) (Foley, 1987), augmented reality (Azuma, 1997), ubiquitous computing (Ubicomp) (Weiser, 1991), pervasive and handheld Interfaces (Saha & Mukherjee, 2003; Satyanarayanan, 2001), lightweight, tacit, or passive interaction styles (Nielsen, 1993), perceptual user interfaces (Turk & Robertson, 2000), affective computing (Picard, 2000), context aware interfaces (Schilit et al., 1994), and speech and multi-modal interfaces (Waibel et al., 1996), and any other interaction styles that base interaction on real world actions, skills, and knowledge.

2.2 Direct Manipulation

WIMP or Direct Manipulation (Hutchins et al., 1986; Shneiderman, 1983) was developed through research at Xerox PARC, beginning with the Alto computer (Thacker, 1979) and followed by the Xerox Star interface (J. Johnson et al., 1989). These were the first interfaces that provided the user with the four ingredients that define this interaction style: Windows, Icons, Menus, and the Pointer. The Alto and Xerox Star were also the first computers that ran operating systems that used the desktop metaphor that has persisted in the modern DM interfaces of Microsoft Windows and the Mac OS.

The desktop metaphor presents a virtual office on the screen of the computer. Almost every feature, from artifacts that represent data to artifacts that allow the user to interact with that data, has some sort of metaphorical connotation to a feature of a real office. This allows users to have a feeling of “I have done this before”, which may alleviate some mistrust between the user and the machine (Hall, 1994). The metaphor is also a means by which to make it easier for users to learn how to use the user interface, because the interface is based on a real-world office structure that is already known by users.

In cognitive psychology, it is widely theorized that people learn more easily if they base the new things they learn on things that they already know (Anderson, Reder, & Simon, 1996; Bransford, Brown, Cocking, & National Research Council (U. S.) Committee on Learning Research and Educational Practice, 2000; Singley & Anderson, 1989). This theory of learning is widely known as Constructivist Theory (Bruner, 1966). This theory states that learners construct new ideas or concepts based on what they already know (Bruner, 1966). Because learning occurs on top of existing knowledge that the user already possesses, a smoother introduction to computer usage occurs than when introducing the user to the command line prompt that arose in early time-sharing systems and is still used in the UNIX operating system (Sweller, 1988).

Hutchins, Hollan and Norman (1986) have proposed that DM is better than the Command-Line Interaction style, because of DM's graphical nature and more direct way of interacting with data than Command-Line interaction. Still though, there is loss in flexibility and the ability to execute commands in batch scripts when using DM, as opposed to command-line interaction.

DM bases all its interactions on some kind of pointing device, such as the mouse, the trackball, or even a touch-screen. While Hutchins, Hollan and Norman (1986) have argued that this interaction style allows the direct manipulation of the data on the computer, the manipulation of data is based on manipulating the pointer through the pointing device and understanding the symbolisms behind the desktop metaphor, and the different types of icons that this metaphor provides. The user needs to know how to perform all the necessary actions like selection and dragging, and equate them to real world actions like grabbing and moving. This conceptual mapping happens more easily when one uses the DM style instead of previous generations of interaction styles, but it is still required if the user is to become an expert. But recent technological advances allow the creation of interaction styles that employ metaphors that are closer to the real world structures they try to mimic. For example, instead of using a “virtual office” metaphor, a “place item in box” metaphor may be employed. Such a metaphor may be more accessible to users, because it does not required them to be familiar to the office environment. However, the end goal is to create interfaces that will not need metaphors to show their function to their users. Rather, research on these interfaces tries to make them invisible to their users (Norman, 1999b; Weiser, 1991), either by completely enmeshing them into the real world to augment it for the benefit of the user or by creating a simulation that convinces users that the interface is the real world. These interaction styles are analyzed in the next section.

2.3 Reality-Based Interaction

Today, leaps in technology have allowed for the creation of Non-WIMP (Jacob et al., 1999) or Post-WIMP (Van Dam, 1997) interaction styles. These share the common characteristic that they try to leverage the knowledge and skills that users already possess, through interactions that are similar to real world interactions. This is facilitated by the use of novel input and output devices that either exist enmeshed in the real world, or perhaps simulate the real world.

Because they leverage real world interactions, skills and knowledge, they have also been called Reality-Based Interaction (RBI) (Jacob, 2004; , 2006).

Various Reality-Based Interaction Styles (RBIs), though, do not seem to have any common threads between them, so researchers do not have much common ground upon which to view the whole landscape of these interaction styles. Now, however, there is an attempt to unify the different paths in RBI research (Jacob, 2004; , 2006). This effort also tries to overcome the problem that each interaction style uses its own ways of communication between the user and the computer. Jacob (2004) proposes that by unifying these different paths in research we may find "gaps or opportunities for future development; and perhaps … provide some explanatory power for understanding what makes them better or worse" (Jacob, 2004, p. 2). Jacob (2004) further proposes that the unification of RBIs should be done by using the fact that "the new interfaces increasingly leverage the user's pre-existing abilities and expectations rather than trained skills" (Jacob, 2004, p. 2).

As proposed by Norman (1986), there are two problems in learning to use an interface: the "Gulf of Execution" and the "Gulf of Evaluation". The "Gulf of Execution" is the mental gulf that users must cross to turn their intentions into commands of the interface to perform a task. The "Gulf of Evaluation" is the mental gulf that users must cross to understand the state of the system from the interface's feedback, after a task has been executed. RBIs try to bridge the two gulfs by using *natural interactions*, interactions that occur in the real world in the same way that they occur in RBIs and have the same effects. Because users know how to perform these interactions, they do not have to interpret their intentions into the interface's language. Instead, they can perform these interactions automatically (Logan, 1988). Users also do not have to translate the system's feedback to understand the system state, because they know what effect their actions have, and can thus evaluate the results based on their pre-existing experience in the real world.

The real power of RBIs, however, is that not only can they provide interactions similar to those in the real world, but they can also augment these interactions, to make them more powerful. For example, walking to move from point A to point B in a navigational interface may be augmented with flying, which is not a real world action for humans. Thus, there is a trade-off between the power of the RBI and how closely it models the real world. If the real world is modeled exactly, then there can be no augmented interactions, but if there are many augmented interactions, then the real world is not modeled closely, and the interface moves feels less real. Thus, one might ask:

- How much augmentation is too much augmentation?
- How does the interface communicate to the user the augmentation of a real world interaction?

These questions are complex in their own right, and completely answering them is beyond the scope of this thesis. However, the thesis will touch upon these questions tangentially, as it tries to define a task modeling framework that allows task evaluation.

Because of the diversity and the apparent disparity between RBIs, there is no way for a researcher to use one solution for the description and the evaluation of distinct interfaces or even of tasks performed in these interfaces. This disparity is a hindering factor in the further improvement of the whole field, because solutions to what may be common problems cannot be communicated between sub-communities developing different RBI models. This thesis aims to provide solutions to some parts of this problem. The next sections present some of the most prevalent RBIs to which this thesis will refer in the following chapters.

2.3.1 Virtual Reality

The goal of VR interfaces (Foley, 1987) is to create a complete virtual a simulation of a world, and allows the user to interact with the virtual world, as one would with the real world. Thus, one major goal of VR is to make the experience of the simulated world feel as “real” or immersive to the user as the real world. According to Sutherland (1965) the main objectives that drive VR research are the following: the use of the display as a window to a virtual world through head-mounted displays, the improvement of image generation until the picture looks real, the maintenance of the world model by the computer in real time, the direct and realistic manipulation of the virtual artifacts by the user, and making the virtual world feel and sound real.

Today, VR employs devices that project images either directly in front of the user’s eyes with a head mounted display (HMD), devices that can project an image all around a user (such as immersive 3D environments), such as the one shown in (Arizona State University), or large displays that try to span the user’s entire visual field (Patrick et al., 2000). The use of these devices tries to convince users that the world that the interfaces present is the real world and that the users are now in another (virtual) world that feels as real as the real world. This is the environment where interaction takes place.

VR user interfaces provide the user with ways to interact that are more natural than those that are used in a DM user interface. This is because the user should be able to interact with the virtual world as if it were real. Through the use of augmented interactions, the interface also allows the user to use “super powers” such as flying or x-ray vision to perform certain tasks (Jacob, Leggett, Myers, & Pausch, 1993).

However, even though interactions may be augmented, and new actions may be created that users cannot perform in real life, still the concept of RBI remains; using real-world interactions, or basing interactions on real-world interactions, allows one to create interfaces that seem as real as possible to the user.

VR is mostly used in programs that can be enhanced if they provide a “real” feel to the user. For example, architectural walkthroughs allow clients to feel as if they are walking in the house, and see it the way it would look as if it were already built (Foley, Van Dam, Feiner, & Hughes, 1990). Another example would be in digital museums where users can directly interact with the exhibits, for example by manipulating a coin or a vase (Shiaw, 2003). This enhances the user’s ability to study the exhibits and unlike in a real museum-- the user can actually see the whole item and study it more carefully. Another application is for collaborative work, where users meet in virtual spaces and interact to perform shared tasks (Deligiannidis, 2000).

VR however still faces many challenges, including providing feedback that feels as real as in the real world, modeling worlds convincingly, and choosing an appropriate display type for each application. Other challenges have to do with current graphics technologies such as providing photo-realistic avatars (Brooks Jr., 1999), and haptic technologies, such as providing the real feeling of texture when users touch a surface in the virtual world (Brooks Jr., 1999). These are a few of the problems that need to be solved before VR is viable for mainstream use.

2.3.2 Augmented Reality

Augmented Reality (AR) is very similar to VR, but instead of trying to immerse the user in a simulated world, it superimposes virtual artifacts on the real world. Azuma defines AR as “an environment which includes both virtual reality and real world elements” (Azuma, 1997). AR is positioned as a technology between VR and Ubiquitous Computing, which is discussed in the next section.

The goal of AR is to provide seamless interaction with the real world and with virtual objects that are superimposed on the real world, without requiring a user’s explicit understanding of the difference between the real and virtual objects. AR tries to create an illusion of an augmented world, which to users seems completely real. AR use aims toward enhancing users’ perception of and interaction with the world, thus amplifying users’ abilities and interactions with the world, in order to make the performance of tasks easier (Azuma, 1997).

Because the object is to allow the user to interact in an environment, the most prevalent allowable action in AR is prehension, the action of reaching and grabbing an artifact. Prehension is one of the few ways that users can manipulate their real-world environment. Because AR provides a view of a computer-enhanced world, interaction with the world should not be extremely modified, as it can be in VR. Rather, it should be grounded on what the user can already do in the real-world environment.

However, just like VR, AR also faces problems in implementation. One of the major problems is superimposing virtual objects correctly onto the real world, so that users do not notice the difference between the real and the virtual objects. Because AR uses cameras to capture the scene that the user sees, and then manipulates the captured images to find the user’s point of view, it is possible to mis-position virtual objects, especially if the point of view of the user changes abruptly. But despite problems such as this one, AR is used in many fields, most importantly in computer assisted surgeries, aiding doctors in navigating their instruments in difficult or delicate surgeries, and is very successful in doing so (Grimson et al., 1997).

2.3.3 Ubiquitous Computing

Ubiquitous Computing (Abowd, Mynatt, & Rodden, 2002; Chung et al., 2004; Dey, Ljungstrand, & Schmidt, 2001; Want et al., 1995; Weiser, 1991; , 1993) (Ubicomp, also known as Pervasive Computing) is actually a vision of how interaction with computers should be done, rather than a particular interaction style. Ubicomp, as defined by Weiser (Weiser, 1991), provides interactions with computers with which users never come in direct contact. Ubicomp is the opposite of VR. Whereas VR puts the person inside a world built by a computer, Ubicomp forces the computer to live in the real world along with people. Weiser (1991) calls this the invisible computer, or a computer that the user does not know exists. Whereas VR tries to capture the user's attention completely and create a virtual world in which the user feels completely immersed, UbiComp wants the computer to become completely unobtrusive, and to allow the user to not pay any attention whatsoever to any of the computer's attributes.

Most of the work in UbiComp has stemmed from projects at Xerox PARC, centered around ubiquitous devices "tabs", "pads" and "boards" (Want et al., 1995). These artifacts are created so that they blend in with today's environment, so that the user would not recognize them as parts of a computer interface. Rather, the artifacts are built to seem as if they are part of the environment to which they can be found (Weiser, 1993).

UbiComp relies heavily on the usage of wireless devices, which can communicate with a computer that holds information about the environment and responds to actions that a user might perform with them (Satyanarayanan, 2001).

2.3.4 Tangible User Interfaces

Tangible User Interfaces (TUIs) are an extension of the ideas presented by Fitzmaurice, Ishii and Buxton (Fitzmaurice, Ishii, & Buxton, 1995) and their graspable interfaces. Bricks (Fitzmaurice et al., 1995), employs actual artifacts to represent digital information. By manipulating these artifacts a user can control the digital information. Ishii and Ullmer (1997) extended the ideas of graspable interfaces put forth in Fitzmaurice (1996) to describe TUIs as the interaction style that uses real artifacts (things that the user can touch) to represent and control digital information.

The interaction model that TUIs use is based on physically represented digital information, like a spatially manipulable physical object, that is coupled to some non-graspable representation of digital information, such as graphics and audio (Cheung, 2002; Fitzmaurice et al., 1995; Ishii, Mazalek, & Lee, 2001; Jacob, Ishii, Pangaro, & Patten, 2002). This coupling is achieved by using four characteristics of TUIs: 1. Physical representations are computationally coupled to underlying digital information, 2. Physical representations embody mechanisms for interactive control, 3. Physical representations are perceptually coupled to actively mediated digital representations, and 4. Physical state of tangibles embodies key aspects of the digital state of a system (Ullmer & Ishii, 2000).

Using these four principles, one can create several different relationships between the artifacts that a user can manipulate and the non-graspable digital information. As Ullmer and Ishii (2000) explain, these relationships can be spatial, in the way that was explained previously. They could be constructive, for example, when the graspable artifacts can be put together to create a bigger structure that conveys meaning to the non-graspable information. They can also be associative, where each artifact has some association with non-graspable information, but the associations cannot be integrated into larger scale relationships.

Holmquist et al. (Holmquist, Redstrom, & Ljungstrand, 1999) suggest a taxonomy of artifacts that can be linked to digital information, including containers, tokens and tools. *Containers* are any artifacts that are used to move information from the physical realm to the digital realm and back. *Tokens* are artifacts that encapsulate digital information in their state, while *tools* are artifacts that allow the user to manipulate the information in the system. Shaer et al. (Shaer, Leland, Calvillo-Gamez, & Jacob, 2004) describe TUIs based on the interaction between the graspable artifacts tokens and the constraints on the tokens. This paradigm takes a step towards creating a high-level description language for understanding and specifying TUIs.

2.4 Summary

This chapter presented interaction styles and the two major currents of interaction in existence today, DM and RBIs. The major difference between these two trends in interaction is that the former presupposes that the computer is a multi-purpose artifact that can be used for the performance of tasks, while the latter supposes that the computer is not an artifact that deserves the user's direct attention. Rather, it is an artifact that is better used when embedded somehow in the nature of the task, and enhances the physical properties of the environment in which the task will take place.

This difference forces different types of physical interaction to be required by each interaction approach. DM assumes that there will be a pointing device and a keyboard, through which all interaction with the computer takes place. On the other hand, RBI assumes that users will interact with the environment, just like they would interact with the real world, or that augmented interactions will be based on users' natural abilities. This, in turn, suggests the study of motions used by humans to manipulate their real-world environment. One first obvious candidate for such study is the grasping action. Because the grasping action is one of the most frequent actions that people use to manipulate and act on their environment (Jones & Lederman, 2006; C. L. Mackenzie & Iberall, 1994), it is the action that has the most potential to be used in RBI as well. Therefore, any model-based evaluation method that tries to evaluate RBI should in turn have a modeling mechanism for this action. Evaluation methods, especially predictive model-based evaluation methods, are the subject of the next chapter.

Chapter 3 Previous Work – Task Analysis

3.1 Task Analysis

Task Analysis (TA) is the analysis of what some person, entity or agent is required to do to complete some task. TA is a structured process by which the major parts of a task are identified, and analyzed in order to understand the task's properties. TA is based upon work by Taylor (1911) and Gilbreth (1911) who analyzed physical actions done by manual labor workers. Diaper (2004) defines TA as "the collective noun used in the field of ergonomics, which includes HCI, for all the methods of collecting, classifying and interpreting data on the performance of systems that include at least one person as a system component" (Diaper, 2004, p. 14).

There are many kinds of TA with differing goals. Some TA methods aim to analyze the usability of a user interface (Annet, 2003; Annet & Duncan, 1967; Carroll & Rosson, 1992; P. Johnson et al., 1988; Turner & McEwan, 2004), while other methods create predictive models to test the efficiency of an interface (Baber & Stanton, 1994; Card et al., 1983; De Carolis, De Rosis, & Pizzutilo, 1996; Gray & Altmann, 2001; Hamilton, 1996; Hartson, Andre, Williges, & Van Rens, 1999; Howes & Young, 1991).

The basic methodology that must be followed to perform a TA is the same for most kinds of TA. First, the tasks that users perform are identified and/or classified. This is usually accomplished either through directly observing users while they perform tasks, or by interviewing users about the tasks that they perform. Then three types of requirements analysis are performed: systems requirements analysis, user requirements analysis and task requirements analysis. The three requirements analyses try to match the system under design with the user types and task types that it will support. For example, an accounting system needs to allow an accountant to open the accounting books for a business, and a clerk to input sales data, but it should not allow the clerk to open accounting books for any business. However, the system may not be able to accommodate all tasks and users that a client may want (Greenburg, 2004). After one has determined the user and task types that will be supported by the system, scenarios or models are developed for each task to be supported. Finally, the designer performs a walkthrough of each scenario, and checks whether there are errors in the proposed task solution, while the modeler runs each task model and predicts the performance of each proposed task.

This chapter presents various model-based TA approaches, major TA models, and theories, and provides the context for considering the CoDeIn framework as a TA method.

3.2 Cognitive Task Analysis

Cognitive Task Analysis (CTA) (Chipman et al., 2000a) is a subdiscipline of TA that focuses primarily on the cognitive aspects of a task, such as modeling the perception of an artifact before its use, instead of modeling only the action of its use. CTA is a set of methods used to identify the cognitive skills that are needed to perform a task proficiently. The results of CTA provide information to designers about how to create systems that can be used easily and learned quickly. CTA techniques today have become much more sophisticated than their century old counterparts (Drury, Paramore, Van Cott, Grey, & Corlett, 1987).

CTA is defined by Chipman, Schraagen and Shalin (2000b) as “the extension of traditional task analysis techniques to yield information about the knowledge, thought processes and goal structures that underlie observable task performance” (Chipman et al., 2000b, p. 3). CTA theories provide support for specific methodologies for gathering and analyzing task data. While both CTA and TA model a task in both cognitive and physical realms, TA methods focus more on the physical part of a task, while CTA methods tend to focus more on the cognitive requirements of a task.

CTA begins with a study of the processes involved in the performance of a job, in order to determine the constituent tasks of that job that should be analyzed (Chipman et al., 2000a). This study involves reading material that pertains to the jobs to be analyzed, and interviewing people who perform the job, so that the CTA analyst understands as much as possible the nature of the job and the processes that comprise it. The second step in CTA is to identify knowledge representations that performers of these tasks have (Chipman et al., 2000a). Knowledge representations are the collection of facts and skills that are required for the execution of a task, and they way these are grouped and processed by task performers. The identification of knowledge representations is also known as knowledge elicitation (Boose, 1989; Cook, 1994; Olson & Biolsi, 1991). There are many different types of knowledge elicitation, but the goal of all is to yield the knowledge representations required for the studied tasks. The final step in CTA is to use the knowledge elicitation techniques that apply, based on the chosen CTA theory to apply towards the analysis of the job (Diaper & Stanton, 2004a).

CTA focuses on modeling procedural or “how to do it” knowledge, by focusing on mapping the task to the actions that constitute the task. More recently CTA techniques have focused upon “goal to task” mappings that require less formal analysis techniques. This analysis is done to describe what people already know about a specific task, and the class of tasks that include each specific task. In the following sections we describe two techniques that have influenced the design of the framework that is presented in this thesis.

CTA models the internal representation and processing that occurs within a human agent for the purpose of designing tasks that can be undertaken more effectively by humans. It is based upon applying cognitive analysis techniques to understand which actions in tasks are cognitive in nature, and how the designer can help the user understand these actions and perform or learn them efficiently.

Many CTA methodologies exist, including Goals, Operators, Methods, Selection Rules (GOMS) by Card, Moran and Newell (1983). GOMS is a model-based CTA methodology that is primarily evaluative in nature and pertains to expert, error-free performance. Other CTA methodologies include Task Knowledge Structures (TKS) (H. Johnson & P. Johnson, 1991; P. Johnson & H. Johnson, 1991; P. Johnson et al., 1988), Cognitive Complexity Theory (CCT)

(Kieras & Polson, 1985) and Task-Action Grammars (TAGs) (Payne & Green, 1986).

The work described in this dissertation is positioned between TA and CTA. CoDeIn uses cognitive knowledge structures (CTA) to describe and predict the performance of a physical task (TA).

3.3 Varieties of Task Analysis Methods

Several TA and CTA methods have influenced CoDeIn or are directly used in CoDeIn.

3.3.1 Fitts' Law

Fitts' Law (Fitts, 1954; Fitts & Peterson, 1964) states that the reaction time required for a pointing movement is proportional to the Index of Difficulty (ID) of the movement. The ID is further defined to be the logarithm of the ratio of the amplitude of the movement to the width of the target, as in the following equation: ID = log(2A/W).

Fitts' Law has been used in many kinds of task analyses, and has always provided estimates closed to measured results. Various experiments show that have been performed have shown that Fitts' Law not only holds for pointing movements with the hands, but also for pointing with other limbs, like the head (Andres & Hartung, 1989), the feet (Glasser & Halcomb, 1980), and even eye-gaze (Miniotas, 2000). It also applies to pointing movements with interaction devices such as the mouse (I. S. MacKenzie, 1992a; , 1992b; I. S. MacKenzie & Buxton, 1994).

MacKenzie has provided a different version of ID, which he called the Shannon formulation: ID = log(A/W + 1) (I. S. MacKenzie, 1992a; , 1992b; I. S. MacKenzie & Buxton, 1994). According to MacKenzie this formulation is more accurate than the original definition, and it is always greater or equal to 1, whereas the original definition may be negative (I. S. MacKenzie & Buxton, 1994). MacKenzie has also shown that Fitts' Law also applies to drag-and-drop type movements using the mouse, and not only to pointing movements (I. S. MacKenzie, 1992a). Fitts' Law is used in several predictive model-based approaches, including GOMS, and it is also used in the CoDeIn framework.

While Fitts' Law is one of the few predictive, hard engineering models that HCI has in its arsenal, it can only predict the completion time of target-acquisition-type movements. However, Accot and Zhai (1997) derive a model for 2-dimensional movements from Fitts' Law that applies to 2-dimensional movements, called the Accot-Zhai steering law. This law can be used to model movements that require following a trajectory rather than target acquisition.

Fitts' Law also cannot be used to model grasping actions. Grasping movements are not at all like target-acquisition movements, but rather consist of three different movements, including one the target-acquisition or transport movement, then a hand shaping movement, and finally the actual grasping action (C. L. Mackenzie & Iberall, 1994). While there are multitudes of models of grasping in the existing literature (i.e. Jones & Lederman, 2006; C. L. Mackenzie & Iberall, 1994), there do not seem to be any straightforward models that predict the completion time of a grasping action based on amplitude of the movement, in the manner that Fitts' Law does with the pointing action. Grasping is a fundamental action in RBIs, so that any task analysis method that wants to model RBIs, should be able to model grasping actions as well.

3.3.2 Norman's Theory of Action

Norman, in his introduction to Cognitive Engineering (Norman, 1986), presented a theory of action (NTOA) that includes the following seven stages:

1. Establishing the Goal
2. Forming the Intention
3. Specifying the Action Sequence
4. Executing the Action Sequence
5. Perceiving the System State
6. Interpreting the System State
7. Evaluating the System State with respect to Goals and Intentions.

Norman based these seven stages on two conceptual structures that the user needs to overcome in order to use a system: "The Gulf of Execution" and "The Gulf of Evaluation" (Norman, 1986).

The two Gulfs are presented as explanations as to why users make mistakes when using an interface. The Gulf of Execution refers to the conceptual shift that users need to make in order to translate their goals and intentions into allowable actions of the interface, in order to perform the required task. This Gulf occurs between steps 2 and 3 of NTOA. When mapping from goals to actions, there is the possibility that a user will choose actions that do not match goals. Because users need to change their goals into specific actions in the language of the interface, however small or large this change may be, it introduces the possibility of users erroneously choosing actions that are not appropriate for the specification of their goals in the interface language. By contrast, the Gulf of Evaluation occurs in another conceptual shift, this time to interpret the system state, by perceiving the feedback of the system. In this case, the Gulf sits between steps 5 and 6 of NTOA. This Gulf represents a different type of error; that of mistranslating the feedback of the system and understanding that something other than what really happened is the result of executing actions that the user chose.

Norman argues that by realizing that these gulfs exist, and by consciously trying to design interfaces that minimize these gulfs, the interfaces become more useable and easier to learn. Hutchins et al. (1986), argue that the reason that DM interaction style is so much better than all of its predecessors is that it makes these gulfs smaller than before. RBIs also attempt to minimize these gulfs by allowing actions that either mimic or extend real-world actions that users already understand (Jacob, 2004; , 2006).

While the problem with NTOA is that NTOA proposes a theoretical framework for treating actions and action sequences, it fails to define a formal framework within upon the action sequences of an interface may be defined and analyzed. NTOA also fails to predict how a “big” gulf may impact the usability of an interface vs. a “small” gulf, because there is no methodology by which to quantify these terms. Because it is a conceptual theory, it has limited use when one’s goal is to produce an evaluation theory with quantifiable, objective results.

3.3.3 GOMS and MHP

Goals, Operators, Methods and Selection rules (GOMS) was developed by Card, Moran and Newell (1983) (CMN-GOMS), to describe expert, error-free performance in the context of a task. GOMS creates a model for a task that can be used for predicting the completion time of the modeled task.

Goals in GOMS are the goals that a user has when trying to accomplish a task, as they would be described in layman's terms. For example, a goal for GOMS could be "delete the word 'they' in the preceding sentence". The operators in GOMS are the actions that one can do in the interface, in order to fulfill the goals, whereas methods are some well-learned actions or sequences of actions comprised of those aforementioned operators, in order to accomplish some goal. The selection rules are rules that a user would use to select between different operators and methods that would allow her to accomplish the same goal, or eventually reach the same goal.

GOMS was created to model expert, error-free performance, and makes the assumption that the operators, methods and selection rules for some task have been learned by its users, have been memorized, and can be recalled and used automatically by users. Even though GOMS can find deficiencies in the design of a user interface or interaction device for the performance of the evaluated task by experts, it can only provide hints as to how to make it more efficient for the expert user. It cannot however, provide any information about the new, inexperienced user, because it does not allow adjustments to its predictions based on the knowledge level of the user using the user interface or interaction device.

GOMS bases the evaluation of a task upon the goals of the user. In contrast, CoDeIn focuses on the knowledge that a user has about how to accomplish a specific goal using a certain interface or device. This allows CoDeIn to be used to analyze and evaluate the behavior of even a new or average user, rather than only the expert user.

GOMS encompasses task-analysis techniques that are related to models of human information processing. There are many different flavors of GOMS, with the simplest one being the Keystroke-Level Model (KLM) (Card & Moran, 1980), CPM-GOMS (B. E. John, 1988) and GOMSL (Kieras, 1999).

KLM-GOMS (Keystroke Level Model) was an attempt by Card and Moran (1980) to create a predictive modeling method for user interface evaluation. It was created by measuring the performance of several “primitive operators”, such as how long it takes a user to press a key on the keyboard or how long it takes to move the mouse on the screen to point to a target. Each of these actions is associated with a time prediction, so that when a model is created using this method, the sum of all the time predictions equals the predicted performance of the whole task. KLM-GOMS operators are used in the more comprehensive CMN-GOMS, and two other variations, GOMSL and CPM-GOMS.

GOMS Language (GOMSL) (Kieras, 1999) is the successor of Natural GOMS Language (NGOMSL) (Kieras, 1996), both created by Kieras. GOMSL defines a standard way that GOMS models are written, by creating a standard notation for actions and a subroutine mechanism, so that a top level description of a task may use models of subtasks as method calls to model the top level task. All the GOMS models in this thesis are written in GOMSL, and use GOMSL's rules for the calculation of predicted completion times of tasks, where possible. GOMSL models can also be compiled by a tool called GLEAN (Kieras, 1999) that produces predictions about the completion times of the tasks described using GOMSL automatically.

CPM-GOMS (Cognitive Perceptual Motor GOMS) was created by Bonnie John (1988). This variation distinguishes between the operators that are used in the performance of a task at a very low cognitive level. CPM-GOMS supports cognitive, perceptual, and motor operators, each of which needs to be initialized and then can be used in the modeling of a task. Because an operator may be initialized at the same time as some other operator is executed, the model of a task is described as a PERT-chart. That resultant chart shows the performance of the task over time through initialization and application of different operators.

This variation is different than the others, because by design, it can accommodate actions that are performed in parallel, while the others cannot. Because of this added capability though, it is also a complex method to apply when modeling a task, therefore its creator suggests that it be used only when another GOMS method cannot be used (B. E. John, 2003; B. E. John & Kieras, 1996a; , 1996b). The capability of modeling parallel actions however, is very important in the modeling of tasks performed in RBIs, because as was described in the previous chapter, RBIs allow the performance of actions just like they are performed in the real world. The actions of walking and chewing gum at the same time, for example, would be impossible to model by GOMS methods other than CPM-GOMS. However, due to its complicated structure, John and Kieras (1996b) recommend not to use CPM-GOMS when modeling simple tasks. This creates a catch-22 situation for the modeler who needs to model simple, parallel tasks in a predictive task modeling method.

Much has been written on the differences between the types of GOMS (Baskin & John, 1998; B. E. John & Kieras, 1996a). John and Kieras also suggest how to choose between GOMS methods for the task to be modeled (B. E. John & Kieras, 1996b).

All of the aforementioned GOMS variations are based on the Model Human Processor (MHP) (Card et al., 1983). MHP is a psychological model of a user as an information processing system. The MHP includes a set of "principles of operation" (Card et al., 1983), which govern the operation of the models created in any GOMS technique. MHP provides a framework that allows one to compute a time estimate for many of the actions that a human may perform while executing a task.

MHP makes parallels between the human brain and a processor, with three sub-processors and four types of memory. The processors are the Cognitive Processor, the Perceptual Processor, and the Motor Processor. The two main memory types are: Long-Term Memory and Working Memory, which includes the Visual Image Store and the Auditory Image Store. Figure 3.1 shows a depiction of MHP.

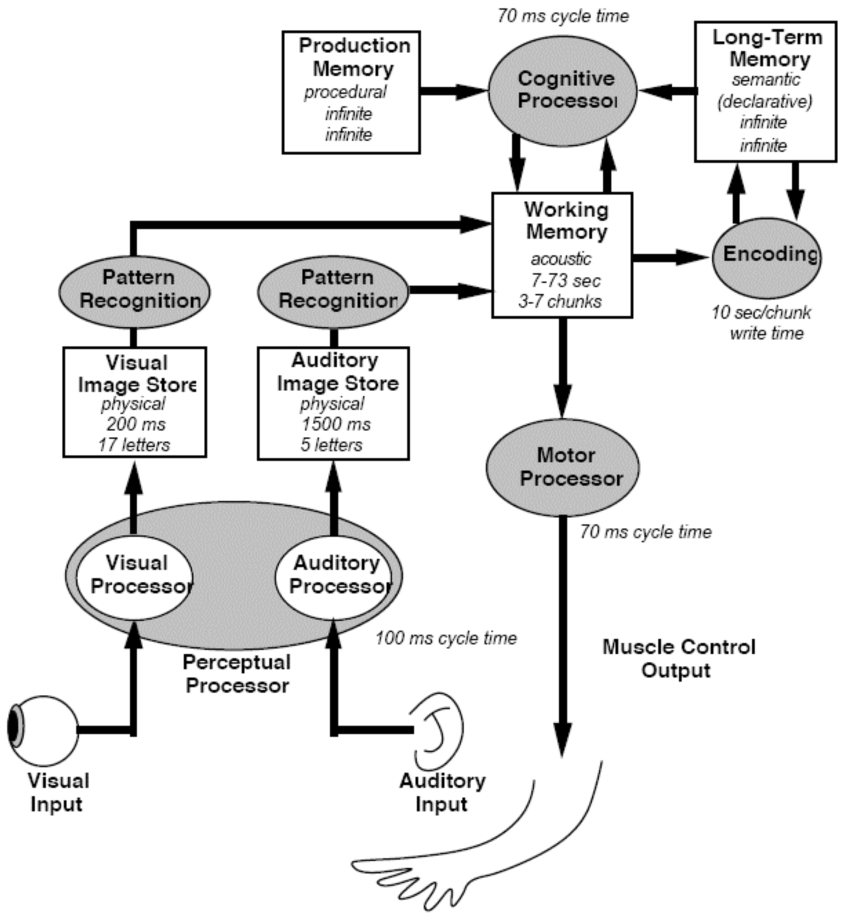


Figure 3.1 The MHP as depicted by Kieras (1999). Used by permission.

There are 10 "principles of operation" proposed by MHP (Card et al., 1983):

1. **Recognize-Act Cycle of the Cognitive Processor**: during each cycle of this processor, working memory contents initiate actions from long term memory, linked to the contents of working memory. These actions, in turn, change the contents of working memory.
2. **Variable Perceptual Processor Rate**: the perceptual processor's cycle time varies inversely to the stimulus intensity, so that the more intense the stimulus, the less time is required to perceive it.

3. **Encoding Specificity**: encoding at the time of perception impacts what and how information is stored. Something that makes an impression is encoded stronger than something that does not. For example, a car accident is encoded with more intensity than a mundane drive on the highway.
4. **Discrimination Principle:** the difficulty of memory retrieval depends on the number of viable candidates in memory that may fit the problem and that can be retrieved.
5. **Fitts' Law** as described previously.
6. **Variable Cognitive Processor Rate:** the cycle time is shorter for more intense tasks, and the cycle time also diminishes with practice.
7. **Power Law of Practice**: the time required for the performance of an action is exponentially proportional to the amount of times the action has been performed.
8. **Uncertainty Principle**: the time to decide increases with the uncertainty about the judgment to be made. For example, choosing between two options that have the same probability to be selected is faster than choosing between 20 options.
9. **Problem Space Principle**: the rational activity in which humans engage to solve a problem can be described in terms of (1) a set of states of knowledge, (2) operators for changing one state into another, (3) constraints on applying operators, and (4) control knowledge for deciding what to apply next.

10. **Rationality Principle:** people act to attain goals through rational action.

3.3.4 Task Knowledge Structures

Task Knowledge Structures (TKS) (H. Johnson & P. Johnson, 1991; P. Johnson & H. Johnson, 1991; P. Johnson et al., 1988) are a theoretical and methodological approach to modeling tasks. The TKS theory assumes that when people learn how to perform a task, the actions and methods used to perform that task are not stored in memory as stand-alone facts. Rather they are stored as grouped knowledge structures that can be remembered and recalled as a coherent whole. This knowledge about tasks is represented in conceptual structures in long term memory, as a coherent structure for each task, called a task knowledge structure. TKS assumes that all of the knowledge about some task is stored in a task knowledge structure, and that this knowledge can be activated when one starts performing a task (H. Johnson & P. Johnson, 1991). TKS also assumes that as people learn to use a system, and while they use a system, they develop more complex knowledge structures. Therefore, users build an action vocabulary (of structures of increasing length) that becomes close to automatic to retrieve.

The theoretical structures to which TKS refers are based upon a cognitive theory that states that the way people fulfill tasks is based on two things: that the task knowledge structures are action sequences that people store in their long term memory, and that people search long term memory to determine whether they know of any action sequence that fulfills the required task. These two assumptions, along with the belief that most behavior is goal-oriented, allow TKS to describe a complete theory of how and why users try to fulfill a task the way that they would.

TKS includes knowledge not only about how to perform a task, but also about objects that are used to accomplish that task. This knowledge allows people to recognize these objects and use them according to the requirements of the task and according to peoples' experience with these objects. Knowledge about these objects then includes knowledge about their "affordances" and "constraints", as explained in Chapter 4.

Therefore, a TKS is a summary representation of the different types of knowledge that are recruited and used during task behavior. The way that TKS is used in usability evaluation is by first deciding upon the "role" to which a person is assigned for a particular task. For example, while one is writing a document on a word processor, one's role could be secretary, student, project manager, etc. A person could also be in any of these roles at any given time; one could be a secretary for one task and a student for another. The theory of TKS assumes that people take many roles through their lives, and that the TKS that they create can be transferred from one role to another. There are tasks that are associated with these roles, and there will be a task knowledge structure for each task that a role may be required to perform (H. Johnson & P. Johnson, 1991; P. Johnson & H. Johnson, 1991).

These TKS may be similar between tasks across roles. This occurs when a person performs a similar task under different roles. In the aforementioned example the person could be a secretary or a student, and in both roles the person knows how to use a word processor. Thus the TKS for the task is similar under both role guises (Hamilton, 1996; H. Johnson & P. Johnson, 1991). In this way, TKS provides an argument for “near transfer of knowledge”, meaning that the knowledge required for a task in one role is very similar to the knowledge required for the task in a different role (Palumbo, 1990).

Within each TKS there is an underlying goal structure that emphasizes the goals and subgoals that one must fulfill in order to successfully complete a task. A goal is the desired state that people must attain according to their conceptualization of the task world as they are performing under a certain role.

TKS models were created as a tool to aid design generation. By modeling user knowledge and using the associated methodology, a designer can use the theory to generate design solutions for interactive systems (Hamilton, 1996). This is in contrast to other task analysis methodologies such as GOMS (Card et al., 1983), which is purely an evaluative modeling method.

TKS models are manipulated through two guiding principles: taxonomic categorization and sequential dependency. The taxonomic categorization principle is that objects that are similar are grouped together, and that users create higher level declarative knowledge structures that allow them to remember action for the group rather than specific objects. The sequential dependency principle is that if people are to carry out a task, after they perform the first action that will bring them closer to the completion of a task, they will not go through every possible action that they know in order to find the next best action to take. Rather, especially if they have performed the task before, they will have a group of actions stored as a sequence that, when performed, will bring them to the desired result. Like taxonomic groupings, actions are grouped together in a meaningful sequence that allows for the creation of subgoals that will lead to the completion of the task (goal).

These principles were tested and evidence was provided to show that they are indeed correct (Hamilton, Johnson, & Johnson, 1998). Even though Hamilton et al. (1998) talk about objects in TKS and hint at the affordances of objects, the object roles are not explicitly defined in terms of a user interface, nor are the affordances and constraints of these objects included in TKS.

Superficially, TKS and GOMS seem to have much in common. There are theoretical differences, though, that distinguish one from the other. First, GOMS is an evaluative method, and not pertinent to direct design generation. TKS allows the designer to generate new designs to accomplish a task. Also, GOMS does not have a high level methodology with which to model user knowledge, although it is capable of modeling low-level knowledge, such as objects in various types of human memory (St. Amant, Freed, & Ritter, 2005). TKS models represent this knowledge, to better leverage it and allow designers to generate designs that fit better with the knowledge that task performers might have.

Finally, for GOMS, usability is how long it takes an expert to complete a task with error free performance. All the elements of a task are considered additive and one can accommodate for “thinking time”. TKS usability is deemed to decrease depending on the level of support for task procedures (H. Johnson & P. Johnson, 1991; P. Johnson & H. Johnson, 1991).

3.4 Summary

The usability evaluation methods that have been presented in this chapter, are the closest theoretically to CoDeIn. CoDeIn is based on the MHP (Card et al., 1983), and espouses the view of TKS that as action sequences are used, they in turn create stronger connections, which makes them more automatic in nature, until the user becomes an expert in performing them.

However, all of these methods have problems that hinder them from modeling RBIs. One obvious problem with most of them is their inability to model parallel actions, which are common in RBIs and the real world. CPM-GOMS can model parallel actions, but it is very complex by nature, therefore discouraging evaluators from using it.

Another problem with current GOMS-like evaluation methods is that they lack the means to model actions that deviate from DM-type tasks. While GOMS was built and was used to model real-life interactions (Gray, John, & Atwood, 1993), not just DM interfaces, it still lacks several key primitive operators for modeling real-world interactions, such as a prehension operator.

On the contrary, usability evaluation methods that deal with the knowledge of users are solely built for evaluating a different aspect of tasks than the efficiency part of the performance of a task. They model whether the steps taken to perform a task are in the correct order, according to the nature of the task, or whether the performance of the modeled tasks includes superfluous or unneeded steps. Therefore, these types of evaluation methods, such as TKS, are not built for the predictive evaluation of the performance of tasks. This means that while these evaluation methods may model RBIs, they cannot provide any quantifiable performance measures, in order to compare different implementations of the same task.

Chapter 4 Previous Work – Affordance and Constraints

This chapter provides descriptions for “affordance” and “constraint”, and explains how they are used in the context of this thesis. Affordance is defined first, according to its use in the field of HCI, and then constraint is defined and contrasted to affordance.

4.1 Affordance

The term “affordance” was coined by Gibson (1977; , 1979) to define properties of objects which allow an actor to act upon them. Norman expanded on this concept, and was essentially the first to present the concept of affordance to the field of Human Computer Interaction (Norman, 1988).

Since then, affordance as a term has been used by many designers and researchers, but as Norman (1999a) explains, many of the uses of the term are vague or unclear, and which is what prompted his writing of his article in the Interactions periodical (Norman, 1999a). In fact, there have been many publications that try to elucidate the term (see Hartson, 2003; McGrenere & Ho, 2002 for more).

Many claim that affordance as was presented in the *Psychology of Everyday Things* (Norman, 1988), was not understood correctly by the HCI community. Gaver (1991), Hartson (2003), and McGrenere and Ho (2002) have also provided their own definitions and extensions of the term affordance, in order to help designers design better user interfaces and interaction devices. For example, the user of a button-based user interface can easily read the labels (sensory affordance) to understand what the button does (cognitive affordance) in order to use it correctly (functional affordance) (Hartson, 2003).

The short example shows how one could address the different types of affordances in Hartson's definition to create a better interface. The concept of affordance is indeed useful in HCI because it forces designers to consider about the information that they impart to the user through the design of the user interface or interaction device they design.

4.1.1 Gibson's Affordance

Gibson (1977) coined the term "affordance" to refer to the actionable properties between the world and an actor (whatever that actor may be) (as cited in Norman, 1999a)). Gibson did not intend the term to refer to any property that may be observable by the actor. Rather, he referred to affordances as all of the properties that allow the actor to manipulate the world, be they perceivable or not. Thus, an affordance is a characteristic of the environment that just happens to make environmental artifacts manipulable or otherwise actable upon by someone. Thus, saying that a designer has added an affordance to a device or an interface does not immediately imply that the device or the interface has become more useable, or that the user would be able to sense the affordance in any way that would help her understand the usage of that device or interface. In Gibson's definition, an affordance is not there to be perceived; it just exists and it is up to the actor to discover the functionality that is offered by the affordance.

4.1.2 Norman's Affordance

Norman (1988) appropriated the term affordance from Gibson, but introduced the concept of "perceived affordance", which defines the clues that a device or user interface gives to the user pertaining to the functionality of an object. Norman calls Gibson's affordance "real affordance". Consider a door that opens when pushed on a flat plate that takes the place of the door handle (Figure 4.1b). The design of the flat plate door handle provides the clue that the door is supposed to be pushed on the flat plate to open it. Conversely, the door handle shown in Figure 4.1a presents the conundrum of where the user should push to open the door. While the affordance that hints towards pushing the door to open it is there, there is no affordance that provides a hint as to where to push to open it. Norman (1988) points out that while this is a design problem of the door handle, it results in people thinking that they cannot figure out how to open a door instead.

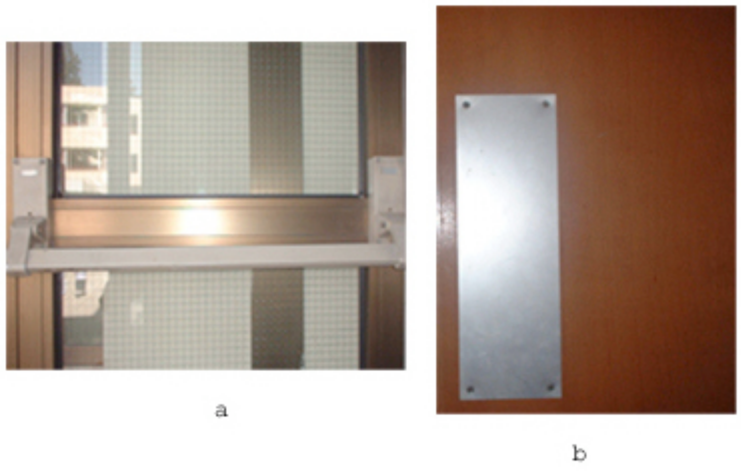


Figure 4.1 Affordance use in two door handle designs

The difference between a perceived affordance and a real affordance (which is the name that Norman uses for Gibson’s definition of affordance) is that the door has the real affordance, which allows it to open in some way. The perceived affordance that the flat panel provides however, is that the door can be opened by pushing it, and therefore the actor receives the way of opening the door through perceiving something from the door’s design. This is the perceived affordance. Norman concludes that well-designed artifacts should have perceived affordances that present the correct clues as to the artifacts’ usage and functionality (Norman, 1988).

4.1.3 Gaver’s Affordance

Gaver (Gaver, 1991) defines affordance by using four different categories instead of Norman’s two. Gaver defines perceptible affordances, false affordances, correct rejections, and hidden affordances, as shown in Figure 4.2. Perceptible affordances are all the affordances for which there is perceptual information for the actor to perceive. This type of affordance would fall under Norman’s perceived affordance definition. If there is information that suggests that an affordance is there when there is none, then that is a false affordance. A hidden affordance is an affordance for which no perceptual information exists. Finally, a correct rejection describes the case when there is no perceptual information and no affordance.

In Gaver’s terms, affordances are a special configuration of properties so that the “physical attributes of the thing to be acted upon are compatible with those of an actor, that information about those attributes is available in a form compatible with a perceptual system, and (implicitly) that these attributes and the action they make possible are relevant to a culture and a perceiver" (Gaver, 1991). In fact, Gaver united the two concepts of a real and a perceived affordance, and named the whole system of the property of an object and the perceivability of that property as the affordance.

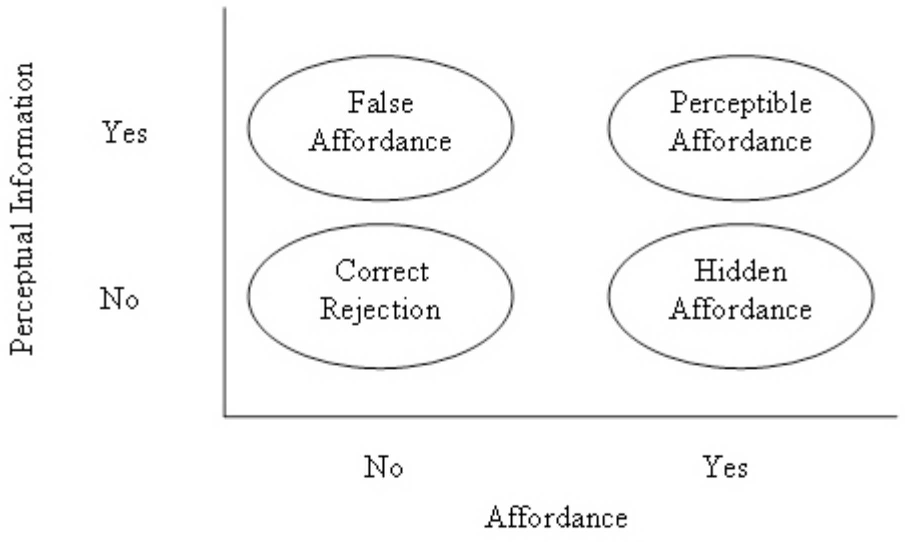


Figure 4.2 Affordances as defined by Gaver (Gaver, 1991)

4.1.4 Hartson's Affordance

Hartson used the concept of affordances to create the User Action Framework (UAF) (Hartson et al., 1999; Hartson, Siochi, & Hix, 1990). He used the concept by basing it on Norman's definition but also redefining it, to make the distinction between each type of affordance that can be encountered more clear. Like Gaver (1991), he refers to four different types of affordances: Physical, Cognitive, Sensory, and Functional (Hartson, 2003).

A physical affordance is a feature of the artifact that allows the actor to do something with it. A cognitive affordance is the information that allows the actor to realize that the physical affordance is there. A sensory affordance is the information that the actor gets before it is processed at a cognitive level, and a functional affordance is the usefulness that the physical affordance gives to the actor (Hartson, 2003). The metal plate on the door from the previous example can be used to elucidate the differences between each affordance type that Hartson proposes. The physical affordance of the plate is the feature which allows the placement of the hand of the user on the door, so that the user can open the door. The cognitive affordance is the combination of information from the user's knowledge and the appearance of the plate that allows the user to realize whether the door is opened by pulling or pushing. If we assume that the metal plate has "Push" engraved on it, then the clarity of the lettering, the size and shape of the letters that allow the user to clearly make them out is the sensory affordance. Finally, the functional affordance is the placement of the plate at the correct position on the door as to allow for the easiest opening of the door if the user pushes on the metal plate.

Hartson goes on to propose the UAF, a framework for designing systems and artifacts based on the four types of affordances that he proposes (Andre et al., 2000; Andre, Hartson, Belz, & McCreary, 2001; Hartson et al., 1999; Hartson et al., 1990). The UAF allows a designer to describe a user's actions in an interface, and the interface's feedback to these actions. Thus, it allows for specification of interfaces, and evaluation of tasks performed in them, at the action level. It also provides mechanisms to describe tasks at a subtask level, rather than directly at an action level. However, the UAF was created specifically for the description of DM interfaces, and therefore currently it lacks the ability to specify RBIs (Hartson et al., 1990).

4.2 Constraints

Constraints are mentioned in many places in the literature (Christou, 2005; Christou & Jacob, 2005; Norman, 1986; , 1988; , 1993; Shaer et al., 2004) as giving helpful clues to the user of an interaction artifact, about how the artifact may and may not be used (Norman, 2002). Constraints are attributes of artifacts that limit the way the artifacts can be used. Consider the example of an elevator button. When pushing the button, it can only be pressed so far. That constraint allows users to realize that they have completed the pressing action, because the full range of the pushing of the button has been reached. Hence, users can infer that they must release the button. In this way, the constraint that limits the range of forward/backward motion of a button also gives a very specific clue to the user; namely that the pushing action is completed. Therefore, the constraint of the button gives a hint to the user as to its usage.

Viewed in this way, constraints are a way that a designer may use to provide knowledge in the world, knowledge that may guide the user to perform an action only in the boundaries that are acceptable for the designed interface. Constraints do not have to be only physical, like the one in the previous example. Norman (2002) mentions many different kinds of constraints, like physical, cognitive, and cultural.

Cultural constraints are constraints that culture imposes on people. For example, red means stop or danger in many Western cultures. The brake lights of all cars are red, and most people in the world realize that when a car’s brake lights are turned on, it means that the car is decelerating, which means that they, in turn, need to decelerate too, or face some kind of danger. This is a constraint that is imposed by culture or rules. Thus, a warning that is accompanied with some kind of red color will be understood more quickly, than if it was accompanied with a green color, which in many cultures is a color that has the exact opposite meaning of red.

Another way to view constraints was proposed by Shaer et al. (2004). Even though their work for the Token and Constraint (TAC) paradigm only refers to TUI interactions, they present an interesting approach as to the view of physical constraints. Constraints in the paradigm are not only viewed as properties of the artifact. Rather, they are viewed as relations between artifacts, where one artifact restricts another artifact’s property in some predefined range. An example of this would be a sphere inside a cube. The cube is viewed as a constraint on the movement of the sphere. The relationship between the two artifacts is defined as a token and constraint (TAC) relationship (Shaer et al., 2004).

4.3 Summary

The theory of affordances and constraints suggests that knowledge can exist not only in the user’s head, but also in the world. While an affordance based on Gibson’s definition is only an attribute of an artifact, all the other definitions of affordance that have been mentioned define affordance as knowledge in the world, which is there to indicate to the user the use of an artifact. This is mostly embodied in Norman’s perceived affordance, which is defined as a hint or a clue in the world towards the use of the artifact that owns the perceived affordance. Therefore, when the user performs a task, the knowledge in the world that pertains to this task can be used to help the user. Unfortunately, as Gray and Fu (2001; , 2004) have pointed out, users tend to disregard perfect knowledge in the world, in favor of knowledge that they have in the head, but which may not be perfect. Thus, users may not utilize knowledge embedded in the task environment when they think that they know how to perform a task, when in fact their knowledge may not be entirely correct. This means that any scheme that plans to evaluate interfaces based on user and world knowledge will have to eventually take this factor into account.

In the following chapters, I present Cognitive Description and Evaluation of Interaction, a predictive, model-based evaluation framework for task analysis, which tries to provide a solution to some of the problems identified in this, as well as in previous chapters.

Chapter 5 Defining Interfaces using User and World Knowledge

The methods described in the previous chapters account for user actions but do not consider user knowledge. Michel Beaudoin-Lafon states that we should strive towards designing interaction, not interfaces (Beaudouin-Lafon, 2004) and this chapter presents a model of interaction based on the knowledge of the user (or knowledge-in-the-head), the presented knowledge in the interface (or knowledge-in-the-world), and the required knowledge for the performance of a task (or task knowledge).

The theoretical basis of the presented model is that a user must have particular kinds of knowledge in order to effectively perform a task using some interface designed in a particular interaction style. Before the development of DM, when command line interfaces were commonly used, the user needed to learn most of the syntax of the interface in order to use it effectively and efficiently. DM introduced the concept of metaphors that allow the user to tap into knowledge already possessed from real world experiences, and understand certain aspects of an interface, without needing to learn everything from scratch (Hutchins et al., 1986). By its nature, however, DM limits the number of actions that a user can perform at a given point in time for a given task. In this way, once the user understands the allowable actions of a DM interface at a particular moment, the issue of effectively using an interface becomes a question of how to put together the allowable actions at that particular moment in order to complete the task.

Looking at this problem from a different angle, DM allows one to reduce the allowable sentences that can be created at any particular moment when using an interface, to those that pertain to the task at hand. This though, can be very limiting, because the user can only follow a certain order in performing a sequence of distinct actions, thereby limiting the flexibility of any interface built in DM. For this reason, hybrid interfaces that include both DM components and command-line components have been created.

5.1 Natural, Simple, and Complex Actions

Understanding what the user needs to learn and already knows requires first the decomposition of the specified task into its constituent actions. There are tasks that have no correspondence to any real world actions, meaning that their description cannot leverage the user's existing knowledge. Thus, these actions must be learned from scratch. At the other extreme, there are "natural actions" that are performed exactly the same whether they are performed in the real world or in the context of a user interface, and that convey the same meaning in both cases. A RBI (Jacob, 2004) is one whose actions are based in reality, but that also leverages computing power to provide solutions to abstract tasks (hence the term *reality-based* and not *reality-like*). Therefore, to make something feel "natural" one has two options: either create a simulation of the real world in which to have the allowable actions in the simulation have the same semantics as in the real world, or to actually use real world actions to carry out computer-based tasks.

One example of a natural action is the “grab” action in Virtual Reality. It possesses the same meaning as in the real world, which is to bring something to the user so that it can be manipulated in some way, and is performed in the same way as in the real world, by reaching for the item in question and placing it in one’s hands. However, the VR grabbing action is based on the same semantics as is the real world grabbing action, but it requires special hardware in order to be performed. Even though the action is based on real-world mechanics, it is not entirely natural, because the user feels or senses the equipment needed to perform the action. A more natural action is the grabbing action in a Tangible User Interface, where the action is performed in the same way as in the real world, carries the same semantics as in the real world, and needs no extra hardware to be performed. In this sense, the TUI grabbing action is more natural than the VR grabbing action.

By thinking about actions in the aforementioned manner, we can place them on a scale of “most natural” to “least natural”, where “most natural” actions mimic every aspect of a real-world action, without any augmentations, while “least natural” actions have no relation to any real-world action whatsoever. However, “most natural” actions, because they carry no augmentations, are not very powerful in conveying extra meaning to a computer interface. Conversely, “least natural” actions, such as script commands in a command line shell, can be extremely powerful. Thus, it seems that the more natural an action, the less power it has in conveying extra meaning to the computer system (Jacob, 2004).

Here I define action as the stimulus-response cycle that begins when the user provides a stimulus to the interface, the interface senses the stimulus, and provides an appropriate response. A "gesture" refers to any stimulus on the part of the user toward the interface that might or might not be perceived by the interface. There are two types of actions: "simple" and "complex".

Simple actions cannot be decomposed into other actions. For example, in DM, simple actions include a left-button click or a right-button click of the mouse. It could be argued that even these actions can be broken down further, e.g. by decomposing the button click into two actions of pushing down the button and then releasing the button, but these two separate events are seldom thought of as separate from each other. Simple actions, though, must have some meaning in the context of the interaction style, and trying to decompose them further might be likened to breaking down the act of writing a character with a pen on a piece of paper into the constituent movements of writing the character. It can be done, but it serves no purpose in the context of writing, because the actions that need to be considered are those that carry some meaning to the performer of the action.

Complex actions, such as the drag-and-drop action in DM, are actions that consist of more than one simple action but are semantically atomic in the sense that only the completed action, with all of its constituents intact, means something to the interface and accomplishes a state change. Complex actions can be considered as a single stimulus by the interface. In order to drag-and-drop, a user needs to select and hold onto the data item that is to be dragged, move the pointer from the initial position to the desired position and then release the mouse button. While the action may be decomposed into the four pieces (select, hold, drag, and release), and each piece provides a perceivable stimulus to the interface, each stimulus by itself accomplishes no state change outside of the action's context.

Surprisingly, we cannot use simple actions to talk about complex actions, because complex actions are still semantically atomic; when their constituent gestures are taken one by one, the constituents carry no semantic meaning in the context of the task. Another example of a complex action is the grabbing action in a TUI. The grabbing action consists of a transport gesture, which takes the hand to the object to be grabbed, and then the fingers close around the target object, to complete the complex action (C. L. Mackenzie & Iberall, 1994). If they are considered by themselves, each constituent gesture in the context of the grabbing task could carry different semantic meaning than the whole complex action, when performed in its entirety. However, each constituent gesture in a different context may be considered a simple action.

Actions can also be seen as productive, non-productive and counter-productive. Productive actions are those stimulus-response pairs that move the user closer to the completion of the task. Non-productive actions have no effect to the performance of the current task, such as painting the case of a server green to fix a network-related problem. It is an action that can be performed but that has no effect towards or against the performance of the fixing the network problem. Counter-productive actions hinder the user's performance of the task. An example is when a system administrator believes that there exists a network problem, when there is none, and changes the network addresses to correct the problem, when that action makes the problem harder to solve. CoDeIn currently only models productive actions in describing tasks, similar to other model-based methodologies, such as GOMS.

5.2 *CoDeIn* Models

Any model created using CoDeIn is a "designer's model" in contrast to a "user's model" of the task; it is intended to aid the designer in thinking clearly about the effects of design choices. Here, notation that was introduced by Nielsen (Nielsen, 1990) may be helpful to explain the difference between different types of models. Nielsen uses Rohr and Tauber's (1984) definition of a relation between a model, a subject and an object R(M,S,O) and modifies it to be R(M,S,O,P) as follows:

- M is the model
- S is the subject who has this model (and who has it for a specific purpose, the 'model function')
- O is the object to be modeled.
- P is the purpose of the model (added by Nielsen).

Nielsen then defines a notation for describing the taxonomy of models, by taking the combination SO above to mean the type of the model. Using this definition now, one can write the following:

- A’s model of B is type AB-model
- A’s model of a type BC model is A(BC) model.
- A’s model of both B and C is a type A(B+C) model. This is how A models the combination of B and C.

The notation is recursive in that one can talk of a type ABC(D+E+F) model which is how A models B’s model of C’s model of the combination of D, E and F. Table 5.1 taken from Nielsen (1990) shows the summary of this notation.

Table 5.1 Model Agents and Types using Nielsen’s Meta Model Notation

Model Agents	**Explanation**
U	User
D	Designer
C	Computer System
R	Software Ergonomics Researcher
T	Task of the user
W	Surrounding world in which user U performs task T
M	The manual (or other documentation) for C
Model Types	
AB	A’s model of B
ABC	A’s model of B’s model of C
A+B	A model having the combination of A and B as objects
A(B+C)	A’s model of the combination of B and C
AB(C+D)	A’s model of B’s model of the combination of C and D

The letters A, B, and C should be replaced with the abbreviations of the actual actors involved with the given model.

Using this taxonomy, the models created by CoDeIn are of the form DU(T+W+C), which means that they are the designer's model of the user's model of the task, the environment of the task and the computer system.

The difference between CoDeIn's model and the actual user model can be demonstrated by an example. Suppose that a task is designed so that it can be executed by performing two parallel actions. Further suppose that the task can also be performed serially, albeit less efficiently. The user may perceive that the task can only be performed serially, thus not reaching optimum efficiency during the task's execution. CoDeIn however, would model the task with parallel actions, reflecting the true nature of the task, and not serial sequencing that is the user's perception of the execution of the task. The task model could also show the serial performance of the task, by considering that the parallel actions would be performed serially. The advantage to this approach is that one task model can model various task executions, thus saving time in the design and research of the task in question.

Unlike models intended to evaluate existing interfaces or interaction devices, CoDeIn's intended use is more for the evaluation of new or novel interaction devices and interfaces; this imposes certain limitations on the model creation process. First, there are no users of a new interaction device or interface, so that any model created must be based on the designer's belief of what the user model will be. This, however, is a still designer model, and not a user model. Also, pragmatic solipsism states that even if there are users of the device or interface, their description of their model can only produce a designer's model of the user model. Even through knowledge elicitation (Cook, 1994), the user model cannot emerge completely and clearly.

This however, is not necessarily an impediment, because for expert users, the designer model and the user model can eventually become virtually the same, because expert users learn how to perform the task as the designer envisioned that users would perform the task. The gradual understanding of the designer model by the users, leads to an increase in their performance, eventually allowing them to reach the optimal performance visualized by the designer (presuming that this is possible).

5.3 Categorizing Task Presentations in User Interfaces

CoDeIn models a task based on the amount of knowledge that users need to possess to perform that particular task. I assume that knowledge about a task can be described by using three sets of knowledge: "task knowledge", "knowledge in the head" and "knowledge in the world".

The first knowledge set is the knowledge that is required for the completion of a task (task knowledge or Kn(t)). This knowledge is the requirement set, and it is not necessarily situated in the user's head or in the environment. The set includes the facts and procedures that must be performed in order for anyone to perform the task in question.

The second knowledge set is the knowledge that the user already possesses that is relevant to the completion of the task. This set only includes knowledge about productive actions. Using Norman's terms (Norman, 2002), I call this set the "knowledge in the head" set, or Kn(h). This set could be empty in the case of a new user who knows nothing about the performance of the task, or it could be equal to the task knowledge set in the case of an expert user, or more commonly, it is a subset of the task knowledge set, which means that the user knows some facts and procedures about the task, but not everything.

The third knowledge set contains knowledge that is provided by the interface in the form of affordances and constraints that pertain to the task (Christou, Jacob, & Cheng, 2006; Gaver, 1991; Hartson, 2003; Norman, 1999a; , 2002), and by the external representations that are created and are designed to hold knowledge for the users, so that users do not have to have every piece of knowledge in their head.

For example, in a blind game of chess that is played with no chessboard and pieces, all the knowledge about the game is in the players' heads. Nobody else can follow the game unless the players impart their knowledge of the game. However, in a normal game of chess, all the knowledge about that particular game is encoded in the positions of the different chess pieces on the chessboard. This knowledge exists in the environment, and it is there to be perceived by anyone who wishes to follow the game. Knowledge that is encoded in this way is placed in the "knowledge in the world" set, or Kn(w), following Norman's terms (Norman, 2002). Kn(w) however, does not include knowledge that exists in the form of user manuals, or tutorials, or that may be gained from different forms of training on the device or the interface in question. Kn(w) only includes knowledge that exists in the immediate task environment of the users.

The basis for the definitions of these sets comes from Task Knowledge Structures (TKS) (P. Johnson & H. Johnson, 1991; P. Johnson et al., 1988), described in Chapter 3. As was explained, TKS assumes that people tend to group together all knowledge that pertains to a certain task, and keep this knowledge as one complete chunk. It further assumes that people know how to perform the task once they have working knowledge of the necessary constituent actions of the task. CoDeIn subscribes to this assumption insofar as that the constituent actions of a task are retrieved as one chunk as one becomes better at performing the particular task. Furthermore, one implication of the TKS assumptions is that if a task is performed enough times, it will eventually become automatic (Anderson, 1992; Logan, 1988).

By viewing the task as the major entity and modeling the required knowledge for the task, and not trying to capture the user's goals and intentions, CoDeIn moves away from the traditional approach involving goal-based evaluation methods, and takes a step closer to the suggestion made by Diaper and Stanton (2004b), who posit that "the whole concept of goals should be abandoned as unnecessary and thus one of the major sources of confusion about task analysis would be removed" (Diaper & Stanton, 2004b, p. 603). Through modeling knowledge required for the task, various conclusions can be drawn about what knowledge the user might be missing, and whether that knowledge exists in the environment. This methodology makes it possible to catch design errors very early in the design process.

In CoDeIn, a task is viewed as a molecular entity composed of atomic actions, either simple or complex. Each action has its own requirements in terms of knowledge, seen as the sole pre-conditions to the execution of the action, and consequences in terms of the state of the interface. Thus, if an action is to be performed, the user needs to know the pre-condition knowledge for the performance of the action. However, actions also create knowledge-in-the-head through their effects on the environment. These effects on the environment change the environment in ways that are perceived by the user, causing the user to change, revise or gather knowledge from the environment. These effects of an action become the action's post-conditions that can be pre-conditions to another action. This idea drives the modeling mechanism that is presented in Chapter 6 of this thesis.

Usually, a combination of the Kn(h) and Kn(w) as perceived by the user, yields all the knowledge that is needed for the completion of the task, as shown in Figure 5.1. The set that represents Kn(t) is a subset of the union of Kn(w) and Kn(h), hence some of the task knowledge is inside the user's head, and some of it is demonstrated in the environment. Also there may be some overlap between the user's knowledge about the task and the environment-provided knowledge, as shown by the intersection of the two sets. An example would be of a user of a word-processing application, who knows how to use the application, but does not remember everything about the task to be accomplished. Therefore, things like labels and icons on menus and buttons, which are perceived affordances of the provided tools, provide hints as to what each menu includes and what each button does, or they even explicitly define specific knowledge that needs to be perceived for the task to be performed, as in the case of a chess game.

Perceived affordances (Norman, 2002) of menus and buttons are one part of the Kn(w) set. Additional Kn(w) is given by the constraints on these items, such as the constraint that a button has two states, or that one menu can be visible at a time. A constraint of menus is that they give multiple choices of which only one can be selected. If the user knows how to perform some of the actions in the task for which there are hints in the environment, then that knowledge belongs in the intersection of the two sets, Kn(h) and Kn(w). Finally, if there are actions for which there are not any affordances in the environment, but the user knows how to perform, then that knowledge belongs to the Kn(h) set. Taking the two sets together, the Kn(t) set is created.

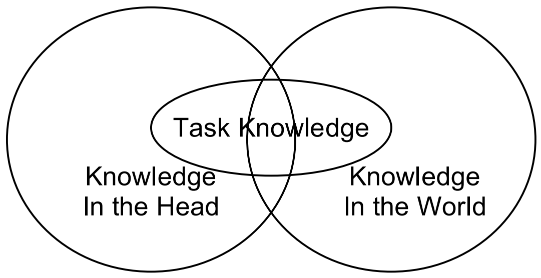


Figure 5.1 Common Knowledge Case 1
The combination of the knowledge-in-the-head set and the knowledge-in-the-world set have some common elements, but the user does not know everything about the world, nor every piece of knowledge that the user knows is described in the world.

A second common case scenario is shown in Figure 5.2. The user has some knowledge about the interaction style, there exists some knowledge in the world (in the interface), and there is specific knowledge that describes how to perform a specific action in the interaction style.

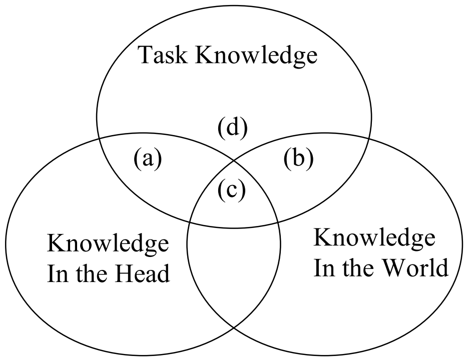


Figure 5.2 Common Knowledge Case 2
Not all the knowledge for the completion of the task is known by the user, or included in the environment

In Figure 5.2, Part (a) represents the knowledge that the user has that pertains to the task that the user is called upon to perform. Part (b) is knowledge in the world (in the form of affordances and constraints) that the user does not have, but can perceive. Part (c) is knowledge that the user has and is also present as affordances and constraints in the world. Finally, part (d) is the task knowledge that the user does not possess, and it is not represented as affordances and constraints of the interface. This is the knowledge that the user must learn in order to bring the task to completion. It is evident that the goal of any design would be to make part (d) as small as possible or even non-existent, reaching knowledge case 1 as shown in Figure 5.1, where anything that the user does not know can be perceived from affordances and constraints in the interface. On the contrary, Figure 5.3 shows the worst case scenario, where the three knowledge sets that have been introduced are not at all connected with each other.

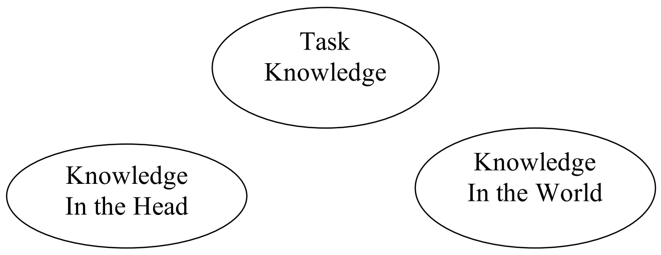


Figure 5.3 Knowledge Case 3: Worst Case Scenario
The user does not know how to perform the task, and the environment does not provide any clues about task performance.

Table 5.2 Different cases of tasks in a user interface shown in set notation.

Knowledge Case	Explanation
$Kn(t) \subseteq Kn(h)$	One of several best case scenarios, where the user knows everything about the performance of the task.
$Kn(t) \subseteq Kn(w)$	One of several best case scenarios, where everything needed for the performance of the task is shown in the environment
$Kn(t) \subseteq (Kn(h) \cup Kn(w))$	One of several best case scenarios, where everything needed for the performance of the task is either known by the user, or is in shown in the world.
$(Kn(h) \cup Kn(w)) - Kn(t) \neq \emptyset$	Common case scenario, where the knowledge for task performance is not all known by the user, even in combination with the knowledge embedded in the environment. Thus, the need for user manuals and tutorials arises.
$(Kn(h) \cup Kn(w)) \cap Kn(t) = \emptyset$	Worst case scenario. The combination of the user's knowledge about the task combined with the knowledge embedded in the environment say nothing about the performance of the task.

In this case, the user does not know how to perform a task, and there are no hints or directions embedded in the environment to help the user gain any of the task knowledge. Therefore, the three sets have no intersection, although each set may hold knowledge chunks about other things that pertain to the interface as a whole. Thus, some outside influence or intervention is necessary (such as a user manual) before progress can be made. While this case seems at first extreme, the case of the MS-DOS or the UNIX command-line interface exemplifies it. The prompt presents no information about the execution of any task, therefore Kn(w) is effectively empty for every task in this environment. If one considers the case of a novice user, then Kn(h) is also effectively empty, meaning that the Kn(t) must be gained from extraneous sources. However, there are several best case scenarios. Table 5.2 shows examples of cases of interface design.

5.4 Defining the User

The previous section defined the three knowledge sets, Kn(h), Kn(w) and Kn(t). Each can be decomposed further into two parts: declarative knowledge (Dkn), or knowledge of facts regarding the task, and procedural knowledge (Pkn), or knowledge regarding the way to perform the different actions that compose the task. Declarative knowledge is distinguished from procedural knowledge in like manner to that used in creating models in cognitive architectures (Gray & Altmann, 2001), such as ACT-R (Anderson & Lebiere, 1998), EPIC (Kieras & Meyer, 1997), or Soar (Laird et al., 1987). Any knowledge that is factual in nature is to be considered declarative, and any knowledge for which a production rule is written is considered to be procedural. A production rule is a piece of code that belongs to a model and captures the performance of an action or task, based on the perceptual, cognitive, and motor abilities of the architecture and the declarative knowledge of the model. In this thesis, the distinction between declarative and procedural knowledge is the result of a knowledge elicitation process for the interface or interaction device under study. A further discussion of what is declarative knowledge and what is procedural knowledge however, is out of the scope of this thesis.

I do not take into account any domain knowledge, because it is constant across interfaces, and I assume that the user already possesses the required domain knowledge for the task. Domain knowledge in this sense however, is knowledge that has nothing to do with how to perform the task using tools. For example, an accountant that knows how to keep accounting books for a company, and knows what accounts the company has but does not know how to use Microsoft Excel, has the required domain knowledge for keeping the books of the company, but does not have the Dkn and Pkn for using Excel to perform this task. Another example would be of a graphic designer, who understands the concept of merging two images to create a collage, but does not know who to perform this action within a specific application.

The knowledge that is taken into account in the CoDeIn framework is interface-specific knowledge, and action-specific knowledge. For example, if the task is to move the pointer using the mouse from point A to point B, then one of the things the user needs to know is that the mouse is bound to the pointer (Christou et al., 2006). This is a declarative knowledge chunk without which the user would not be able to complete the task. The user would also need to know how to move the pointer using the mouse. Because this is knowledge about the way that the task is performed, it is represented by a procedural knowledge chunk.

Declarative knowledge can further be decomposed into three different types of knowledge. The first type is “What is it” knowledge. This type of fact refers to the objects that are manipulated in the context of an action or task. For example, in a folder-opening task, where the user is required to double-click the folder icon, the “what is it” knowledge needed would be that of what is the mouse, what is the pointer, and what is the folder icon.

The second type of declarative knowledge is “meaning conveyed” knowledge. This type of knowledge allows the user to distinguish different meanings conveyed in different contexts, through the same action. In the case of the previous example, double-clicking on a folder icon opens the folder for viewing. Double clicking on an application icon on the other hand, starts the application. Thus the same action has different meanings in two different contexts.

The last type of declarative knowledge in the framework is binding knowledge. Bindings are the “connections” between different objects in an interaction style, and the connections may be permanent or temporary. In the folder clicking example, the user needs to know that the mouse is bound to the pointer, and that the mouse movement translates to pointer movement. This would be an example of a permanent, or static binding. Such bindings cannot be broken by the user’s volition. On the other hand, in a drag-and-drop task, when the user holds down the mouse button to drag an icon, the connection between the icon and the pointer is only transient. It exists only for the duration of the action. These bindings are temporary, or dynamic, because they can be created on the fly, during the execution of the task, usually when the user requests it. Table 5.3 summarizes the types of knowledge in the CoDeIn framework.

Table 5.3 A summary of the four different types of knowledge in the CoDeIn Framework.

Type of Knowledge	Description
"How to do it"	Procedural knowledge, or knowledge about how an action or a task is performed
"What is it"	Declarative knowledge about what is an object that is manipulated in the context of a task or action.
"Meaning Conveyed"	Declarative knowledge about what the action's meaning is in the context of the task.
Binding	Declarative knowledge about how different objects of an interaction style are connected, as well as about the duration of this connection.

The decomposition of task knowledge into declarative and procedural parts, allows the categorization of different types of users. Table 5.4 shows the categories of users created by knowledge classification. The categories in Table 5.4 show a preliminary categorization, based on absolute categories. The first category defines the set of expert users of a task. These users know everything there is to know about a task, and can perform the task without any help from the environment. There are however, some interaction styles where such a user cannot exist. For example, consider the DM interaction style, where a user that knows exactly where all the menus and menu items are for the performance of a task. If one of the required menu items changes position for some reason, as may happen during version changes of an interface, then the expert user will not be able to perform the task without searching for the updated knowledge in the environment. Conversely, in a command-line interface the user needs to have Kn(t) = Kn(h), as there is no way that the environment can give information about the execution of any task (excluding command completion), without the knowledge that there may be a help command, or searching for that knowledge in a user manual, because as mentioned earlier, these are not types of knowledge that belong in the Kn(w) set.

Table 5.4 A categorization of users according to their knowledge about a task.

Knowledge of the User	User Category
$(DKn(h) = Dkn(t)) \wedge (PKn(h) = Pkn(t))$	Expert user who knows everything about the performance of the task.
$(DKn(h) = Dkn(t)) \wedge ((PKn(h) \cup Pkn(t)) = \emptyset)$	A user who knows the facts, or the theory behind performing some task, but does not know how to use the various interface tools to execute the task (knows which menu to use, but does not know how to use the mouse to choose the appropriate menu).
$((DKn(h) \cup Dkn(t)) = \emptyset) \wedge (PKn(h) = Pkn(t))$	A user who knows how to use interface tools, but does not know the facts about the performance of the task (knows how to use the mouse and how to operate menus, but does not know which menu to use).
$((DKn(h) \cup Dkn(t)) = \emptyset) \wedge ((PKn(h) \cup Pkn(t)) = \emptyset)$	A novice user who does not know anything about the task.

The categories shown in Table 5.4 are absolute, in that they only allow a binary categorization of users. Either the user belongs to a category or not. Most of the time though, a binary categorization is not sufficient, and this is the case here as well. The user may know something about a task, but not everything there is to know about it. Therefore one needs a way of representing the amount of knowledge of the user, compared to the sets of Table 5.4; one that reflects “how well” the user knows how to perform a task.

One such measure is an efficiency rating (ER) that represent how a novice user compares to a hypothetical expert user. The efficiency of an agent versus another agent is a ratio of the rates with which the agents can complete tasks: novice rate / expert rate. A rate is the reciprocal of task time, so that this becomes (1/novice performance) / (1/expert performance) = expert performance / novice performance. For example, one could think of a user class relative to a task that is considered 50% expert if users belonging to this class can perform the task in twice the time of an expert user.

The ER can also be applied to each of the knowledge chunks that users need to know to perform one of the actions that constitute the task, thus modifying the completion time of the action based on the ER of the modeled users. However, models created with CoDeIn are not intended for the evaluation of existing devices or interfaces, but rather of new or novel devices or interfaces. Therefore, one may not be able to find expert users that can be tested on the device and receive a measure of expert user completion time. To overcome this I suggest that non-expert users may be tested, and their performance be compared to a similar device, on which the tested users are experts. For example, if an ER is required for a virtual reality glove, one could test non-expert users and yield a completion time estimate, and then use the tested users' performance on the mouse as a lower bound for the possible performance on the virtual reality glove.

I believe that using performance on existing devices may provide a good estimate for the purpose of approximating expert user performance on the new devices or interfaces. The ER may even be made up for the purposes of calculating preliminary estimates about the device or interface, to provide evidence to consider if further pursuing the designed artifact should be considered. Another benefit of having the freedom to create such profiles is that one model created for a task may provide estimates for different types of users, according to different ER estimates.

5.5 Defining the Task and Design Generation

Using the proposed method, the researcher or the designer of a task is able to recognize where in the design the task model may not fit the user model. This happens when the user may be missing knowledge for performing certain actions in the interface, with no Kn(w) to compensate. Thus, the conceptual model of the task can be examined, and may be corrected where necessary. It is well known that the researcher's model of a task is different from the designer's model of a task, and both are different from the user's model of a task (Shneiderman, 1998). Using Nielsen's meta-model (Nielsen, 1990) notation, U(T+C) is different from D(T+C) and R(T+C).

Therefore, analyzing a task using the CoDeIn framework may provide evidence for further clarification and may even help in the generation of a new design for an interface; one that requires less knowledge on the part of the user. One of the purposes of such an analysis is so that the designer or researcher can create a design whose proposed user population will need to learn as few things as possible to be able to use the proposed design. This occurs through analysis of the knowledge required to perform tasks using the proposed design. The analysis of the knowledge needed for the performance of the task and the specification of the model user have as a purpose to uncover the places where constraints and affordances may be placed to provide Kn(w), thus augmenting the user's knowledge. Hence, while not the topic of investigation in this thesis, CoDeIn may provide evidence for alternative or new interface designs that fit a task better, given specific or generic user profiles (St. Amant et al., 2005).

5.6 Summary

This chapter presented the intended use of the CoDeIn framework, along with a categorization of user interfaces through the modeling of the required knowledge for performance of tasks in a user interface. Three different knowledge sets were defined, Kn(t), Kn(h) and Kn(w). These three sets are constituted from declarative and procedural facts, denoting a finer decomposition of the sets into Dkn(t) and Pkn(t), DKn(h) and PKn(h), and DKn(w) and PKn(w) respectively.

A task is defined as a series of actions, where each action requires knowledge pre-conditions and generates knowledge post-conditions. Each of the pre- and post-conditions is a declarative or procedural knowledge chunk, and all of these knowledge chunks comprise the Kn(t) set. The Kn(h) set is composed of chunks that the user knows, and the Kn(w) set is composed of chunks that are affordances, constraints, or embedded in structures in the environment of the task.

Several user categories are defined in set notation. These user categories become more varied with the proposal of ER, a rating that allows the modeling of non-expert users according to their performance versus that of an expert user. The ER is defined as the ratio of the completion rate of an expert user over that of a non-expert user in the performance of a task. This allows predictive calculations that estimate the completion time of tasks by a defined user class.

Finally some benefits for generating new interface or device designs are discussed. These benefits are gained by building models in CoDeIn that show the required knowledge for the performance of tasks with the proposed device or on the proposed interface, allowing the designer or researcher to consider further pursuing the proposed design..

The next chapter will use these definitions to create a predictive task-modeling method based on task knowledge.

Chapter 6 Modeling the Task: Definition of CoDeIn

This chapter presents the nomenclature of the CoDeIn framework and the diagrammatic notation for description of tasks. The nomenclature is used to describe the constituents of an interface and their relationships, while the diagrammatic notation is used to describe interfaces in terms of the knowledge required to use them.

Because RBIs are still researched and developed in isolation from each other, they suffer from the lack of a common language for their description, and each interaction style defines its constituent elements through different terminologies. Because one of the goals of the CoDeIn framework is to provide a stepping stone towards the unification of RBIs, this chapter attempts to provide nomenclature that is broad enough to allow for the description of most RBIs, and anchor the usage with common terms that are used in DM, to give a sense of cohesion to the field.

6.1 Nomenclature

Every interaction style has some sort of representation for data. Icons are the most prevalent in the direct manipulation interaction style, with more diverse representations in RBIs. Each interaction style also has its own ways that allow the user to interact and control the data. CoDeIn provides a common vocabulary to researchers, which describes representations of data and the ways of interaction for all interaction styles that fall under the RBI banner. The following sections present the three categories of objects that may exist in a user interface: Data, Interaction, and Intermediary Objects. These concepts allow CoDeIn to express the constituent parts of different interaction styles via one unified terminology.

6.1.1 Data Objects

Every interaction style defines some sort of representation for the actual data that resides in the system. Even though all data is ultimately described by 1's and 0's, each interaction style has some sort of representation of the logical groupings of that data, be that graphical, physical, textual or otherwise. The representations are used to represent distinct data entities such as files, folders, database tables, applications, etc.

A Data Object (DO) is defined as the representation of a distinct data entity or group of entities in an interaction style. DOs may take many different forms and may be used under different guises, but their sole purpose is to allow the user to manipulate the data entities that they represent. DOs may be comprised of other DOs, and they can be combined to create more complex DOs.

Examples of DOs in DM are the icons that represent files, folders, applications etc. In fact, a folder is a DO that includes other DOs (the file icons). The problem with icons though, is that the user needs to know, for example, that a folder has a specific icon, and that when double-clicked it will open and show its contents in a window. Only after users acquire this knowledge they will be able to effectively use an interface that uses DOs such as in the DM interaction style.

Another example of a more intuitive DO that is comprised by other DOs is a document page in Microsoft Word. The page can be considered a DO because it is the representation of the document data in a form that allows users to manipulate that data. A page is comprised by sections, paragraphs, sentences, words, and characters, which are also DOs themselves.

Consider a VR application such as the digital library application created by Shiaw (2003). This application presents Greek vases from a certain era in various ways, one of these being a depiction of the vases on pedestals. The user may approach the vases, grab them and manipulate them in order to get a better sense of the exhibits than one would in a regular museum where manipulation of the artifacts is strictly forbidden.

In this application, the DOs are the vases as depicted in VR, because these are the items that the user will manipulate. In this case, the interaction is smooth and intuitive because users may interact with the DOs as they would interact with the real artifacts in the real world. The easiness of the task comes from the ability of RBIs to take actions from the real world, actions that the user already knows how to perform, such as the grabbing action and head tilting actions, and incorporate them in the context of interfaces. Therefore, instead of users manipulating vases using the mouse, and trying to map two dimensional motions of the mouse to move three dimensional objects in 3-D space, users can perform the actions they know to manipulate the vase and concentrate on studying the artifact rather than on the manipulation of the artifact.

6.1.2 Interaction Objects

In many interaction styles there are artifacts, real or virtual, that allow the user to interact with the DOs. For example, in DM the user may use the pointer to drag and drop a file from one folder to another. In this type of interaction, the user uses the pointer, which is the interaction object (IO) to manipulate the icon of a file, which is the DO.

IOs are the objects that are perceived by the user to be the means of interaction with the DOs in an interaction style. They are the tools that are provided by the interaction style that allow the manipulation of properties, states and attributes of the DOs. IOs do not have to be manipulated directly by the users, although in many RBIs this is possible.

IOs do not have to act on only one DO. They may act on a group of DOs, or on specific properties or attributes of a DO. IOs may only work on a particular type of DO as well. Also, more than one IOs may be used together to achieve a desired action.

IOs are used when the user cannot directly control or manipulate the DOs, or their properties and attributes. IOs may also be used in the case when the interface wants to explicate to the user a specific way of manipulation, or an interaction with an attribute of a DO.

In many RBIs the hands are the IOs of the interface. However, this statement is truer for some RBIs and not as true for others. In VR for example, while the simulation tries to convince the user that the IOs are the hands, the actual IOs are the representations of the hands in the virtual environment and their quality influences the quality of the experience.

In interfaces such as flight simulators, or even real environments such as those of an aircraft, the IOs are not software representations. Rather, the interfaces are controlled through the use of joysticks, and consoles. Each of the controls then, becomes an IO to the DO that it manipulates. For example, when pilots want to manipulate the course of an aircraft, they manipulate the rudder, which in turn manipulates the ailerons. Thus, the ailerons are the IOs, because those are responsible for the change of the course of the airplane. The rudder is an intermediary object, as explained below.

6.1.3 Intermediary Objects

Intermediary objects (INs) are the physical objects used by the user to manipulate the IOs. INs are usually physical artifacts in any interaction style. These artifacts are used to manipulate the IOs, when the IOs cannot be manipulated directly.

INs are never virtual. They are always real physical objects that are always bound to some IO in the interface. Through this binding they allow the manipulation of the interaction object by the user. As mentioned in the previous section, the rudder of an airplane is the IN through which the pilot controls the INs, the ailerons, which allow the change in one of the DOs of the airplane, its course.

Having an IN control the IO that controls the DO adds a third level of abstraction to the system, which in turn presents the added challenge to the user of learning and understanding the various ways that the binding between the IN and the IO works. Even though this is a simple observation, it is a crucial one in CoDeIn, as the framework deals with the knowledge that the user needs to have to use an interaction style with some degree of efficiency and expertise. However, users may notice the use of some INs more than others.

A different example of an IN is the mouse in DM, which is bound to the pointer. The mouse is the physical artifact manipulated by the user to control the pointer. The pointer in this case is the IO which allows the manipulation of the DOs. But any device that controls the pointer in DM is an IN, like track balls, the touchpad, or even the cursor keys on the keyboard. There are however, several devices that are not INs, because they do not control the pointer, but rather become the pointer themselves, like light pens, or they allow the user to manipulate the DOs directly, like the touch screen.

In a TUI environment, usually there are no INs, because the DOs and IOs are manipulated directly by the user. In Virtual Reality, on the other hand, the INs take many forms. In the case of immersive Virtual Reality, the head-mounted display is an IN, which allows the immersion of the user into the virtual environment. Even though the user does not use it directly to manipulate anything, the head-mounted display will change the user's view when the user moves around and turns her head left, right, up or down. Thus the head-mounted display is an integral part of the virtual experience and it is the IN through which users orient themselves in the virtual environment. Another IN is the device which positions the user's hand or hands in the virtual environment. This could be a data glove, a polhemus or some other tracking device, which also provides the user with the ability to act on the environment. In such cases, the mapping to the user is more direct, because in the case of a data glove, the user just feels the glove, and does not realize that the glove is an IN. Rather, the IN's role is partially hidden from the user.

One issue that emerges from the use of an IN to control the IO is how the manipulation of the IN affects the IO. Usually, the user learns by experience that a specific manipulation of the IN affects the IO in a specific way, and with enough experience the translation of IN manipulations to IO manipulations becomes automatic. This type of knowledge is called Binding knowledge, and it is described in the next section.

6.1.4 Bindings

Binding is another term that is used in CoDeIn that signifies the places where the IO and the DO directly connect in order to carry out an action by the user. When an IO touches or in some other way comes in contact with the DO, we say that the IO is bound to the DO.

When an IO manipulates or controls one or more DOs, or one of a DO's properties, the IO is said to be bound to the DO. Binding an IO to a DO means that the DO has the focus of the user, and it is going to be manipulated or it is being manipulated. Bindings are the points at where interaction happens between the user and the computer interface, because only through these bindings can the user manipulate or control the DOs.

There can be two types of bindings: static and dynamic. Static bindings are bindings that hold throughout the use of the particular interface between two objects, and dynamic bindings are created and destroyed according to user actions. Static bindings are usually found between INs and IOs, such as the binding of the mouse to the pointer. Dynamic bindings though, are usually found between IOs and DOs.

A static binding example is of the mouse and the pointer. The IN (the mouse) allows the user to control the pointer, throughout the lifetime of an application. Dragging a file icon with the pointer though, creates a dynamic binding between the icon and the pointer. This binding is only persistent during the lifetime of the action.

Static bindings do not stop existing because of a change in the attributes of the IO. Consider the case of a video game that changes the pointer shape. The pointer is still manipulated through its IN, therefore the binding still applies.

6.1.5 Diagrammatic notation

Using the previous definitions of the sets of knowledge and the knowledge distinction between declarative and procedural chunks in chapter 3, and the nomenclature in the previous section, we define a diagrammatic notation to describe and evaluate interaction tasks. This notation is inspired from several other diagrammatic notations, like Statecharts (Harel, 1987) and Augmented Transition Networks (ATNs). The notation uses an ATN to represent the flow of the task from one action to the other, and allows zooming into task states to see how they are carried out. The full notation for the proposed method is shown in Figure 6.1, and an example model is shown in Figure 6.2.

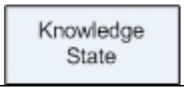
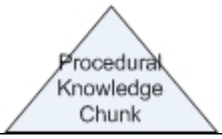
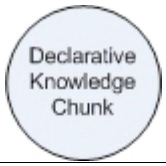
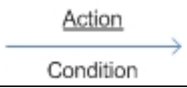
Description	Symbol
Knowledge State	
Procedural Knowledge Chunk	
Declarative Knowledge Chunk	
Action	

Figure 6.1 The diagrammatic components that make up the syntax of the proposed evaluation method

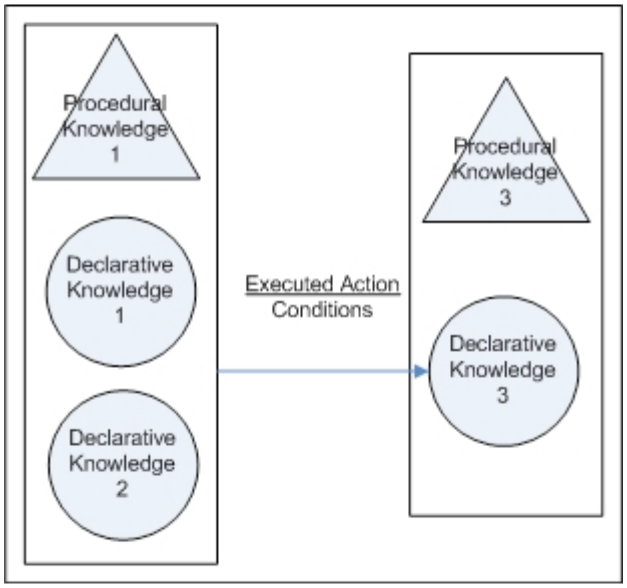


Figure 6.2 An example model of a one-action task in CoDeIn

Figure 6.1 includes four symbols that make up the diagrammatic notation of CoDeIn. The first symbol, the rectangle, represents a knowledge state. Because I propose modeling of the task, and not modeling of the interface, the knowledge state does not represent a specific state of the interface. Rather, it represents a state in the process of the task, if the task is viewed as happening serially over time. Each knowledge state in a task encapsulates the knowledge that the user must have to perform the action that would allow the task execution to proceed. For example, in Figure 6.2 the first knowledge state of the task includes three different declarative and procedural knowledge chunks. These are required for the execution of the action, as shown by the arrow labeled "Executed Action".

A declarative knowledge chunk is represented by a circle, with the description of the chunk inside it. When a declarative chunk is included in a state, then that chunk is a precondition for any action coming out of the knowledge state.

A procedural chunk represents the procedural knowledge that the user needs to possess, to be able to perform the action represented by the arrow coming out of the knowledge state. The procedural knowledge in the notation is not broken down into its how-to components, but it can be thought of as a procedure in a cognitive architecture, like ACT-R (Anderson & Lebiere, 1998) or Soar (Laird et al., 1987). Resolving procedural knowledge into its finer components is beyond the scope of the proposed method, as explained in Chapter 5. I consider that the required knowledge for each action is atomic, in that it cannot be broken down any further. For this knowledge I propose using the ER (as defined in Chapter 5) that allows the attribution of a number that represent how well the user knows how to perform the action.

The reason for using different symbols for declarative and procedural knowledge is because of how each type of knowledge is viewed in the context of CoDeIn. Declarative facts are things that the user either knows, or does not know. I assume that there are no gradations of *how well* the user knows a particular fact. For example, users either know that the mouse controls the pointer, or they do not. However, there are gradations on the time required for the recall of this type of knowledge. Expert users for example, may be able to recall everything that is required for the performance of a task almost automatically (Logan, 1988), whereas novice users may have to recall each fact independently from the others, or they may need to recall one fact more times than others.

On the other hand, I claim that one can test how well a user can perform an action (Christou & Jacob, 2003). For example, one can administer a test to see how well users know how to move the mouse. Thus, procedural knowledge can be graded using a scale from 0 to 1, with 0 being no knowledge and 1 being expert error-free performance. This is the use of the ER as applied to allow the predictive calculations of estimates of task completion time.

Finally, the arrow represents an action that the user performs. The name or the description of the action is written over the arrow and, if applicable, a condition is written under the arrow. The condition of the action represents when the transition will be followed, and not when the action will be performed. In any knowledge state, when an arrow exists, it may be followed (that is, the action may be performed) as long as the condition below the arrow is satisfied, and the knowledge requirements of the knowledge state are satisfied from the set of knowledge in the head or knowledge in the world. The condition of the action is optional, and if it is not included, the action may be performed as soon as the user enters a state that allows the action. Figure 6.2 shows how the action creates the declarative knowledge chunk that is required in the next knowledge state, by pointing directly at that knowledge chunk.

6.1.6 Examples

In this section I provide some examples of how the notation may be used to describe various cases of pre-requisite knowledge and consequent actions. Figure 6.3 demonstrates the case where an action may be performed as soon as the user has all the required knowledge about its performance.

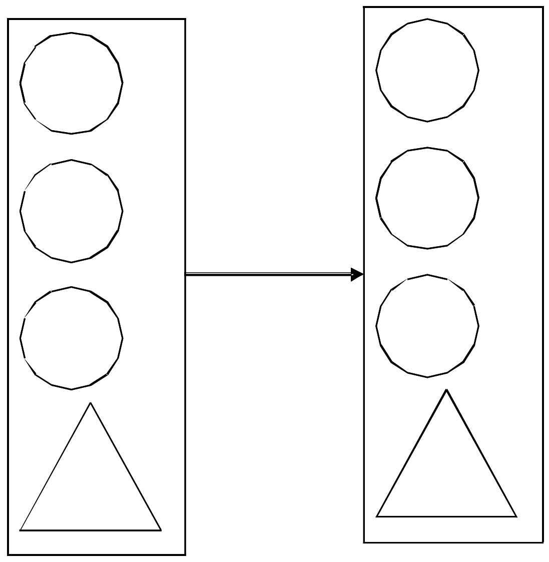


Figure 6.3 Pre-requisite knowledge and consequent action.

Figure 6.4 demonstrates the case where the action performed has a direct effect on a knowledge chunk, and Figure 6.5 demonstrates the case where the action has a direct effect on multiple knowledge chunks. This may happen when the user is moving the pointer. The user's knowledge of where the pointer is, is updated by the performance of the action "moving the pointer".

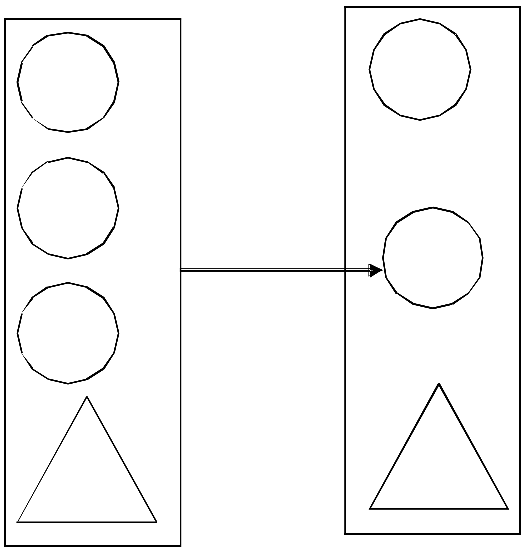


Figure 6.4 Action performance modifies knowledge.

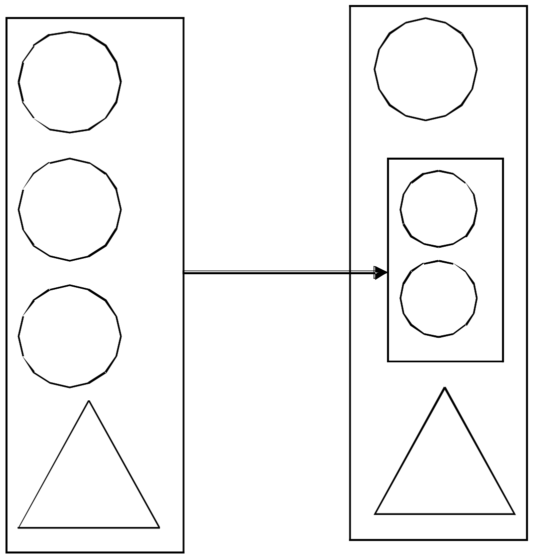


Figure 6.5 Action affecting multiple knowledge chunks.

There are cases where the user may be able to perform two or more actions at the same time, such as when using a multi-modal interface. Figure 6.6 displays a case where the performance of the two actions requires the same pre-condition knowledge.

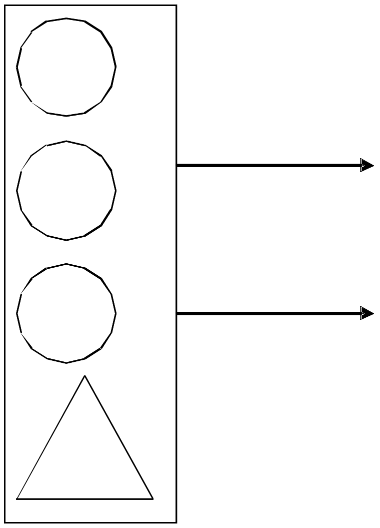


Figure 6.6 Performance of two actions in parallel.

Figure 6.7 displays the case of the performance of parallel actions that do not have the same preconditions, but must be performed together during the execution of a task. The actions performed are not required to lead users to the same knowledge state. Rather, users may be able to choose in which knowledge state they will be next, after they perform the parallel actions.

6.2 Using the notation for description of a task

The previous section presented the diagrammatic notation of CoDeIn. In this section I show various examples of how this notation can be used to describe a task. The task used here is that of moving a file from one folder to another, in the DM interaction style. The task is also presented in more detail in a later chapter of this thesis, where the framework is used to not only describe it, but also evaluate it and compare it to a similar task in a different interaction style.

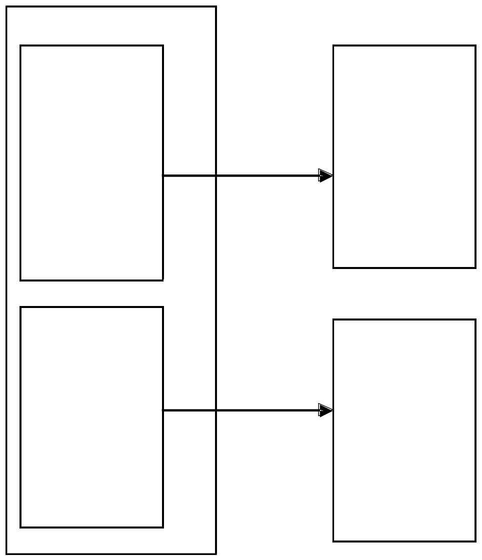


Figure 6.7 Parallel actions requiring different knowledge and leading to different knowledge states.

The framework assumes that some kind of knowledge acquisition and elicitation process has been followed, by employing any of the processes reviewed in Chapter 3. As the knowledge acquisition and elicitation part is outside the scope of this thesis, I assume that it has been done, and continue to the description of the task, using the results of the knowledge elicitation process.

The first step in the evaluation of a task is to create a summary diagram. This diagram's goal is to show a simplified view of the task, without all the knowledge chunks and action-condition pairs, and it represents the execution of a task over time. Figure 6.9 displays such a summary diagram for a "File move" task.

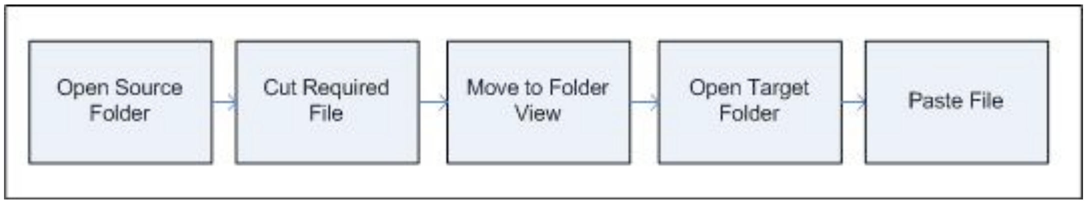


Figure 6.8 A summary diagram for a "File move" task.

Each subtask can be analyzed and yield a diagram with its constituent knowledge and actions. Because of the hierarchical decomposition of tasks, the process resembles that of Hierarchical Task Analysis (Annet, 2003; Annet & Duncan, 1967). Figure 6.10 shows an enlargement of the "Open source folder" subtask.

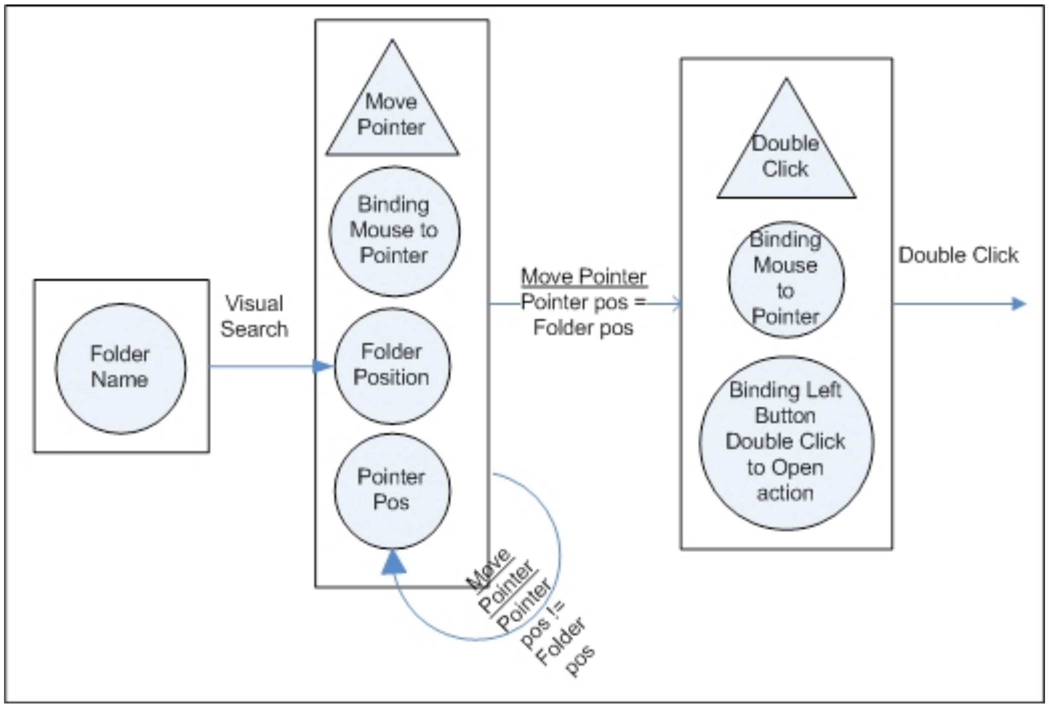


Figure 6.9 The "Open source folder" knowledge state from the summary diagram. Enlarged here to show the knowledge and constituent actions for its completion

There are three knowledge states in Figure 6.10. The first one shows that the user performs a visual search to find where the source folder is on the screen, and is followed by the "Visual Search" action. Notice that the visual search action creates the required knowledge chunk in the second knowledge state. Once the visual search action is completed, the user moves to the second knowledge state. The second one describes moving the pointer from its current position to over the source folder. This requires the user to know four chunks: the pointer's position, the binding of the mouse device to the pointer, the source folder's position (which are all shown as declarative knowledge), and how to move the pointer (which is procedural knowledge). If any of these four chunks is missing, then the user will not be able to perform the "Move pointer" action, which is denoted as arrows coming out of the subtask.

There are two arrows that come out of the second knowledge state, both marked as the action "Move pointer". The topmost arrow includes the condition that signifies that if the pointer's position is over the source folder, then the user can move on to the next knowledge state to complete the task. The second (lowermost arrow) shows that while the pointer's position is not inside the source folder's position, then the user will need to continue moving the pointer. Notice how here too, the lowermost arrow points back to a specific knowledge chunk. This is because the move pointer action updates the position of the pointer, thus the action updates the specific knowledge chunk.

The arrow that represents the double click action is the same as the arrow that is shown in the summary diagram, which takes the user from the "Open folder" state to the next state called "Cut Required File". In the same way, the rest of the subtasks in the summary diagram are expanded and presented in Figures 6.11 – 6.14.

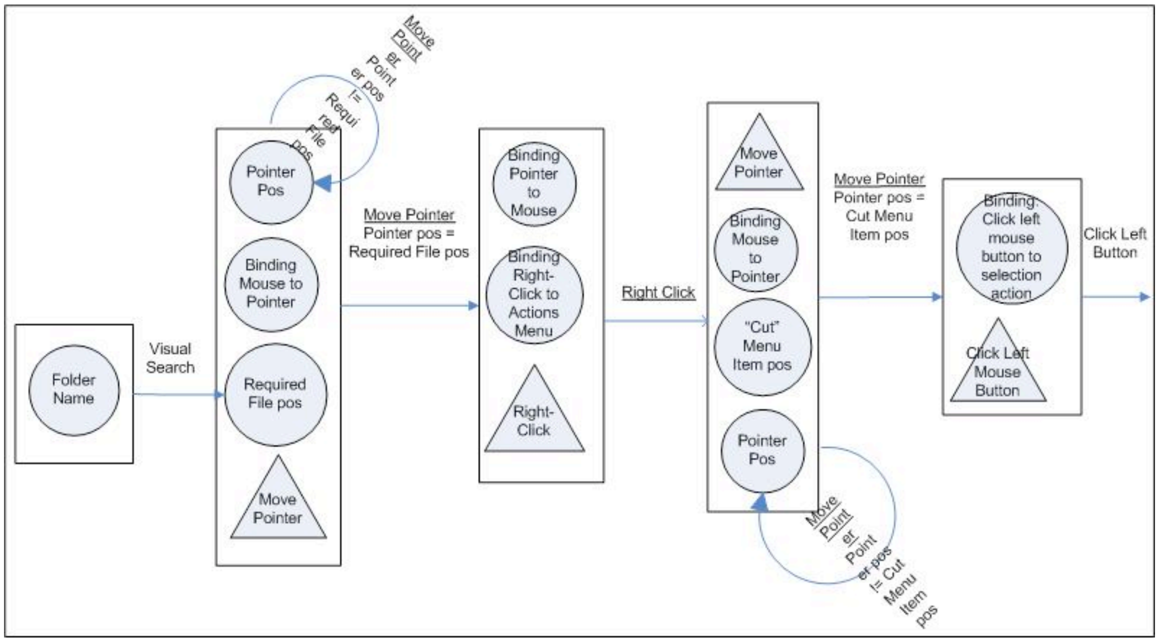

Folder Name
Visual Search
Pointer Pos
Move Pointer Pointer pos != Required File pos
Binding Mouse to Pointer
Required File pos
Move Pointer
Move Pointer Pointer pos = Required File pos
Binding Pointer to Mouse
Binding Right-Click to Actions Menu
Right-Click
Right Click
Move Pointer
Binding Mouse to Pointer
"Cut" Menu Item pos
Pointer Pos
Move Pointer Pointer pos != Cut Menu Item pos
Move Pointer Pointer pos = Cut Menu Item pos
Binding Click left mouse button to selection action
Click Left Mouse Button
Click Left Button

Figure 6.10 The analysis of the "Cut File" state in the summary diagram of Figure 6.3.

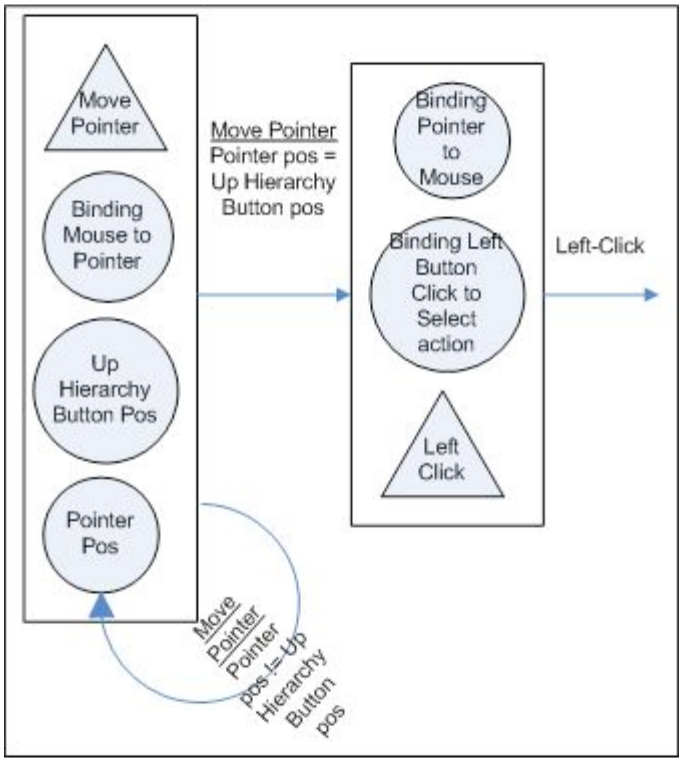

Move Pointer
Binding Mouse to Pointer
Up Hierarchy Button Pos
Pointer Pos
Move Pointer Pointer pos != Up Hierarchy Button pos
Move Pointer Pointer pos = Up Hierarchy Button pos
Binding Pointer to Mouse
Binding Left Button Click to Select action
Left Click
Left-Click

Figure 6.11 Analysis of the "Move to Folder View" state in Figure 6.3

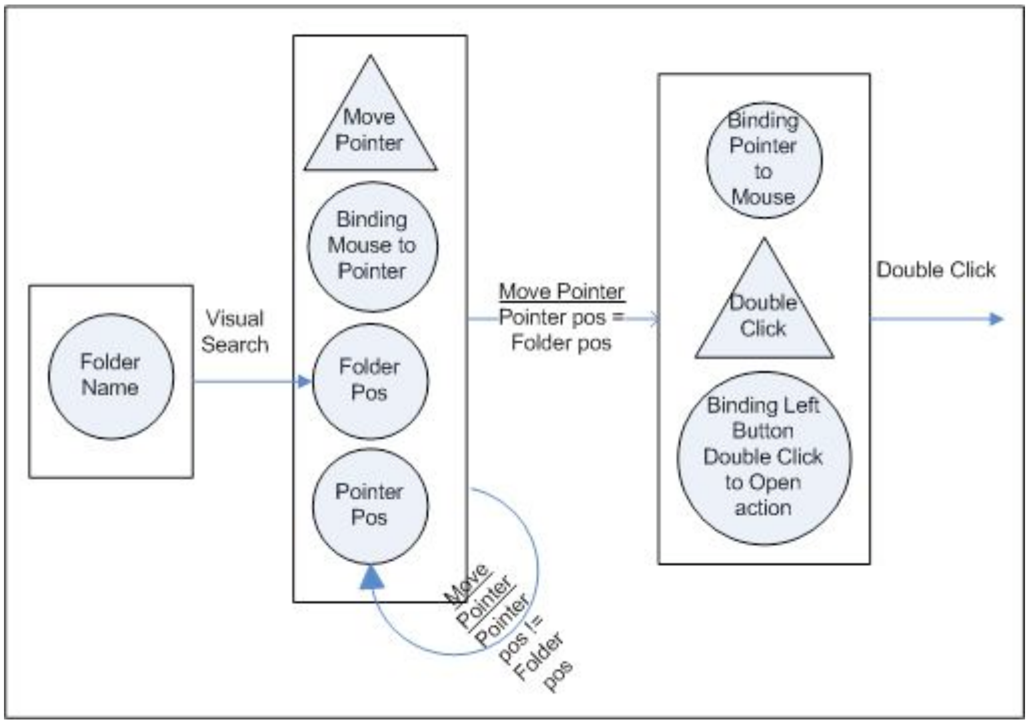
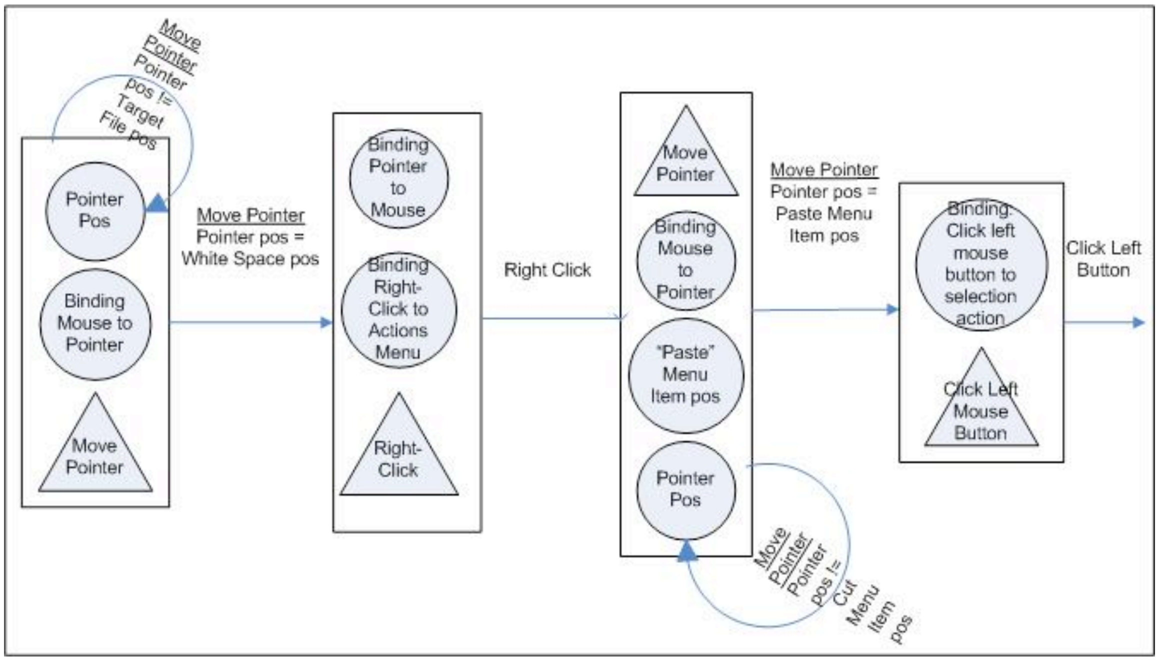


Figure 6.12 Analysis of the "Open Target Folder" state in Figure 6.3



Move Pointer Pointer pos != Target File pos
Pointer Pos
Binding Mouse to Pointer
Move Pointer
Move Pointer Pointer pos = White Space pos
Binding Pointer to Mouse
Binding Right-Click to Actions Menu
Right-Click
Right Click
Move Pointer
Binding Mouse to Pointer
"Paste" Menu Item pos
Pointer Pos
Move Pointer Pointer pos != Cut Menu Item pos
Move Pointer Pointer pos = Paste Menu Item pos
Binding: Click left mouse button to selection action
Click Left Mouse Button
Click Left Button

Figure 6.13 Analysis of the "Paste File" state in Figure 6.3

As is evident from the figures, there are knowledge states that are repeated many times with very few differences. For example, the knowledge state of moving the mouse pointer is the same in all subtasks, their only difference being the target of the motion. Such knowledge states may be considered design patterns (Gamma, Helm, Johnson, & Vlissides, 1994), and as such they can be reused without drawing their constituent knowledge, nor any actions that lead back to the same knowledge state. Complete tasks may also be considered design patters. For example, the whole “File move” task described in Figures 6.9 to 6.13 can be represented as a single knowledge state in a larger diagram that requires or allows the performance of this task. These knowledge states represent one coherent unit of knowledge that can be applied in many different situations. This is also supported by other task analysis theories, such as TKS (H. Johnson & P. Johnson, 1991; P. Johnson & H. Johnson, 1991; P. Johnson et al., 1988). TKS supports that knowledge that represents a coherent whole can be retrieved together as one chunk of knowledge, thus justifying the use of one knowledge state to represent the performance of a task.

The knowledge chunks in Figures 6.10 to 6.13 mention specific IOs and INs, such as the pointer and the mouse. However, when expert users are presented with a touchpoint or a touchpad, they are immediately able to use these instead of the mouse to control the pointer. Also, the mouse may not control the pointer, as is frequently the case in first-person shooter games. Rather, the mouse controls the point-of-view of the user. But expert users are able to control point-of-view with the mouse in the same manner they control the pointer. Thus, instead of referring to bindings of mouse to pointer, and clicking on folders and files, I replace these with the generic terms DO, IO, and IN.

This generalization provides a graphical way of showing how users may create knowledge structures that apply to different situations with some similarities. As these structures become more general, they apply to more situations, showing that the user abstracts the knowledge required for the performance of a task, which many believe is a direct effect of experience (Anderson, 1992; Logan, 1988; Palumbo, 1990). The generalization is contrasted in Figures 6.14 and 6.15. While they are the same, Figure 6.15 uses the generic terms instead of the specific terms used in Figure 6.14.

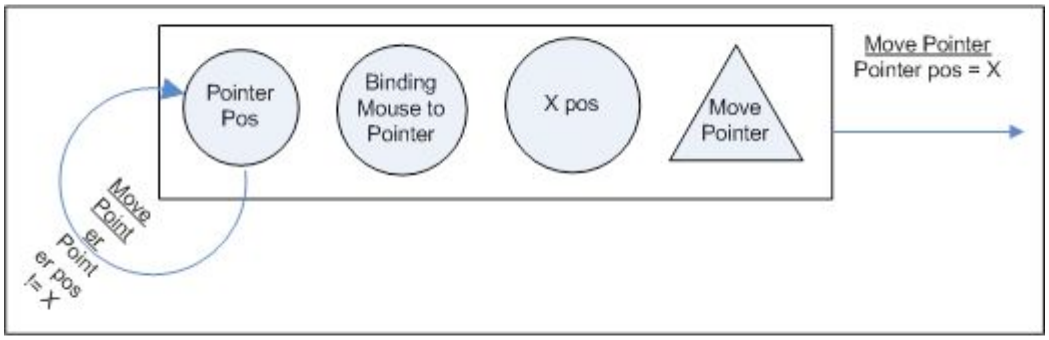


Figure 6.14 The Knowledge State representing the move pointer action, using a variable. Variable X is used to represent the target of the pointer movement, without being instantiated.

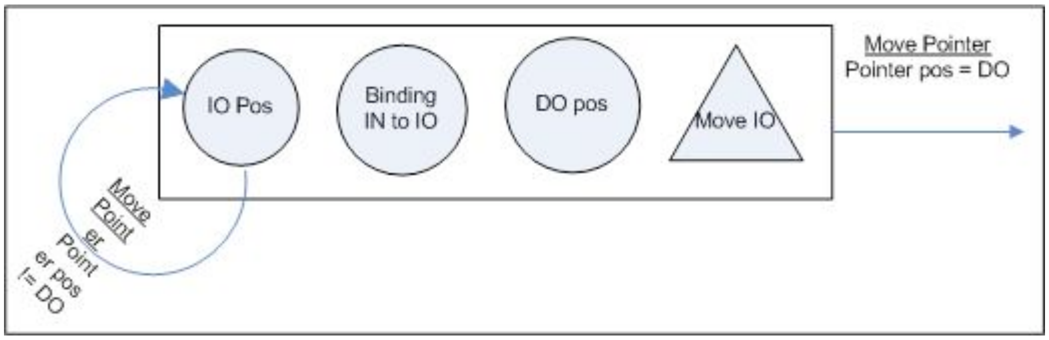


Figure 6.15 The Knowledge state representing the 'Move Pointer' action with instantiated variables. All the references to specific interface elements have been replaced with generic references to Data, Interaction and Intermediary Objects to reflect the generic nature of the knowledge state.

6.3 Chunking and Automatic Processes

Considering the case of users who have some knowledge about the performance of a task but lack other knowledge, in CoDeIn the chunks that are not known are considered to add infinite retrieval time to the model, and thus make the performance of the task impossible. Once users learn the chunks that they do not know, then they retrieve each chunk of knowledge from long-term memory. This retrieval is associated with a time cost. Users gain experience in the performance of the task by repeatedly performing it, and this leads to the increment of their ER and decreases the amount of time associated with the retrieval of knowledge from long-term memory. The explanation in the Cognitive Science literature for this process is that through this process the task becomes automatic, or close to automatic (Anderson, 1992; Logan, 1988; Palumbo, 1990; Ritter & Schooler, 2001). Chapter 8 of this thesis describes an experiment that demonstrates this effect, and shows how CoDeIn models it.

Once these users know all the declarative chunks of knowledge that the task requires, their ability to perform the procedural aspects of the task is equal to that of an expert user, denoted by an efficiency rating equal to 1.

The diagramming method of CoDeIn shows this process concretely by associating retrieval times to all the knowledge chunks that are independently retrieved from long-term memory, and as users gain expertise the retrieval time decreases, relative to the users' increasing ER. Therefore, similar to other theories of cognition, CoDeIn utilizes the concepts of chunking and automatic performance. However, CoDeIn shows the components of the task and what is learned more fully than previous representations, because the description of tasks includes the knowledge required for the task's performance.

6.4 Summary

This chapter presented the nomenclature and notation used in the CoDeIn framework. The terms DO, IO, and IN were introduced to refer to the constituents of a user interface, without using interaction-style-specific notation. The concept of bindings was also introduced to mean the “connections” between DOs and IOs or IOs and INs.

CoDeIn’s diagrammatic notation was presented with examples that demonstrate how the notation allows the creation of models of task performance. The primary difference between CoDeIn’s notation and the notation of other task analysis methods is that CoDeIn presents and bases models created with it on the presumed task-performance knowledge users are required to possess.

A simple DM “File move” task is used to exemplify the procedure of describing a task using CoDeIn. The example is also used to show how general actions can be described using notation that is independent of interaction style.

It was then suggested that knowledge states and task models may be used as design patterns (Gamma et al., 1994) allowing the creation of smaller, high-level models.

Finally, the concepts of chunking and of task automatization are discussed and it is shown how CoDeIn explains the occurrence of this process. The next chapter uses these concepts for the evaluation of an experimental task.

Chapter 7 CoDeIn vs. GOMSL

This chapter presents an example of CoDeIn's task-completion-time prediction capabilities. An experiment is performed that times the completion of a task in a DM interface and a TUI interface. The experimental results then are compared to the completion time prediction of a GOMSL model (Kieras, 1999) and to the completion time prediction of CoDeIn.

One objective of this experiment is to examine whether CoDeIn can model tasks performed in interfaces designed in different interaction styles. A second objective is to determine whether CoDeIn can estimate the completion time of modeled tasks with similar accuracy to estimates of existing evaluation methods, such as GOMSL (Kieras, 1999).

The experimental task is the "file move" task described in Chapter 6. Here it is designed and tested in two different interaction styles, DM (Hutchins et al., 1986; Shneiderman, 1983) and RBI (Jacob, 2004; , 2006). Because RBIs are a whole class of interaction styles as discussed in Chapter 2, I chose a Tangible User Interface (TUI) (Ishii & Ullmer, 1997; Ullmer & Ishii, 2000; , 2001) as a representative for RBIs. I analyze the two conditions by using GOMSL (B. E. John & Kieras, 1996a; Kieras, 1999) and Fitts' Law (Fitts, 1954; Fitts & Peterson, 1964). Then, I use the CoDeIn framework and compare its estimates for each interface to those of GOMSL (Kieras, 1999). CoDeIn provides better predictions for both the DM and the RBI conditions than GOMS and Fitts' Law combined.

The rest of the chapter is organized as follows. First, I present the "file-move" and introduce the two conditions of the experiment, DM and RBI, together with models of each condition's task in GOMSL and CoDeIn. Two more experiments are then presented that provide additional information required for estimating the completion time of the task for each condition. Subsequently I discuss the findings of all three experiments and draw conclusions about the ability of CoDeIn to estimate the completion times of tasks, compared to the GOMSL's same ability.

7.1 Experiment 1

The experiment consisted of having a group of 13 experienced MS Windows XP™ participants perform a file manipulation task in two conditions.

7.1.1 Participants

The 13 participants, 9 male and 4 female, were all undergraduate Computer Science Students at the Computer Science department of Cyprus College and their age varied between 19 to 30 years old. All the participants were enrolled in an introductory Human-Computer Interaction course at the time of the experiment and participated in the study for course credit.

7.1.2 Materials and Design

The task was designed in two different interaction styles, as shown in Figures 7.1 and 7.2: one condition was based on MS Windows XP™ representing the DM interaction style (Figure 7.1), and the second condition was designed as a TUI using a mockup of the actual interface (Figure 7.2). Because all that is of interest is the interaction with the interface, it was unnecessary to build the computer-supported tangible user interface to support the task.

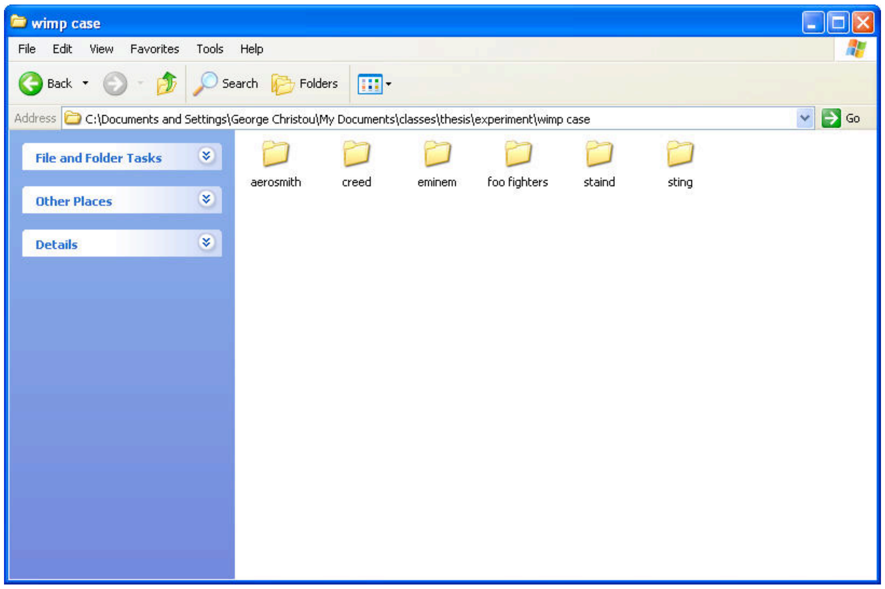


Figure 7.1 The WIMP condition layout of the experiment

The TUI condition's interface was built as a "Wizard of Oz" interface. The folders were represented by cardboard boxes, with 16cm length and 14cm width. The name of each folder was written on the front side of each box. Inside each box there were regular paper cutouts of file objects, measuring 7.2cm wide and 5.4cm long. Each folder had a minimum of 3 and a maximum of 11 file objects. The file objects represented the files that were inside each folder; the name of each file they represented was printed in a box in the middle of the paper cutout. The file objects are shown in Figure 7.3.

Each experimental condition had one task to be performed, and the song/album pairs were the same for all subjects, and for both conditions. All participants performed the task in both conditions, and performed the tasks in the two conditions in random order. Each participant performed ten trials per condition.

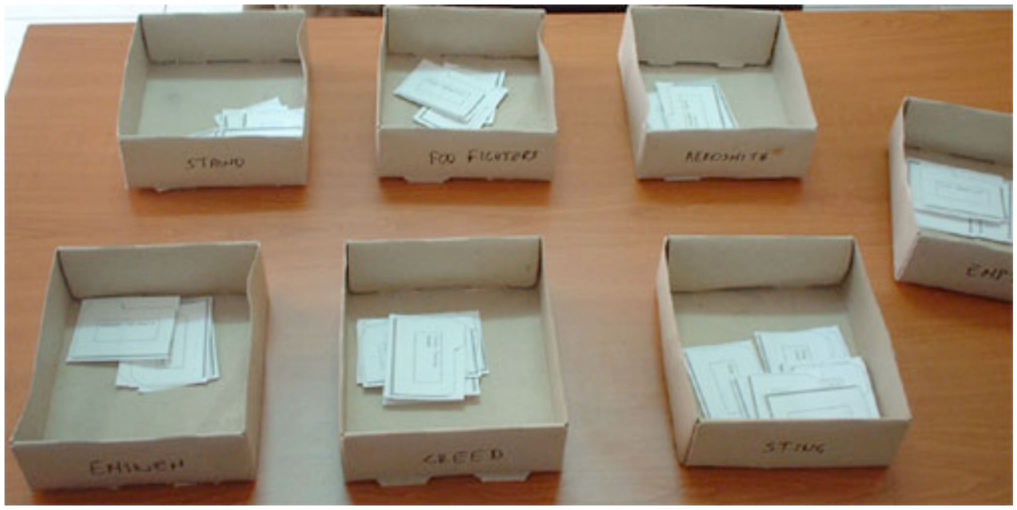


Figure 7.2 The TUI condition layout of the experiment

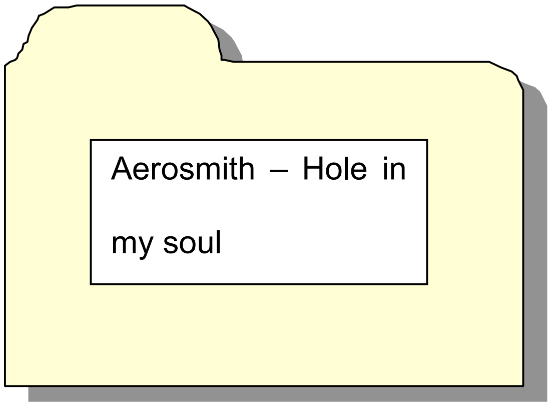


Figure 7.3 The regular paper cutout used in the TUI condition.

7.1.3 Procedure

Participants were asked to perform the following task: “Find the mp3 file called X, in folder Y and move it from folder Y to folder Z”, where the mp3 file X, the source folder Y and the target folder Z were disclosed to the participants prior to the beginning of each trial. For example, a participant might be asked: “Find the mp3 file called ‘Cleaning my closet’ by Eminem, in folder ‘Sting’ (source folder), and move it to folder ‘Eminem’ (destination folder)”.

Figure 7.1 shows the layout of the DM condition of the task. The participants began each trial with the pointer at the right side of the window, not shown in Figure 7.1. This was considered to be the pointer’s home position, and all subsequent Fitts’ Law calculations for the first movement of the condition’s task are calculated from that position. The participants proceed to move the pointer above the source folder, double-click it, position the pointer over the target file, right-click, and select cut from the popup menu. This completed the first subtask of selecting the target mp3 file. Then the participants had to move the pointer onto the “up folder” icon on the window toolbar, double-click on the destination folder icon, and right-click inside of the white-space area of the destination folder window, and select paste. This would signal the end of the trial. The task was modeled in GOMSL (Kieras, 1999), with the model shown in Table 7.1. GOMSL was chosen from the various GOMS methods based on recommendations given by John and Kieras (1996b).

The task was timed using a handheld stopwatch. The experimenter signaled the participant to start the trial by saying ‘Go!’, and at the same time began timing the task. The experimenter stopped the stopwatch when the icon of the target file appeared in the target folder.

Figure 7.2 shows the layout that was used for the TUI condition of the experiment. The TUI condition’s task was performed in the following manner: the participants reached into the source folder, grabbed the file objects, and physically searched through them. When they found the target mp3 file they placed it into the destination folder. This signaled the end of the trial. The TUI condition's task is shown as a GOMSL model in Table 7.2.

7.1.4 Results

The results of the experiment are shown in Table 7.1. The results in the DM and TUI columns represent the task completion times in the DM and TUI conditions of the experiment, respectively.

Table 7.1 Average completion times (in seconds) by participant for the two experimental conditions

Participant	DM	TUI
1	8.00	1.80
2	7.5	5.53
3	13.35	4.03
4	12.33	2.36
5	7.37	4.02
6	21.33	6.07
7	31.22	8.05
8	7.86	2.24
9	9.12	3.01
10	11.00	6.88
11	24.00	5.68
12	15.00	6.74
13	7.00	7.16
Average	13.47	4.89
Std. Dev.	7.59	2.10

The first result we can note in Table 7.1 is that the TUI condition task can be completed faster than the DM condition, and the standard deviation of the completion time of the DM condition's task is greater than that of the TUI condition. This was expected, because the tasks are not entirely equivalent. For example, the TUI condition's folder representation was always open, whereas in the DM condition's task the participants needed to perform actions to open the required folders.

7.2 GOMSL Models

Tables 7.2 and 7.3 show the GOMSL (Kieras, 1999) models for the tasks of the DM and TUI conditions, respectively. Each step and each method in the two tables has next to it, in parentheses, the estimated completion time for the task. The execution time for each GOMSL method is calculated by summing up all the execution times of the primitive operators and adding 0.05 seconds for each step in the method (Kieras, 1999) that represents the mental decision to execute that step.

The primitive operators are "Keystroke Level Method" (KLM) operators as defined by Card et al. (1983). For each "method for accomplishing goal" step, 0.05 seconds are added, and the same applies for each selection rule. Any 'Decide' operators take 0.05 seconds to execute, regardless of the number of conditions. Next to each step in the GOMSL methods, the average time (in seconds) for the performance of the operator is given in parentheses as computed by Card et al. (1983). The total time needed for the performance of a method is in parentheses next to the beginning of the method. The time required for a step in the method to be executed, as well as the time needed for calls to methods, is not shown, but is included in the total time calculated for each method.

Table 7.2 The GOMSL model for the DM condition of experiment 1

Top Level Method (16.15 s)
Method for Goal: Move file from source folder to destination folder
Step 1: Accomplish Goal Open Source Folder (1.70 s)
Step 2: Accomplish Goal Cut Required File (4.20 s)
Step 3: Accomplish Goal Go back to folder list (2.75 s)
Step 4: Accomplish Goal Open Destination folder (3.00 s)
Step 5: Accomplish Goal Paste file (4.15 s)
Step 6: Return with goal accomplished

Method for Goal: Open Source Folder (Total time = 1.70 s)
Step 1: Point to Source Folder (P = 1.1 s)
Step 2: Double-click mouse button (BB = 0.4 s)
Step 3: Return with goal accomplished

Method for Goal: Cut Required File (Total time = 4.20 s)
Step 1: Recall that required file is Y (negligible)
Step 2: Locate Y on the screen (M = 1.2 s)
Step 3: Point to Y (P = 1.1 s)
Step 4: Click mouse button (B = 0.2 s)
Step 5: Point to “Cut”. (P = 1.1 s)
Step 6: Click mouse button (B = 0.2 s)
Step 7: Return with goal accomplished

Method for Goal: Go back to folder list (Total time = 2.75 s)
Step 1: Locate “Up Folder” Button (M = 1.2 s)
Step 2: Point to “Up Folder” (P = 1.1 s)
Step 3: Click mouse button (BB = 0.2 s)
Step 4: Return with goal accomplished

Method for Goal: Open Destination Folder (Total time = 3.00 s)
Step 1: Recall that the Target folder’s name is X (negligible)
Step 2: Locate X on the screen (M = 1.2 s)
Step 3: Point to X (P = 1.1 s)
Step 4: Double-click mouse button (BB = 0.4 s)
Step 5: Return with goal accomplished

Method for Goal: Paste file (Total time = 4.15 s)
Step 1: Point to white space (P = 1.1 s)
Step 2: Click mouse button (B = 0.2 s)
Step 3: Locate “Paste” Menu Item (M = 1.2 s)
Step 4: Point to “Paste” (P = 1.1 s)
Step 5: Click mouse button (B = 0.2 s)
Step 6: Return with goal accomplished

The TUI condition' task modeled in GOMSL, with three GOMS methods, shown in Table 7.3. The three methods capture the three parts of the task: (1) finding the source folder and grabbing the file objects, (2) searching through the pile of file objects to find the required one, and (3) placing the required file object in the target folder and the rest of the file objects back in the source folder.

In the model shown in Table 7.2, there are two peculiarities. The first is that in two methods, "Find Folders and Grab File Objects", and "Place File in Destination Folder", I use a primitive operator that is not included in either the documentation of GOMSL (Kieras, 1999), nor in the KLM primitive operators (Card et al., 1983). This operator is signified with the letter R in step 2 of the first method and steps 2 and 3 of the second method, and it represents the prehension action that users perform when they reach and grasp an artifact. GOMS does not have any operators that represent hand or arm pointing actions, and the literature suggests that the prehension action cannot be modeled by Fitts' Law (Jones & Lederman, 2006; C. L. Mackenzie & Iberall, 1994). However, the surveyed literature does not seem to provide a model that, like Fitts' Law, allows the prediction of the prehension movement execution time based on some function of the distance of the actor from the target. Thus, another experiment was necessary in order to find a model that estimates the completion time of the prehension operator in step 2 of the 'Move File from Source Folder to Destination Folder' method, and step 2 of the 'Place File in Destination Folder' method in Table 7.2. I refer to this as Experiment 2.

Table 7.3 The GOMSL model for the TUI condition of Experiment 1

Top Level Method (8.95 s):
Method for Goal: Move file from source folder to destination folder
Step 1: Accomplish Goal Find Folders and Grab File Objects (2.25 s)
Step 2: Accomplish Goal Find Required File (3.5 s)
Step 3: Accomplish Goal Place File in Destination Folder (2.95 s)
Step 4: Return with Goal Accomplished

Method for Goal: Find Folders and Grab File Objects (2.25 s)
Step 1: Locate source and target folders (M = 1.2 s)
Step 2: Reach into source folder and grab files (R = 0.75 s)
Step 3: Examine the name of the file object on top of the file object pile (negligible)
Step 4: Decide: If the filename matches the required filename Then Accomplish Goal: 'Place File in Destination Folder' (2.95 s) Else Accomplish Goal: 'Find required file' (3.5 s) (Step = 0.05 s)
Step 5: Return with goal accomplished

Method for Goal: Find Required File (3.5 s)
Step 1: Move file object from top of file object file to bottom.
Step 2: Examine the name of the file object on top of the file object pile
Step 3: Decide: If the filename matches the required filename Then 'Return with goal accomplished' Else Accomplish Goal: 'Find Required File'

Method for Goal: Place File in Destination Folder (2.95 s)
Step 1: Locate destination folder (M = 1.2 s)
Step 2: Place file in the destination folder (R = 0.75 s)
Step 3: Place remaining files in source folder (R = 0.75 s)
Step 4: Return with goal accomplished

The second peculiarity is that GOMSL cannot compute the completion time of the method "Find Required File", because there are no primitive operators defined for the steps of this method, and because of its iterative nature. Therefore, a third experiment was performed to calculate the completion time of this method.

7.3 Experiment 2

Experiment 2 was performed to find an average for the completion time for the grasping action in the TUI condition of the task.

7.3.1 Participants

Experiment 2 involved 18 new participants, ages between 20 and 30 years old, 15 males and 3 females, all college students of the Department of Computer Science of Cyprus College. The participants were asked to perform the reaching action of the TUI condition of the first experiment.

7.3.2 Materials and Design

The participants were seated in front of a desk and were asked to place their dominant hand on the edge of the desk. The office was measured on its width, and we placed a scale with markings at every 5cm from the edge closest to the participant, to the other edge of the desk. The scale was marked from 5cm to 50cm.

7.3.3 Procedure

Each trial was performed as follows: a box that was the same as the one in experiment 1 was placed on one of the markings. Inside the box we placed a stack of 11 file objects, like the one shown in figure 3. The participant was asked to reach into the box as fast as they could on the verbal cue of the experimenter, and grab the stack of file objects. The experimenter timed the participant, using a stopwatch, and noted the completion time of the movement. The experimenter stopped the stopwatch when the participant had a firm grip on the stack of file objects. If the participant did not manage to grab the file objects, or did not grab all the file objects at once, the trial was performed again. Each participant performed the task ten times, with the placement of the box being random for each trial.

7.3.4 Results and Analysis

The average completion times for each distance, along with the associated Index of Difficulty and task completion time are shown in Table 7.4.

Table 7.4 Distance, average completion time and Index of Difficulty for Experiment 2. The distance is measured in centimeters and the time in seconds.

Distance	ID	Time	Standard Deviation
5	0.39	0.68	0.12
10	0.70	0.70	0.14
15	0.95	0.72	0.28
20	1.17	0.72	0.13
25	1.36	0.69	0.12
30	1.52	0.74	0.16
35	1.67	0.74	0.15
40	1.81	0.82	0.16
45	1.93	0.82	0.22
50	2.04	0.84	0.16
Average Completion Time		**0.75**	**0.16**

Figure 7.4 shows the relationship between the ID and the completion time of the action, with the regressed line passing through the data. The results of the regression analysis on ID vs. time indicated a reliable relationship (R^2 = .73, F = 22.0, p < .005).

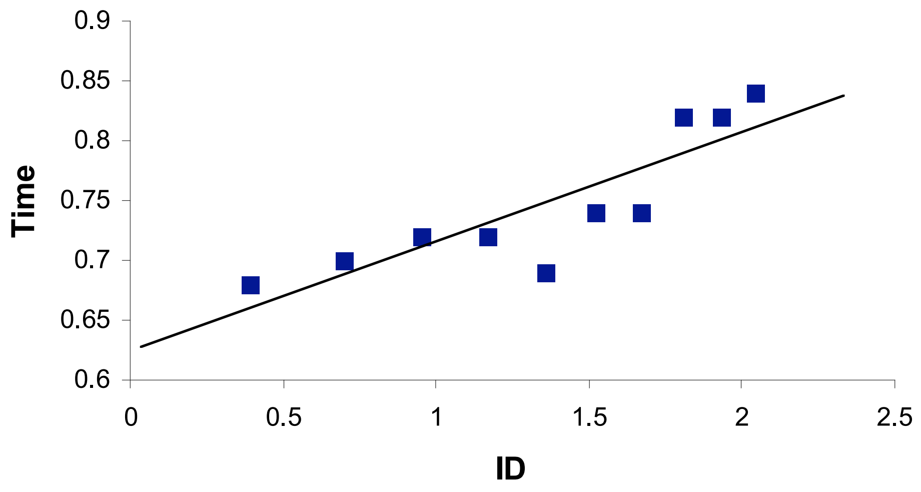


Figure 7.4 Experiment 2's ID vs. Task Completion Time.
The straight line represents the trend of the data, as calculated with regression analysis.

Figure 7.5 shows the relationship between distance and time in graphical form, with a regressed line passing through the results. The regression analysis we performed on distance vs. time also indicates a more reliable relationship ($R^2 = .82$, $F = 36.81$, $p < .0005$) than that between the ID and the completion time.

A curve estimation analysis was performed to investigate whether Fitts' Law may explain the same results, using a logarithmic model in SPSS, on distance vs. time. However, the results were not statistically significant ($R^2 = .64$, $F = 14.21$, $p > .005$). The logarithmic model is shown as a red line in Figure 7.5.

The three regression models may seem to have similar R^2 values (.73, .82 and .64), but the fitness of the model of distance versus time is statistically significant at $p < .0005$, whereas the other two models are statistically significant at $p < .005$. The statistical significance of the linear model is greater than the other two by one order of magnitude.

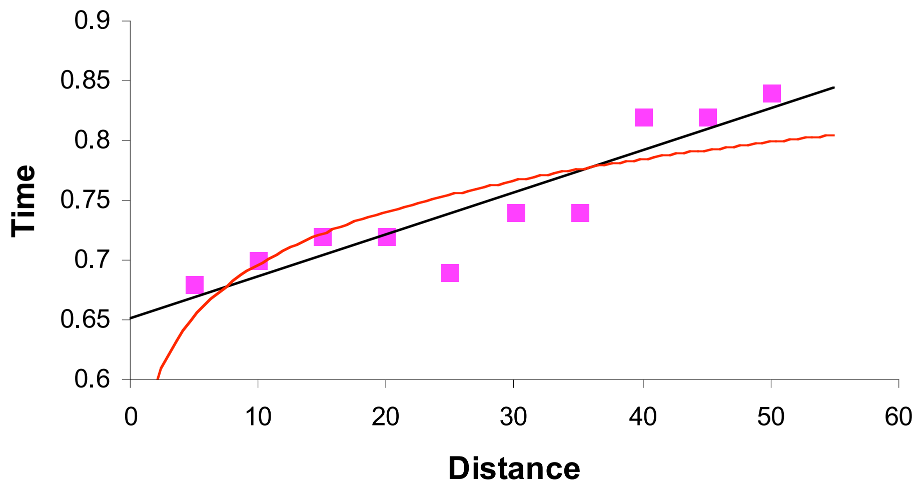


Figure 7.5 Experiment 2's Distance vs. Task Completion Time. The straight line represents the trend of the data, as calculated with regression analysis.

Therefore, this experiment resulted in the corroboration of the literature about prehension motions (Jones & Lederman, 2006; C. L. Mackenzie & Iberall, 1994) that posit that these motions are not as fast as Fitts' Law would suggest. Rather, the prehension motion completion time is linearly correlated to the distance of the actor from the object.

As confirmed in experiment 2, the assumption that the prehension motion cannot be modeled by Fitts' Law. Hence, the experimental average of experiment 2 was used for the calculation of the TUI condition completion time in the GOMS and CoDeIn models.

7.4 Experiment 3

Experiment 3 was performed to find an estimate for the search strategy that participants used during their performance of the TUI condition of experiment 1. Because the file objects were represented by pieces of paper with the filename printed on them, when the participants grabbed the files, they held all the objects like a stack. To find the target file object, the participants examined the topmost object, and if it was not the required one, then they moved the object to the bottom of the pile. They iteratively performed this procedure until they found the required object.

Due to the iterative nature of the participants' search, the search time seemed to be based on the number of objects they had to go through until they found the required object. Therefore, the hypothesis of this experiment is that the time to find the target object is directly proportional to the number of objects, as follows: ECT = constant * (Number of Objects), where the ECT is the Expected Completion Time of the experiment. In this formula, I assume that there exists a linear relationship between the completion time of the search and the number of file objects above the required file object in the pile. This is because of the increased number of times that the participant must move the other file objects to the bottom of the pile, until the target file object is reached. I also assume that the reading time of each filename is approximately constant, as is the time to move an object from the top of the pile to the bottom.

7.4.1 Participants

The supplementary experiment involved 10 new participants, ages from 20 to 30, 7 males and 3 females, who were asked to only perform the search through the tangible file objects in the aforementioned way, until they found the target object.

7.4.2 Materials and Design

The file objects that were used were identical to the ones used in the TUI condition of experiment 1, as shown in Figure 7.3. The time was measured using a handheld stopwatch, operated by the experimenter.

7.4.3 Procedure

Each participant was asked to find the required file object in a pile of 10 other file objects, for a total of 11 file objects in the pile. The required file object was inserted at a specific position by the experimenter, from 2nd to 10th in the pile. The presentation order of the pile was random, to avoid order effects. For example, the participant would be presented first with a pile that had the required file object 4th, then 10th, then 6th, and so on. The experimenter made sure that the participants did not see where the required file object was placed before each trial. Each participant performed 10 searches at each of the 9 possible positions of the required file object in the pile.

The participants started each trial with the stack of file objects in their hands, without knowing where the target object was in the pile. As soon as the experimenter gave the verbal cue to the participant to start, the participant would go through the stack of objects using the required search strategy described previously, and the participant would say “Stop” when they would find the required file object. At that point the experimenter would stop the stopwatch and record the time and the position of the target file object.

7.4.4 Results and Analysis

The result of this experiment gave the average time for the search that is used in the GOMS method “Find Required File”, shown in Table 7.3. Additionally, Table 7.5 shows the average completion time vs. the position of the required file, and Figure 7.6 shows a graph with the relationship between the average time taken to find the required file object in the pile, against the required file object’s position in the pile. Figure 7.6 shows that the relationship appears to be linear on the position of the required file object in the pile. Also, linear regression analysis on the data, shown in Table 7.6, results in a very strong linear relationship, ($R^2 = .999$, $F = 15{,}434.05$, $p = 5.77 \times 10^{-13}$).

The linear regression equation that calculates the Completion Time (CT) from the experiment results is CT = -0.02 + 0.59 * (Position of target). The acquired formula for CT closely resembles the hypothesized formula (ECT) for the estimation of the completion time of the search. The formula has a very small error, in that the time taken for the task if there are no file objects cannot be negative, whereas the y-intercept of the equation is negative. However, the standard error of the experiment is 0.03. Using the standard error, the estimate for the y-intercept is found to be in the range of [-0.05, 0.01]. I operate under the assumption that if there are no file objects, then the subject will not need any time to search, which means the time for 0 objects should be 0 seconds, which is inside the range given for the intercept.

I conclude that the relationship between the time and the position of the item in the pile is a linear one.

Table 7.5 Average Search Time vs. Required File Object Position in Pile

Required File Object Position in Pile	Average Completion Time (in seconds)
2	1.1
3	1.7
4	2.3
5	2.9
6	3.5
7	4.1
8	4.6
9	5.2
10	5.8
Average	**3.5**

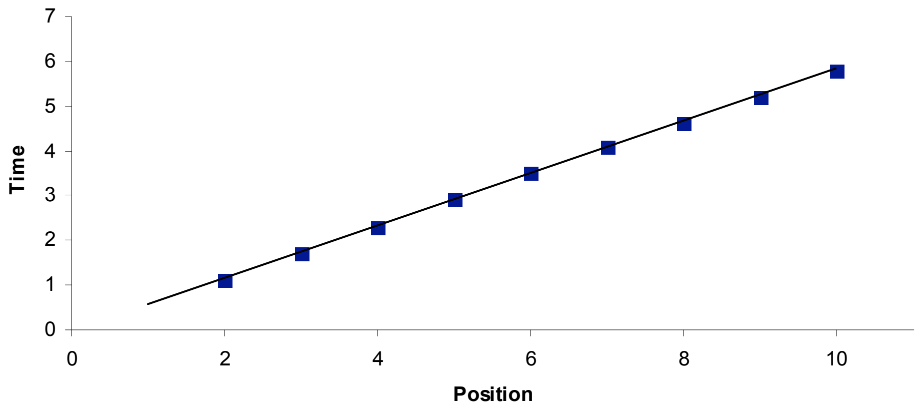


Figure 7.6 Experiment 3's Required file object's position vs. average search time. The times shown are in seconds, and the line showed is the regressed line found using linear regression analysis.

7.5 Refinement of the GOMSL Models

The results of the two experiments (experiment 2 and experiment 3) were used to refine the GOMSL model. The result of experiment 2 was used for the calculation of the R operator in Table 7.3, and the result of experiment 3 was used as the completion time for method “Find Required File” in Table 7.3.

Fitts’ Law was also used to make the GOMSL model of the DM condition (Table 7.2) more accurate. The pointing form of Fitts’ Law was used as indicated by Mackenzie (1991): MT = 230 + 166*ID, because the task only involved pointing motions with the mouse. The participants were specifically asked not to use the drag-and-drop method of moving files in the Windows environment.

In the above equation MT is the Movement Completion Time and ID is the index of difficulty for the task using the Shannon formulation (I. S. MacKenzie & Buxton, 1994), as was discussed in Chapter 3. Each movement’s ID was calculated independently, because each movement had different start and end points, and different targets. Using the distances of pointer travel on the screen, I computed the results shown in Table 7.6 that also delineates the GOMSL Method Steps of Table 7.2 whose times were replaced by the Fitts’ Law calculations. The total calculated time using Fitts’ Law and GOMSL combined is 12.98 seconds.

Table 7.6 DM Condition's Fitts' Law calculations.

GOMSL Method Step Replaced	Description of Fitts’ Law Result
Method Open Source folder, step 1: from 1.1 to 0.80s	Moving from the home position of the cursor to the source folder (ID = 3.46, MT = 0.80s).
Method Cut Required File, step 3: from	Moving from the current position of the cursor

1.1 to 0.82s	to the required file (ID = 3.53, MT = 0.82s).
Method Cut Required File, step 5: from 1.1 to 0.63s	Moving from the file icon to the "Cut" menu item (ID = 2.44, MT = 0.63s).
Method Go Back to Folder List, step 2: from 1.1 to 0.72s	Moving from the "Cut" menu item position to the "up hierarchy" button on the window toolbar (ID = 2.96, MT = 0.72s).
Method Open Destination Folder, step 3: from 1.1 to 0.52s	Moving from the "up hierarchy" button to the destination folder (ID = 1.75, MT = 0.52s).
Method Paste File, step 1: from 1.1 to 0.52s	Moving from the destination folder position to the unused area of the folder window (ID = 1.75, MT = 0.52s).
Method Paste File, step 4: from 1.1 to 0.52s	Moving from the current pointer position to the "Paste" menu item (ID = 1.75, MT = 0.52).

7.6 CoDeIn Models

The same Fitts' Law calculations were used in the CoDeIn model, shown in Figures 7.7 – 7.10. As explained in the previous chapter, the model is built by first creating a summary diagram that shows the constituent subtasks of the task. This is shown in Figure 7.7. The summary diagram can be used like the top level method of the GOMSL model as well. By using it, one can calculate the time that the user spends at every state, and how long it takes for each transition from one state to the other. By the addition of all the calculated times, a prediction for the completion time of the task is found. Because this diagram has the same states as the steps of the top level method of the GOMSL model of Table 7.1, the time of each step can be assigned to each of the states, and reach the same completion time estimate as that of the GOMSL model.

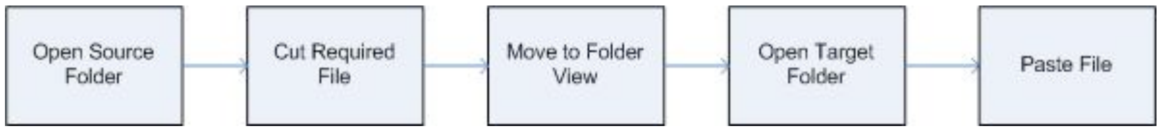


Figure 7.7 A summary diagram of the DM condition of the experiment

Figures 7.8 – 7.10 show refined views of the subtasks in Figure 7.7. These show generalized knowledge states, because the assumed user that performs the task is classified as an expert who has already built the abstract knowledge structures required for the experimental task. In Figure 7.8 there are two knowledge states that refer to the IO and IN of the DM condition (i.e., the pointer and the mouse). Because of the commonality of these devices the correspondence is assumed and therefore omitted. The variable X_DO used in Figures 7.8 to 7.10 is instantiated to the appropriate DO during the performance of each action. Figure 7.8, for example, may be used to represent both the “Open Source Folder” subtask and the “Open Target Folder subtask”; X_DO is bound either to the Source folder or the Target folder, according to which sub-task is being modeled. Because the task of opening folders is the same, the only difference being the target of the task, one may consider the performance of the opening any folder as a general procedure, with only the target of the procedure being different. This is captured in the model of Figure 7.8 by using the variable X_DO. Next to each binding of the X_DO variable, in parentheses, I include the time calculated (by Fitts’ Law) to execute the action that includes them as targets (in this case, the “Move IO” action).

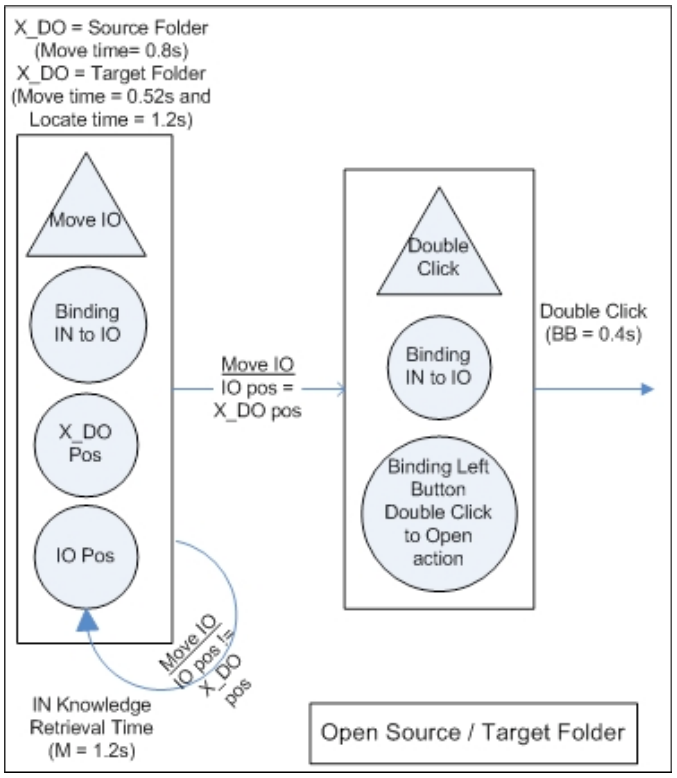


Figure 7.8 The "Open source/target folder" knowledge state from the summary diagram of Figure 7.4, enlarged to show the knowledge and constituent actions for its completion

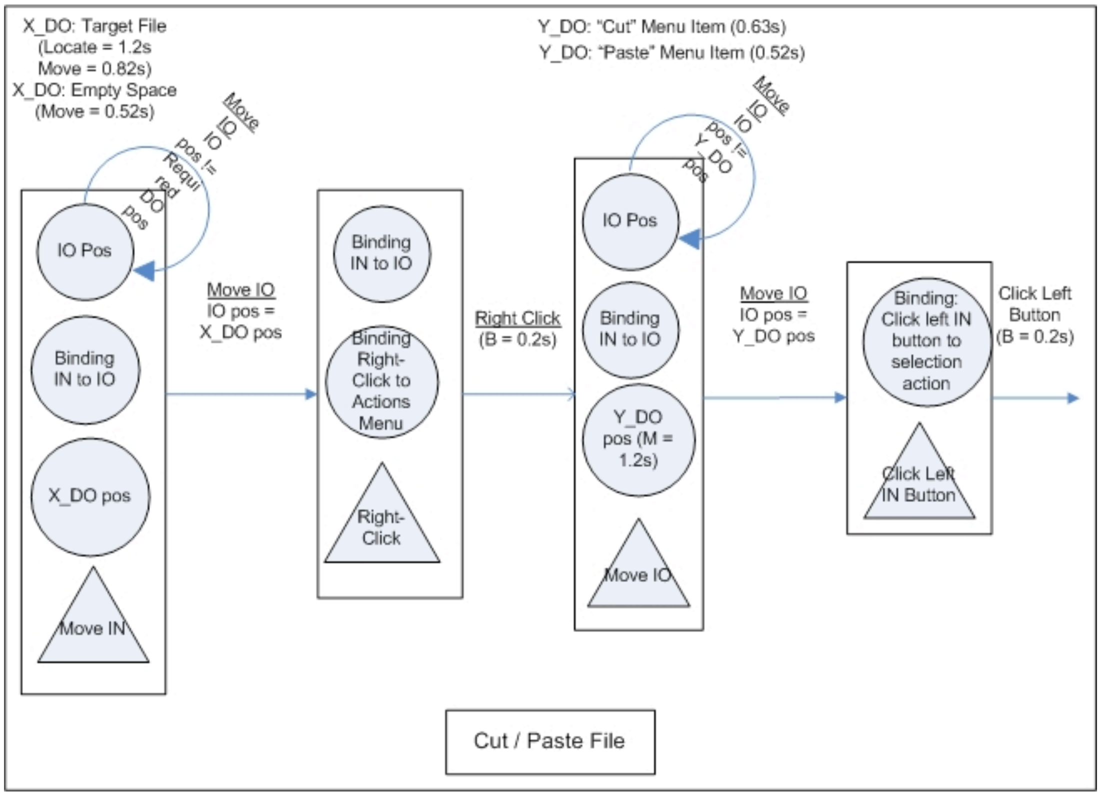


Figure 7.9 The analysis of the "Cut/Paste File" states in the summary diagram of Figure 7.4.

There are two arrows that come out of the first knowledge state, both marked as the action “Move IO”. The topmost arrow includes the condition that signifies that if the IO’s position is over X_DO, then the user can move on to the next knowledge state to complete the task. The second (lowermost arrow) shows that while the IO’s position is not equivalent to the X_DO’s position (I use equivalent with a certain amount of freedom here), then the user needs to continue moving the IO.

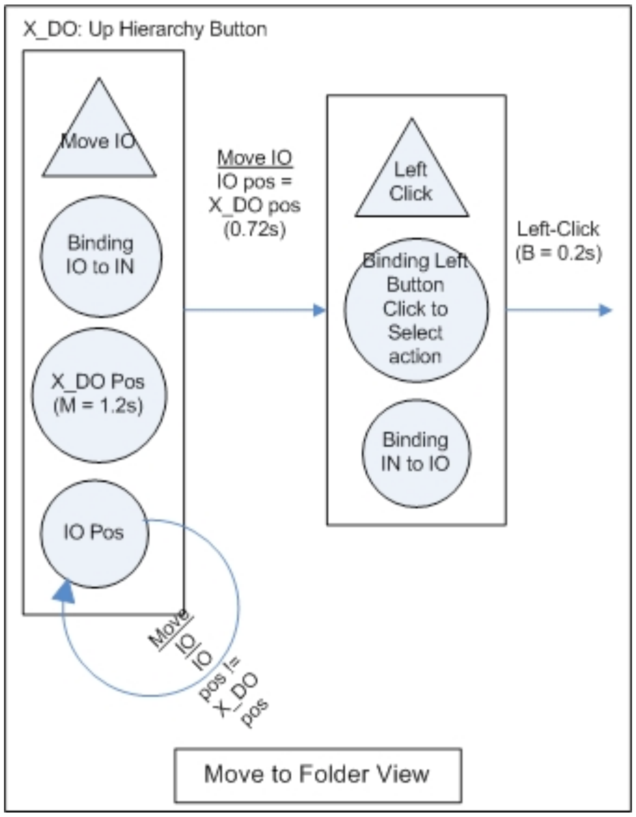

X_DO: Up Hierarchy Button
Move IO
Binding IO to IN
X_DO Pos (M = 1.2s)
IO Pos
Move IO IO pos = X_DO pos (0.72s)
Left Click
Binding Left Button Click to Select action
Binding IN to IO
Left-Click (B = 0.2s)
Move IO IO pos != X_DO pos
Move to Folder View

Figure 7.10 Analysis of the Move to Folder View state in Figure 7.4

There is no search time for the position of X_DO when bound to the source folder, because the experiment began with the participants able to view all the folders. Therefore the participants knew the position of the source folder before the beginning of the experiment, so visual search was not part of the completion time of the task.

Figure 7.11 shows the CoDeIn model for the TUI condition's task. Because this model was smaller than the one required for the DM condition's task, the complete model is shown in one figure.

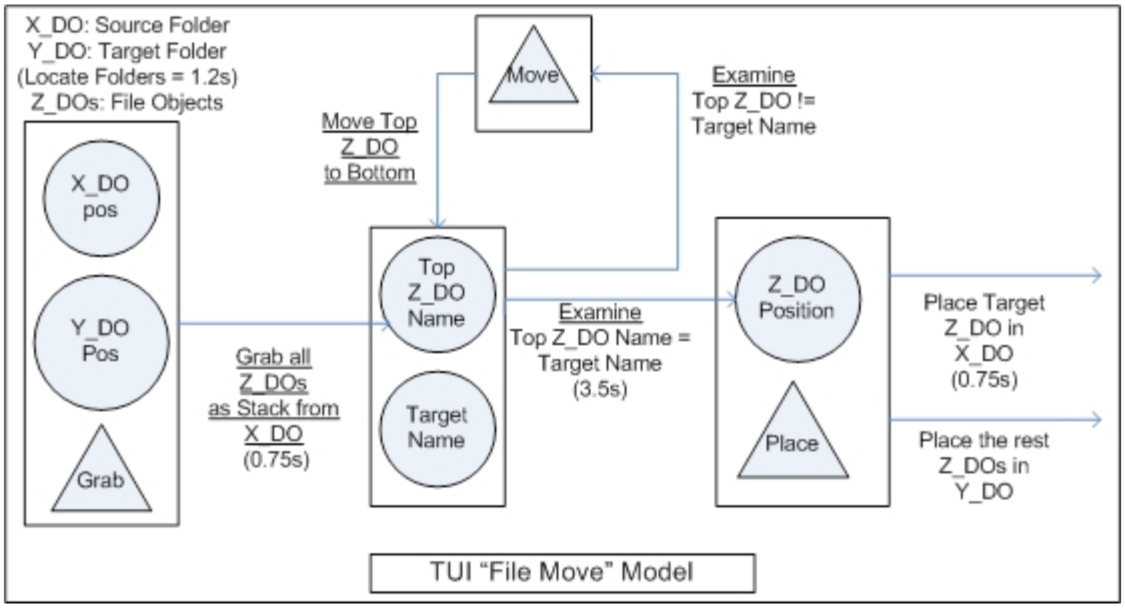


Figure 7.11 The complete analysis of the task in the TUI condition

7.7 Discussion

In this section I compare the results taken from the experiment 1 with the estimates of the GOMSL and CoDeIn models for each condition. Table 7.7 shows the results of each condition with the results of the two analyses. The predicted completion time by GOMS for the DM condition overshoots the actual experimental average by 2.63 seconds. When Fitts' Law is used in the GOMSL model it more accurate: 0.49 seconds deviation less than the experimental average. CoDeIn though, yields the most accurate result for the DM condition, with a deviation from the experimental average of 0.06 seconds.

Table 7.7 Summary of the results of the two conditions of experiment 1. All times shown are in seconds.

	DM	Absolute Deviation	TUI	Absolute Deviation
Experiment 1 Average	13.47	-	4.89	-
GOMSL	16.15	2.63	-	-
Refined GOMSL	12.98	-0.49	8.95	4.06
CoDeIn	13.53	0.06	6.20	1.31

Tables 7.8 and 7.9 show a method by method breakdown of the GOMSL model for each of the two conditions, and they are compared with the equivalent knowledge states of CoDeIn. Table 7.8 shows the differences between the two methods in the evaluation of the DM condition. There are several small differences that are evident in the table. The primary reason in the difference between the three models, the GOMSL model, the refined GOMSL model, and the CoDeIn model, is that when using CoDeIn there is no extra cost of execution for each action, whereas in the two GOMS models, each step in each method requires some decision time to be executed. The large differences between the GOMSL model and the other two models can be attributed to the fact that the GOMSL model uses the pointing operator P that averages short and large distances traveled with the pointer. By substituting the pointing operator's estimated completion time of the pointing actions in the GOMSL model with the estimates using Fitts' Law, the resultant estimate of the model becomes more accurate.

It is a well established fact that GOMS with Fitts' Law can accurately model the performance of tasks designed in the DM interaction style. However, even in the DM environment, CoDeIn outperforms the prediction of GOMS by 430 milliseconds. This result suggests that CoDeIn can model DM tasks and provide predictions with the same or better accuracy as GOMS. However, the CoDeIn model does not only provide a prediction; it also provides information about the knowledge requirements of the task, something that is not evident in the GOMS model. While the GOMS model delineates the procedure required for the performance of the task, the declarative knowledge required for the performance of the task is not shown. On the contrary, in the CoDeIn model, one may declare the procedure for the task as well as both the declarative and procedural knowledge requirements of the task. This allows comparative evaluations between different designs of the task, in the same interaction style, or in different interaction styles. For example, the comparison of the DM condition vs. the TUI condition can be performed on the basis of required knowledge instead of speed of performance using the CoDeIn model. Using GOMS, one would only be able to compare the two designs on the basis of number of steps required for the performance of the task, or on the basis of methods required for the performance of the task. These two measures however, may not be representative of the difficulty of the task, because more or less steps or methods for the performance of the task may not convey the fact that the user may need to learn many more facts in the design that requires fewer steps or methods for its completion. Modeling the task in CoDeIn though, provides this information as well.

Table 7.8 DM condition's subtask breakdown of the analysis of each modeling method. All times shown are in seconds.

Subtask	CoDeIn Estimate	GOMS Estimate	GOMS with Fitts' Law
Open Source Folder	2.40	1.70	1.40
Cut Target File	4.25	4.20	3.45
Go Back to Folder List	2.12	2.75	2.37
Open Destination Folder	2.12	3.00	2.42
Paste File	2.64	4.15	2.99
Top Level Method Execution	-	0.35	0.35
Total Completion Time	13.53	16.15	12.98

Table 7.9 shows that the greatest difference in the two methods for the TUI condition's models is that CoDeIn takes into account that replacing the file objects in the destination folder can be performed in parallel with the replacement of the rest of the file objects in the original folder. There also is a slight difference between each model's estimate in the "Find Folders and Grab File Objects" subtask.

Because GOMSL, the variant of GOMS used in this analysis, lacks the capability of modeling parallel actions, due to its step by step modeling of tasks, it cannot describe the TUI condition correctly. CPM-GOMS (Gray et al., 1993; B. E. John, 2003; B. E. John & Kieras, 1996a; , 1996b) may be more accurate than the GOMSL model shown here, because it can handle parallel actions. However, even the creators of CPM-GOMS suggest that when modeling simple tasks, such as the experimental task here, one should not use CPM-GOMS because of the complexity of model creation, even for simple tasks.

Table 7.9 TUI condition's subtask breakdown of the analysis of each modeling method. All times shown are in seconds.

Subtask	CoDeIn Estimate	GOMS Estimate
Find Folders and Grab File Objects	1.95	2.25
Find Required File	3.50	3.50
Place File in Destination Folder	0.75	2.95
Top Level Method Execution	-	0.25
Total Completion Time	6.40	8.95

7.8 Proposal of the prehension operator in GOMSL

As mentioned before, a corollary result that is worthy of mention is that GOMS does not have a prehension operator that would provide an estimate of the completion time of a reach-and-grasp action. This strongly hinders GOMS from modeling RBIs such as TUIs and VR interfaces, because these interfaces rely on or use extensively the prehension operation for the execution of several actions. As was shown in experiment 2, and posited in motor control literature, Fitts' Law cannot be used to model prehension.

Through the study of other models, and evidence from experiment 2, I propose that a new operator be included in the primitive operator set of GOMSL that captures prehension actions, based on a straight line equation: $CT = \alpha + \beta D$, where CT is the completion time, α and β are constants that have to be calculated according to the task, and D is the distance of the actor from the target.

Having an prehension operator based upon a linear equation also suggests that TUIs and VR interfaces may be slower to use than DM. TUI and VR use prehension as the main manipulation action, instead of pointing as DM does. Prehension’s completion time seems to be directly proportional to the distance of the actor from the target, and not logarithmically proportional as pointing is. Figure 7.12 shows this effect graphically. While at shorter distances prehension may be as fast, or even faster than pointing, as distance becomes greater, the time required for the completion of a prehension action becomes greater than that of a pointing action.

I believe that this effect may not be as noticeable as expected, because TUIs and VR are not used in the same way as DM. The study of this phenomenon is proposed as a future direction directly stemming from this work, because to establish the importance of speed of prehension vs. the importance of the speed of pointing, a survey of the TUI, VR and DM literature is required, to examine how each action is used and in what context each action is used. Then a comparison should be performed, if the situations where RBIs use the prehension action require the same speed of execution as the situations where DM interfaces use the pointing or drag-and-drop action that also obeys Fitts’ Law. Also, it would seem that this result applies less to VR than to TUIs, because VR does not completely rely on the prehension action, as TUI does. Hence a more comprehensive examination is needed to draw conclusions about the comparative speed of execution of equivalent tasks in RBI and DM.

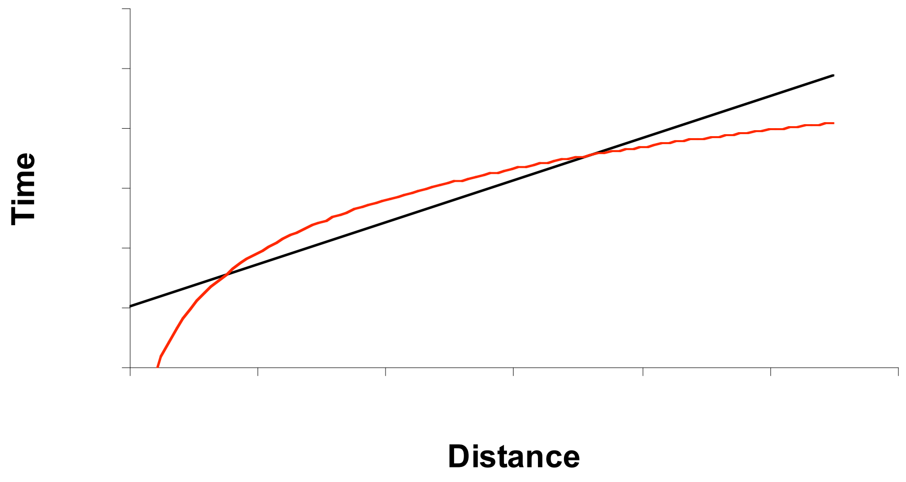


Figure 7.12 Example of linear vs. logarithmic operators.
The black line represents a linear operator and the red line a logarithmic one.

7.9 Conclusions

This chapter presented three models of a task designed in two different interaction styles. First, models of the two conditions of the tasks in GOMS and CoDeIn were created, and then an experiment was performed to provide actual completion times for each condition, so that each model's estimated completion time could be compared. Because of the lack of operators of GOMS that could be applied to the TUI condition's task, two more experiments were performed. Those were used to identify average completion times for the two actions that the KLM model used by GOMS could not provide.

The results of the models and the experimental average found through experiment 1 are shown in Table 7.7. The first notable result was that the participants performed the TUI condition faster than the DM condition’s task. Also, there was less variability in execution time in the TUI condition’s task than in the DM condition’s.

When GOMS was used to analyze the DM condition it estimated a slower completion time, overshooting the actual experimental average completion time. But when GOMS and Fitts’ Law were combined, a much stronger model emerged, which calculated a completion time that was closer to the actual experimental result than GOMS had originally calculated on its own. This is attributed to the fact that the KLM operator P, used to represent pointing actions, estimates a much larger time frame for pointing actions than required in the experimental task. Thus, the use of Fitts’ Law which modeled the pointing actions more accurately, allowed the better estimation.

For the TUI condition GOMS needed an operator to describe prehension, and it also needed a way to calculate completion time of the iterative search in the pile of file objects. Two more experiments were performed to calculate the necessary completion times required for the GOMS model, but even when using the calculated times GOMSL calculated a completion time that was twice that of the experimental average. This is attributed mostly to the fact that GOMSL does not have a way to describe parallel actions, something that was already known, but the difference between the experimental and the calculated result was larger than expected.

CoDeIn on the other hand, managed to estimate the completion time of the DM condition and the TUI condition with more accuracy than the GOMSL model. This is attributed to three differences between GOMS and CoDeIn. The first difference is CoDeIn's ability to model parallel actions, while GOMSL and CMN-GOMS (Card et al., 1983) (the original incarnation of GOMS) cannot. Even though CPM-GOMS (Gray et al., 1993; B. E. John, 1988; B. E. John, 2003) would have been able to correctly model the parallelism in the TUI task, its complexity does not lend it to the development of models for such simple tasks (B. E. John & Kieras, 1996a; , 1996b).

The second difference is that CoDeIn is designed specifically to handle RBI tasks. Therefore it includes operators that can handle real-world actions, such as prehension.

The third difference between GOMS and CoDeIn is a difference in the theoretical view of task performance. CoDeIn regards each action as an implicit part of the knowledge application process. It does not view each action as a separate step in the performance of a task. This assumption allows CoDeIn to remove step execution times as they are found in GOMSL models.

Finally, through the procedure of experiment 2, I proposed a new operator for the description of prehension in GOMS, using a linear model with the completion time being directly proportional to the distance of the actor to the target. The implications of this finding may be more significant for TUIs in general, because if prehension is slower than pointing, and TUIs base interaction on prehension, then they are inherently less efficient than DM. This question is left open for future work, as a survey is needed to see in which situations TUIs are used, vs. the situations that DM is used. Then, a comparison should be made to see whether these situations require such a fast performance as the pointing action would suggest, or if the prehension action would offer adequate performance as well.

Also, as shown in Figure 7.12, prehension actions are inherently slower than pointing actions after some threshold distance is exceeded. This would not be an issue if, in the real world, actor-to-target distances were limited, like they are on a computer screen. On a screen the maximum distance of an artifact from the pointer is limited by the resolution of the screen and the translation of mouse movement to pointer movement. In the real world however, distances between the actor and an artifact may vary infinitely, and while users would probably not try to grab something that is infinitely far from them, there are no guarantees about the distance that an artifact may have from the user, and whether the user will decide to try to grab the artifact or to move towards the artifact (thereby decreasing the distance) and then grab it.

Chapter 8 Non-Expert User Modeling

In previous chapters of this dissertation, I have discussed the basis of the CoDeIn framework and how it can be used to estimate the completion time of a task. In this chapter I attempt to show how CoDeIn may be used to predict the performance of non-expert users. The task presented in this chapter is a simple pointing task, but what is of interest is not the task itself; the task is only used to examine CoDeIn's performance in modeling non-expert users.

The pointing task is performed with two different interaction devices, the mouse and the virtual glove (Virtual Realities, 2006). The primary question of the experiment is whether CoDeIn can use the performance of one pointing device to predict the performance of another, using techniques that describe the users' knowledge, as discussed in Chapter 5.

A working hypothesis of the framework, as proposed in Chapter 5, is whether a user's Efficiency Rating (ER) can be used to penalize the user's estimated completion time. Because most contemporary model-based evaluation methods pre-suppose expert performance, it is very difficult to estimate the completion time of a task by a novice or other non-expert user. In the case of the pointing task, normally Fitts' Law (Fitts, 1954; Fitts & Peterson, 1964) would have been used to predict the completion time of the movement. However, Fitts' Law assumes expert performance of a fast, targeted movement towards a target, an assumption that is not true for one of the two conditions of the experiment. GOMS cannot be used here either, because it supposes expert, error-free performance. Therefore, while GOMS and Fitts' Law can be used to model the condition in which users are experts, they cannot be used to model the condition in which users do not have experience (Fitts & Peterson, 1964; B. E. John, 2003).

This is a general problem with contemporary model-based evaluation methods, because they base their evaluation on assumptions that may not be true anymore. RBIs use interaction devices that may not be familiar to users, such as walk-up-and-use interfaces that cannot expect expert performance. They also allow the use of computers to a large number of users who may never become experts, because they do not need to become experts. Thus, the assumption of model-based evaluation methods that require expert, error-free performance cannot be upheld most of the time.

CoDeIn, however, tries to overcome this problem by modifying expert performance to predict non-expert performance. The way that modification takes place is by increasing the performance time of the non-expert users according to the non-expert users' ER.

However, this ability to do this stems from the assumption that users gain experience and skill from performing a task repeatedly, and their ER increases as they gain this experience. Thus, the second part of this chapter presents a continuation of the non-expert condition of the experiment to examine whether CoDeIn can actually model the results of such a gain in experience.

8.1 Background and Motivation

Considerable research work has been devoted to the study of input devices (Buxton, 1983; Card, Mackinlay, & Robertson, 1991), especially on the differences between 3-D and 2-D interaction devices. Early in the investigation of these devices, Foley and Wallace (1974) proposed the distinction between the input task and the input device, and considered that the input task was independent of the input device. This provided a first step towards the study of interaction tasks and devices, but formal experimentation (Jacob et al., 1993; Jacob & Sibert, 1992; Jacob, Sibert, McFarlane, & Mullen Jr., 1994) and experience showed that the task was indeed connected with the device used to perform it.

Jacob and Sibert (1992) examined how task performance varies when the task is matched with the appropriate interaction device. They concluded that for best performance of a task, the task's perceptual structure should be matched with the perceptual structure of the device.

Hinckley, Tullio, Pausch, Proffitt and Kassell (1997) compared two 2-D interaction devices with two 3-D interaction devices on a 3-D rotation task. The object of the task was to rotate an object shown in 3-D on the computer's display, so that it would match the desired orientation. They found that even though participants did not have any previous experience with the 3-D interaction devices, they finished the task faster than using the 2-D interaction devices, and with no statistically significant difference in accuracy between the two conditions.

Ware and Jessome (1988) compared the performance of a 6-D interaction device that they named the bat, with a conventional mouse in a 3-D object placement task. The object placement task consisted of using the interaction device to move an object shown in 3-D on a computer display from one position to another. They conclude that the 6-D interaction device allowed better performance than the conventional mouse for this task.

From the aforementioned studies, it can be concluded that some devices are better suited to a certain task than others. However, these studies were performed to research how the device matches tasks, and not how users performed the tasks they were presented with the studied devices. The object of the study presented in this chapter is to examine whether CoDeIn can model the performance of users that perform a task using different interaction devices, and to study whether CoDeIn can capture how users learn to use an interaction device.

8.2 Experiment – Phase I

The purpose of this experiment is twofold. The first purpose is to test CoDeIn's ability to model non-expert user performance in a task with an interaction device that participants have never used before, and the second is to show that the predictions of CoDeIn follow an established prediction model of performance according to gained experience, such as the Power Law of Practice (Card et al., 1983).

8.2.1 Participants

Ten participants took part in the first phase of the experiment. They were all students of the department of Computer Science of Cyprus College, of ages between 19 and 25 years old. All the participants were taking a comparative programming languages course and were given extra credit in the course for their participation in the experiment. All of the participants had had substantial previous experience with the DM interaction style, and Microsoft Windows in particular. None of the participants had any experience in using the Virtual Glove prior to the experiment.

8.2.2 Materials and Design

The task was presented on a PC with a Philips 107E60 monitor and a Pentium 4, 3.00GHz CPU. The task was programmed using Java. The program presented a shape from a pool of three different shapes (a triangle, a square and a circle) and allowed the participants to drag that shape to one of three bins shown in the program's window. The Java program window is shown in Figure 8.1. The display resolution was set at 1024x768 pixels and the Java program window resolution at 800x600 pixels. The mouse that was used for the mouse condition was a Microsoft Wheel Optical Mouse. The glove that was used for the glove condition was a P5 Virtual Glove (Virtual Realities, 2006).

The virtual glove is worn on the hand of the user like a normal glove, and through a sensor column allows the user to move the pointer on the screen. Users can left-click by bending and straightening the index finger of the hand in quick succession. Dragging an item is performed by placing the pointer on top of the required item, bending the index finger, moving the pointer to the required position to drop the item, and straightening the index finger at that point. When the ring and pinky fingers of the hand are bent, the pointer's position is locked on the screen, so that the user may reposition the hand to a more comfortable spatial location. This posed a problem to the participants, who would instinctively bend their two fingers, which would trigger this effect.

The Java program measured the time taken by each participant to drag the shape from its initial position to the correctly labeled bin, and logged the coordinates of the pixels over which the pointer moved during the drag-and-drop action. The program also measured two error conditions: the number of times the shape was released in a different bin from the correctly labeled one, and performance of a single or double click of the mouse instead of the required drag-and-drop action.

8.2.3 Procedure

During the first phase of the experiment the participants were asked to carry out a basic drag-and-drop task, under two conditions. The first condition required the participants to use a mouse to drag the shape from the home position to the corresponding bin. In the second condition, the participants used a Virtual Reality glove (Virtual Realities, 2006) to drag the shape. The first condition of the experiment will be referred to from here onwards as the 'mouse condition', and to the second condition as the 'glove condition'. All 10 participants performed the task in both experimental conditions. The order of presentation of the two conditions to each participant was randomized.

The Java program that implemented the task presented the participants with one of three shapes at random. The three shapes were a circle, a square or a triangle. Each participant performed twenty trials and each trial presented a new, random shape from the three available. The participants had to drag the presented shape to one of three bins, each labeled with the name of a shape. When the shape was dropped over any bin, the trial would end, and the result would be tabulated. The bin positions and the bin labels were held constant throughout the experiment. Figure 8.1 shows a screenshot of the Java program, where the shape presented is a circle.

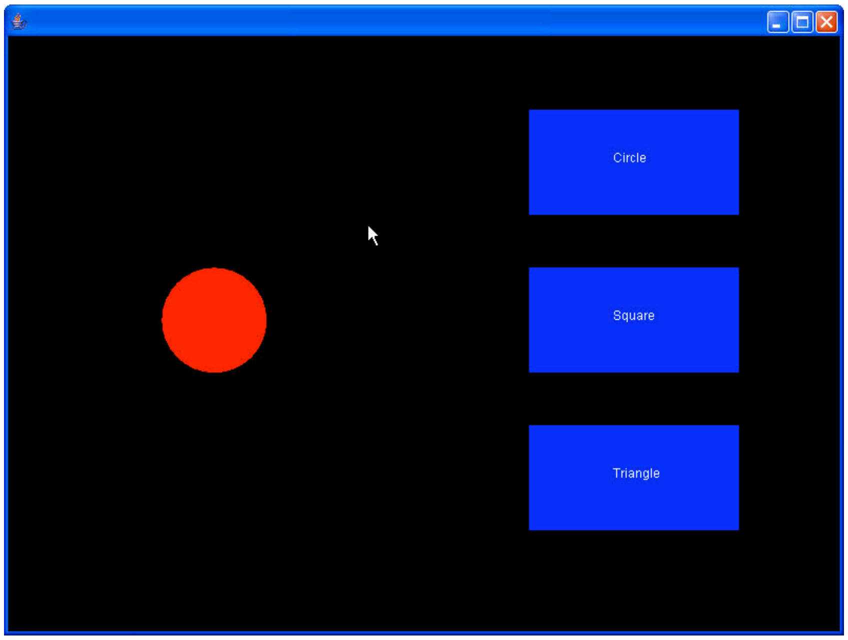


Figure 8.1 A screenshot of the Java program on which the task was executed.

Each trial began with a message box that let the participant know that when they pressed the OK button the trial would begin. The message box always appeared at the same position to allow the calculation of pointer movement. Up to this point, the shape was invisible, to disallow any preprocessing of the trajectory required for the completion of the trial by the participants. Once the participants pressed the OK button, the shape would appear. The participants would then have to navigate the pointer over the shape and drag-and-drop the shape over the bin of their choice. Timing of the trial would start when the participant would press the OK button, and would finish when the dragged shape was dropped over any bin, not just the correct bin.

If the shape was dropped over an incorrect bin, the result was noted, but was flagged as an error. Erroneous results were kept for later analysis, but were not used during the calculation of the average completion time of the task. The only other error that was logged was the participants' accidental single- or double-clicking instead of performing the drag-and-drop movement. This error never occurred in the mouse condition, but did occur during the glove condition. Any time measurements taken from a trial that resulted to one of the two error types was not used in the calculation of the average completion time of the task by the participant.

8.2.4 Results and Analysis

The time taken for each participant to move the shape from its initial location to the correct bin is shown in Table 8.1. There are two striking differences between the two conditions at first glance. The mouse condition is much faster than the glove condition, and there is much greater variation in completion times in the glove condition than in the mouse condition, as shown in Table 8.1. The first difference (i.e. the speed of execution) was expected, because the participants had never used the virtual glove before the experiment, while they all considered themselves experts in the use of the mouse.

The difference in variation of task completion using the glove may also be justified by the fact that the participants had for the first time used a device that allowed movement of the arm in 3-D space, and not in 2-D space. Therefore, the participants had to become accustomed to the fact that they should provide stabilization of the movements using their arm muscles, although in the second phase of the experiment, participants realized that they did not need to use their whole arm to move the glove, but just their forearm and wrist. However, this is something they could not have known nor been taught, but rather that they had to discover and learn through their experience with the glove.

Table 8.1 Average Completion Times of the two conditions, shown by shape. The standard deviation for each condition is also shown. Times are in milliseconds.

	Mouse	Std. Dev.	Glove	Std. Dev
Circle	1483	689	7457	5096
Square	1338	470	6041	2678
Triangle	1347	346	7044	2887
Average	1389	501	6847	3553

Given the experimental setup the circle shape had to be moved diagonally upwards on the screen; the square shape required a straight, left-to-right movement; the triangle required the participants to move the shape diagonally downwards. From the table, it can be noted that for the mouse condition, the straight line trajectory necessary for the placement of the square shape into the correct bin is the fastest motion on average, followed by the diagonally downward motion for the placement of the triangle shape. The slowest motion is the diagonal upward motion for the circle shape. What is interesting is that even though the glove requires different movements than the mouse, the same speed pattern emerges. The fastest motion is that of the straight line, followed by the diagonal downward motion, and then the diagonal upward motion.

One of the major differences in task performance of the two conditions is that the glove motions are performed while supporting the weight of the whole arm, whereas the mouse motions are performed on a flat surface, where the effects of gravity are less felt by the user. The motion to move the glove is done with the hand pointing towards the screen, with the arm slightly extended in front of the body. Hence, moving the arm requires the user to balance the weight of the whole arm, not just the hand. Also, the movements required to control the pointer through the glove are larger than those required when using the mouse. In fact, the arm movement is translated as one to one movement of the pointer. Therefore, to move the pointer from the leftmost part of the display to the rightmost part, users would need to move their arm in a line that covers the actual distance of their display.

This is untrue when using the mouse. Relatively smaller motions of the mouse are translated as larger motions of the pointer on the display. Also, the motions of the mouse are constrained by the surface on which the mouse is placed, so the user need only perform motions that use the arm below the elbow, without supporting the weight of the arm.

Table 8.2 shows the average completion time over all shapes for each participant in each of the two conditions. The performance of the mouse condition’s task is faster than that of the glove condition’s. The difference between the faster and slower motions may be attributed to the different types of motion required for each interaction device, and the distances that have to be traveled by each device to produce the same resultant movement of the pointer on the display. The mouse condition's standard deviation is also smaller than that of the glove condition's.

Table 8.2 The average completion time of the first phase's conditions. All times are in milliseconds.

Participant	Mouse	Glove
1	1665	7544
2	1254	11206
3	2308	6490
4	1121	3788
5	1097	10221
6	1341	5305
7	1221	8943
8	1219	6806
9	993	4612
10	1096	5384
Average	**1332**	**7030**
Std. Dev.	**389**	**1962**

8.2.5 Mouse Condition Models

The GOMSL (Kieras, 1999) model of Table 8.3 estimates the completion time of the mouse condition to be 5400 ms. Compared to the actual experimental average, as shown in Table 8.2, the GOMSL model predicts an inaccurate completion time. The reason for the discrepancy is that the model uses two default KLM operators, the pointing operator P and the mental time operator M (Kieras, 1999), that detract from the model's accuracy. The P operator is inaccurate because the pointing actions in the experiment are shorter than those that the P operator considers to produce the average for the completion time of a generic pointing action. The visual search times required to find the shape and the correct bin for the shape during each trial are, again, shorter than the average value proposed by the M operator. To correct the first problem, Fitts' Law is used to replace the P operator with a more accurate estimation of the pointing action's completion time.

Table 8.3 The GOMSL model for the first phase of the experiment.

```
Top Level Method
Method for Goal: Drag shape (Total time = 5.40 s)
Step 1: Find Shape (M = 1.2 s)
Step 2: Point to shape (P = 1.1 s)
Step 3: Press the mouse button (B = 0.2 s)
Step 4: Find correct bin (M = 1.2 s)
Step 5: Drag to correct bin (P = 1.1 s)
Step 6: Release mouse button (B = 0.2 s)
Step 7: Return with goal accomplished
```

The specific Fitts' Law model used to express the pointing action's movement completion time (MT) is taken from Mackenzie (1992a) and is calculated using two methods: the drag-select method, which states that MT = 135 + 249*ID, and the point-select method which states that MT = 230 + 166*ID, where ID is the Index of Difficulty as calculated via the Shannon formulation: ID = $\log_2(D/W + 1)$, where D is the distance of the manipulated object from the target, and W is the width of the target along the axis of motion. Therefore the MT of the pointing action consists of the two movements of the pointer: one movement from the OK button to the shape, and one movement from the home position of the shape to the correct bin. However, while both are pointing actions, they are different; one is a point-select action, and the other is a drag-select action, as defined by Mackenzie (1992a). For the first action, the point-select version of Fitts' Law proposed by Mackenzie (1992a) was used, and for the second, the drag-select version. Adding the two components of the motion yields the following equation:

$$MT = 135 + 249 * \log_2(400/200+1) + 230+166* \log_2(200/200+1) = 926 \text{ ms}$$

Fitts' Law predicts 926 milliseconds for the completion of both pointing actions. This prediction is very close to the real experimental average, but does not take into account the mental processing time of the participants when a shape is presented to them. The participants must first identify the shape, identify the correct bin, and then drag the shape to the correct bin. That is why there is approximately one half second of difference between the experimental average and Fitts' Law's prediction.

By substituting step 2's completion time in the GOMSL model of Table 8.2 to be 396 milliseconds as calculated by the point-select version of Fitts' Law (I. S. MacKenzie, 1992a), and step 4's completion time to be 530 milliseconds as calculated by the drag-select version of Fitts' Law (I. S. MacKenzie, 1992a), the resulting estimated time becomes 4126 milliseconds. This result is still far from the average completion time of the mouse condition of the experiment.

To correct the second problem in the analysis, I applied the Model Human Processor's (MHP) (Card et al., 1983) reaction time analysis. The participants perceived the shape, processed it in visual store, and then initiated motion to move the pointer to the shape (TT_{shape}). Then, they perceived the bins, processed them in visual store, decided which one is correct, and initiated motion towards the decided bin (TT_{bin}). This results in the following analysis:

$TT_{shape} = \tau_p + \tau_c + \tau_m = 100 + 70 + 70 = 240$ ms

$TT_{bin} = \tau_p + 2*\tau_c + \tau_m = 100 + 2*70 + 70 = 310$ ms

$TT = TT_{shape} + TT_{bin} = 240 + 310 = 550$ ms

where TT is the total mental processing time or thinking time required by the participants to perform the perception and recognition of the shape and the bin, and the decision of how the dragging motion will be performed. τ_p is one cycle of the perceptual processor of MHP, τ_c is one cycle of the cognitive processor of MHP, and τ_m is one cycle of the motor processor of MHP (Card et al., 1983).

By substituting these findings into steps 1 and 4 of the GOMSL model in Table 8.2 and including the Fitts' Law calculation, the resultant total time is 2276 milliseconds, which is even closer to the experimental average.

The same analysis was performed with CoDeIn, with the model shown in Figure 8.2. In this analysis, I only include the final results of Fitts' Law and the MHP calculation, and the CoDeIn analysis yields a completion time of 1876 milliseconds. The results of the two models for the mouse condition are shown in Table 8.4. It is evident from Table 8.4 that the closest prediction comes from CoDeIn.

Table 8.4 The experimental average of the mouse condition and the models' estimates.

	Completion Time	Deviation	Percentage Deviation
Experimental Result	1389	-	
GOMS	5400	4011	289%
GOMS and Fitts' Law	4126	2737	197%
GOMS, Fitts' and MHP	2276	887	64%
CoDeIn	1876	487	35%

However, none of the GOMS variations (Card et al., 1983; B. E. John & Kieras, 1996a; , 1996b; Kieras, 1999) nor Fitts' Law can be used to evaluate the glove condition of the experiment. GOMS can only model expert, error-free performance (Card et al., 1983; B. E. John, 2003), and Fitts' Law expects a fast, targeted motion towards the object of interest (I. S. MacKenzie, 1992a; , 1992b; I. S. MacKenzie & Buxton, 1994). The reason that GOMS is inappropriate for use is that the participants of the study are all novices in the use of the virtual glove, thus not fitting the profile of the user modeled by GOMS. As for Fitts' Law, while at first it may seem that the motion performed by the glove's users is a fast targeted motion, there are factors that may detract from the task's performance with the requirements of Fitts' Law. One reason that was observed by personal use and reported by participants in the experiment, is that the hand muscles tend to relax while performing the dragging motion, thereby bending the ring and pinky fingers of the hand that controls the glove. This however, leads to unbinding the glove from the pointer, to accommodate changing the position of the hand and arm while keeping the pointer at the same position on the screen.

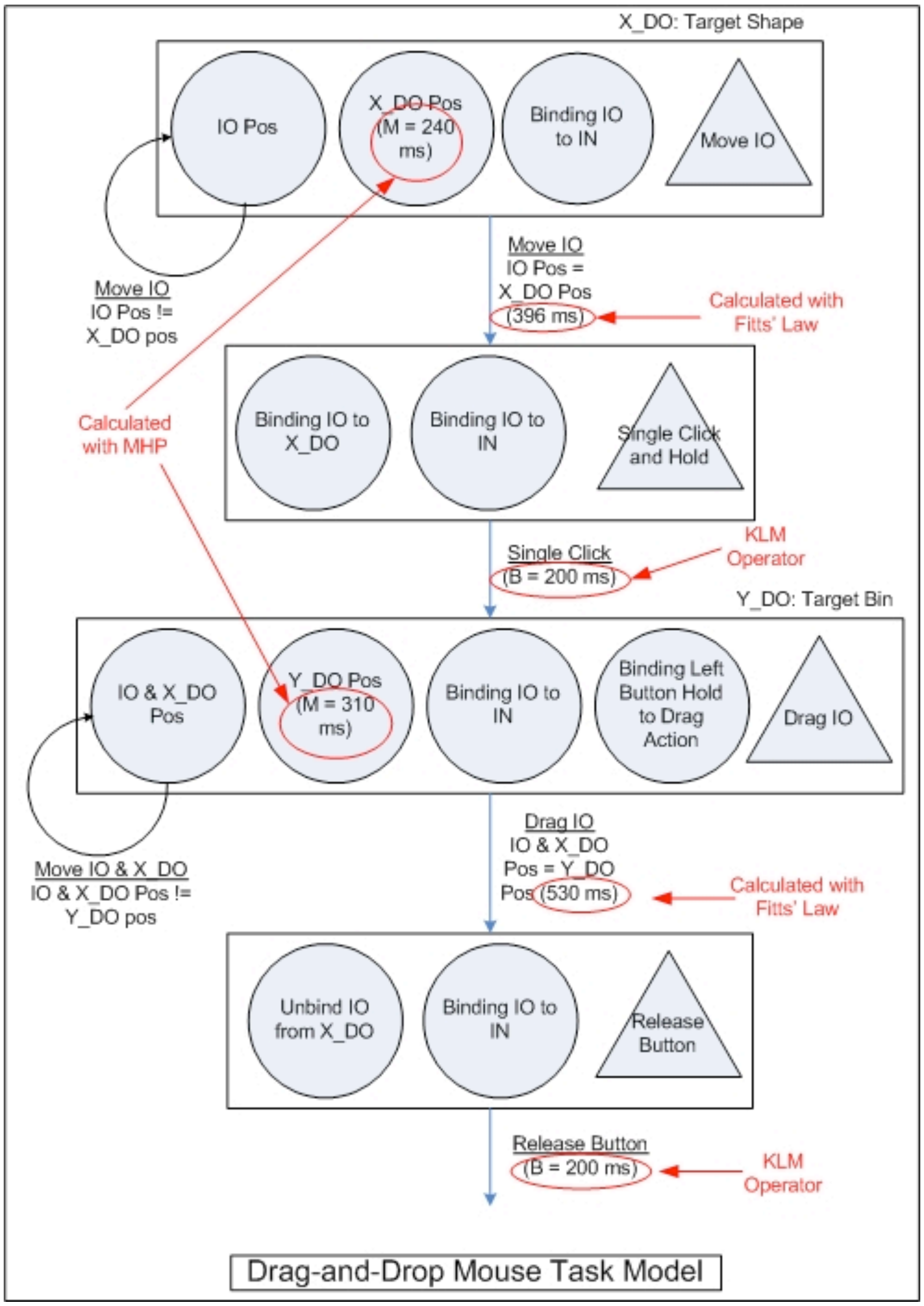


Figure 8.2 The CoDeIn analysis of the mouse condition of the experiment. The annotations in red are explanatory for the values shown in the model. They do not constitute part of the model.

As was later observed during the experimental trials, the novice users made this error frequently, leading to degraded performance. The degradation in performance is something that should be addressed by the novice user model, thus the following section describes how CoDeIn accommodates this performance degradation by using the user model method proposed in Chapter 5 of this thesis.

8.2.6 Glove Condition Model

The participants of the study were novices in the use of the virtual glove. However, they understood the concepts of drag-and-drop, selection, and glove to pointer binding, based on their knowledge of the DM interaction style, and the use of the mouse in particular. Thus, the participants had the declarative knowledge required for the completion of the task. However, their knowledge of the binding between the glove and the pointer was flawed, because, based on their model of the how the mouse works, this binding is supposed to be static. For this reason, and because people often prefer to use knowledge-in-the-head, even if that knowledge is incomplete (Gray & Fu, 2001; , 2004), their knowledge of how to use the mouse transferred negatively to the use of the glove (Singley & Anderson, 1989), because the glove's binding to the pointer is dynamic, requiring an action by the user to uphold the binding. Because the participants had the mouse knowledge, they needed to remind themselves that they were responsible for upholding the binding between the IN and IO in the case of the glove, a fact that led to degraded performance. Also, because the participants had never used the glove before, they did not know exactly how their arm movement would translate into pointer movement. Further, because the glove is a multi-axis device, it behaves differently if used on- or off-axis.

The two problems provide the basis for deciding what parts of the task should be penalized to predict the completion time of the glove movement. The procedural knowledge for the “Move” action and the procedural knowledge for the “Drag” action should be penalized in some way, as the movements’ performance was not at the level of an expert user. This is a basic difference between the CoDeIn framework and other model-based evaluation methods. Instead of assuming some kind of performance, like GOMS assumes expert, error-free performance, CoDeIn allows the creation of a model that may be used for expert, error-free performance, and then provides mechanisms to modify that performance to model non-expert users.

There are three mechanisms that CoDeIn allows for the modification of performance of expert users to estimate the performance of non-expert users. The first mechanism is based on the assumption that a test may be used to gauge how well the user knows something. This assumption is based on the fact that all formal knowledge-imparting processes require some kind of testing to gauge the learner’s understanding of the imparted knowledge. Thus, one tests a sample of users on a particular piece of knowledge, and then uses that performance as the average performance for the whole population of users that belong in the same class as that of the tested users.

The second way is to arbitrarily decide that the users who will perform the evaluated task have an ER with value X. Obviously, this arbitrary decision will be based on analysis that leads the modeler to believe that the users do indeed have that X ER. This type of evaluation is more suited to tasks that do not require great precision.

The third way, which is used in this analysis, is to perform a test, and then use the results of the test to calculate the ER of the users. In this instance, the completion times (CT) of the two conditions will be used for the penalization of the glove condition's procedural knowledge. The resultant CT of the mouse condition will be considered expert performance, and the resultant CT of the glove condition is the deviation from expert performance. The result of their division is the ER of the tested users. While the two conditions may differ in several areas, such as the distance traveled by each IN, or the muscles used in each condition, the end result is that both are pointing tasks that are governed by Fitts' Law. The distance traveled when applying Fitts' Law is the distance between the pointer and the target bin on the display and not the distance traveled by the INs. That is the reason why the mouse condition's result was chosen as an estimate of the virtual glove's expert performance.

In this experiment, the participants' performance with the mouse was used as the lower bound of the glove performance (i.e. glove performance <= mouse performance). The participants' mouse condition average completion time was used as the benchmark for their expert-performance average glove condition completion time (CT). Supposing that the glove condition's CT could be the same as the mouse condition's CT, if the participants were expert users of the virtual glove, then the ratio of the mouse condition's CT over that of the glove condition's CT gives the ER of the participants. The average completion time of both conditions is shown in Table 8.1, yielding a ratio of 0.2. Thus the CT of the mouse condition is 0.2 times that of the glove condition. Thus, the ER can be used to modify the mouse CT (T_{exp}) to calculate the glove condition's CT (T_{glove}). The modification process can be performed by the formula: $T_{glove} = T_{exp} / ER_{glove}$. In this case there is only one chunk of knowledge affected by the modification, that of the first knowledge state's 'Binding IO to IN' declarative chunk, of the model shown in Figure 8.3.

Figure 8.3 shows the glove condition's CoDeIn diagram, with the penalization percentage of the two actions, because of the penalization of their procedural knowledge. The M operator used in the first knowledge state's 'Binding IO to IN' knowledge chunk is relevant because the participants needed to remind themselves how to keep the binding, that, unlike when using the mouse, was dynamic and could have been broken by failing to keep their two last fingers straight.

Because the participants' ER is 20%, the framework's method for calculating non-expert performance states that there should be a T_{exp} / ER_{glove} slowdown in the performance of the actions of the users. Earlier it was explained that while the users have all the declarative knowledge for the performance of the task, they do not have enough experience using the glove. Thus, they are not experts of the procedural knowledge required for the performance of the task. The slowdown is therefore aimed at the procedural aspect of the task, rather than the declarative aspect. Therefore, the required completion time is multiplied by ER in the two pointer moving actions, to reflect this slowdown.

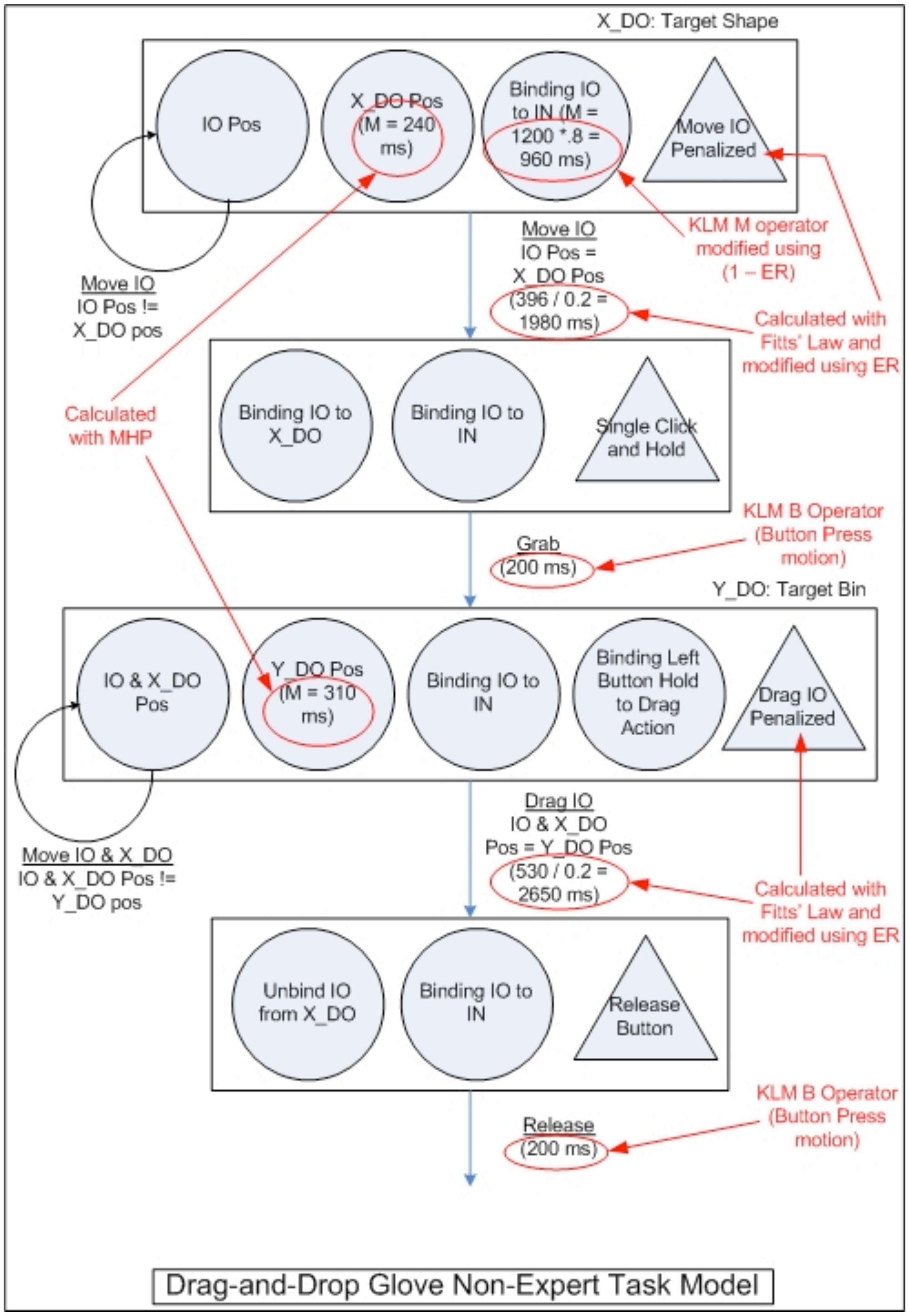

X_DO: Target Shape
IO Pos
X_DO Pos (M = 240 ms)
Binding IO to IN (M = 1200 *.8 = 960 ms)
Move IO Penalized
Move IO IO Pos != X_DO pos
Move IO IO Pos = X_DO Pos (396 / 0.2 = 1980 ms)
KLM M operator modified using (1 – ER)
Calculated with Fitts' Law and modified using ER
Calculated with MHP
Binding IO to X_DO
Binding IO to IN
Single Click and Hold
KLM B Operator (Button Press motion)
Grab (200 ms)
Y_DO: Target Bin
IO & X_DO Pos
Y_DO Pos (M = 310 ms)
Binding IO to IN
Binding Left Button Hold to Drag Action
Drag IO Penalized
Move IO & X_DO IO & X_DO Pos != Y_DO pos
Drag IO IO & X_DO Pos = Y_DO Pos (530 / 0.2 = 2650 ms)
Calculated with Fitts' Law and modified using ER
Unbind IO from X_DO
Binding IO to IN
Release Button
KLM B Operator (Button Press motion)
Release (200 ms)
Drag-and-Drop Glove Non-Expert Task Model

Figure 8.3 The CoDeIn diagram for the glove condition of the experiment. The red annotations are not part of the model; they are explanative of the values found in the model.

Also, the M operator is multiplied by 0.8, because the participants are 20% experts (= ER) and thus, 1-ER of the time the participants will *not* behave like experts, and therefore will need to remind themselves how to uphold the binding between the glove and the pointer.

Table 8.4 shows the experimental completion time for both conditions of the experiment and the predictions of the CoDeIn models for each condition, as well as the deviation of the prediction from the actual result.

Table 8.5 The two conditions' average completion time and the CoDeIn predictions. Times shown are in milliseconds.

	Experiment Completion Time	CoDeIn Prediction	Deviation	GOMSL Prediction	Deviation
Mouse Condition	1389	1876	487 (35%)	2276	887 (64%)
Glove Condition	6847	6230	-517 (7.5%)	--	--

Taking the deviation as a percentage of the experimental completion time, CoDeIn erred 35% on the mouse condition, but only 7.5% on the glove condition. One possible explanation is that CoDeIn's prediction of the mouse condition completion time is based on the theoretical structures of MHP (Card et al., 1983). On the contrary, the glove condition model is based on a test that provided the expert user completion time, and from there the model modifies the expert performance appropriately, coming up with a more accurate prediction.

8.3 Experimental Phase 2

The second phase was performed to provide evidence towards the validity of two basic assumptions of the framework. The first assumption is that CoDeIn can use ER to model the performance of any type of user, as long as there exists some completion time for the task that can be used as an expert performance benchmark. This assumption was used in the first phase of the experiment to predict the completion time of the glove condition. The benchmark used in the first phase of the experiment was to calculate the expert performance time of the movements with Fitts' Law, and then penalize them according to the ER of the participants relative to the expert user set.

8.3.1 Using Efficiency Ratings in the Power Law of Practice

In Chapter 3, the Power Law of Practice (Card et al., 1983) was introduced as one of the principles of operation of MHP (Card et al., 1983). This law states that the logarithm of the completion time of a task decreases linearly with the logarithm of the number of times the task is performed. In formula form: $T_n = T_s * (\text{number of trials})^{-\alpha}$, where T_n is the completion time of the task after (trial number) trials, T_s is the completion time of the task before any training was given, and α is the rate of learning. For MHP the rate of learning α is set in the range: $\alpha \in [.2, .6]$, with .4 taken as an average value.

It was also mentioned that the law is also one of CoDeIn's principles of skilled performance. Hence, this principle should be supported by showing that

CoDeIn's predictions (as users gain more experience by performing the task more times) follow a power law as well.

This assumption is intimately tied to the theory that knowledge or skill can be transferred from one context to another, either negatively or positively. Because there are arguments both for (Anderson et al., 1996; Bransford et al., 2000; Palumbo, 1990; Singley & Anderson, 1989), and against (Greeno, Smith, & Moore, 1992; Lave & Wenger, 1991) this assumption, there was some need to provide some evidence towards the validity of the assumption, at least in the context of this framework.

From the power law, a form can be derived that uses the ER instead of the number of trials to compute the completion time of a task. The proof follows:

1. Let $ER_a = T_{exp} / T_a$, where ER_a is the ER of a user relative to the expert user set that completes a task in time T_a, and T_{exp} is the expert, error-free performance completion time. Thus:
2. $T_a = T_{exp} / ER_a$, by exchanging the terms ER_a and T_a in step 1.
3. The Power Law can be stated as: $T_b = T_a * (n_b – n_a)^{-.\alpha}$, where n_b is the number of trials required for completion time T_b, when it is known that after n_a trials the completion time of the task is T_a.
4. Therefore, by dividing both parts with T_a : $T_b / T_a = (n_b – n_a)^{-.\alpha}$
5. Substituting step 2 for T_a and T_b into step 4 yields:

   $(n_b – n_a)^{-.\alpha} = (T_{exp} / ER_b) / (T_{exp} / ER_a)$

   $= ER_a / ER_b$
6. Substituting the result of step 5 into the Power law yields:

$$T_b = T_a * (ER_a / ER_b)$$

Step (6) shows the final form of the derived power Law with ERs instead of number of trials for the calculation of new performance based on previous performance.

8.3.2 Participants

The second phase of the experiment only included 6 (2 female and 4 male) out of the original 10 participants for phase 1; 4 of the participants could not commit to 3 weeks of the everyday training required. The participants volunteered to participate in the experiment without any reward to compensate for their time.

8.3.3 Materials and Design

The second phase of the experiment was performed on a computer with the same specifications as that of the first phase's. A free version of a game "Beach Head 2002" (Atari Co.) was used to train the participants with the virtual glove (Virtual Realities, 2006). The object of the game is to destroy as many hostiles as possible, by controlling a machine gun, as shown in Figure 8.4.

The machine gun is controlled by the motion of the glove, and the user shoots the machine gun by performing a single-click. The hostiles may come from any direction. The participants were tested at the beginning of every week, for three weeks, using the Java program used in the first phase of the experiment, and once before training started, as part of the first phase of the experiment.

Figure 8.4 A screenshot from "Beach Head 2002",
the game used to train participants in the use of the virtual glove

8.3.4 Procedure

The participants played the game for 20 minutes every day, 5 days per week. On each Monday of every week that the experiment ran, the participants were tested on the drag-and-drop task, and then played the game for the required 20 minutes. This testing was performed to gauge whether there was any significant skill transfer from the game to the point-and-click task, and to examine whether there was any improvement in the completion time of the experimental task.

The game does not train the user on the drag-and-drop action, but rather on the point-and-click action. The null hypothesis for this experiment was that there would not be any significant transfer in the skill required for the point-and-click task of the game to the drag-and-drop task of the experiment.

8.3.5 Analysis

Table 8.6 shows the results of the experimental testing that took part every week. Table 8.6 shows the average results of each participant's completion time of the task. The mouse completion time is also shown for comparison. The average completion time of the participants is decreased every week, along with the standard deviation of their performance (except for an irregularity after 1 week of training). However, the standard deviation after 3 weeks of training becomes similar to that of the mouse. Figure 8.5 shows this graphically.

Table 8.6 The results of the second phase of the experiment by week. The results of the mouse condition are shown for reference. All times shown are in milliseconds

Participant	Mouse	Glove Week 0	Glove Week 1	Glove Week 2	Glove Week 3
1	1665	7544	5834	3934	3642
2	1254	11206	6773	3148	3958
3	2308	6490	10041	4709	4527
4	1097	10221	4132	5212	4131
5	1221	8943	7443	6895	3498
6	1219	6806	4126	4126	3474
Average	1461	8535	6392	4671	3872
Std. Dev.	458	1913	2241	1296	413

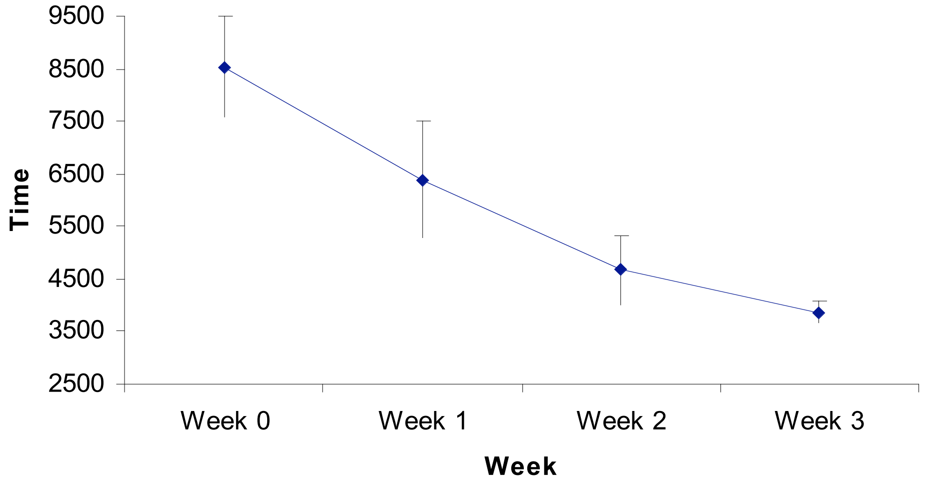


Figure 8.5 Average Completion Time of the Glove Condition by week. Week 0 represents the results of testing at the beginning of the first week, before any training had taken place, week 1 shows the results of testing after one week of training, etc. All times shown are in milliseconds.

Figure 8.6 shows the amount of variation in the completion times of the participants, again by week. As the weeks go by and training progresses, it is evident that not only the participants become faster, but they also converge to very similar performance, as the variation becomes smaller every week. In fact, after three weeks of training, the variation in the completion time of the drag-and-drop task becomes similar to that of the mouse condition.

Figure 8.6 reveals another important characteristic of the performance of the participants, when combined with the trend shown in Figure 8.5. While the participants' glove completion time variation becomes almost as low as that of their performance with the mouse, the same does not happen with the glove performance completion time. Rather, in Figure 8.5, the completion time starts plateauing after three weeks of training, possibly hinting to the fact that the virtual glove is inherently harder to control than the mouse.

Figure 8.6 Boxplots of the completion time for the glove condition of the experiment. The Mouse condition boxplot is included for reference as well.

As was mentioned earlier, there is the possibility that because the mouse is a more constrained device than the glove, the task is easier to perform, because fewer variables need to be controlled by the user, compared to when performing the same task with the glove. Again, this is hinted at by the theoretical substratum of the framework that more affordances and constraints make a task easier to perform or a device easier to control.

What is immediately evident from the graph is that there is a decrease in the completion time of the drag-and-drop task, consistent with the Power Law of Practice (Card et al., 1983). This fact suggests that skill transfer has taken place. However, because there was no control group that did not practice on the game, it is hard to say how much of the decrease in the completion time of the participants is a result of the training on the game, and how much is a result of performing the experimental task multiple times. The same is true for the decrease in standard deviation.

However, I believe that there was skill transfer from the training task to the experimental task, in that the participants learned to better understand the binding between the movements of the pointer according to the movements of the limb, and to keep the binding between the glove and the IO by keeping the last two fingers of the hand straight. These was reinforced by the requirement of the game to keep the machine gun in constant motion to dispose as many hostiles as possible, as fast as possible. Therefore, the better the participants became at using the glove, the better their performance on the game, and as they anecdotally suggested: “the better you get, the more fun [the game] is”.

The movement of the limb required by the game is not in a straight line, from left to right. Rather, players must move the pointer all around the screen, in seemingly random ways. This fact seemed to inhibit the participants' performance after one week of training, creating more variation in the completion time of the task as shown in Figure 8.6. But at the beginning of the second week, the effect disappeared. The way that the effect was gauged was by plotting the participants' trajectories using pixel points gathered by the experiment program. Figure 8.7 shows the trajectories that one participant had performed during the completion of the mouse condition of the experiment. Figures 8.8 to 8.11 show the trajectories of the glove condition of the experiment of each trial by the same participant for each week the participant was tested. All the graphs use the same scales for the x and y axes, to convey equivalent information.

Figures 8.8 to 8.11 show how the trajectories become straighter and more concentrated together from the home position to the targets, as the participant gains more experience in the use of the glove. At the end of the third week of practice, the trajectories become very similar to those that the participant performed using the mouse. The trajectories towards the top of the graph were created from the presentation of the circle shape, the middle ones from the presentation of the square shape, and the bottom ones by the presentation of the triangle.

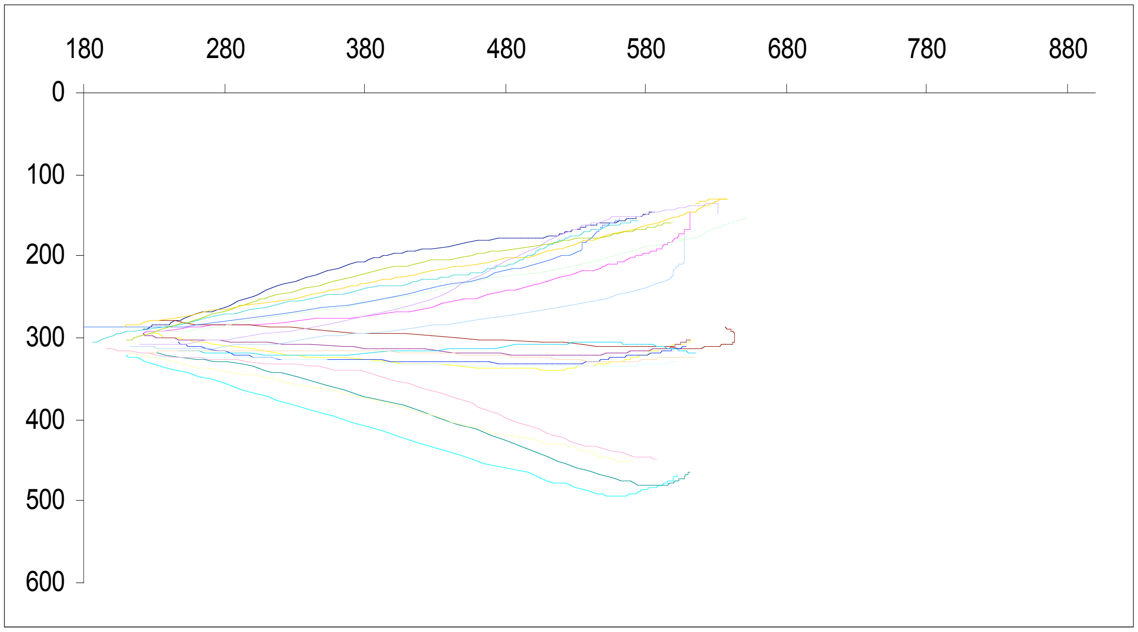


Figure 8.7 The mouse condition trajectories of the trials of the experiment.

Each line represents a different trial.

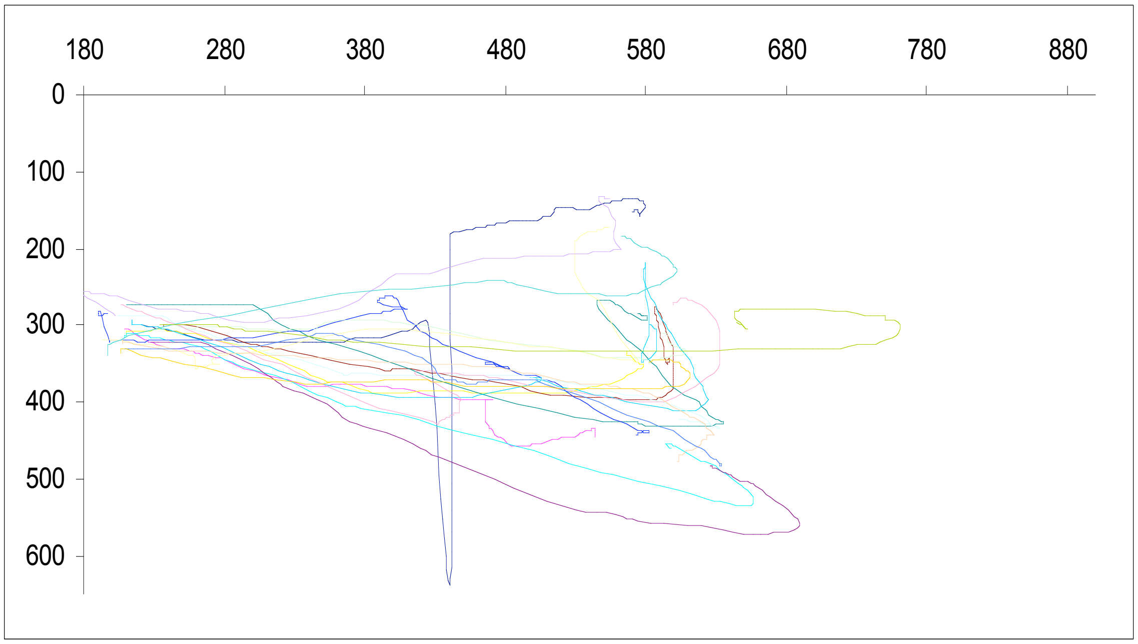


Figure 8.8 The initial test before any practice, of the Glove condition of the experiment. Each line represents a different trial.

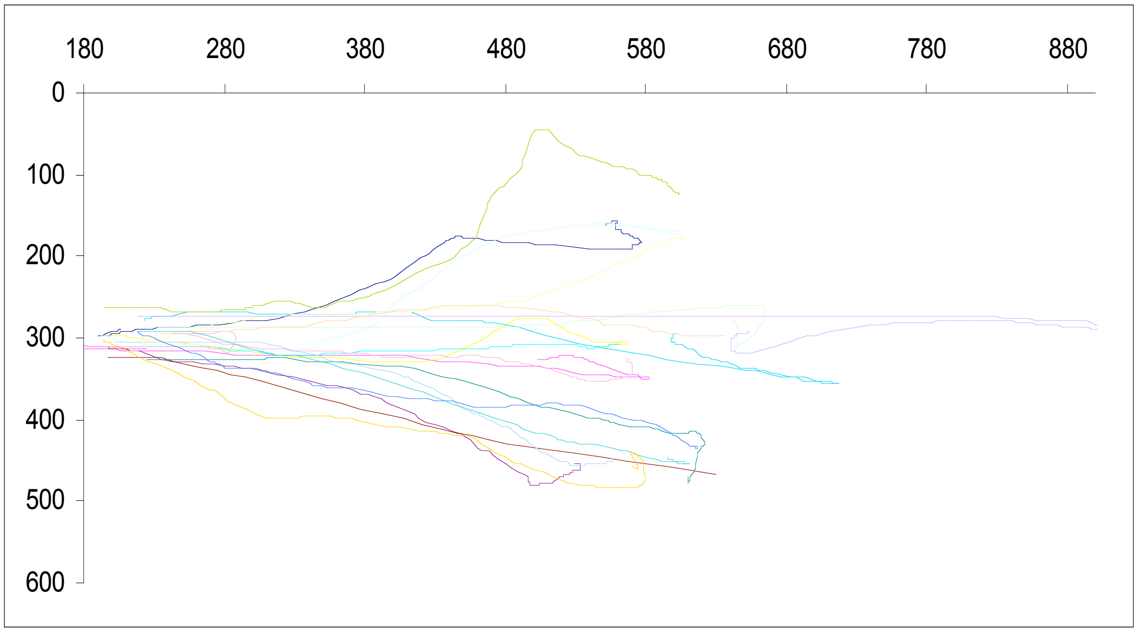


Figure 8.9 The test after one week of practice, of the Glove condition of the experiment. Each line represents a different trial.

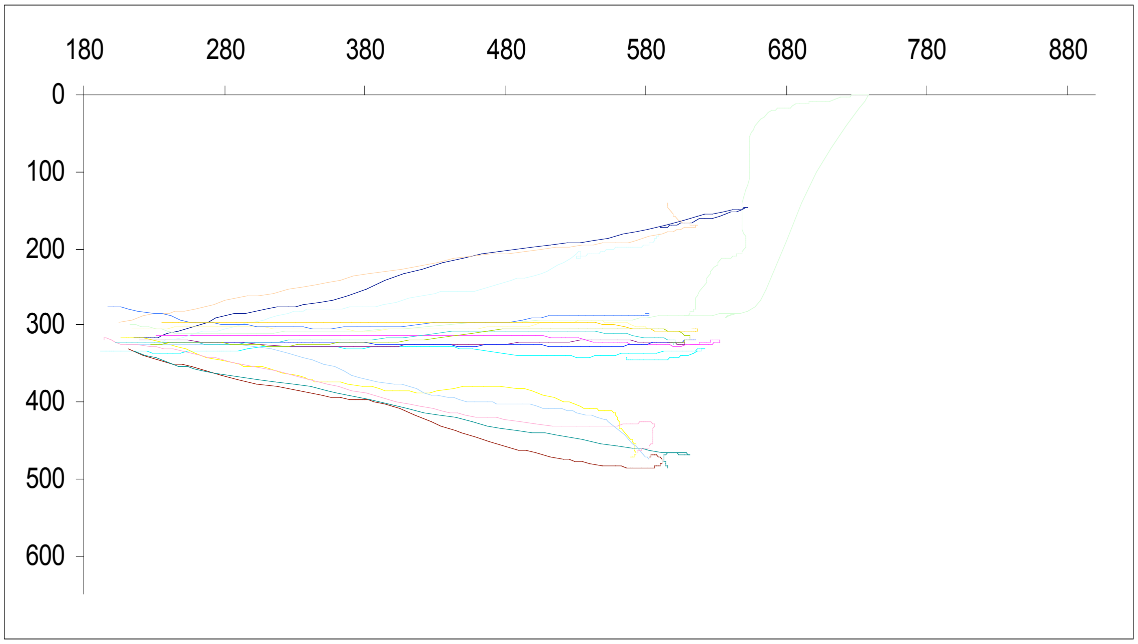


Figure 8.10 The test after two weeks of practice, of the Glove condition of the experiment. Each line represents a different trial.

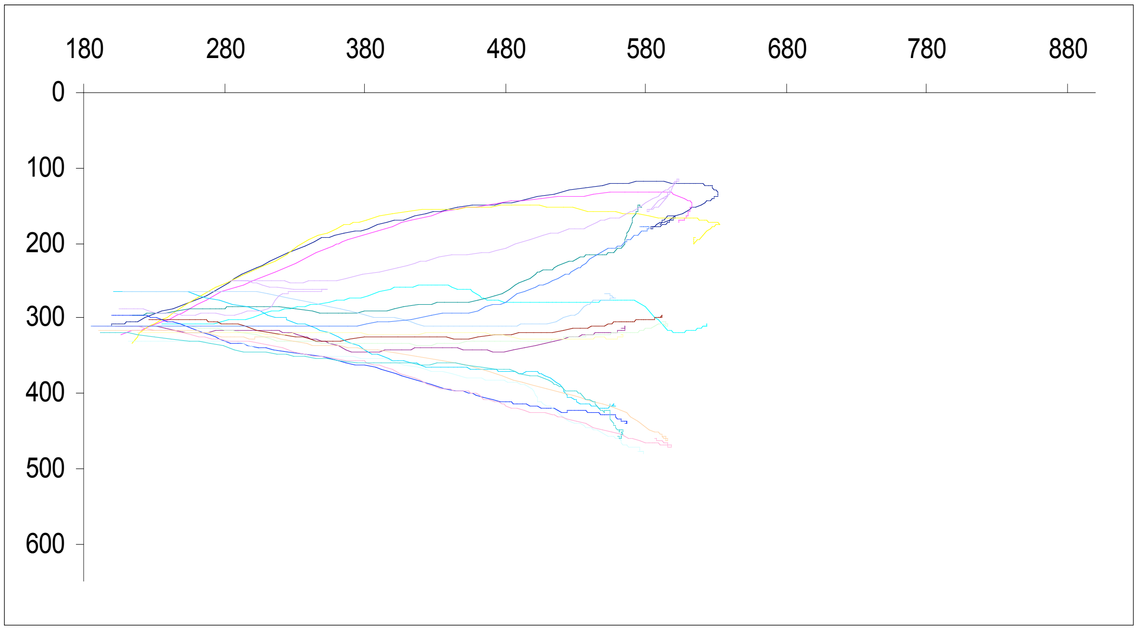


Figure 8.11 The test after three weeks of practicing the Glove condition of the experiment. Each line represents a different trial.

Summing up the evidence that is presented in Table 8.6 and Figures 8.5 to 8.11, I conclude that while the participants improved their performance in the procedural part of the task, and the accuracy of their movement approached the movement accuracy when using the mouse, the velocity of the movement still remained slower with the virtual glove than with the mouse. Because the experiment could not have been continued for longer than it did, the question that remains to be answered is whether the accuracy is what is acquired first in the evolution of such a skill, and then the speed of the movement is developed, or if indeed the virtual glove is inherently a slower pointing device than the mouse. It is not evident, though, that there is significant transfer from the performance of one task (that of playing the game) to the performance of the other (the drag-and-drop task), because there was no control group to test for this.

8.3.6 Models

There are two ways that the completion time of the experiment can be predicted. The first way is by using the first performance of the task as the benchmark, and using CoDeIn's version of the Power Law of Practice, as derived in section 8.3.1. This version of the Power Law of Practice uses the ER of the class of users whose performance needs to be predicted. The Power Law has one drawback, though, because it needs the ER for the benchmark completion time to predict any other completion times.

To predict the performance of each week using the Power Law, I used as benchmark the first test completion time (Week 0 completion time), and then used two ways to validate the performance of the model. The first was to consider that the ER of the participants was 40% as calculated in the first phase of the experiment, and assumed that each week the participants' ER had a relative increase of 40%, calculated as $ER_{weekB} = 1.4 * ER_{weekA}$, where week A and week B are two consecutive weeks.

The second was to use the Power Law directly into the glove condition model, shown in Figure 8.3, and modify the three values using different ERs. This was done again supposing a relative 40% increase in the ER per week. Table 8.7 shows the predicted values using the first way of calculation, and table 8.8 shows the predicted values using the second way. In both Tables 8.7 and 8.8 T_a and ER_a are the initial completion time and ER of the participants. T_b is the predicted completion time.

Table 8.7 Predictions of the derived Power Law of Practice

	Experimental Result	**Power Law Prediction**	**Deviation**	**Deviation Percentage**
Week 0	8535	T_a = 8535 ER_a = .2		
Week 1	6392	T_b = 8535 * (.2 / .28) = 6096	296	5%
Week 2	4671	T_b = 8535 * (.2 / .39) = 4377	294	6%
Week 3	3872	T_b = 8535 * (.2 / .55) = 3104	768	20%

Table 8.7 shows the power law predictions that are on average 10% smaller than the experimental results. However, the predictions are based on assumptions about the nature of the users, and not on actual data gathered from the users. Therefore the results are expected to have some deviation, but even

with 10% deviation, they still show the learning trend, and thus capture the essence of the increase in experience as decrease in performance time well. This can be seen in Figure 8.12, where the predictions of the derived power law are the series labeled as 'Power Law'.

Table 8.8 shows the predicted results that are generated by penalizing the procedural knowledge areas of the CoDeIn model in Figure 8.3. The same results are shown graphically in Figure 8.12 as "CoDeIn" for comparison with the experimental results and the predicted results of the derived power law.

Table 8.8 Predicted results by the application of the Derived Power Law in the model of Figure 8.4

	Glove Week 0	**Glove Week 1**	**Glove Week 2**	**Glove Week 3**
	8535	6392	4671	3872
Efficiency Rating	20%	28%	39%	55%
CoDeIn Model Changes and Predictions				
'Move IO' action	396 / .2 = 1980	396 / .28 = 1414	396 / .39 = 1015	396 / .55 = 720
'Drag IO' action	530 / .2 = 2650	530 / .28 = 1893	530 / .39 = 1359	530 / .55 = 963
'Binding IO to IN' chunk	1200 *.8 = 960	1200 *. 72 = 864	1200 *. 61 = 732	1200 * .45 = 540
Unchanged Parts	950	950	950	950
CoDeIn Prediction	6540	5121	4056	3173
Deviation %	23%	20%	13%	18%

Figure 8.12 shows the results of the two predictions and the experimental results. Power Law's prediction of week 0 testing is the same as that of the

experimental result, because that is the benchmark for the application of the derived power law. CoDeIn's week 0 result seems far from the average of the second phase of the experiment, but a fact that has to be considered is that the experiment was only performed with 6 participants. Had all 10 participants been used in the second phase, the initial experimental result would have been 6847 milliseconds, as shown in Table 8.5; this changes the accuracy of the model from 23% deviation to 7.5%. I believe that the same would have happened with the rest of the predicted results. An extrapolation such as this cannot be made for the derived power law, because the derived power law would have to use a different benchmark completion time for its calculations. However, I believe that the derived power law would hold in that case as well.

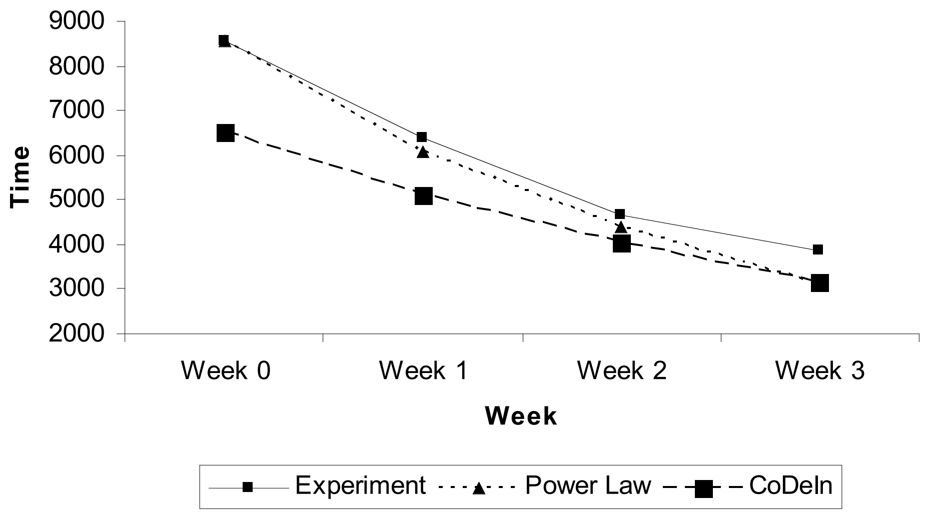


Figure 8.12 The actual and predicted results from the derived power law. All times shown are in milliseconds.

The general form of the results though, corroborates the supposition that CoDeIn's predictions follow an existing model of performance calculation, namely the Power Law of Practice (Card et al., 1983). Therefore, while the predictions may seem inaccurate, because of a large difference between the predicted and the actual experimental values, the effect of decreasing completion time in a decreasing exponential fashion is evident. Also, as mentioned above, the deviation may be misleading due to the small sample size of participants in the second phase of the experiment.

8.4 Conclusions

This chapter presented the second experiment that was performed for the validation of the CoDeIn framework. The aims of the experiment were first to show that CoDeIn can handle the evaluation of different categories of users other than experts with error-free performance. The second aim was to show that CoDeIn's predictions follow a decreasing exponential trend as suggested by existing models of performance prediction that take into account the effects of increased experience in a task, such as the Power Law of Practice (Card et al., 1983).

The first aim was accomplished by using one of the basic suppositions of the framework that tests may be performed on a sample of the population of interest, and the results used to model the performance of the larger population. The model utilized the ER of the participants to modify the procedural knowledge of the task. The results provided a close estimate to the actual performance of the participants. This finding is crucial to the CoDeIn framework, as one of the major differences of CoDeIn and other model-based evaluation methods is that while CoDeIn can model expert performance, its main use is to model non-expert user performance.

The second aim was accomplished by following up the glove condition of the experiment with supplementary training for some of the original participants of the experiment. The participants of the second phase of the experiment practiced the use of the virtual glove, and after each week of practice they were tested to examine whether that practice had affected their performance in the experimental task. The result was that there was indeed a decrease in the completion time of the task, hence an increase in the participants' efficiency of performing the task, in a way that conforms to the Power Law of Practice (Card et al., 1983). However, whether this is owed to knowledge transfer from the game or it is owed to the performance of the experimental task many times cannot be determined, because there was no control group that would not perform the training, but only the tests.

The results were then compared with the CoDeIn model's predictions and assumption of rise in performance, occurring as a decrease in task completion time. The result of the comparison was that CoDeIn's predictions did follow a decreasing exponential trend, as suggested by the Power Law of Practice (Card et al., 1983). However, the difference between the experimental results and the predicted results was larger than expected. This is attributed primarily to the small sample size for the second phase of the experiment. The initial average completion time of the sample used in the second phase of the experiment was significantly larger than that of the sample used in the first phase. However, the results of the model converge to the actual performance of the sample as the effects of training become more evident towards the end of the training period.

Obviously, a large scale study that applies CoDeIn to a real task with many participants is imperative, to validate the framework in real-world conditions. Such a study was outside the scope of this thesis, but it will be the subject of future research.

Chapter 9 Conclusions and Future Work

This chapter presents a summary of the findings, results and conclusions of this work. This thesis deals with the problem of model-based evaluation of Reality-Based Interaction. RBIs present different challenges in predictive task evaluation than the previous generations of interaction styles because they leverage parallel actions and utilize a wider range of interaction devices than previous generations of interaction styles. Hence, it is imperative for an evaluation method (that is designed to evaluate tasks built in an RBI) to be able to accommodate the evaluation of expert and non-expert users alike. Also, because RBIs base their interactions on real-world interactions, and because real-world interactions can be parallel, a model-based evaluation method must have a mechanism that allows the modeling of such actions.

9.1 Summary of Contributions

In this thesis I have presented the following contributions:

- In chapter 5 I proposed a way for modeling actions and specifying user performance according to the knowledge required for the performance of those actions. I have followed that with the proposal of the Efficiency Rating (ER), a measure of the expertise of a user, based on expert, error-free performance.
- In chapters 6 to 8 I have designed, developed, tested, and validated a new task analysis framework, called CoDeIn, that

allows modeling of tasks and allows completion time estimates of expert and non-expert performance of these tasks. This approach has been found to be more accurate in the context of new interaction styles or devices than GOMSL.

- In chapter 8 I have proposed that the reach-and-grasp action may be modeled similarly to Fitts' Law, based on the distance of the actor to the object-to-be-grasped. Contrary to Fitts' Law, the model that I propose for this action is a linear one. Based on this model, I proposed a new operator for GOMS that models this action.

CoDeIn is based on the theory of the MHP (Card et al., 1983). It assumes that knowledge in the performance of one task may be transferred to the performance of another, either positively or negatively (Singley & Anderson, 1989). It also assumes that through experience in the performance of a task, the performance eventually becomes automatic (Logan, 1988).

CoDeIn leverages the level of knowledge of each user type (novice, expert, etc.) and bases task evaluation on models that reflect that level of knowledge. Thus, task evaluation is based on the knowledge that is required for the performance of a task, and the amount of knowledge that a particular class of users has. Because CoDeIn uses task knowledge to model a task, a formal method for the description of knowledge pertaining to the task in an interface was proposed. The knowledge of the users pertaining to the task, as well as the knowledge required for the performance of the task, and the knowledge that exists in the environment about the task, constitute the three sets that CoDeIn uses to describe the level of the user. For example, if the user knows all the required knowledge for the performance of the task, then the user is classified as an expert in the performance of the task. This procedure is followed to define user categories. By measuring the ER of users, a way of modification of expert models was devised to allow the aforementioned claim that CoDeIn can model any type of user.

The prediction of expert task performance occurs by the application of MHP's primitive operators (Card et al., 1983). These operators provide a completion time for each action that is performed for the execution of a task. However, because MHP's primitive operators were primarily created for the evaluation of DM interfaces (Card & Moran, 1980), work was performed to provide operators for actions that do not occur in the DM interaction style. I found that one of the more important actions for which there is no operator in MHP is the prehension operator.

Experiment 1 in Chapter 7 was performed to examine whether CoDeIn's predictions can compete with the predictions of an established model-based evaluation method. The task in experiment 1 compared CoDeIn's predictions on a "file-move" task to GOMSL's (Kieras, 1999) predictions. The task was performed in two interfaces, one designed using the DM interaction style, and another designed as a TUI. The model of the TUI condition required the use of a prehension operator. Because no simple models were found in the surveyed literature (Jones & Lederman, 2006; C. L. Mackenzie & Iberall, 1994), experiment 2 was performed that led to the creation of a model that predicts the completion time of the grasping action based on the distance of the actor from the artifact to be grasped. This experiment yielded a reliable linear relationship between the distance of the actor from the artifact ($R^2 = .82$, $F = 36.81$, $p < .0005$). With the help of this operator the task was modeled in both GOMSL and CoDeIn. CoDeIn had a prediction error of 60 milliseconds from the average experimental result in the DM condition, whereas GOMSL's prediction had an error of 480 milliseconds. In the TUI condition, CoDeIn' prediction error was 1.61 seconds, while GOMSL's prediction had an error of 4.06 seconds. The results of the models in experiment 1 provided evidence to support the initial claim that CoDeIn more accurately predicted the completion time of the task for both the TUI interface and the DM interface than GOMSL.

Another assertion of this work is that CoDeIn may model any type of user, from novice to expert, making it more suitable for the evaluation of RBIs. As was argued RBI users exhibit more varied behavior than that of users of DM interfaces, because RBIs include interfaces that specifically target users who are not experts, such as walk-up-and-use interfaces.

The experiment described in Chapter 8 was performed to test CoDeIn's ability to perform such an evaluation. This experiment was performed in two phases. The first phase was used to study the performance of non-expert users, and the second phase to study non-expert user performance as users gained more experience in the experimental task.

The experimental task for both phases of the experiment asked users to perform a drag-and-drop task, and was designed in the DM interaction style. The experiment included two conditions: one where participants used a conventional mouse to perform the drag-and-drop task, and a second where participants used a virtual glove (Virtual Realities, 2006). The condition using the mouse was considered the expert performance condition, and the condition using the glove was considered the novice performance condition. GOMSL was only used to model the expert condition of the experiment, because due to its assumption of expert, error-free performance of a task, it could model novice performance. However, CoDeIn again yielded a more accurate prediction than GOMSL in the mouse condition of the experiment, with an error of 487 milliseconds, while GOMSL's error was 887 milliseconds. Also, CoDeIn had an error of 517 milliseconds in its prediction of the glove condition average completion time. The importance in the result of the glove condition was that the prediction was found by using the mouse condition as a benchmark, and by penalizing the benchmark time through efficiency ratios, as was claimed in Chapter 5.

The second phase of the experiment was performed to evaluate whether CoDeIn's predictions during the increase in the expertise of a task follow an established prediction model, the Power Law of Practice (Card et al., 1983). The second phase involved a subset of the original participants to the experiment, and asked them to train for 3 weeks on a different task than that of the experiment, using the virtual glove. At the beginning of each week for the duration of the experiment, the participants were tested on the experimental task to examine whether there was any difference in their performance.

Several conclusions were reached from the second phase of the experiment. Following the implications of the Power Law of Practice the trend of learning that CoDeIn should predict should have a decreasing exponential form. Using the modification of expert performance techniques proposed in the first phase of the experiment, CoDeIn was able to capture the trend of learning that the participants exhibited as they gained experience with the task that followed the Power Law of Practice and was matched by the experimental findings. Also, a new form of the Power Law of Practice was derived from the original that uses CoDeIn's ER of a class of users instead of the number of repetitions of the task that were performed by the users.

The experiment presented in Chapter 8 also suggests that expertise may be demonstrated as a decrease in the standard deviation of the performance of a task. As shown in Figure 8.6, as users gained expertise in the performance of the task, the standard deviation of their completion time became smaller. However, this finding has not been investigated further, therefore no conclusions can be drawn about it yet.

There are also some implications for the design of interaction devices that emanate from this thesis. The experiment performed in Chapter 7 demonstrates that the speed with which one may perform actions in a point-and-click environment follows a logarithmic law versus the amplitude of the motions performed, while in a reach-and-grasp environment, the completion time of actions may follow a linear law versus the amplitude of the motions.

Another implication is that actions may be associated with knowledge; knowledge that users are required to know in order to execute the action. This knowledge can be quantified and represented in a model, thus allowing designers to have a guide as to what their users will have to know in order to use the designers' proposed design. This allows the comparison of proposed devices with existing devices before the building of prototypes, but only using the actions and tasks that the devices will be designed to accomplish.

9.2 Future Validation Work

The work in this thesis was done in a closed laboratory environment and therefore the results of the experiments may not generalize to large scale, real-world tasks. Hence there is still the need for the validation of CoDeIn through the evaluation of a real-world, large scale task. This undertaking however, is left as a future research direction. The performance of a large scale experiment is also required to examine whether CoDeIn models scale well as tasks become larger, or they become large and unwieldy, making them unsuitable or inconvenient for complex task modeling. The supposition of this thesis is that the models will scale well because of the diagrammatic notation's basis in Statecharts (Harel, 1987), especially if summary diagrams are used to describe higher-level tasks. However, this cannot be investigated until CoDeIn is used in the predictive evaluation of more complex tasks than considered here.

Further, the framework has only been tested in two interaction styles: DM and TUI. While I believe that the framework will be able to model tasks in other interaction styles, this still remains to be validated. As such, I suggest that more validation work is required by modeling tasks designed in interaction styles other than the ones used in this thesis.

The tasks that CoDeIn modeled in this thesis did not contain any decisions on the part of the user, in terms of choosing one of several possible paths of actions for the completion of a task. More work is required in this area, to demonstrate that CoDeIn can model choice between several paths of actions, and to demonstrate that CoDeIn can produce performance estimates based on such models.

9.3 Future Research Directions

Experiment 2 in Chapter 7 presented evidence that prehension is an action that cannot be modeled using Fitts' Law (Fitts, 1954; Fitts & Peterson, 1964). However, the kinesiology literature surveyed in this thesis does not provide a model of prehension that allows the calculation of the completion time of the action, even though there are models for the explanation of the three phases of prehension as discussed in Chapter 7. While the experiment performed provided a linear model for the prehension action based on the distance of the actor from the artifact, there is need for more experimentation with different reach-to-grasp directions, and with various objects of different weights and sizes to be grasped. Experiment 2 in Chapter 7 was performed to find a specific model for the task of experiment 1, and not a generally applicable model. Also, Figure 7.5 shows a dip between the 20 and 30 centimeter distances of the actor from the artifact. This may be a random event due to sample bias, or it may be a more general event that occurs because of the convenience of that particular range in the execution of the grasping action. Again, more experimentation is needed to ascertain the source of this phenomenon.

CoDeIn was able to predict the completion times of the tasks presented in Chapters 7 and 8 of this dissertation. However, the predictions only involved the prediction of error-free performance on the part of the users. The reality though is that users perform mistakes. It is my belief that CoDeIn's knowledge model is capable of predicting the frequency of errors in the performance of tasks, but this is left as another future research direction.

Another research direction that is left open, but is required for CoDeIn to be considered a complete task analysis framework, is to validate its predictions with tasks that involve choice or selection in the performance of a task. This may happen when a task consists of several sub-tasks where the performance of each sub-task is not a pre-condition in the performance of the others, such as in the case of programming a video player (Gray, 2000).

Work done on the description of actions based on knowledge points to the possible creation of a theory of action independent of the environment in which the action is performed. Rather, the description of the action will only be based on the knowledge required for its performance. The underpinnings of such a theory have been proposed in Chapter 5, but more work is necessary to fully develop it. Using such a theory, CoDeIn could be extended to provide estimates for the performance of errors and counter-productive actions (as defined in Chapter 5).

Finally, CoDeIn can be extended to model tasks that are performed by more than one user in the context of Computer Supported Cooperative Work (CSCW). Environments that support cooperative work may allow the same actions, but the knowledge for their performance does not belong to just one user. Therefore, I believe that CoDeIn may be extended to model and estimate cooperative tasks using work that investigates how knowledge structures are shared among cooperative users.

Bibliography

Abowd, G. D., Mynatt, E. D., & Rodden, T. (2002). The Human Experience. *IEEE Pervasive Computing, 1*(1), 48-57.

Accot, J., & Zhai, S. (1997). *Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks.* Paper presented at the CHI'97 Conference on Human Factors in Computing Systems, Atlanta, GA.

Anderson, J. R. (1992). Automaticity and the ACT* Theory. *American Journal of Psychology, 105*, 165-180.

Anderson, J. R., & Lebiere, C. (1998). *Atomic Components of Thought*. Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson, J. R., Reder, L. M., & Simon, H. A. (1996). Situated Learning and Education. *Educational Researcher, 25*(4), 5-11.

Andre, T. S., Belz, S. M., McCreary, F. A., & Hartson, H. R. (2000). *Testing a Framework for Reliable Classification of Usability Problems.* Paper presented at the 44th Annual Meeting of the Human Factors and Ergonomics Society, San Francisco, CA.

Andre, T. S., Hartson, H. R., Belz, S. M., & McCreary, F. A. (2001). The User Action Framework: A Reliable Foundation for Usability Engineering Support Tools. *International Journal on Human-Computer Studies, 54*(1), 107-136.

Andres, R. O., & Hartung, K. J. (1989). Prediction of Head Movement Time Using Fitts' Law. *Human Factors, 31*(6), 703-714.

Annet, J. (2003). Hierarchical Task Analysis. In D. Diaper & N. Stanton (Eds.), *The Handbook of Task Analysis for Human Computer Interaction* (pp. 67-82). Mahwah, NJ: Lawrence Erlbaum Associates.

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

Arizona State University. Decision Theater. http://dt.asu.edu/index.php?cat=1&subcat=9: March 7th, 2007.

Atari Co. Beach Head 2002. http://www.atari.com/us/games/beach2002/pc: 8th February 2007.

Azuma, R. T. (1997). A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments, 6*, 355-385.

Baber, C., & Stanton, N. (1994). Task Analysis for Error Identification. *Ergonomics, 37*, 1923-1942.

Baskin, J. D., & John, B. E. (1998). *Comparison of GOMS Analysis Methods.* Paper presented at the CHI 98 Conference on Human Factors in Computing Systems, Los Angeles, CA, USA.

Beaudouin-Lafon, M. (2004). *Designing Interaction, Not Interfaces.* Paper presented at the Conference on Advanced Visual Interfaces, Gallipoli, Italy.

Boose, J. H. (1989). A Survey of Knowledge Acquisition Techniques and Tools. *Knowledge Acquisition, 1*, 3-37.

Bransford, J., Brown, A. L., Cocking, R. R., & National Research Council (U. S.) Committee on Learning Research and Educational Practice. (2000). *How People*

Learn: Brain, Mind, Experience, and School: Expanded Edition: National Academies Press.

Brooks Jr., F. P. (1999). What's Real about Virtual Reality? *IEEE Computer Graphics and Applications, 19*(6), 16-27.

Bruner, J. (1966). *Toward a Theory of Instruction*. Cambridge, MA: Harvard University Press.

Buxton, W. (1983). Lexical and Pragmatic Considerations of Input Structures. *Computer Graphics, 17*(1), 31-37.

Card, S. K., Mackinlay, J. D., & Robertson, G. G. (1991). A Morphological Analysis of the Design Spaces of Input Devices. *ACM Transactions on Information Systems, 9*(2).

Card, S. K., & Moran, T. P. (1980). The Keystroke Level Model for User Performance Time with Interactive Systems. *Communications of the ACM, 23*, 396-410.

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

Carroll, J. M., & Rosson, M. B. (1992). Getting Around the Task-Artefact Cycle: How to Make Claims and Design by Scenario. *ACM Transactions on Information Systems, 10*, 181-212.

Cheung, P. (2002). *Designing Sound Canvas: The Role of Expectation and Discrimination.* Paper presented at the CHI '02 Extended Abstracts on Human Factors in Computing Systems, New York, NY.

Chipman, S. F., Schraagen, J. M., & Shalin, V. L. (2000a). *Cognitive Task Analysis*. Mahwah, NJ: Lawrence Erlbaum Associates.

Chipman, S. F., Schraagen, J. M., & Shalin, V. L. (2000b). Introduction to Cognitive Task Analysis. In S. F. Chipman, J. M. Schraagen & V. L. Shalin (Eds.), *Cognitive Task Analysis*. Mahwah, NJ: Lawrence Erlbaum Associates.

Christou, G. (2005). The Use and Evolution of Affordance in HCI. In C. Ghaoui (Ed.), *The Encyclopedia of Human Computer Interaction* (pp. 668-672). Hershey, PA: Idea Group Reference.

Christou, G. (2007). *Towards a New Method of Evaluation for Reality-Based Interaction Styles.* Paper presented at the Extended Abstracts of CHI 07 Conference on Human Factor in Computing Systems, San Jose, CA.

Christou, G., & Jacob, R. J. K. (2003). *Evaluating and Comparing Interaction Styles.* Paper presented at the DSV-IS 2003: 10th Workshop on the Design, Specification and Verification of Interactive Systems, Funchal, Portugal.

Christou, G., & Jacob, R. J. K. (2005). *Identifiability: A Predictive Quantitative Measure for the Comparison of a Task Designed in Different Interaction Styles.* Paper presented at the HCI International, Las Vegas, NV.

Christou, G., Jacob, R. J. K., & Cheng, P. L. (2006). *Modeling the Task: Leveraging Knowledge-in-the-Head at Design Time.* Paper presented at the ICEIS 06 8th International Conference on Enterprise Information Systems, Paphos, Cyprus.

Chung, E. S., Hong, J. I., Lin, J., Prabaker, M. K., Landay, J. A., & Liu, A. L. (2004). *Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing.* Paper presented at the Design of Interactive Systems 2004, Cambridge, MA, USA.

Cook, N. J. (1994). Varieties of Knowledge Elicitation Techniques. *International Journal on Human-Computer Studies, 41*(6), 801-849.

De Carolis, F., De Rosis, F., & Pizzutilo, S. (1996). Formal Description and Evaluation of User Adapted Interfaces. *International Journal on Human-Computer Studies, 49*, 95-120.

Deligiannidis, L. (2000). DLoVe: A Specification Paradigm for Designing Distributed VR Applications for Single or Multiple Users: Tufts University.

Dey, A. K., Ljungstrand, P., & Schmidt, A. (2001). *Distributed and Disappearing User Interfaces in Ubiquitous Computing.* Paper presented at the CHI 01 Conference on Human Factors in Computing Systems, Seattle, WA, USA.

Diaper, D. (2004). Understanding Task Analysis for Human Computer Interaction. In D. Diaper & N. Stanton (Eds.), *The Handbook of Task Analysis for Human Computer Interaction* (pp. 5-47). Mahwah, NJ: Lawrence Erlbaum Associates.

Diaper, D., & Stanton, N. (2004a). *The Handbook of Task Analysis for Human-Computer Interaction*. Mahwah, NJ: Lawrence Erlbaum Associates.

Diaper, D., & Stanton, N. (2004b). Wishing on a sTAr: The Future of Task Analysis. In D. Diaper & N. Stanton (Eds.), *The Handbook of Task Analysis for Human Computer Interaction* (pp. 603-619). Mahwah, NJ: Lawrence Erlbaum Associates.

Drury, C. G., Paramore, B., Van Cott, H. P., Grey, S. M., & Corlett, N. (1987). Task Analysis. In G. Salvendy (Ed.), *Handbook of Human Factors* (pp. 370-401). New York, NY: Wiley.

Fitts, P. M. (1954). The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology, 47*(6), 381-391.

Fitts, P. M., & Peterson, J. R. (1964). Information Capacity of Discrete Motor Responses. *Journal of Experimental Psychology, 67*(2), 103-112.

Fitzmaurice, G. W. (1996). Graspable User Interfaces: Department of Computer Science, University of Toronto.

Fitzmaurice, G. W., Ishii, H., & Buxton, W. (1995). *Bricks: Laying the Foundations for Graspable User Interfaces.* Paper presented at the CHI 95 Conference on Human Factors in Computing Systems, Denver, CO.

Foley, J. D. (1987). Interfaces for Advanced Computing. *Scientific American, 257*(4), 127-135.

Foley, J. D., Van Dam, A., Feiner, S. K., & Hughes, J. F. (1990). *Computer Graphics: Principles and Practice*. Reading, MA: Addison Wesley.

Foley, J. D., & Wallace, V. L. (1974). The Art of Natural Graphic Man-Machine Conversation. *Proceedings of the IEEE, 62*(4), 462-471.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. New York, NY: Addison-Wesley.

Gaver, W. (1991). *Technology Affordances.* Paper presented at the Conference on Human factors in computing systems: Reaching through technology, New Orleans, Louisiana.

Gibson, J. J. (1977). The Theory of Affordances. In R. E. Shaw & J. Bransford (Eds.), *Perceiving, Acting and Knowing*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Boston, MA: Houghton Mifflin Co.

Gilbreth, F. (1911). *Motion Study*. New York, NY: D. Van Nostrand Company.

Glasser, H. J., & Halcomb, C. G. (1980). *Foot placement and response latency: A test of Fitts' law.* Paper presented at the Human Factors Society 24th Annual Meeting, Santa Monica, CA.

Gray, W. D. (2000). The Nature and Processing of Errors in Interactive Behavior. *Cognitive Science, 24*(2), 205-248.

Gray, W. D., & Altmann, E. M. (2001). Cognitive Modeling and Human-Computer Interaction. In W. Karwowski (Ed.), *International Encyclopedia of Ergonomics and Human Factors* (Vol. 1, pp. 387-391). New York: Taylor & Francis, Ltd.

Gray, W. D., & Fu, W. (2001). *Ignoring Perfect Knowledge In-The-World for Imperfect Knowledge In-The-Head: Implications of Rational Analysis for Interface Design.* Paper presented at the CHI Conferance on Human Factors of Computing Systems, Seattle, WA, USA.

Gray, W. D., & Fu, W. (2004). Soft Constraints in Interactive Behavior: the Case of Ignoring Perfect Knowledge in-the-World for Imperfect Knowledge in-the-Head. *Cognitive Science, 28*, 359-382.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance. *Human Computer Interaction, 8*(3), 237-309.

Greenburg, S. (2004). Working Through Task Centered System Design. In D. Diaper & N. Stanton (Eds.), *The Handbook of Task Analysis* (pp. 49-65). Mahwah, NJ: Lawrence Erlbaum Associates.

Greeno, J. G., Smith, D. R., & Moore, J. L. (1992). Transfer of Situated Learning. In D. Detterman & R. J. Sternberg (Eds.), *Transfer on Trial: Intelligence, Cognition and Instruction* (pp. 99-167). Norwood, NJ: Ablex.

Grimson, W. E. L., Ettinger, G. J., Kapur, T., Leventon, M. E., Wells III, W. M., & Kikinis, R. (1997). Utilizing Segmented MRI Data in Image-Guided Surgery. *International Journal of Pattern Recognition and Artificial Intelligence, 11*, 1367-1397.

Hall, W. (1994). Ending the Tyranny of the Button. *IEEE Multimedia, 1*(1), 60-68.

Hamilton, F. (1996). *Predictive Evaluation using Task Knowledge Structures.* Paper presented at the CHI Conference on Human Factors in Computing Systems, Vancuver, BC.

Hamilton, F., Johnson, P., & Johnson, H. (1998). *Task-Related Principles for User Interface Design.* Paper presented at the Schaerding Workshop on Task Analysis, Schaerding, Austria.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*(8), 231-274.

Harrison, M. A. (1978). *Introduction to Formal Language Theory* (First ed.). Boston, MA: Addison-Wesley Longman.

Hartson, H. R. (2003). Cognitive, Physical, Sensory and Functional Affordances in Interaction Design. *Behaviour & Information Technology, 22*(5), 315-338.

Hartson, H. R., Andre, T. S., Williges, R. C., & Van Rens, l. (1999). *The User Action Framework: A Theory-Based Foundation for Inspection and Classification of Usability Problems.* Paper presented at the HCI International '99 8th International Conference on Human Computer Interaction.

Hartson, H. R., Siochi, A. C., & Hix, D. (1990). The UAN: A User-Oriented Representation for Direct Manipulation. *ACM Transactions on Information Systems, 8*(3), 181-203.

Hinckley, K., Tullio, J., Pausch, R., Proffitt, D., & Kassell, N. (1997). *Usability Analysis of 3D Rotation Technique.* Paper presented at the UIST '97 Symposium on User Interface Software and Technology, Banff, Alberta, Canada.

Holmquist, L. E., Redstrom, J., & Ljungstrand, P. (1999). *Token-Based Access to Digital Information.* Paper presented at the Proceedings of the First International Symposium in Handheld and Ubiquitous Computing, Karlsruhe, Germany.

Howes, A., & Young, R. M. (1991). *Predicting the Learnability of Task-Action Mappings.* Paper presented at the Conference on Human Factors in Computing Systems, New Orleans, LA, USA.

Hutchins, E., Hollan, J., & Norman, D. (1986). Direct Manipulation Interfaces. In D. A. Norman & S. W. Draper (Eds.), *User Centered System Design: New Perspectives in Human-Computer Interaction* (pp. 87-124). Hillsdale, NJ: Lawrence Erlbaum Associates.

Ishii, H., Mazalek, A., & Lee, J. (2001). *Bottles as a Minimal Interface to Access Digital Information.* Paper presented at the CHI '01 Extended Abstracts on Human Factors in Computing Systems, New York, NY.

Ishii, H., & Ullmer, B. (1997). *Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms.* Paper presented at the CHI Conference on Human Factors in Computing Systems, Atlanta, GA, USA.

Jacob, R. J. K. (1995). Natural Dialogue in Modes Other Than Natural Language. In R. J. Beun, M. Baker & R. M. (Eds.), *Dialogue and Instruction* (pp. 289-301). Berlin, Germany: Springer-Verlag.

Jacob, R. J. K. (2004, September 22, 2006). Reality-Based Interaction: Understanding the Next Generation of Human-Computer Interfaces. from http://www.cs.tufts.edu/~jacob/theory/

Jacob, R. J. K. (2006). *CHI 2006 Workshop Proceedings: What is the Next Generation of Human Computer Interaction?* Medford, MA: Tufts University.

Jacob, R. J. K., Deligiannidis, L., & Morrison, S. (1999). A Software Model and Specification Language for Non-WIMP User Interfaces. *ACM Transactions on Computer-Human Interaction, 6*(1), 1-46.

Jacob, R. J. K., Ishii, H., Pangaro, G., & Patten, J. (2002). *A Tangible Interface for Organizing Information Using a Grid.* Paper presented at the CHI 02 Conference on Human Factors in Computing Systems, Minneapolis, MS.

Jacob, R. J. K., Leggett, J. J., Myers, B. A., & Pausch, R. (1993). Interaction Styles and Input/Output Devices. *Behaviour & Information Technology, 12*(2), 69-79.

Jacob, R. J. K., & Sibert, L. E. (1992). *The Perceptual Structure of Multidimensional Input Device Selection.* Paper presented at the CHI'92 Conference on Human Factors in Computer Systems.

Jacob, R. J. K., Sibert, L. E., McFarlane, D. C., & Mullen Jr., M. P. (1994). Integrality and Separability of Input Devices. *ACM Transactions on Computer-Human Interaction, 1*(1), 3-26.

John, B. E. (1988). *Contributions to engineering models of human-computer interaction. (volumes i and ii).* Carnegie Mellon University, Pittsburgh, PA.

John, B. E. (2003). Information Processing and Skilled Behaviour. In J. M. Carroll (Ed.), *HCI Models, Theories, and Frameworks* (pp. 55-101). San Francisco, CA: Morgan Kaufmann.

John, B. E., & Kieras, D. (1996a). The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Transactions on Computer-Human Interaction, 3*(4), 320-351.

John, B. E., & Kieras, D. (1996b). Using GOMS for User Interface Design and Evaluation: Which Technique? *ACM Transactions on Computer-Human Interaction, 3*(4), 287-319.

Johnson, H., & Johnson, P. (1991). Task Knowledge Structures: Psychological basis and Integration into System Design. *Acta Psychologica, 78*, 3-26.

Johnson, J., Roberts, T., Smith, D. C., Irby, C. H., Beard, M., & Mackey, K. (1989). The Xerox Star: A Retrospective. *IEEE Computer, 22*(9), 11-29.

Johnson, P., & Johnson, H. (1991). Knowledge Analysis of Tasks: Task Analysis and Specification for Human-Computer Interaction. In A. Downton (Ed.), *Engineering the Human-Computer Interface* (pp. 117-144). London, U.K.: McGraw-Hill.

Johnson, P., Johnson, H., Waddington, R., & Shouls, A. (1988). Task Related Knowledge Structures: Analysis, Modelling and Application. In D. M. Jones & R. Winder (Eds.), *People and Computers: From research to implementation* (pp. 35-62). Cambridge: Cambridge University Press.

Jones, L. A., & Lederman, S. J. (2006). *Human Hand Function*. New York, NY, USA: Oxford University Press.

Kieras, D. (1996). A Guide to GOMS Model Usability Evaluation Using NGOMSL. Retrieved September 18, 2006, 2006, from http://www.eecs.umich.edu/~kieras/goms.html

Kieras, D. (1999). A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3. Retrieved September 20th, 2006, from http://citeseer.ist.psu.edu/kieras99guide.html

Kieras, D., & Meyer, D. E. (1997). An Overview of the EPIC Architecture for Cognition and Performance with Application to Human-Computer Interaction. *Human Computer Interaction, 12*, 391-438.

Kieras, D., & Polson, P. G. (1985). An Approach to the Formal Analysis of User Complexity. *International Journal of Man-Machine Studies, 22*, 365-394.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence, 33*(1), 1-64.

Lave, J., & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge, MA: Cambridge University Press.

Logan, G. D. (1988). Toward an Instance Theory of Automatization. *Psychologoical Review, 95*, 492-527.

Mackenzie, C. L., & Iberall, T. (1994). *Advances in Psychology: The Grasping Hand*. Amsterdam: Elsevier Science BV.

MacKenzie, I. S. (1991). *Fitts' Law as a Performance Model in Human-Computer Interaction.* Toronto, Ontario, Canada: Ph.D. Thesis, University of Toronto.

MacKenzie, I. S. (1992a). Fitts' law as a research and design tool in human-computer interaction. *Human Computer Interaction, 7*, 91-139.

MacKenzie, I. S. (1992b). *Movement time prediction in human-computer interfaces.* Paper presented at the Graphics Interface '92, Toronto, Canada.

MacKenzie, I. S., & Buxton, W. (1994). The prediction of pointing and dragging times in graphical user interfaces. *Interacting with Computers, 6*, 213-227.

Marcus, A. (2006). From KidCHI to BabyCHI. *Interactions, 13,* 52-53.

McGrenere, J., & Ho, W. (2002). *Affordances: Clarifying and evolving a concept.* Paper presented at the Graphics Interface 2000, Montreal.

Miniotas, D. (2000). *Application of Fitts' Law to Eye-Gaze Interaction.* Paper presented at the CHI 00 Extended Abstracts on Human Factors in Computing Systems, The Hague, Netherlands.

Nielsen, J. (1990). A Meta-Model for Interacting With Computers. *Interacting with Computers, 2*(2), 147-160.

Nielsen, J. (1993). Noncommand User Interfaces. *Communications of the ACM, 36*(4), 83 - 99.

Norman, D. (1986). Cognitive Engineering. In D. Norman & S. W. Draper (Eds.), *User Centered Systems Design: New Perspectives on Human-Computer Interaction* (pp. 31-62). Hillsdale, NJ: Lawrence Erlbaum Associates.

Norman, D. (1988). *The Psychology of Everyday Things*: Basic Books.

Norman, D. (1993). *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine.* Cambridge, MA: Perseus Books.

Norman, D. (1999a). Affordance, Conventions, and Design. *Interactions, 6*(3), 38-43.

Norman, D. (1999b). *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. Cambridge, MA, USA: The MIT Press.

Norman, D. (2002). *The Design of Everyday Things*. New York, NY: Basic Books.

Olson, J. R., & Biolsi, K. J. (1991). Techniques for Representing Expert Knowledge. In K. A. Ericsson & J. Smith (Eds.), *Toward a General Theory of Expertise*. Cambridge, UK: Cambridge University Press.

Overbeeke, K. C. J., & Wensveen, S. A. G. (2003). *From perception to experience, from affordances to irresistibles.* Paper presented at the DPPI 2003: International Conference on Designing Pleasurable Products and Interfaces, Pittsburgh, PA.

Palumbo, D. (1990). Programming Language/Problem Solving Research: A Review of Relevant Isssues. *Review of Educational Research, 60*(1), 65-89.

Patrick, E., Cosgrove, D., Slavkovic, A., Rode, J. A., Verratti, T., & Chiselko, G. (2000). *Using a Large Projection Screen as an Alternative to Head-Mounted Displays for Virtual Environments.* Paper presented at the CHI 00 Conference on Human Factors in Computing Systems, The Hague, Netherlands.

Payne, S. J., & Green, T. R. G. (1986). Task Action Grammars: A Model of the Mental Representation of Task Languages. *Human Computer Interaction, 2*(2), 93-133.

Picard, R. W. (2000). *Affective Computing*. Cambridge, MA: The MIT Press.

Psychology. http://en.wikipedia.org/wiki/Psychology: April 21st, 2007.

Ritter, F. E., & Schooler, L. J. (2001). The Learning Curve. In N. J. Smelser & P. B. Baltes (Eds.), *International Encyclopedia of Social and Behavioral Sciences* (pp. 8602-8605). Amsterdam: Pergamon.

Rohr, G., & Tauber, M. J. (1984). *Representational Frameworks and Models for Human-Computer Interfaces.* Paper presented at the 2nd European Conference on Cognitive Ergonomics, Gmunden, Austria.

Saha, D., & Mukherjee, A. (2003). Pervasive computing: a paradigm for the 21st century. *IEEE Computer, 36*(3), 25 - 31.

Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications, 8*(4), 10-17.

Schilit, B. N., Adams, N. I., & Want, R. (1994). *Context-Aware Computing Applications.* Paper presented at the Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA.

Sebesta, R. W. (2006). *Concepts of Programming Languages* (Seventh ed.). Boston, MA: Pearson Eductation.

Shaer, O., Leland, N., Calvillo-Gamez, E. H., & Jacob, J. K. R. (2004). The TAC Paradigm: Specifying Tangible User Interfaces. *Personal and Ubiquitous Computing, 8*(5), 359-369.

Shiaw, H. (2003). New Interaction Techniques for the Digital Library: 3D Focus+Context Interactive Visualization: Tufts University.

Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer, 16*(8).

Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human Computer Interaction*: Addison Wesley.

Singley, M. K., & Anderson, J. R. (1989). *The Transfer of Cognitive Skill* (Vol. 9). Cambridge, MA: Harvard University Press.

Sipser, M. (2006). *Introduction to the Theory of Computation* (Second ed.). Boston, MA: Thomson Course Technology.

St. Amant, R., Freed, A. R., & Ritter, F. E. (2005). Specifying ACT-R Models of User Interaction with a GOMS Language. *Cognitive Systems Research, 6*, 71-88.

Stanford Encyclopedia of Philosophy: Cognitive Science. http://plato.stanford.edu/entries/cognitive-science/: April 2nd, 2007.

Stanford Encyclopedia of Philosophy: The Frame Problem. http://plato.stanford.edu/entries/frame-problem/: April 2nd, 2007.

Steedman, M. (2002). *Formalizing Affordance.* Paper presented at the 24th Annual Meeting of the Cognitive Science Society, Fairfax, VA.

Sutherland, I. E. (1965). *The Ultimate Display.* Paper presented at the IFIP Congress 65.

Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science, 12*, 257-285.

Taylor, F. (1911). *Scientific Management*. New York, NY: Harper and Row.

Thacker, C. P. (1979). *Alto: A Personal Computer*. Palo Alto, CA: Xerox Palo Alto Research Center.

Turk, M., & Robertson, G. (2000). Perceptual User Interfaces. *Communications of the ACM, 40*(2), 63 - 67.

Turner, P., & McEwan, T. (2004). Activity Theory: Another Perspective on Task Analysis. In D. Diaper & N. Stanton (Eds.), *The Handbook of Task Analysis for Human-Computer Interaction* (pp. 423-439). Mahwah, NJ: Lawrence Erlbaum Associates.

Ullmer, B., & Ishii, H. (2000). Emerging Frameworks for Tangible User Interfaces. *IBM Systems Journal, 39*(3&4), 915-931.

Ullmer, B., & Ishii, H. (2001). Emerging Frameworks for Tangible User Interfaces. In J. M. Carroll (Ed.), *Human Computer Interaction in the New Millenium* (pp. 579-601): Addison-Wesley.

Van Dam, A. (1997). Post-WIMP User Interfaces. *Communications of the ACM, 40,* 63-67.

Virtual Realities. (2006). P5 Glove. http://www.vrealities.com/P5.html: 2nd February 2007.

Waibel, A., Tue, M., Duchnowski, P., & Manke, S. (1996). Multimodal Interfaces. *Artificial Intelligence Review, 10*(3-4), 299-319.

Want, R., Schilit, B. N., Adams, N. I., Gold, R., Petersen, K., Goldberg, D., et al. (1995). An Overview of the PARCTAB Ubiquitous Computing Experiment. *IEEE Personal Communications, 2*(6), 28-43.

Ware, C., & Jessome, D. R. (1988). Using the Bat: A Six-Dimensional Mouse for Object Placement. *IEEE Computer Graphics and Applications, 8*(6), 65-70.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American, 265,* 94-104.

Weiser, M. (1993). Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM, 36*(7), 74-83.

Woods, W. A. (1970). Transition Network Grammars for Natural Language Analysis. *Communications of the ACM, 13*(10), 591-606.