

**CODEIN – A NEW NOTATION FOR GOMS TO HANDLE EVALUATIONS OF  
REALITY BASED INTERACTION STYLE INTERFACES**

**Corresponding Author:**

Georgios Christou  
Department of Computer Science & Engineering  
European University Cyprus  
6 Diogenes St.  
2404, Engomi, Nicosia  
Cyprus  
Phone: +357-22713104  
Email: g.christou@euc.ac.cy

Frank E. Ritter  
College of Information Sciences and Technology  
Penn State University  
University Park, PA 16802  
U.S.A.  
Phone: +1 (814) 865-4453  
Email: frank.ritter@psu.edu

Robert J. K. Jacob  
Dept. of Computer Science  
Tufts University  
Halligan Hall  
161 College Avenue  
Medford, Mass. 02155  
U.S.A.  
Phone: +1 (617) 627-2225  
Email: jacob@cs.tufts.edu

## **ABSTRACT**

We propose a new diagrammatic notation system for GOMS, called Codein, with extensions to support the evaluation of Reality Based Interaction Styles (RBIs). The proposed notation gives added power to GOMS to model and evaluate the task completion time of parallel actions during the performance of a task, something that was previously only possible using CPM-GOMS, which is far more complicated to use. Codein's evaluative power is verified through an experiment. The first condition of the experiment compares the completion time predicted by a GOMSL model, a Codein model, and the actual completion time of participants in a direct manipulation task. The second compares the completion time of participants in a Tangible User Interface task with predictions by a GOMSL model and a Codein model. Predicted task times by Codein in both conditions come close to the actual experimental results.

## INTRODUCTION

There are many methods to evaluate Direct Manipulation (DMI) interfaces (Hutchins, Hollan, & Norman, 1986; Shneiderman, 1983, Alonso-Ríos et al., 2010), with one of the most prominent being the GOMS (Goals, Operators, Methods, Selection Rules) family (Card, Moran, & Newell, 1983; John & Kieras, 1996a). But as new interaction styles are created to take advantage of quickly evolving technology, can the existing evaluation methods be used with these new interaction styles? Also, most established evaluation methods mostly apply to the DMI style, and are not able to handle continuous, parallel interactions, because they are not used in the DMI style. New interaction styles however, such as Virtual Reality (VR), Tangible User Interfaces (TUI; Ishii & Ullmer, 1997), Ubiquitous Computing (Weiser, 1991), and others, do use them. To refer to all the new interaction styles, we use a term proposed by Jacob et al. (2008), namely Reality-Based Interfaces (RBI).

Here, we propose a new notation system for GOMS (Card, Moran, & Newell, 1983), with extensions to support the evaluation of RBIs. The notation provides tools to evaluate a task based on the knowledge that users need to possess to perform that task. This is done through a diagrammatic notation that enables the designer to model the interactions inside a RBI interface, given the interface actions and the knowledge that the user needs to have to perform each action.

To evaluate how well the proposed notation performs compared to GOMS, we present an experiment that compares the findings of the proposed notation to GOMS augmented with Fitts' Law. Fitts' Law models human pointing movements

and predicts that the time required to point to an object is a function of the size of the target object and the distance of the target to the actor (Fitts, 1954; Fitts & Peterson, 1964; MacKenzie, 1991). The result is that the proposed notation performs at least as well as GOMS and Fitts' Law combined. We then describe another experiment to test whether the proposed notation can be used to evaluate a TUI (Ishii & Ullmer, 1997). We show that while GOMS is unable to model this task, through the proposed notation with the addition of an operator, the task can be modeled with predicted completion time close to the actual completion time of the task.

## **BACKGROUND**

GOMS is one of the most widely known evaluation methods in HCI. Based on the goal of the user to perform a task, it analyzes the task using operators, methods and selection rules. Kieras (2006) has written extensively on how GOMS can be used to evaluate how interfaces support tasks.

GOMS, though, has some well known problems. First, it only applies to expert, error-free performance (John, 2003), dealt with by other methods (MacDorman et al, 2011), excluding evaluation for occasional users who are the most frequent users of RBIs. Second, only one of its varieties, CPM-GOMS (Gray, John, & Atwood, 1993), allows the evaluation of parallel tasks, something common in RBIs (Jacob et al., 2008). However, CPM-GOMS is very complex for most evaluation analyses (John & Kieras, 1996b). With the proposed notation we aim to provide an easier approach to modeling parallel tasks.

GOMS is frequently augmented with Fitts' Law (Fitts, 1954; Fitts & Peterson, 1964; MacKenzie, 1991). A Fitts' Law formulation often used is  $MT = \alpha + \beta \cdot \log(1 + D/W)$ , where MT is the time to point to an object, D is the distance from the object, W is the size of the object on the plane of observation, and  $\alpha$  and  $\beta$  are experimentally defined constants. It is common to replace the pointing operator in GOMS with a Fitts' Law model of the pointing task in question (Christou, 2007).

The proposed notation uses Fitts' Law to evaluate pointing movements and KLM-GOMS (Card, Moran, & Newell, 1983) operators to evaluate other portions of the task. The notation is influenced by State Transition Networks (Newman, 1969), Statecharts (Harel, 1987), and cognitive architectures, such as ACT-R (Anderson & Lebiere, 1998) and Soar (Laird, Newell, & Rosenbloom, 1987).

State Transition Networks are graph theoretic structures denoting system states as circles, and transitions between states as arrows in a diagram. The transitions (arrows) are labeled with the user actions that trigger the transition and the system response. Statecharts model state diagrams with the addition of allowing each state diagram to transition inside of a state, creating super-states, without affecting the states in the diagram. Last, cognitive architectures are theories that aim to explain human cognition. A fundamental argument for some cognitive architectures is the distinction between procedural and declarative knowledge, where procedural knowledge is modeled as productions, and declarative knowledge is modeled as a propositional network. The interested

reader is referred to Anderson (1982) and Anderson and Lebiere (1998) for further information about declarative and procedural knowledge.

## **CODEIN: THE PROPOSED NOTATION**

We propose a diagrammatic notation that extends GOMS so that it can be used in the evaluation of RBIs, as well as DMI interfaces. We name this diagrammatic notation COgnitive Description and Evaluation of INteraction (Codein). The notation is based on the categorization of task knowledge into two types: declarative and procedural (Anderson & Lebiere, 1998, Christou, Ritter, & Jacob, 2009). Working with this notation presupposes that domain knowledge is constant across interfaces, and that the user already possesses the required domain knowledge for the task. For example, if the task is to cook a chicken using a specific recipe, the notation does not model the recipe knowledge, as this is considered known by the task performer. What is modeled is the knowledge required for the use of the oven that will be used to cook the chicken. As the recipe will be the same no matter what kind of oven is used, the domain knowledge is considered to remain constant in the modeled scenarios.

The distinction between declarative and procedural knowledge is shown in the following example: Suppose the goal is to move the pointer from point A to point B on the screen. Then one of the things the user needs to know is that the mouse is bound to the pointer, 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. This is a procedural knowledge chunk.

But why the user needs to perform this task is not modeled, as this would be domain knowledge.

## **Terminology**

The proposed notation includes a few terms to refer to different devices and user interface constructs in the same manner, over different interaction styles. This is necessary to enable the notation's users to talk about designs in different interaction styles. For example, when talking about a user interface construct in a Tangible User Interface vs. one in a Virtual Environment, the terminology will help disambiguate the role of the constructs. This in turn, leads to clearer conversations between designers of interfaces in different interaction styles.

### ***Data Object***

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

A Data Object is defined as the representation of a distinct data entity or group of entities in an interaction style. Data Objects 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. Data Objects may be comprised of other Data Objects, and they can be combined to create more complex Data Objects.

Examples of Data Objects in DMIs are the icons that represent files, folders, applications, etc. In fact, a folder is a Data Object that includes other Data Objects (the file icons). An example Data Object that includes other Data Objects is a document page in Microsoft Word. The page can be considered a Data Object because it is the representation of the document data in a form that allows users to manipulate that data. A page is comprised of sections, paragraphs, sentences, words, characters, and pictures, which are also Data Objects.

A VR museum application, created by Shiaw, Jacob and Crane (2004), presents ancient Greek vases in various ways, one of these being a depiction of the vases on pedestals. The user may approach the vases, grab them in the virtual world and manipulate them to get a better sense of the exhibits than one would in a real world museum where manipulation of the artifacts is strictly forbidden.

In this application, the Data Objects are the vases as depicted in VR, because these are the items that the user will manipulate. Here the interaction is smooth and intuitive because users may interact with the Data Objects as they would interact with the real artifacts in the real world.

### ***Interaction Objects***

In many interaction styles there are artifacts, real or virtual, that allow the user to interact with the Data Objects. For example, in DMI the user may use the pointer to drag and drop a file from one folder to another. In this case the user uses the pointer, which is the Interaction Object to manipulate the icon of a file, which is the Data Object.



Interaction objects are the objects that are perceived by the user to be the means of interaction with the Data Objects 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 Data Objects. Interaction objects do not have to be manipulated directly by a user, although in many RBIs this is possible.

Interaction objects do not have to act on only one Data Object. They may act on a group of Data Objects, or on specific properties or attributes of a Data Object. Multiple Interaction objects may be used together to achieve a desired action, and Interaction objects may only work on a particular type of Data Object as well.

In many RBIs the hands are the Interaction objects 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 Interaction objects are the hands, the actual Interaction objects are the representations of the hands in the virtual environment and the quality of their simulation influences the quality of the experience. The devices used to track the hands are Intermediary Objects, as explained below.

### ***Intermediary Objects***

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

Intermediary objects are never virtual. They are always real physical objects that are bound to some Interaction object in the interface. Through this binding they allow the manipulation of the interaction object by the user. As mentioned in the previous section, a data glove can be an Intermediary object through which the user controls her hands' representations, which allow changes on the Data Objects in the virtual environment. Having an Intermediary object control the Interaction object that controls the Data Object 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 Intermediary object and the Interaction object works.

Another example of an Intermediary object is the mouse, which in DMI is bound to the pointer on the screen. The mouse is the physical artifact manipulated by the user to control the pointer. The pointer in this case is the Interaction object that allows the manipulation of Data Objects. But any device that controls the pointer in DMI is an Intermediary object, such as track balls, the touchpad, or even the cursor keys on the keyboard. There are, however, several devices that are not Intermediary objects, because they do not control the pointer, but rather become the pointer themselves, such as light pens, or they allow the user to manipulate the Data Objects directly, such as the touch screen.

In Virtual Reality, on the other hand, the Intermediary objects take many forms. In the case of immersive Virtual Reality, the head-mounted display is an Intermediary object, 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 the head-mounted display is the Intermediary object through which users orient themselves in the virtual environment. Another Intermediary object is the device that positions the user's hand or hands in the virtual environment. This could be a data glove 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 Intermediary object. Rather, the Intermediary object's role is partially hidden from the user.

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

### ***Bindings***

A binding signifies the places where the Interaction object and the Data Object directly connect to execute an action by the user. When an Interaction object touches or in some other way comes in contact with the Data Object, we say that the Interaction object is bound to the Data Object.

When an Interaction object manipulates or controls one or more Data Objects, or one of a Data Object's properties, the Interaction object is said to be bound to the Data Object. Binding an Interaction object to a Data Object means that the Data Object has the user's focus, and it is going to be manipulated or it is being manipulated.

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 Intermediary objects and Interaction objects, such as the binding of the mouse to the pointer. Dynamic bindings though, are usually found between Interaction objects and Data Objects.

A static binding example is of the mouse and the pointer. The Intermediary object (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 Interaction object. Consider the case of a video game that changes the pointer shape. The pointer is still manipulated through its Intermediary object, therefore the binding still applies.

### **Codein's Notation**

Codein's notation uses a State Transition Network to represent the flow of the task from one action to the other, and allows zooming into task states to see their

internals, similar to Statecharts (Harel, 1987). The full notation for the proposed notation is shown in Figure 1.

**[Figure 1 goes about here]**

Figure 1 presents the four components that comprise the notation. First, the rectangle represents a knowledge state. Each knowledge state may hold other knowledge states, or knowledge that the user must have to perform the action coming out of the state. An example is the knowledge state that users are in every time they perform a double-click to open a folder. They summon knowledge about how to open a folder using the left mouse button that will be translated into the command “open folder” by the user interface.

A procedural chunk (Anderson & Lebiere, 1998) represents the procedural knowledge that the user needs to possess to perform actions allowed at that state. Procedural knowledge is how-to knowledge, and in Codein it is not decomposed into its specific components. Rather, we consider that the required knowledge for each action is atomic, in that it cannot be decomposed any further. Declarative knowledge chunks can be thought of as procedures in cognitive architectures, such as ACT-R (Anderson & Lebiere, 1998) or Soar (Laird, Newell & Rosenbloom, 1987).

A declarative knowledge chunk (Anderson & Lebiere, 1998) is denoted by a circle, with a description of the chunk inside it. Declarative knowledge is knowledge about facts. Thus, when a declarative chunk is included in a state, then that chunk is a fact that needs to be known (a precondition) for any action coming out of that state.

Last, an arrow represents an action that the user performs. The description of the action is written as an arrow label and is underlined, and, if applicable, a condition is written in the label, without underlining. The condition of the action defines when the transition will be followed. In any knowledge state, when an arrow exists, it may be followed (i.e., the action may be performed) as long as the knowledge requirements of the knowledge state and any action conditions are satisfied. Action conditions are optional.

### **Building Codein Models**

Codein models are constructed similarly to GOMSL models (Kieras, 2006). As with GOMS, Codein model construction begins after task analysis. Once the designer defines the user interface tasks to be modeled, and decomposes those tasks into actions and knowledge to execute those actions, a summary diagram can be built. This diagram displays the knowledge states that users will go through and the actions they will perform to reach the ending knowledge state, the state where the user knows that the task was completed successfully. The designer then draws diagrams for each knowledge state, denoting the procedural and the declarative knowledge required for the performance of each action as those are shown in the summary diagram. This knowledge must be included in each action's preceding knowledge state. The resultant diagrams thus describe both the procedural and declarative knowledge required for the completion of the task. Task completion time can then be predicted by assigning Keystroke Level Model operators (Card et al., 1983) to the actions and the mental steps

performed. This process is best demonstrated by example, which we do in the next section.

## **EVALUATION OF CODEIN**

The experiment performed to compare task completion time predictions of Codein and GOMSL (Kieras, D., 2006) consisted of having a group of 13 experienced MS Windows XP™ participants perform a file manipulation task in two conditions. We then compare the participant's completion times to GOMSL and Codein's predictions.

### **Participants**

The 13 participants, 9 male and 4 female, were all undergraduate Computer Science Students at the Computer Science department of the European University Cyprus. Their ages varied between 19 to 30 years old ( $M = 22.8$ ,  $SD = 3.5$ ). All the participants were enrolled in an introductory HCI course at the time of the experiment and participated in the study for course credit.

### **Materials and Design**

The task was designed in two different interaction styles, as shown in Figures 2 and 3: one condition was based on MS Windows XP™ representing the DMI style (Figure 2), and the second condition was a mockup of a TUI (Figure 3). The DMI condition's display size was 1024x768 pixels on a CRT monitor, and the window size, which was kept constant over all trials, was 610x460 pixels.

**[Figures 2 and 3 go about here]**

The TUI condition's interface was built as a Wizard of Oz interface (a term coined by Kelley, 1983), as shown in Figure 3. Wizard of Oz interfaces are mockups of the real interface, with a human confederate driving the interface reacting to the actions of participants. Research on TUIs often uses Wizard of Oz systems as mockups, because it allows ideas to be tested before the actual interface can be built (Consolvo et al, 2007). Because a new interface to a common problem (file management) is examined, the results of the experiment should not be affected. The folders were represented by cardboard boxes (16 x 14 cm). 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 (7.2 x 5.4 cm). 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 4.

**[Figure 4 goes about here]**

Each experimental condition consisted of a single task. The song/album pairs were constant across participants and conditions. All participants performed both conditions but with the tasks in random order. Each participant performed ten trials per condition, with different song/album pairs.

### **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 at



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)”.

The participants began each trial with the pointer at the right side of the window. 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 proceeded to move the pointer above the source folder, double-clicked it, positioned the pointer over the target file, right-clicked, and selected cut from the popup menu. This completed the 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 signaled the end of the trial. The task was timed using a program that recorded the participants’ movements and kept timing information for the complete duration of the task.

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. In this condition, the task was recorded using a video camera, and the time required for the completion of the task was found through frame-by-frame analysis of the recording. The software used was Virtual Dub (Lee, <http://www.virtualdub.org/>).

Because the software used for the analysis allowed viewing of one frame per 300–350ms, the results recorded were rounded to the nearest half-second.

## Results

All participants completed the tasks without errors. Table 1 shows their results. The GOMSL and Codein models' results come from the models described and discussed in subsequent sections. The deviation columns in Table 1 show by how much the models' predictions either overshoot (positive deviation) or undershoot (negative deviation) the actual results.

### **[Table 1 goes about here]**

The first result to note in Table 1 is that the TUI condition task can be completed faster than the DMI condition, and the standard deviation of the completion time of the DMI condition's task is greater than that of the TUI condition. This was expected, because the tasks are not completely equivalent. While the obvious difference is that one is designed to be performed on a computer and the other is performed in a TUI, there are further differences. For example, the TUI condition's folder representation was always open, and some of the content of each folder was always visible to the participants. On the other hand, in the DMI condition's task the participants needed to perform actions to open the required folders, and in the folder view the contents of the folders were not visible to the participants. Also, the participants were able to perform actions using both hands in the TUI condition, whereas in the DMI condition participants are restricted to the use of one pointer on the screen.

## CONSTRUCTION OF THE GOMS MODELS

GOMSL was chosen from the various GOMS methods based on recommendations given by John and Kieras (1996b). The DMI task was modeled in GOMSL (Kieras, 2006), with the model shown in Table 2. The TUI condition's task is shown as a GOMSL model in Table 3. Each step and each method has an estimated completion time. The execution time for each GOMSL method is calculated by summing up the execution times of the primitive operators and adding 0.05 seconds for each step in the method (Kieras, 2006) that represents the mental decision to execute that step.

### [Tables 2 and 3 go about here]

The primitive operators are Keystroke Level Model operators (Card et al., 1983). For each "method for accomplishing goal" step, 50 ms are added, and the same applies for each selection rule and Decide operators, 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 from Card et al. (1983). The total time 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.

For each pointing task, we used Fitts' Law to predict the movement time. The pointing form of Fitts' Law was used as indicated by Mackenzie (1991):

$$MT = 230 + ID*166.$$

The coefficients  $\alpha$  and  $\beta$  were taken directly by Mackenzie's (1991) experiments with the mouse as a pointing device. During his experiments, the constants for

Fitts' Law to match mouse pointing movements were  $\alpha = 230$  ms and  $\beta = 166$  ms. The ID in the above equation is the index of difficulty for the task. The ID for each movement was calculated using the distances of pointer travel on the screen, using the Shannon formulation (MacKenzie, 1991), which has  $ID = \log_2(D/W + 1)$ , where D is the distance from the target, and W is the width of the target on the axis of motion. Because the distance from the initial position of the pointer to its target position changes every time, a different ID is calculated for each situation. The task has 7 distinct pointing movements:

1. Moving from the home position of the cursor to the source folder  
(ID = 3.46, MT = 0.80 s).
2. Moving from the current position of the cursor to the required file  
(ID = 3.53, MT = 0.82 s).
3. Moving from the file icon to the "Cut" menu item (ID = 2.44, MT = 0.63 s).
4. Moving from the "Cut" menu item position to the "up hierarchy" button on the window toolbar (ID = 2.96, MT = 0.72 s).
5. Moving from the "up hierarchy" button to the destination folder  
(ID = 1.75, MT = 0.8 s).
6. Moving from the destination folder position to the unused area of the folder window (ID = 1.75, MT = 0.52 s).
7. Moving from the current pointer position to the "Paste" menu item  
(ID = 1.75, MT = 0.52 s).

Table 3 shows the TUI condition task modeled in GOMSL, with three GOMS methods. 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 of Table 3, there are two peculiarities. First, in two methods, “Find Folders and Grab File Objects”, and “Place File in Destination Folder”, we use a primitive operator that is not included neither in GOMSL (Kieras, 2006) nor in the KLM (Card, Moran, & Newell, 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. It represents the prehension action (reach) that users perform when they reach and grasp a physical 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). Thus, we developed a prehension operator (Christou, Ritter, & Jacob, 2008), and used it to calculate a prediction for the prehension action.

Second, 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 supplementary experiment was performed to calculate the completion time of this method. A summary of the study and data supporting this is given in Appendix 1.

Using the results from the supplementary experiment, we calculate the completion time of the TUI task to be 8.95 s. This predicted time is 80% higher than the data and is outside the GOMS suggested range of +/-20% and nearly 2

SDs from the data mean. This is outside the limits of the average + 1 standard deviation of the experimental study. We believe that the discrepancy for the GOMS analysis is that the participants performed some actions in parallel, such as finding the source and target folders at the beginning, and placing the target object in the target folder and the rest of the file objects back in the source folder. Because GOMS does not allow modeling of parallel actions, it is not surprising that one of the most commonly used GOMS methods cannot accurately calculate the completion time of the TUI task.

### **CONSTRUCTION OF THE DMI CODEIN MODEL**

The same Fitts' Law calculations were used in the Codein model of the DMI condition, shown in Figures 5 to 8. The Codein model is summarized in Figure 5, using knowledge states to describe the sequence of actions in the task. The summary diagram is used as the top level method of the GOMS model. 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.

**[Figure 5 goes about here]**

Each knowledge state in the summary diagram can be analyzed further, and yields a diagram with its constituent knowledge and actions. Figures 6 to 8 show enlargements of each of the subtasks in Figure 5. In these figures we use the acronyms DO, IO and IN to signify Data Object, Interaction Object, and Intermediary Object respectively.

**[Figures 6, 7 & 8 go about here]**

The folders and files are recognized as Data Objects, the pointer as an Interaction object, and the mouse as an Intermediary object. In the TUI condition, there is no Intermediary object, and the hands of the participants are the Interaction objects. There is a static binding between the pointer and the mouse, knowledge without which the participants would not be able to perform the DMI action, and there are dynamic bindings between the pointer and the Data Objects every time that a Data Object is moved using the Interaction object. These facts lead to the analysis as shown in Figures 6, 7, and 8, where the diagrams include “variables”, denoted as X\_DO and Y\_DO, to show that the targets of the cut and paste mechanism may be any Data Object representation in the DMI condition. The Intermediary object, again could have been any pointing device and the diagrams would remain the same. The only thing that would change is the Fitts’ Law function constants to calculate the pointing time.

Taking figure 6 as an example of how knowledge state diagrams are constructed, we decompose the “Open source/target folder” task into two actions: moving the pointer and clicking with the mouse, yielding two knowledge states. The two arrows coming out of the state represent the action of moving the pointer until the pointer is over the required folder. This allows moving to the next knowledge state that represents double clicking with the Intermediary object on the folder to open it. Notice how the two arrows here signify the same action (move IO), but with different conditions and results.

To calculate the completion time of this subtask, we use Fitts’ Law for all the pointer moving actions and traditional KLM operator times for other actions. We

also take into account that the participants in the experiment were experts in the DMI style. This means that they would know how to use the mouse, so we only add an M operator for the retrieval of all the knowledge associated with the mouse. Thus, in the subsequent states that need mouse knowledge, we do not add retrieval from long-term memory of knowledge for the use of the mouse. Taking this into account, we find that the folder position needs to be found by visual search, thus adding another M operator. The movement time is already calculated by Fitts' Law in a previous section (0.80 s), yielding a total of 3.2 s ( $M+M+0.8$  s) for the first knowledge state in figure 6. The second knowledge state in the same figure requires only mouse knowledge, which was already recalled. The double click mouse button action is represented by operator BB = 0.4 s, totaling the completion time of the knowledge state 'Open folder' for the source folder case to 3.6 s. The rest of the knowledge states shown in figures 7 and 8 are calculated in the same way with the operators shown in the figures. The operator times in the figures are shown in the knowledge chunks and in the action descriptions. The completion time required for the whole task of figure 5 is calculated by adding up the completion times of the subtasks.

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. The completion times are as follows: Open Source Folder: 2.4 s;



Cut File: 4.25 s; Move to Folder View: 2.12 s; Open Target Folder: 2.12 s; and  
Paste File: 2.64 s. The completion time of the whole task is 13.53 s.

### **CONSTRUCTION OF THE TUI CODEIN MODEL**

The TUI condition's summary diagram is shown in Figure 9. The TUI condition task is decomposed into four subtasks: Finding the source and target folders and grabbing all the file objects from the source folder, searching for the target file object, and replacing the file objects to their respective final locations.

**[Figure 9 goes about here]**

Figure 10 is an expanded description of Figure 9. Here, the knowledge for the performance and the flow of tasks are shown. Of note are the parallel actions that occur during the task's performance, observed during the experiment. The parallel actions occur after the last knowledge state, from which two arrows emerge. In this case the two arrows have different names, suggesting that the two actions are different, and that they occur in parallel. The total time required for the TUI task as calculated is 6.2 s.

**[Figure 10 goes about here]**

### **RESULTS AND DISCUSSION**

Table 3 shows the results of each condition with the results of the two analyses. The combination of GOMS and Fitts' Law gives an accurate estimated completion time for the DMI condition. However, even when they are used together, GOMS and Fitts' Law do not provide an accurate estimation for the TUI condition. The reason is that GOMS cannot model parallel actions. One could suppose that the two actions of placing the target file in the destination folder,

and placing the rest of the files in the source folder in the 'Place File in Destination Folder' method of the TUI GOMSL model could be replaced by one action and get a completion time result closer to that of Codein. However, if one operator were used, then the model would describe a task different than the one performed. Namely, it would describe a task where the actor only performs one action to place both the stack of data objects back in the source folder and the selected item in the target folder. But the GOMSL model should show that two actions occur, not one.

Codein provides a comparable result to the experimental average of the DMI condition, close to the GOMS result. Also, for the TUI condition, Codein provides a more accurate result than GOMS.

It is a well established fact that GOMS with Fitts' Law can accurately model the performance of tasks designed in the DMI style. However, as can be seen from Table 4, Codein also predicts the completion time of the DMI condition at least as accurately as the GOMSL model. The Codein model, though, not only provides a prediction, but it also provides information about what the user needs to know to perform a task, something that is less 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 represent 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 DMI 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.

**[Table 4 goes about here]**

Table 5 shows that the greatest difference in the two models is the 'Place File in Destination Folder' procedure. The reason is that Codein considers that when two actions occur in parallel, only the action with the longest completion time will be counted towards the total completion time (as shown in the last knowledge state of Figure 10). On the contrary, GOMSL considers that each action's completion time must be calculated and added to the task's completion time. There also is a slight difference between each model's estimate in the "Find Folders and Grab File Objects" subtask.

**[Table 5 goes about here]**

As discussed earlier, GOMSL lacks the capability of modeling parallel actions, and thus cannot describe the TUI condition correctly. CPM-GOMS (Gray, John, & Atwood, 1993; John, 2003; 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. We posit that the proposed notation provides this added functionality using simpler notation than CPM-GOMS, creating a more versatile GOMS tool, especially in the evaluation of those that allow or require parallel actions.

## **CONCLUSIONS**

In this article we presented a new notation, Codein, that is based on GOMS and that extends GOMS' capabilities in two ways. The first way is that Codein provides an easy to use representation parallel actions, while in most GOMS notations this is not possible. The second is that it provides more information about the performance of a task than a GOMS model. Specifically, declarative knowledge is not included in GOMS models, but it is included in Codein.

To examine whether a model built with the proposed notation performs as well as the GOMS model, we performed a file manipulation task experiment in two different interaction styles: DMI and TUI. It was found that the predicted task completion times are comparable in the DMI condition. It was further found that the proposed notation's model was more accurate when used to predict the task completion of a task performed in the TUI condition. This was because the new notation models parallel tasks. We thus believe that the added capabilities make Codein suitable to model Reality-Based Interfaces.

The limits of Codein are that its graphical representation is more bulky, because each action of a task may be composed of one or more knowledge states, leading to large diagrams. However, we expect that users of the notation will eventually develop their own templates for commonly performed actions, thus reducing the number of diagrams required to model tasks and a completely textual representation could be easily created. Another limitation is that there is no automated tool for the calculation of completion times or for the creation of task diagrams. It is in our plans to develop such tools for this notation in the near future.

This work shows a usability approach for modeling new types of interfaces. This report is just one example of its use and one test of its prediction accuracy. Future reports will need to test a wider range of interfaces to make it more robust and to provide examples of use.

## **ACKNOWLEDGMENTS**

Portions of this research were supported by the National Science Foundation (Grant IIS-0414389). We also want to thank Junya Morita and Panayiotis Zaphiris for their valuable input and suggestions on drafts of this paper.

## REFERENCES

- Alonso-Ríos, D., Vázquez-García, A., Mosqueira-Rey, E., & Moret-Bonillo, V. (2010). Usability: A Critical Analysis and a Taxonomy. *International Journal of Human Computer Interaction*, 26(1), 53-74.
- Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89(4), 369-406
- Anderson, J. R., & Lebiere, C. (1998). *Atomic Components of Thought*. Mahwah, NJ: Erlbaum.
- Card, S. K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human Computer Interaction*. Hillsdale, NJ: Erlbaum.
- 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, pp.2165-2170
- Christou, G., Ritter, F. E., & Jacob, R. J. K. (2008). *Modeling Prehensile Actions for the Evaluation of Tangible User Interfaces*. In the *Proceedings of ITI 08 International Conference of Information Technology Interfaces*, Cavtat/Dubrovnik, Croatia, pp.415-420. IEEE XPLORE.
- Christou, G., Ritter, F. E., & Jacob, R. J. K. (2009). *Knowledge-based usability evaluation for Reality-Based Interaction*. Paper presented at the CHI 2009 Workshop on Challenges in Evaluating Usability and User Experience in Reality-Based Interaction, Boston, MA.
- Consolvo, S., Harrison, B., Smith, I., Chen, M. Y., Everitt, K., Froehlich, J., & Landay, J. A. (2007). *Conducting In Situ Evaluations for and With Ubiquitous Computing Technologies*. *International Journal of Human Computer Interaction*, 22(1-2), 103-118.
- 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.
- 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.

- Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*. 8(3), 231-274.
- 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: Erlbaum.
- 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., Girouard, A., Hirshfield, L. M., Horn, M. S., Shaer, O., Solovey, E. T., et al. (2008). *Reality-Based Interaction: A Framework for Post-WIMP Interfaces*. Paper presented at the CHI 08 Conference on Human Factors in Computing Systems, Florence, Italy, pp.201-210
- 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.
- John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). *Predictive Human Performance Modeling Made Easy*. Paper presented at the CHI 2004 Conference on Human Factors in Computing Systems, Vienna, Austria, pp.455-462
- Jones, L. A., & Lederman, S. J. (2006). *Human Hand Function*. New York, NY: Oxford University Press.
- Kelley, J. F. (1983). *An empirical methodology for writing user-friendly natural language computer applications*. Paper presented at the CHI '83 Conference on Human Factors in Computing systems, Boston, MA, pp.193-196
- Kieras, D. (1988). Towards a Practical GOMS Model Methodology for User Interface Design. In M. Helander (Ed.), *The Handbook of Human Computer Interaction* (pp. 135-158). Amsterdam: North-Holland.

- Kieras, D. (2006). A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN4. Retrieved March 1st, 2011, from [ftp://www.eecs.umich.edu/people/kieras/GOMS/GOMSL\\_Guide.pdf](ftp://www.eecs.umich.edu/people/kieras/GOMS/GOMSL_Guide.pdf)
- Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 33(1), 1-64.
- Lee, A. Virtual Dub. Retrieved 24<sup>th</sup> January 2008 from <http://www.virtualdub.org/>.
- MacDorman, K., Whalen, T., Ho, C., & Patel, H. (2011). *An Improved Usability Measure Based on Novice and Expert Performance*. *International Journal of Human - Computer Interaction*, 27(3), 280-302.
- 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.
- Newman, W. M. (1969). A System for Interactive Graphical Programming. *Proceedings of the 1968 Spring Joint Computer Conference*, 47-54.
- Shiaw, H. Y., Jacob, R. J. K., & Crane, G. R. (2004). *The 3D Vase Museum: A New Approach to Context in a Digital Library*. In *Proc. JCDL 2004 Joint Conference on Digital Libraries*, Tucson, Arizona, USA, pp.125-134
- Shneiderman, B. (1983). Direct Manipulation: A step beyond programming languages. *IEEE Computer*, 16(8). 57-69.
- Weiser, M. (1991). The computer for the 21st Century. *Scientific American*, 265, 94-104.



## **APPENDIX I - SUPPLEMENTARY EXPERIMENT**

The experiment was performed to calculate the completion time of the TUI search.

### ***Participants***

The experiment involved 10 participants different from those used in the Codein vs. GOMS experiment. Their ages were from 20 to 30 (std. dev = 2.2), 7 males and 3 females, who were asked to perform the search through the tangible file objects in the way described in the method 'Find Required File' in Table 3.

### ***Procedure***

Each participant was given a pile of 11 file objects, and was asked to find the target file object. The target object was inserted at a specific position by the experimenter, from 2<sup>nd</sup> to 10<sup>th</sup> in the pile. The first position was not used, as during the experiment described in the main article, the target object was never placed on top of the pile. Each participant performed 90 trials, 10 searches at each of the 9 possible positions of the target file object, randomly ordered.

### ***Materials and Design***

The file objects used were identical to the ones in the TUI condition of the main experiment. The time was measured in the same way, and with the same limitations, as described for the main experiment's TUI condition.

### ***Analysis***

Figure 11 shows the results of this experiment. The graph shows that the relationship between the average time taken to find the non-top, required object in the pile, against the required file object's position in the pile is linear on the position of the target object in the pile. Linear regression analysis on the data gives  $R^2 = 0.999$ ,  $F(8, 9) = 15,434$ ,  $p < 0.001$ . The average time taken for the participants to find the target object was 3.5 seconds, which we used in the GOMSL analysis of the TUI condition as well as in the Codein analysis.

**[Figure 11 goes about here]**

**Table 1 Average completion times for the experiment and models in seconds (SD)**

|                    | <b>DM</b>       | <b>Deviation</b> | <b>TUI</b>     | <b>Deviation</b> |
|--------------------|-----------------|------------------|----------------|------------------|
| Mean Task Time     | 13.47<br>(7.59) | -                | 5.00<br>(2.10) | -                |
| GOMSL + Fitts' Law | 13.26           | -0.21            | 8.95           | 3.95             |
| Codein             | 13.53           | 0.06             | 6.20           | 1.2              |

**Table 2 The GOMSL model for the DMI condition of the experiment**

---

**Top Level Method (13.26 s)**

Method for Goal: Move file from source folder to destination folder

Step 1: Accomplish Goal Open Source Folder (1.40 s)

Step 2: Accomplish Goal Cut Required File (3.45 s)

Step 3: Accomplish Goal Go back to folder list (2.37 s)

Step 4: Accomplish Goal Open Destination folder (2.70 s)

Step 5: Accomplish Goal Paste file (2.99 s)

Step 6: Return with goal accomplished

Method for Goal: Open Source Folder (Total time = 1.40 s)

Step 1: Point to Source Folder (Fitts' = 0.8 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 = 3.45 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 (Fitts' = 0.82 s)

Step 4: Click mouse button (B = 0.2 s)

Step 5: Point to "Cut". (Fitts' = 0.63 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 = 2.37 s)

Step 1: Locate "Up Folder" Button (M = 1.2 s)

Step 2: Point to "Up Folder" (Fitts' = 0.72 s)

Step 3: Click mouse button (BB = 0.2 s)

Step 4: Return with goal accomplished

Method for Goal: Open Destination Folder (Total time = 2.70 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 (Fitts' = 0.8 s)

Step 4: Double-click mouse button (BB = 0.4 s)

Step 5: Return with goal accomplished

Method for Goal: Paste file (Total time = 2.99 s)

Step 1: Point to white space (Fitts' = 0.52 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" (Fitts' = 0.52 s)

Step 5: Click mouse button (B = 0.2 s)

Step 6: Return with goal accomplished

---

**Table 3 The GOMSL model for the TUI condition of the experiment**

**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'

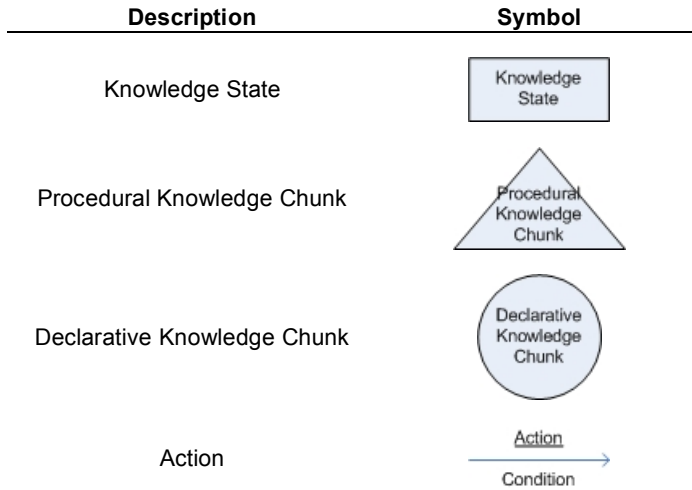
Method for Goal: Place File in Destination Folder (2.95 s)  
 Step 1: Locate destination folder (M = 1.2 s)  
 Step 2: Place file in the destination folder (R = 0.75 s)  
 Step 3: Place remaining files in source folder (R = 0.75 s)  
 Step 4: Return with goal accomplished

**Table 4 DMI condition's subtask breakdown of the analysis of each modeling method, in seconds.**

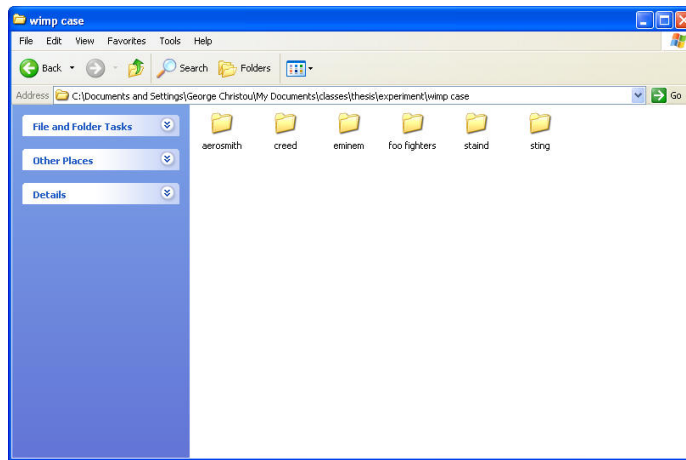
| <b>Subtask</b>             | <b>Codein Estimate</b> | <b>GOMS with Fitts' Law</b> |
|----------------------------|------------------------|-----------------------------|
| Open Source Folder         | 2.40                   | 1.40                        |
| Cut Target File            | 4.25                   | 3.45                        |
| Go Back to Folder List     | 2.12                   | 2.37                        |
| Open Destination Folder    | 2.12                   | 2.70                        |
| Paste File                 | 2.64                   | 2.99                        |
| Top Level Method Execution | -                      | 0.35                        |
| Total Completion Time      | 13.53                  | 13.26                       |

**Table 5 TUI condition's subtask breakdown of the analysis of each modeling method.**

| <b>Subtask</b>                     | <b>Codein Estimate</b> | <b>GOMS Estimate</b> |
|------------------------------------|------------------------|----------------------|
| 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              | 6.20                   | 8.95                 |



**Figure 1** The components of the diagrammatic notation



**Figure 2** The DMI condition layout of the experiment



Figure 3 The TUI Condition of the experiment

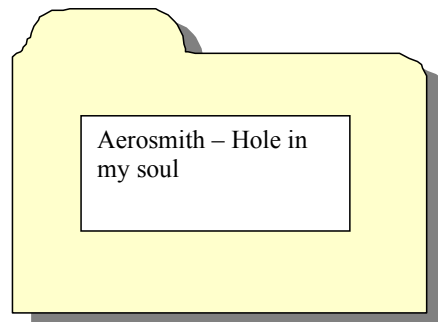


Figure 4 The regular paper cutout used in the TUI condition.

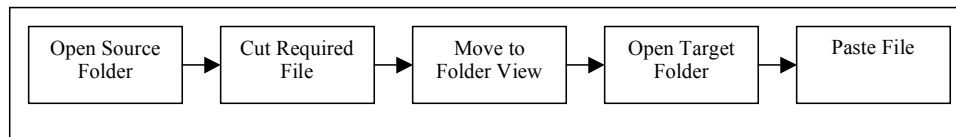
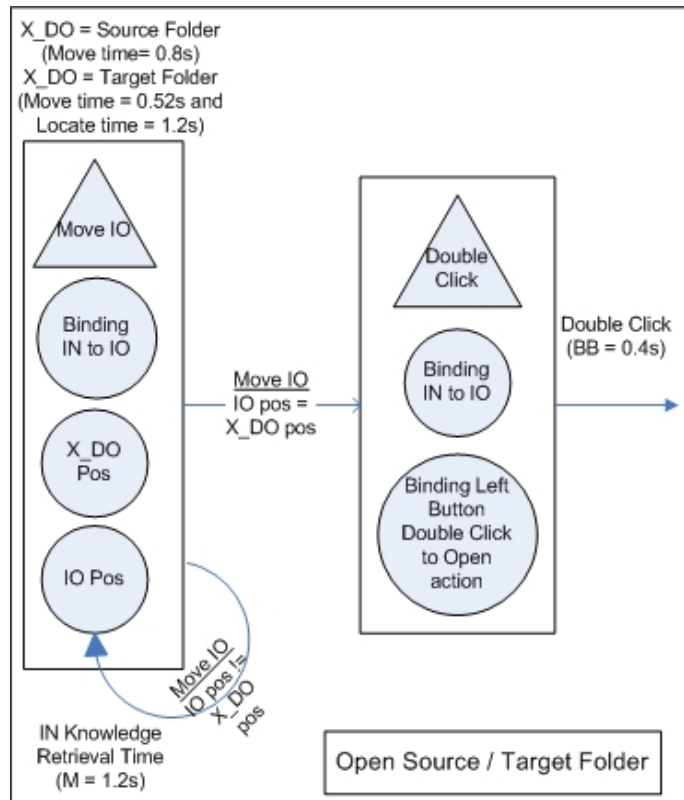


Figure 5 A summary diagram of the DMI condition of the experiment.





**Figure 6 The "Open source/target folder" knowledge state from the summary diagram of Figure 5, enlarged to show the knowledge and constituent actions for its completion**

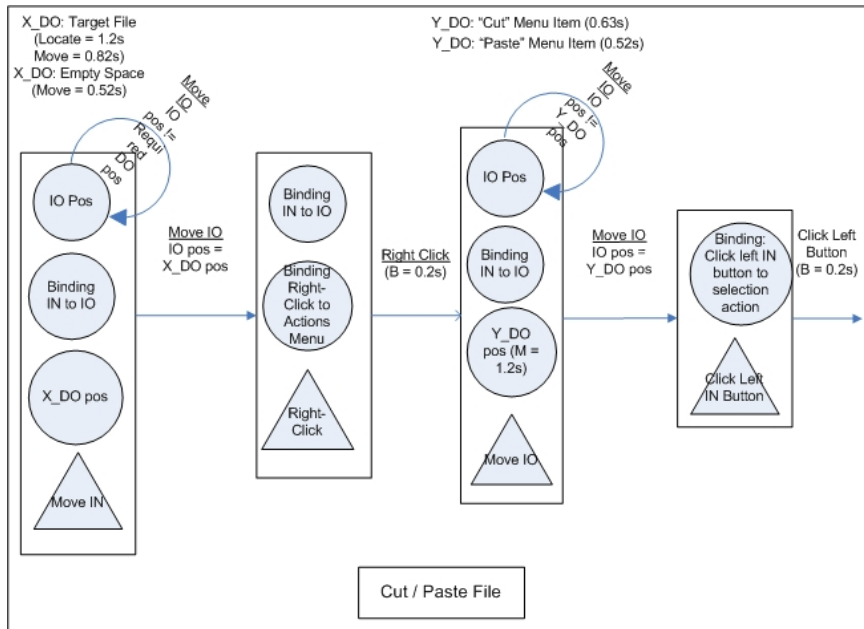


Figure 7 The analysis of the "Cut/Paste File" states in the summary diagram of Figure 5.

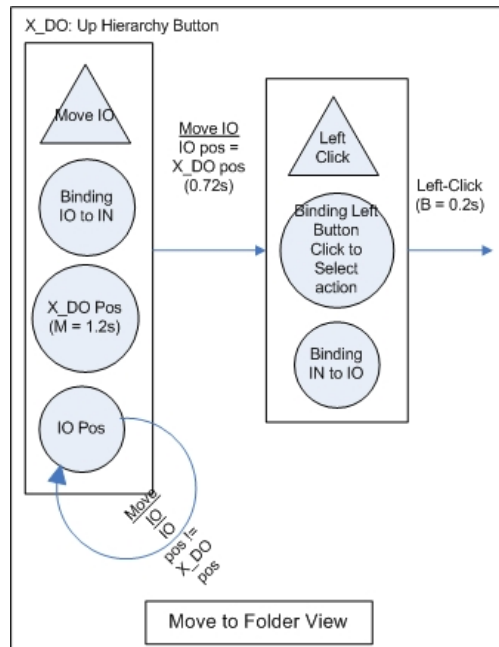


Figure 8 Analysis of the Move to Folder View state in Figure 5.

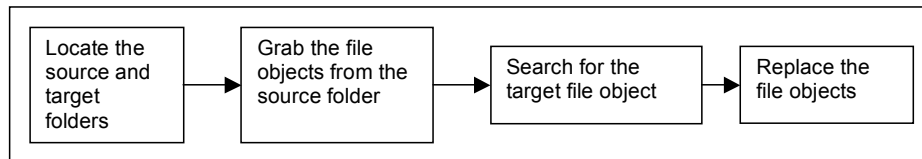


Figure 9 Summary diagram for the TUI condition.

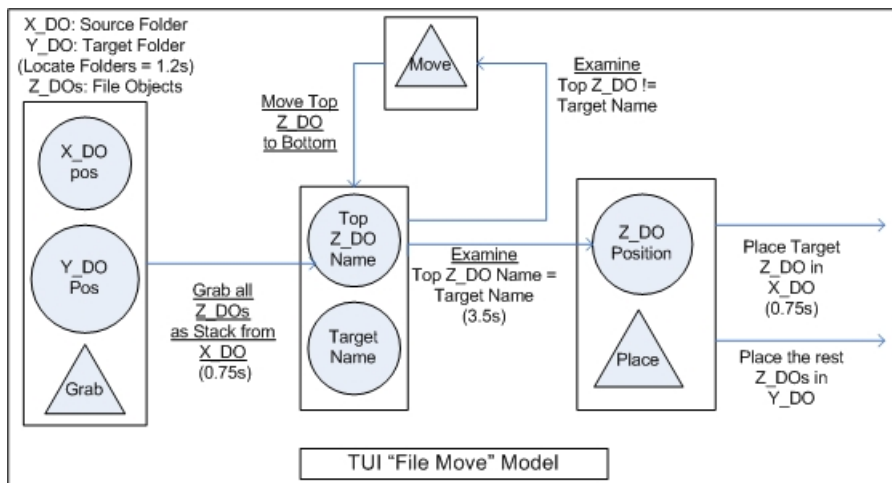


Figure 10 The Codein analysis of the task in the TUI condition

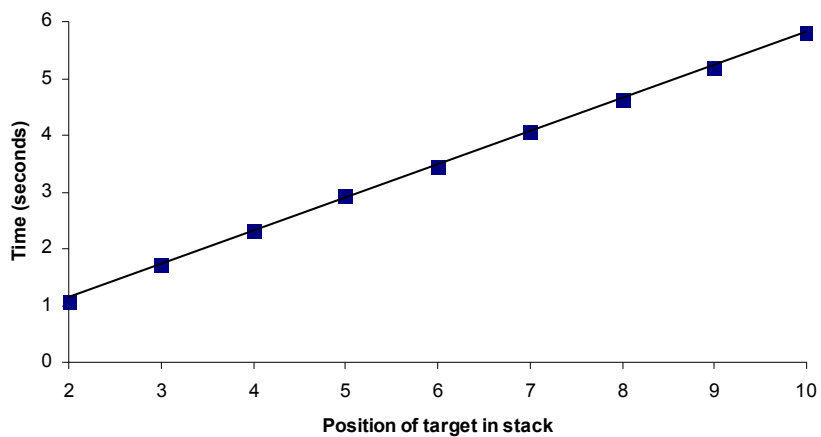


Figure 11 The average search time vs. the target object's position in the pile, with regression line.