Task Mapping Model (TMM) Analysis Manual

Kevin Mayo

TR 93-07

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

February 24, 1993

Task Mapping Model (TMM) Analysis Manual

— *A Reference* —

Version 1.6

Kevin A. Mayo
Department of Computer Science, Virginia Tech (VPI&SU)
Blacksburg, VA 24060
mayo@cs.vt.edu

I. Introduction

This guide presents the task mapping model (TMM), a synthesis/analysis methodology for aiding the interface specialist in designing interfaces with better usability. This guide does not motivate the necessity or impact of this model with regards to user interface development, that is presented elsewhere (Mayo & Hartson, 1993).

What is the Task Mapping Model (TMM)?

Briefly, TMM describes necessary knowledge for user task completion, and analyzes if this knowledge is supported by the user or interface. In order to examine user task knowledge, a model of the users' task is needed. TMM provides a framework for describing and analyzing tasks. This framework consists of various abstract levels of task decomposition and description. Each level of abstraction contains level-specific objects, operations, and sub-tasks for task description and analysis. TMM focuses on the mappings users make among domains (during task performance) to provide designers with specific information about task structure and user knowledge requirements. This model allows descriptions of tasks using hierarchical decomposition coupled with abstraction of user knowledge requirements.

How does TMM effect the user interface development process?

The overall goal of TMM is to support the user interface development process in relation to both **initial design** and **redesign** by deriving new interface design requirements. Because TMM is task oriented, analysis for initial design and redesign support are implemented in two ways: **global analysis, situational analysis**. Global analysis techniques examine whole systems—including all user tasks, input-output devices, and user interfaces. Global analysis examines all aspects of systems and is very costly in terms of time and effort. Situational analysis is involved with examination and analysis of a few specific user tasks and interface usability problems found by evaluators during formative evaluation cycles. During the initial design process both global and situational analysis can help the interface specialist focus on user- and task-centered issues. The redesign process has the benefit of formative evaluation findings, and thus, situational analysis is very cost-effective. TMM employs both analyses types for initial design and redesign during user interface development.

TMM may also be used as a **synthesis** or **analysis** method. As a synthesis method, TMM supports interface design throughout the design process by focusing interface specialists on user-centered needs. Using TMM descriptions of tasks that users had problems with, interface specialists can direct their attention on improving interface usability and human performance. Through analysis of specific situations (situational analysis) during formative evaluation, TMM synthesizes new interface design requirements.

As an analysis method, TMM helps interface specialists investigate what users need to know during task performance. With a TMM task description, user knowledge requirements can be analyzed in detail, based on task structure and relationships among domain contents. Further, TMM analysis points outs interface deficiencies and provides a synthesis of missing knowledge that should be made available to the users, and new interface design requirements are then derived to support that missing knowledge.

Where does TMM fit in the overall picture?

Human-computer interaction specialists often conceive their work two different viewpoints: the **constructional** view and the **behavioral** view. The constructional outlook is focused on implementation issues in HCI—where to put button, how to open a window, how can anyone program using X? While the behavioral view focuses on user and system behavior—how did the user do this, how did the system respond? Because of the user- and task-centered approach of TMM, it fits into the behavioral view.

Who uses TMM?

TMM can be used by interface specialists for both synthesis and analysis during human-computer interface design. Formally, TMM is designed for human-computer interface specialists; however, the simplicity and readability of the model's descriptions allow more general use, including 'walkthrough'-style evaluations by non-specialists.

The following sections provide a general introduction to TMM, and outline the domain structures, knowledge elements, and task description formats used in TMM. A section on TMM usage outlines associated methodologies, and finally, an example is provided to illustrate the model's use. This guide also has a glossary defining commonly used terms.

II. Overview of TMM Task Descriptions

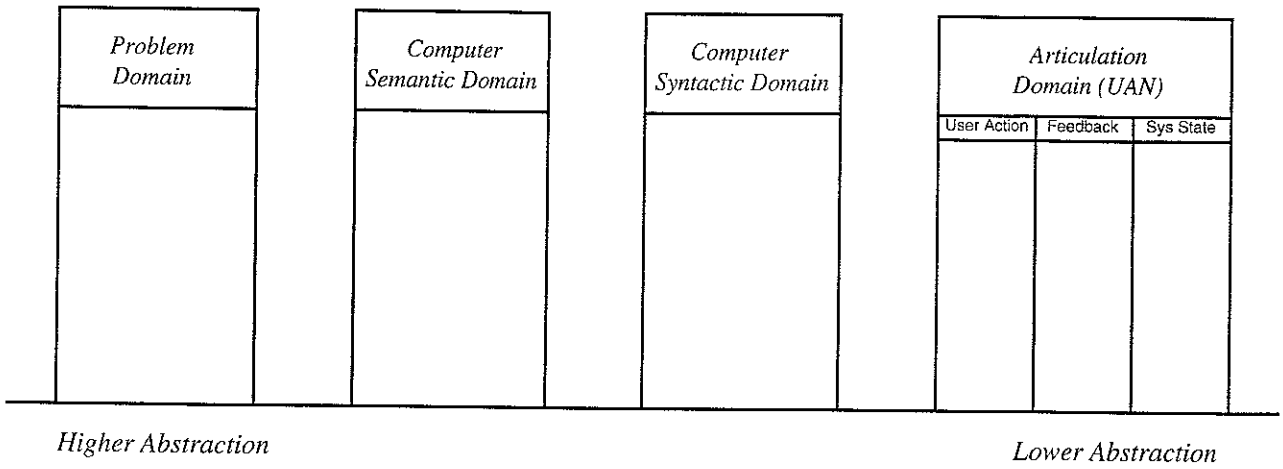
This section gives a general overview of describing tasks with TMM.

Domains

The abstract levels of TMM task descriptions are referred to as **domains**. A domain contains entities specific to the abstraction level. These entities are collectively referred to as **domain items**. A domain item is either an **object**, **operations**, or **sub-tasks**. Domain objects are manifestations of physical or conceptual *things* related to the task. Operations are actions defined within the domain's level of abstraction. Sub-tasks represent subordinate execution sequences. All three components are (examples are given in the following section).

Domains range from 'higher' to 'lower' levels of task abstraction. The domains reflect different levels of abstraction at which users perceive task artifacts (domain items). (Shneiderman, 1979; Shneiderman, 1987; Moran, 1981; Nielsen, 1986) Simply put, a user works at several levels when performing a task on a computer—from the highly abstract problem domain down to the low abstraction level of physical manipulation.

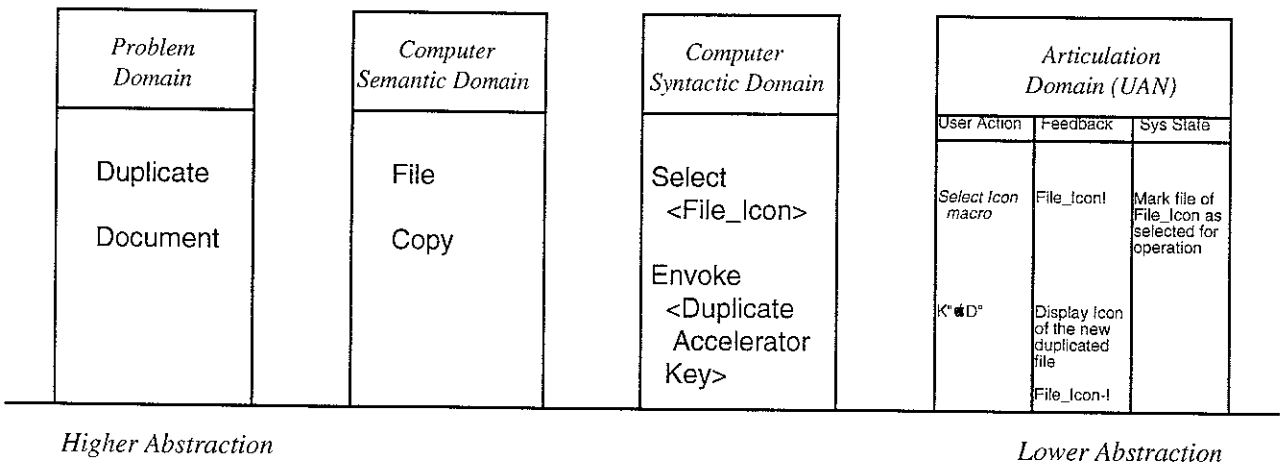
The domains used in TMM task descriptions are: **problem domain**, **computer semantic domain**, **computer syntactic domain**, and **articulation domain**. These domains are discussed in more detail, with examples, in the following section. The following figure depicts the domains of task abstraction.



Pictorial Representation of Relationship among Domain Levels

Ideally, users wish to work in the problem domain, where all domain items are problem related; however, once a task is computerized, users must understand the translations between their understanding of the task and the computers representation of the task.

For example, when a user attempts the task of `DUPLICATE A DOCUMENT` (problem domain task), the user must reconceptualize it into `COPY A FILE` (computer semantic domain). Many of these tasks can be viewed as action-object pairs that exist within different domains, and are both representations of the task at hand. Also, `COPY A FILE` can be reconceptualized further into the 'computer' grammatical components (computer syntactic domain items representing the 'copy command' and the actual filename) and input sequence (articulation domain specification of users' actions).



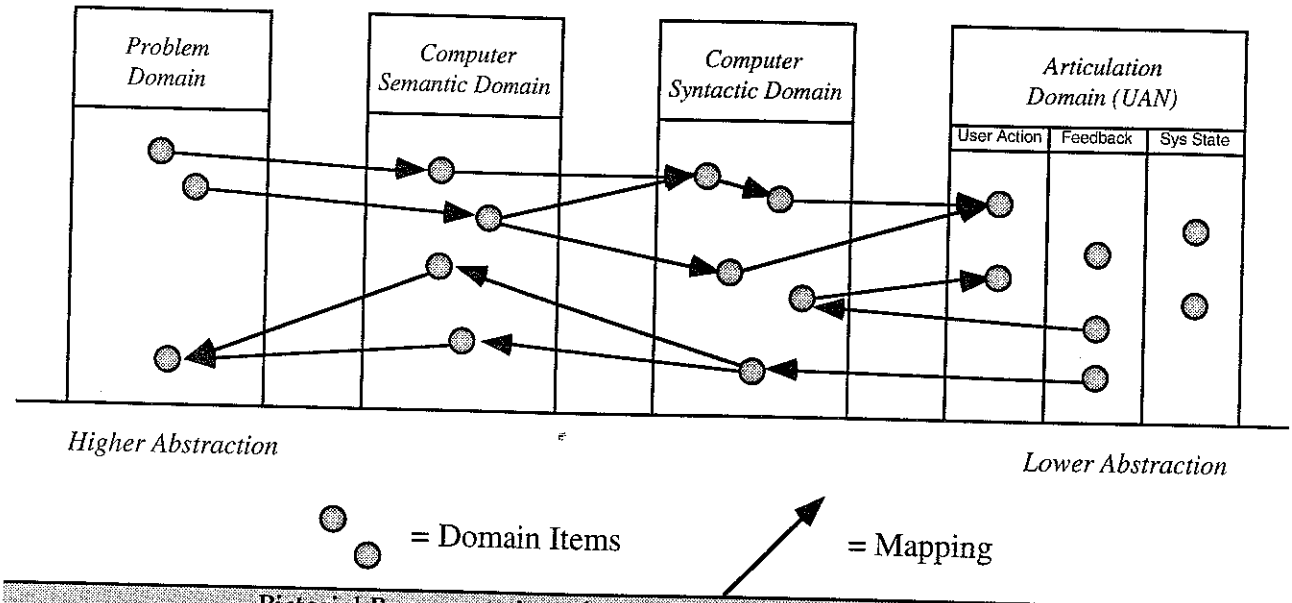
Example Task (`DUPLICATE A DOCUMENT`) within Domain Levels

Mappings

Using these domains to represent the different abstract levels that users perceive tasks, mappings are formed to relate items across domain boundaries. A **mapping** refers to a specific reconceptualization or translation of a domain item from one domain to another. Mappings can exist from higher to lower or lower to higher abstract domains levels. The sequence of mappings through these domains are task **paths**, and may be either **execution** or **evaluation**. Mappings from higher to lower levels of abstraction represent execution paths. Here the user maps conceptual task goals into actual physical actions. The converse, examining physical results to determine is a goal has been achieved, is an evaluation path—users map from lower to higher levels of abstraction.

In the previous example, `DUPLICATE A DOCUMENT` was reconceptualized (translated) to `COPY A FILE`. During task performance the user maps `DUPLICATE` to `COPY` and `DOCUMENT` to `FILE`. TMM notation uses an arrow, ⇒, to indicate a 'maps to' relationship, e.g., `DUPLICATE` ⇒ `COPY` and `DOCUMENT` ⇒ `FILE`. These are two of the necessary mappings for task translation in this example.

In general, mappings link domain items across domain boundaries. The following figure depicts the relationships of mappings and domain items among domains:



Pictorial Representation of Mappings within Domain Structure

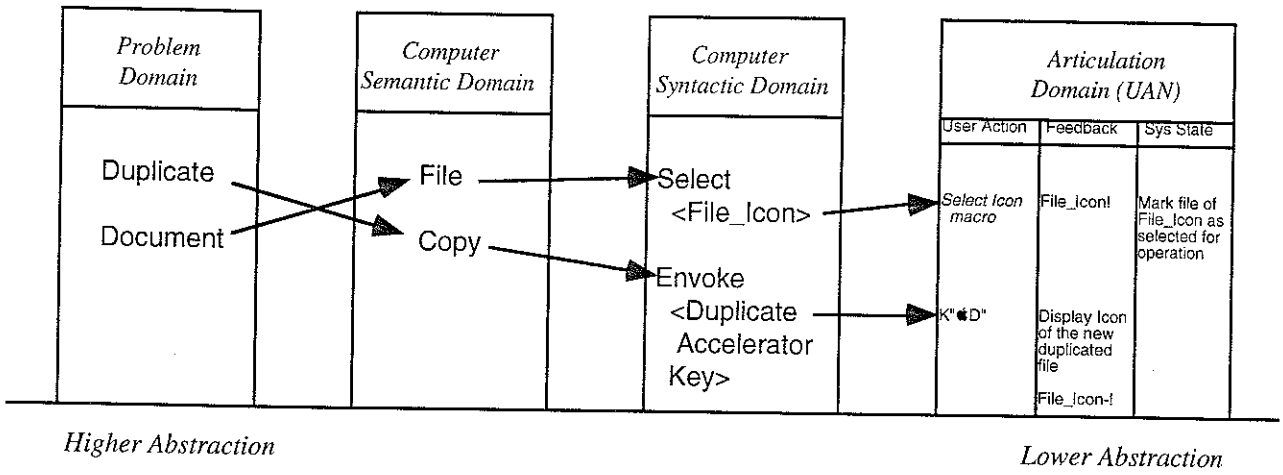
This example shows mappings among different domains. Mappings within domain boundaries are further decompositions of domain items; while, mappings among items from different domains boundaries represent changes in abstraction levels.

This example illustrates a 'one-to-many' mapping from the single problem domain item to two computer semantic domain items. This could represent a possible alternative method for performing the same task, or these mappings can indicate a second possible computer semantic representation of the problem domain item. 'One-to-many' mappings can occur between any two domains.

'Many-to-one' mappings are also possible. Our example shows two computer semantic domain items mapping to a single computer syntactic domain item. This could represent a collapsing of the task into a common set of grammatical components or actions. For example, consider error conditions where once an error is identified a sequence of actions common to all errors must be taken for error resolution. Observe the errors associated with reading-from or writing-to a file—

once the error is determined the user must check the media and then remount it. 'Many-to-one' mappings can occur between any two domains.

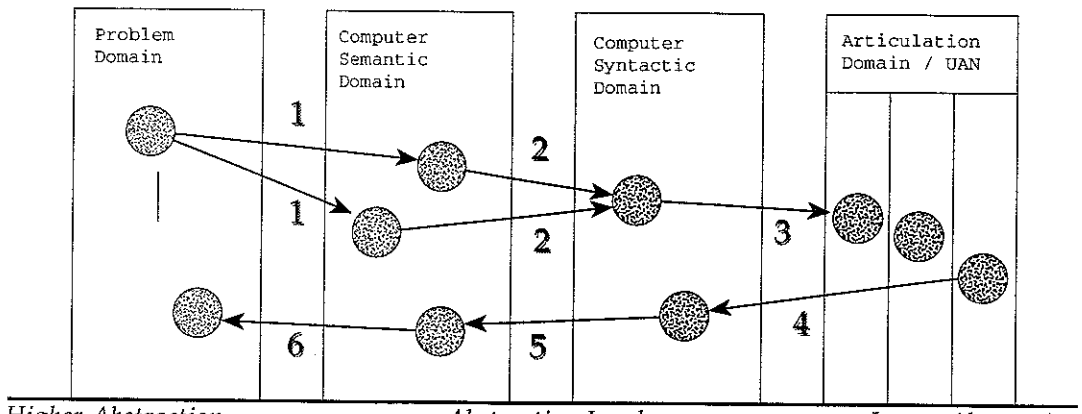
With mappings, our previous example task of DUPLICATE A DOCUMENT might look like:



Example Task (DUPLICATE A DOCUMENT) with Mappings

In this example, DUPLICATE ⇒ COPY ⇒ actual system 'Copy' command (e.g., 'cp', 'Duplicate' menu item) ⇒ users' physical actions. This series of mappings illustrates the user's reconceptualization of a problem domain task down into physical user actions.

Mapping one domain item into another item in a different domain is not altogether straightforward. For instance, what is involved when the user performs DOCUMENT ⇒ FILE? The user requires certain knowledge to perform this mapping. Therefore, sets of necessary knowledge elements are associated with each mapping, representing the knowledge requirements for the mapping. Also, a mapping may have alternative knowledge sets. The following figure depicts this concept with example generic mappings:



Key

Domain items are represented as circles, and mappings are arrows.

Knowledge requirements for the execution path
(higher to lower abstraction level mappings)

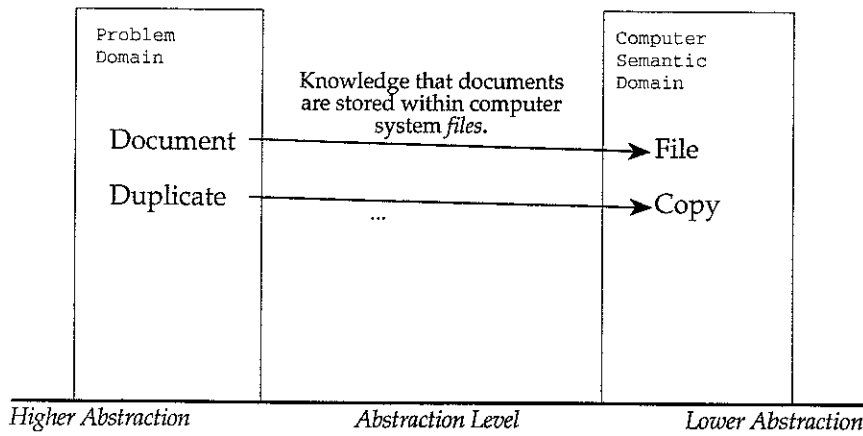
1: Knowledge required to map a problem domain item into a computer semantic domain item.
2: Knowledge required to map a computer semantic domain item into a computer syntactic domain item.
3: Knowledge required to map a computer syntactic domain item into an articulation domain item.

Knowledge requirements for the evaluation path
(lower to higher abstraction level mappings)

4: Knowledge required to map an articulation domain item (system feedback) into a computer syntactic domain item.
5: Knowledge required to map a computer syntactic domain item into a computer semantic domain item.
6: Knowledge required to map a computer semantic domain item into a problem domain item.

Pictorial Representation of Mappings with Knowledge Requirements

In the previous figure notice that mappings connect domain items across domain boundaries. The knowledge necessary for each mapping is 'external' to TMM's domains. In other words, domains contain domain items, while knowledge requirements are *generally* defined outside or between the domains. In the DUPLICATE A DOCUMENT example, consider the mapping DOCUMENT ⇒ FILE, one possible knowledge requirement could be the factual knowledge that *documents are stored within files.*, as shown in the following figure:



Example Task (DUPLICATE A DOCUMENT) with Mappings and Knowledge

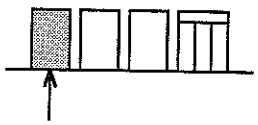
This example illustrates that users must understand that documents are stored in system files in order to map document to file. This may seem trivial, but a user without this understanding is doomed.

This section has presented a brief overview of TMM. The following sections depict the domains, knowledge elements, and task description issues.

III. TMM Task Description Domain Framework

This section further defines the domains introduced in the previous section: problem domain, computer semantic domain, computer syntactic domain, and articulation domain.

III.1. Problem Domain



The problem domain contains items (objects, operations, and sub-tasks) directly related to user task performance described in specific real-world terminology. Thus, the scope of this domain is defined by the users' task or job. Domain items are the physical and conceptual real-world entities that comprise the users' task, and are often not related to computer systems. The problem domain does not contain anything defined outside this scope.

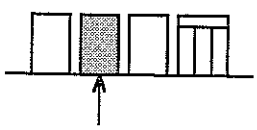
Example problem domain items

Problem Domain	Objects	Operations	Sub-Tasks
DOCUMENT PREP.	• LETTER • DOCUMENT	• EDIT • READ	• MOVE LINE 1 TO 5 • EDIT DOCUMENT • DISCARD LETTER • CHECK SPELLING
PERSONAL FINANCES	• CREDIT CARDS • DRAFT ACCT. • SAVINGS ACCT. • DEPOSIT SLIP • BALANCE	• DEPOSIT • WITHDRAW • TRANSFER	• TRANSFER $20 TO SAVINGS • BALANCE CHECKBOOK

This table shows two different problem domains with representative domain items. Obviously, this is not a complete table—e.g., documents can also be printed, reformatted, etc. This table does illustrate that all domain items are problem defined, and there is no hint of possible computer implementations or solutions. The objects defined are within the problem domain—a letter is a physical world 'letter' and not a sequence of ones and zeros in a computer somewhere.

As can be seen, there are no items concerned with computer hardware or software in this domain. This level of abstraction requires that only the problem domain and tasks within the problem domain need to be understood.

III.2. Computer Semantic Domain



The computer semantic domain contains items (objects, operations, and sub-tasks) that build a representation of the users' tasks with abstract computer concepts. This domain is a layer of abstraction between the problem domain task representation and the specific computer syntactic domain task representation. I.e., this domain serves as an generic computer "middle-ground" in the users' translation of tasks between the real world and a specific computer system.

Example computer semantic domain items

Objects	Operations	Sub-Tasks
• FILE • DISK • KEYBOARD • WINDOWS	• MOVE • DELETE • RENAME	• MOVE A DATA FILE FROM APPLICATION FOLDER TO DATA DISK • INSERT A DISK

Items within this domain are strictly computer related, but their definition is not limited to any particular hardware or software platform. Items in this domain do not include the actual interface features, but this domain does include the generalized concepts represented by these features. For example, FILEs are common to most systems but their physical representations among these systems vary considerably.

The TMM has chosen the taxonomy reported by Lenorovitz as the standard language within the domains. (Lenorovitz, Phillips, Ardrey, & Kloster, 1984) By adopting this taxonomy as standard, the TMM helps eliminate ambiguities from designers' natural language task descriptions—while not relying on constrictive grammars. This also gives a common language by which designers and analysts can communicate.

The taxonomy is divided into four sub-taxonomies of the User-System Interface (USI): *computer-output, computer-internal, user-input,* and *user-internal taxonomies.* The user-input and user-internal taxonomies are of interest to TMM—these form the basic language used in TMM task descriptions. This language is used throughout the problem-, computer-semantic-, and computer-syntactic domains. These two taxonomies are outlined hierarchically:

PERCEIVE	ACQUIRE	DETECT SEARCH SCAN EXTRACT CROSS-REFERENCE
	IDENTIFY	DISCRIMINATE RECOGNIZE
MEDIATE	ANALYZE	CATEGORIZE CALCULATE ITEMIZE TABULATE
	SYNTHESIZE	ESTIMATE INTERPOLATE TRANSLATE INTEGRATE FORMULATE PROJECT OR EXTRAPOLATE
	ASSESS	COMPARE EVALUATE
	DECIDE	
COMMUNICATE	TRANSMIT	CALL ACKNOWLEDGE RESPOND SUGGEST DIRECT INFORM INSTRUCT REQUEST
	RECEIVE	

User-Internal Taxonomy

CREATE	ASSOCIATE	NAME GROUP
	INTRODUCE	INSERT
	ASSEMBLE	AGGREGATE OVERLAY
	REPLICATE	COPY INSTANCE
INDICATE	SELECT *(POS/OBJ)*	
	REFERENCE	
ELIMINATE	REMOVE	CUT DELETE
	STOP	SUSPEND TERMINATE
	DISASSOCIATE	RENAME UN-GROUP
	DISASSEMBLE	SEGREGATE FILTER SUPPRESS SET-ASIDE
MANIPULATE	TRANSFORM	
ACTIVATE	EXECUTE __ *FNT*	

User-Input Taxonomy

Taxonomies of user Actions (Lenorovitz, et al., 1984)

The definitions are given in the following table (still showing the hierarchy):

PERCEIVE	ACQUIRE	DETECT	Discover or notice an occurrence (usually unsolicited)
		SEARCH	Purposeful exploration or looking for specified item(s).
		SCAN	Glance over quickly, usually looking for overall patterns or anomalous occurrences (not details).
		EXTRACT	Directed, attentive reading, observing, or listening with the purpose of gleaning the meaning or contents thereof.
		CROSS-REFERENCE	Accessing or looking up related information usually by means of an indexing or organized structuring scheme set up for that purpose.
	IDENTIFY	DISCRIMINATE	Roughly classify or differentiate an entity in terms of a gross level grouping or set membership—frequently on the basis of only a limited number of attributes.
		RECOGNIZE	Specific, positive identification of an entity.
MEDIATE	ANALYZE	CATEGORIZE	Classify or sort one or more entities into specific sets or groupings, usually on the basis of a well-defined classification scheme.
		CALCULATE	Reckon, mentally compute, or computationally determine.
		ITEMIZE	List or specify the various components of a grouping.
		TABULATE	Tally or enumerate the frequencies or values of the members of an itemized list or table.
	SYNTHESIZE	ESTIMATE	Mentally gauge, judge, or approximate, often on the basis of incomplete data.
		INTERPOLATE	Assign an approximate value to an interim point based upon knowledge of values of two or more bracketing reference points.
		TRANSLATE	Convert or change from one form or representational system to another according to some consistent *mapping* scheme.
		INTEGRATE	Pull together, and mentally organize a variety of data elements so as to extract the information contained therein.
		FORMULATE	Generate and put together a set of ideas so as to produce an integrated concept or plan.
		PROJECT OR EXTRAPOLATE	Assign an approximate value to a future point based upon the value(s) of preceding point(s).
	ASSESS	COMPARE	Consider two or more entities in parallel so as to note relative similarities and differences.
		EVALUATE	Determine the value, amount, or worth of an entity, often on the basis of a standard rating scale or metric.
	DECIDE		Arrive at an answer, choice, or conclusion.
COMMUNICATE	TRANSMIT	CALL	Signal to a specific recipient or set of recipients that a message is forthcoming.
		ACKNOWLEDGE	Confirm that a call or message has been received.
		RESPOND	Answer or reply in reaction to an input.
		SUGGEST	Offer for consideration.
		DIRECT	Provide explicitly authoritative instructions.
		INFORM	Pass on or relay new knowledge or data.
		INSTRUCT	Teach, educate, or provide remedial data.
		REQUEST	Solicit, query, or ask for.
	RECEIVE		Get, obtain, or acquire an incoming message.

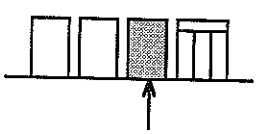
User-Internal Taxonomy

Definitions for Lenorovitz Taxonomies (Lenorovitz, et al., 1984)

CREATE	ASSOCIATE	NAME	Give title to or attach label to for purposes of identification/reference.
		GROUP	Link together or associate for purposes of identification.
	INTRODUCE	INSERT	Make space for and place an entity at a selected location within the bounds of another such that the latter wholly encompasses the former, and the former becomes an integral component of the latter.
	ASSEMBLE	AGGREGATE	Combine two or more components so as to form a new composite entity.
		OVERLAY	Superimpose one entity on top of another so as to affect a composite appearance while still retaining the separability of each component layer.
	REPLICATE	COPY	Reproduce one or more duplicated of an entity (no links to *master*).
		INSTANCE	Reproduce an original (*master*) entity in such a way as to retain a definitional link to the master—i.e., such that any subsequent changes or modifications made to the master will automatically be reflected in each and every *instance* created therefrom.
INDICATE	SELECT *(POS/OBJ)*		Opt for or choose an entity (e.g., a position or an object) by *pointing* to it [and possibly other user actions].
	REFERENCE		Opt for or choose an entity by invoking its name.
ELIMINATE	REMOVE	CUT	Remove a designated portion of an entity and place it in a special purpose buffer (residual components of the original entity usually close in around *hole* left by *cut-out* portion).
		DELETE	Remove and (irrevocably) destroy a designated portion of an entity.
	STOP	SUSPEND	Stop a process and temporarily hold in abeyance for future restoration.
		TERMINATE	Conclude a process such that it cannot be restarted from the point of interruption, only by complete re-initiation.
	DISASSOCIATE	RENAME	Change an entity's title or label, without changing the entity itself.
		UN-GROUP	Eliminate the common bond or reference linkage of a group of entities.
	DISASSEMBLE	SEGREGATE	Partition and separate an entity into one or more component parts such that the structure and identity of the original is lost.
		FILTER	Selectively eliminate one or more layers of an overlayed composite.
		SUPPRESS	Conceal or keep back certain aspects or products of a process without affecting the process itself (i.e., affects appearance only).
		SET-ASIDE	Remove entire contents of current (active) work area and store in a readily accessible buffer (for future recall).
MANIPULATE	TRANSFORM		Manipulate or change one or more of an entity's attributes (e.g., color, line type, character font, size, orientation) without changing the essential content of the entity itself.
ACTIVATE	EXECUTE __ *FNT*		Initiate or activate any of a set of predefined utility or special purpose functions (e.g., sort, merge, calculate, update, extract, search, replace).

User-Input Taxonomy
Definitions for Lenorovitz Taxonomies (Lenorovitz, et al., 1984)

III.3. Computer Syntactic Domain



Computer syntactic domain items (objects, operations, and sub-tasks) represent actual syntactical components related to both computer semantic and articulation domain items. These items represent user interface software entities (e.g., specific filenames, interface objects) and actions (e.g., user physical actions, interface commands) employed during task performance with particular computer or software packages.

This domain outlines the syntax—the components of articulation and their order. In other words, the computer semantic domain describes ***what*** *to express* to the computer, whereas the computer syntactic domain describes ***how*** *to express* to the computer. Thus, items in the computer syntactic domain are *somewhat* specific to general interaction styles and techniques.

Many components are not fully specified at this level of task description because of ambiguities in users' actions. (E.g., the interface specialist may not know the exact filename or command.) Therefore, variables may be used at this level of task description. Domain items with brackets, < >, are variables in TMM task descriptions that represent actual system equivalents. For example, <FILENAME> would reduce to the actual system filename like 'exp.dat', and <FILE ICON> is the system-displayed icon for a particular file used within the task. With variables a task description still maintains adequate expressive power, while the 'noise' of variable qualification is left to user actions.

It is important to understand that computer syntactic domain items represent syntactical items for articulation—not the manipulation or articulation of the syntactical items. By the vary nature of command line interfaces (CLI), the computer syntactic domain items will fully specify the commands; however, in direct manipulation (DM) interfaces only the components of the commands are specified. To illustrate the differences between a CLI and DM style interfaces, consider the task ERASE FILE:

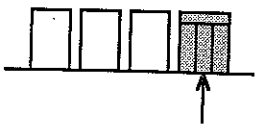
Example computer semantic items

Computer Semantic Items	Interaction Style	Computer Syntactic Items
• ERASE • FILE	*COMMAND LINE INTERFACE*	
	• UNIX	• INVOKE 'RM' • SELECT <FILENAME>
	• MS-DOS	• INVOKE 'DEL' • SELECT <FILENAME>
	DIRECT MANIPULATION	
	• MACINTOSH-OS	• SELECT <FILE ICON> • MOVE <FILE ICON> TO <TRASHCAN ICON> • DE-SELECT <FILE ICON>
	• MICROSOFT WINDOWS	• SELECT <FILE ICON> • INVOKE <DELETE> MENU ITEM FROM <FILE> PULLDOWN MENU

These example syntactic domain items are, in effect, a meta-language for the articulation. The items specify the articulation in terms of actual commands and arguments (albeit through variables), but they do not specify the articulation. For instance, the UNIX-CLI syntactic items are INVOKE 'RM' and SELECT <FILENAME>, but the articulation of these items is not specified (in fact, it may be either by keyboard, voice, or some other form). The syntax for the syntactical domain items is fully defined and described by the interface specialist.

Syntactical representations described within this domain map into the articulation domain. The specification for the articulation needs only be described once. For example, the tasks ERASE FILE and COPY FILE share a common computer syntactic domain item: SELECT <FILENAME>. The articulation for this syntactic domain item needs only be described once, and then referenced in the future. These representations of syntactical items can be kept in a dictionary of articulation descriptions.

III.4. Articulation Domain



The articulation domain contains the specification of the users' physical interaction while communicating computer syntactic items to and from the interface. This domain models the users' actions with the syntax of the interaction defined within the computer syntactic domain. Because of its ability to specify users' actions and system feedback, the User Action Notation (UAN) (Hartson, Siochi, & Hix, 1990) is used within this domain.

The articulation domain contains both system feedback to users and user inputs to the system. The system's interpretation of user inputs is a purely constructional issue, and therefore not important to this research. However, the users' ability to understand system feedback is a behavioral issue very important within TMM.

Users must understand system feedback because it usually affects subsequent actions. Hence, evaluation paths in TMM task descriptions mostly originate from system feedback. TMM helps determine the usefulness of the feedback by identifying ambiguous, misleading, or nonexistent feedback based upon the evaluation paths and users' knowledge requirements. For example, the feedback `FILE ID=0092KAM REMOVED` could be understood better as `LETTER TO JOE SCHMALTZ DISCARDED`. In the first case the feedback is in cryptic computer-syntactic domain terms; while in the second case, feedback is provided in problem domain specific terms. The TMM task description for this task identifies several mappings users must perform to relate `ID=0092KAM` back to `LETTER TO JOE SCHMALTZ`. The second feedback example (problem-domain-specific) removes many of the file name mappings identified by TMM, and hence makes the feedback easier for the user to understand. Of course, feedback can not always be represented within the problem domain—a `DISK FULL` error is a good example.

The articulation level can also help in identifying interface characteristics that could inhibit task performance for certain classes of users. For example, how would a seeing person deal with a Braille screen, or a speech impaired individual with a voice activated system? Examining user class profiles in conjunction with the articulation domain could identify usability problems, this is discussed in the section TMM View of User Classes.

III.5. Domain Skipping

TMM task descriptions capture the flow of user actions in terms of tasks and sub-tasks, as well as the necessary knowledge for these actions. It is possible, given the TMM domain framework, that users can 'skip' a domain. Skipping a domain implies that the user does not have or want an understanding of the task within the domain. For example, it is possible for a user to have memorized a keystroke sequence for a particular task without an understanding of the underlying commands that are being issued. TMM analyses supports this style of translation; however, a user with a *fragile task understanding* represents usability problems waiting to happen.

IV. TMM's Description of User Knowledge Requirements

Each mapping is associated with a set of knowledge requirements. This set of knowledge requirements represents the necessary information that users must possess to perform that

mapping. Knowledge takes several forms within TMM task descriptions: **factual knowledge**, **conceptual knowledge**, and **procedural knowledge** (much like *schemas*—see (Evans, 1988)). Knowledge from these categories may be necessary for successful task completion. It is not the intention of this research to define mental models defining what users believe, how they store/retrieve information, or the relationship structure among user knowledge. These categories serve to classify for knowledge, but not the ability to model knowledge manipulation or relationships.

IV.1. Factual Knowledge

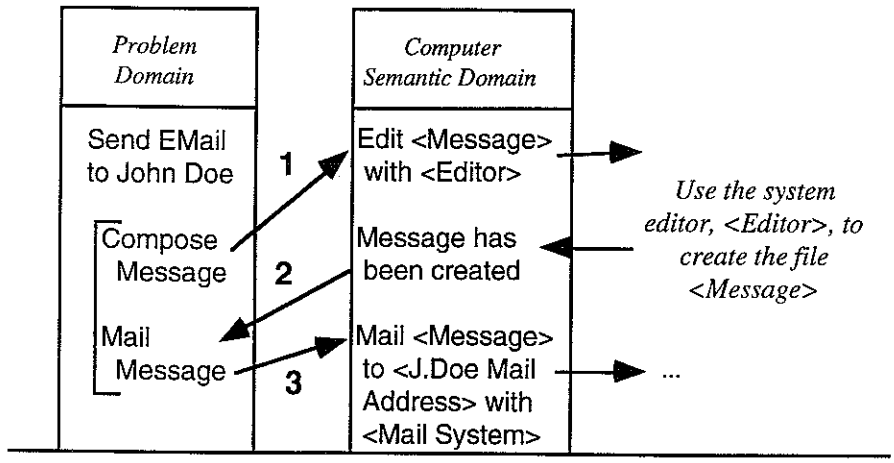
Definition: **Factual knowledge** is comprised of single or collections of declarative facts users need for mapping items from one domain to another.

E.g.

- problem-computer semantic mapping that requires the factual knowledge: `FAMILY TREE INFORMATION IS STORED IN A DATABASE.`
- computer semantic-computer syntactic mapping that requires the factual knowledge: `COLUMN DELETION COMMAND IS 'COL-DEL'.`
- computer syntactic-articulation mapping that requires the factual knowledge: `KEYBOARD IS QWERTY STYLE.`

The first and most straightforward category of knowledge is factual knowledge. Factual knowledge is information necessary to the user. The information can be a single statement or a series of statements taken as a whole. Both `FILES ARE STORED ON DISK` and `THE FIRST COLUMN IS MONTHLY SALES` are examples of factual knowledge elements. These examples help to illustrate that factual knowledge is necessary for task completion, and also that the requirement for specific factual knowledge can be associated with a mapping between any domain levels.

The notation that TMM uses for factual knowledge is `FK: <STATEMENT>`. The knowledge, `<STATEMENT>`, is a natural language representation of a single fact. As an example of a mapping requirement with corresponding factual knowledge, consider the task: `SEND EMAIL TO JOHN DOE`:



Key:

1: FK: Messages are contained within files
FK: Files are changed though editing
2: FK: Message is prepared
3: FK: All EMail must have an address
FK: John Doe has an EMail address
FK: EMail is sent through mail system

Example of Factual Knowledge

In this superficial example the user must know about system files, edit and mail systems, and user email addressing. These represent a few of the necessary factual knowledge elements for this task. This example also shows factual knowledge used in both execution and evaluation paths.

IV.2. Conceptual Knowledge

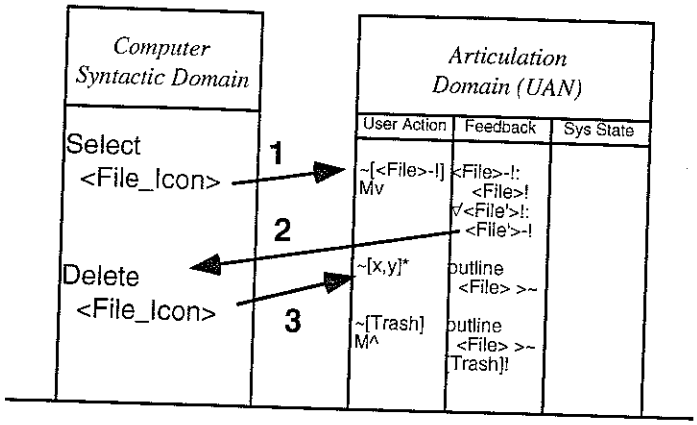
Definition: **Conceptual knowledge** is the understanding of relationships between collections of factual knowledge, ideas, and other conceptual knowledge.

E.g.

- problem-computer semantic mapping that requires conceptual knowledge about BIRDS (including: general characteristics, color, shape, wings, etc.)
- computer semantic-computer syntactic mapping that requires conceptual knowledge about FILES (including: general characteristics, commands, contents, etc.)
- computer syntactic-articulation mapping that requires conceptual knowledge about VOICE ACTIVATED SYSTEMS (including: general characteristics, commands, initialization, etc.)

The second category of knowledge is conceptual knowledge. A concept is defined to be a collection of ideas, characteristics, and/or other concepts. For example, a user has a conceptual understanding of FILES, and therefore, the user may understand operations and general characteristics of FILES but*not necessarily* the underlying system representation and operations.

Conceptual knowledge can also be used when the definition of the users' factual knowledge is too cumbersome. For example, it would be very difficult to generate the entire list of facts associated with FILES in a TMM task description, so conceptual knowledge can be employed as a shorthand notation. It also is used to indicate *general* knowledge of a topic without reference to specifics. If a user is known to possess given conceptual knowledge, then sometimes possession of certain factual knowledge can be deduced or inferred. Like factual knowledge, conceptual knowledge can be used throughout a TMM task description.



Key:

1: CK: Direct manipulation interface styles
CK: Desktop metaphor
FK: Noun-Verb interaction
2: FK: File associated with <File_Icon> is selected
3: CK: Deleting in desktop metaphor DM interface
(E.g., FK: Files in the <Trash> are deleted)
PK: Movement is performed by dragging

Example of Conceptual Knowledge

The previous figure depicts an example of using a conceptual knowledge element within a mapping. Here the task is to delete a file from a Macintosh-like interface. This involves moving the file icon to a deletion (trashcan) icon. The conceptual knowledge element represents knowledge of direct manipulation interfaces. This knowledge encompasses movement, selection, and the direct manipulation paradigm of object-action pairs.

IV.3. Procedural Knowledge

Definition: **Procedural knowledge** represents possible courses of action, user goals, task ordering and structure, needed during task performance.

E.g.

- problem-computer semantic mapping that requires procedural knowledge: `TASK: LOCATE DATABASE ASSOCIATED WITH QUARTERLY REPORT.`
- computer semantic-computer syntactic mapping that requires procedural knowledge: `TASK: PERFORM VISUAL SEARCH FOR PRINT COMMAND.`
- computer syntactic-articulation mapping that requires procedural knowledge: `TASK: MOVE POINTER/CURSOR TO RIGHT-HAND SIDE OF BOX OUTLINE (SCREEN POSITION: 655, 231).`

The final knowledge category is procedural knowledge. Procedural knowledge represents possible courses of action, user goals, task ordering and structure, needed during task performance. In simpler terms, this knowledge represents 'where-am-I', 'how-did-I-get-here' and 'what-do-I-do-next' task knowledge. There are two different types of procedural knowledge: knowledge of possible task paths, and knowledge to correctly choose a path.

The first type of procedural knowledge outlines the possible paths users may take. Much of this knowledge is embedded into the task decomposition. In particular, users must be aware of tasks that have multiple orderings in time. For example, task A requires both task B and C to be completed (in either order)—there are two paths for task A: B then C, or C then B . This type of procedural knowledge identifies timing relationships among tasks—which defines possible task paths.

Knowledge about task paths also includes methods that help the users' mapping process among task abstraction domains. Consider the problem of locating an interface artifact, such as a menu item, the associated procedural knowledge identifies the need for information about searching menus. Said another way, procedural knowledge also describes auxiliary tasks and information that users employ to assist their own task mapping needs.

The second type of procedural knowledge is concerned with conditional task execution and evaluation. Simply put, a task may conditionally execute. A user should know and understand the condition that controls task execution. For example, if the condition is `RECORD RECENT DATA ONLY` for the task `STORE DATA`, then the user must be aware of the condition and be able to evaluate it.

Tasks can have several outcomes, and users faced with evaluating the results. Procedural knowledge includes the conditions that control the task outcomes. For example, the task `FIND CORRECT FILE` has both a positive (`CORRECT FILE FOUND`) and negative (`FILE NOT FOUND`) outcomes. Users must know the correct path *based on the condition* to follow (i.e., where-do-I-go-next). Procedural knowledge for this task includes both outcomes and the paths the users take. Conditions associated with the outcomes are specified using natural language, and can indicate either an execution or evaluation paths.

Users can maintain multiple methods for the same task. However, this modeling does not attempt to place conditionals on equally likely method selection based on user class profiles.

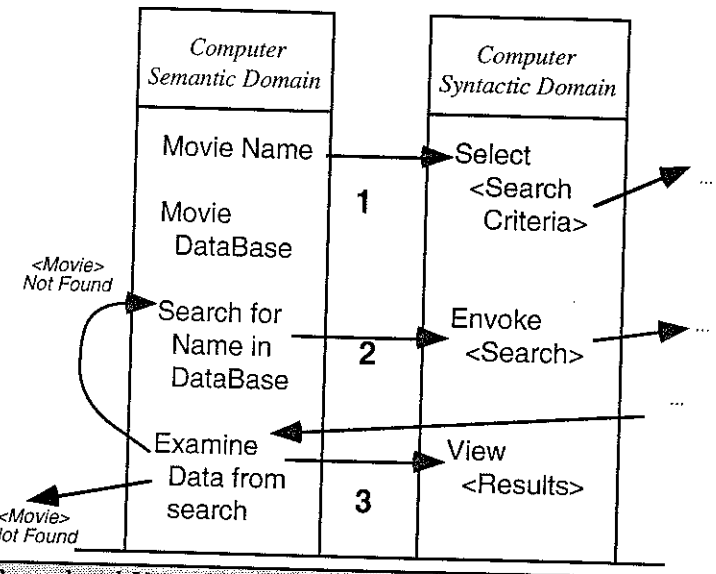
This is because TMM does not model mental processes but only identifies knowledge necessary for task accomplishment.

As an example, consider the problem domain task of LOCATE A KNOWN MOVIE IN MOVIE STORE COMPUTER, this maps into the computer semantic domain items: MOVIE NAME and SEARCH. (For this task, it is assumed the user knows the movie name.)

Key:

1: FK: The movie name, <Movie>, is <Criteria> for search
2: FK: <Search> is the command to retrieve data from database
3: PK: <Results> are examined by user visual search

Also,
PK: Structure of task: Locate a known movie in video store computer
PK: Where to go after task: Examine Data from search



Example of Procedural Knowledge

In this task, the user must specify the criteria (the movie name) and invoke the search. Factual knowledge represents the actual command names, but procedural knowledge describes the necessary task structure. Further, once output is generated from the search, the user must examine the data to check for accuracy in the search. Either the movie data was found or it wasn't (notice the condition within the task description). This is another example of procedural knowledge. The user must know to verify the output, and also know the next action base on the verification.

V. TMM Task Ordering Issues

Correct and accurate task description also involves modeling the ordering constraints among tasks. Tasks may be ordered in many different ways. In command line interfaces a sequential task ordering is standard; however, with direct manipulation windowing systems and the increase of desk-top computer power, users are now challenged with performing several tasks at once, possibly sequentially, concurrently, or interleaved. User interface designs have taken these new interaction possibilities to users, and in many cases, seriously changed the complexity of user tasks.

If a user may perform a given task in several different ways, then TMM should capture that ability in the task descriptions. TMM provides several notations to model timing relationships.

TMM task descriptions use brackets to delineate tasks and sub-tasks into related aggregates. The timing relationships defined in this section can be applied to either single tasks or aggregates of tasks grouped within brackets.

Sequential tasks

If tasks are serially ordered in time, then they are **sequential tasks**. This timing relationship among tasks is very common. For example, consider using your car's computer to SET CRUISE CONTROL SPEED TO 55 MPH. The sub-tasks are (1) GET CAR TO 55 MPH, (2) SET CRUISE CONTROL SPEED. These tasks are ordered in time. Setting the cruise control before the correct speed is attained leads to a task execution error. TMM task description notation for this example is:

```
[ Task: Get Car to 55 mph.
  Task: Set Cruise Control Speed
```

Example of Sequential Tasks

Conditional task performance

Often a task is performed based on a condition that the user evaluates. For example, consider the task EXAMINE DATASET FOR ERRORS. The sub-tasks could include: EVALUATE DATASET, and CORRECT DATASET. However, the execution of correct dataset is *dependent on whether an error exists in the dataset*—a user evaluated conditional. The TMM notation for this uses 'IF <condition>' as in:

```
Task: Examine dataset for errors
[ Task: Evaluate Dataset
    [ IF an error exists in the dataset
      Task: Correct Dataset
```

Example of Conditional Task Performance

Conditionals can also be associated with task mappings—an alternative notation for conditional task performance. These notations allow designers to build up conditional execution sequences (using nesting) based on user evaluation of conditionals.

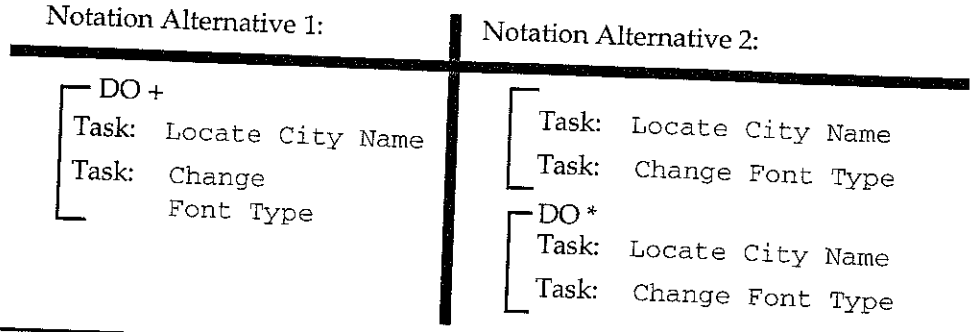
Repeated tasks

If a task is to be performed several times, then it is defined to be a **repeating task**. For example, if the task is CHANGE FONT TYPE FOR CITY NAMES which are dispersed throughout a document, this could be defined as the grouped tasks LOCATE CITY NAME and CHANGE FONT TYPE repeated zero or more times. The TMM notation for this uses 'DO *' as in:

```
[ DO *
  Task: Locate City Name
  Task: Change Font Type
```

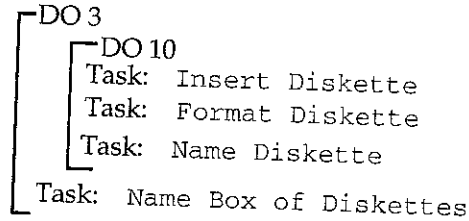
Example of Performing a Task Zero or More Times

The '*' represents zero or more occurrences of the tasks within the brackets. Sometimes it is known that the task(s) will be executed at least once, and in this case, the '+' notation represents one or more executions of the tasks within the brackets. The following example provides the same task timing relationships but with two different notations; one uses the 'DO +' operator, while the other uses the 'DO *' operator.



Example of Performing a Task One or More Times

Of course it is also possible that a task is repeated a specific number of times. In this case, a numeric count, or a range of counts, is applied to the bracket,. For example, imagine three new boxes of diskettes and the task `FORMAT DISKETTES`. There are ten diskettes to a box, so the task is described as:



Example of Performing Specific Repetitions of Tasks

This example also shows how task groupings can be nested.

Conditional looping of tasks

The previous notation is used when the number of repetitions is known, however, in many cases the number of repetitions is controlled by a condition. Conditional task repetition can be represented by two different task description structures: a **test-before** loop and a **test-after** loop (much like computer programming looping constructs). A test-before looping construct describes tasks that are performed only if a condition is evaluated first, whereas, a test-after looping construct describes tasks that are performed first and then the condition is evaluated. These task description structures differ from the previous notations in that the repetition is conditionally controlled.

A test-before loop structure that TMM uses in task description is '`DO-WHILE`'. Natural language[1] statements represent the conditions associated with the looping structure. The conditions are evaluated by users during task performance, and the user determines if the tasks are to be executed again. For example, if the task is to update files the description could be:

[1] While natural language (English, or other human-spoken language) is used to specify conditions, an imposed syntax would be helpful. This syntax is left to the interface specialist.

DO WHILE more <File_Icon> to Update
Task: `Locate next file, <File_Icon>`
Task: `Update <File_Icon>`

Example of Test-Before Looping of Tasks

The test-after loop description structure used by TMM is 'DO-UNTIL'. Like the test-before loop condition, the test-after loop condition is specified in natural language which the user should be able to evaluate. One difference between the test-before loop and the test-after loop is that the tasks will be performed at least once in a test-after loop. Our previous example could be restated with a test-after loop, if and only if, there is an assumption that there is always at least one <File_Icon> to update:

DO
Task: `Locate next file, <File_Icon>`
Task: `Update <File_Icon>`
UNTIL no more <File_Icon> to Update

Example of Test-After Looping of Tasks

Notice in both of these examples, the test-before and test-after looping constructs, that the conditions are evaluated by the user. The user determines if the loop is executed or re-executed. This implies that the user must know the condition and how to evaluate it. These conditions are included in procedural knowledge.

Order independent tasks

The previous examples all involve sequential tasks, but many higher-level tasks provide time independent sub-tasks where the selection order does not matter. In this case, the task is composed of disjoint sub-tasks—non-concurrent tasks that are not ordered in time.

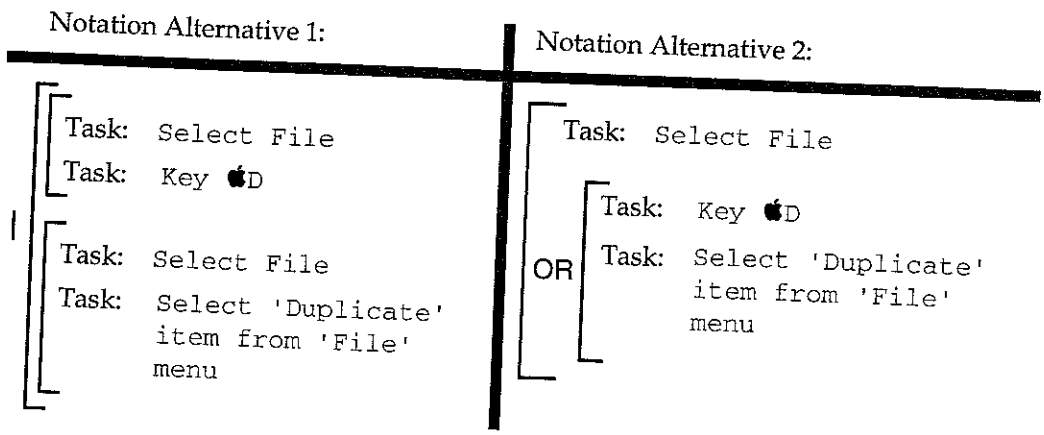
For example, if the task were `SET VCR CLOCK TO WEDNESDAY 7:21PM`, then the sub-tasks represent setting the hour, minute, and weekday. However, their execution order is not important—the user may set the weekday before setting the hour, or whatever order is desired. If the order of sub-task execution is not important (or left to user discretion) then the '&' operator is used:

&
Task: `Set VCR Weekday to Wednesday`
Task: `Set VCR Hour to 7 pm`
Task: `Set VCR Minute to 21`

Example of Non-Sequential, Non-Concurrent Tasks

Alternative method selection

Many higher-level tasks may be performed by any one of several different methods. Each method should be captured within a TMM's task description. TMM's notation for **alternative method selection** is '|' or 'OR'. As an example, consider the task `DUPLICATE FILE` in the Macintosh operating system. One of two duplication alternatives can be used, either the '⌘D' short-cut or the 'DUPLICATE'FILE-menu item is selected:

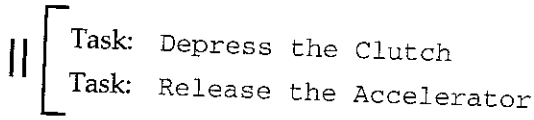


Example of Alternative Method Selection

In this example, it can be seen that the user must first select the file for duplication. After this selection, however, the user may duplicate the file in one of two ways, either through the 'fast-key command access' or through the 'File' menu. This example shows both methods, but it also demonstrates how the common prefix task `SELECT FILE` can be factored out with task grouping for readability and simplicity.

Concurrent tasks

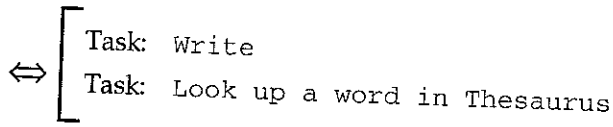
There can also be **concurrent tasks**—performance of tasks is perceived to happen at the physical same time. These can be much more difficult to describe notationally, but they too must be captured and modeled. For example, consider the automotive task `SHIFT FROM 3RD TO 4TH GEAR`. To perform this task the user must first depress the clutch while releasing the accelerator concurrently. These two sub-tasks are marked as concurrent using the '||' operator:



Example of Concurrent Tasks

Interleaved tasks

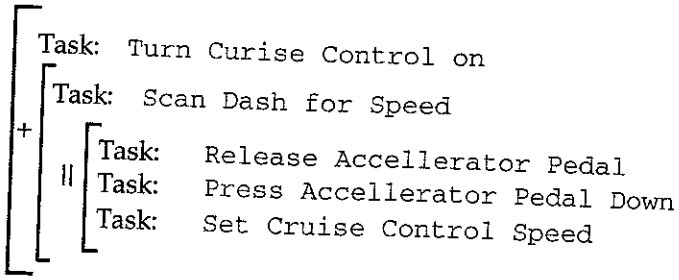
The final timing relationship involves **interleaved tasks**. In this case, a task might be suspended and then resumed after some intervening actions or tasks. Like concurrency, this is also very difficult to represent within a task description. Consider the tasks of writing and using a thesaurus—at any moment the task of writing could be suspended to look up a word, and then resumed. The notation for interleaved tasks is '⇔' as in the example:



Example of Interleaved Tasks

Combinations of timing operators

Of course, combinations and nesting of timing operators over task groups is possible. Let's return to the example task `SET CRUISE CONTROL SPEED TO 55 MPH`. Consider the following description:



Example of a Combination of Timing Relationships

This example has a double nested task timing relationship. The first action is to turn cruise control on. Then, at least once, the speed is assessed and the proper action is chosen among three alternatives. That example shows how timing relationships can be combined in a task described using TMM notation.

VI. TMM View of User Classes

The previous discussions assume that a single user (or single *class* of users) performs a given task. However, this is not always the case; there may be several distinct user classes for a given system or task. These various user classes can have different abilities and characteristics which are collectively referred to as **user class profiles**. For example, an interface may need to accommodate both novice and frequent users, yet the user class profiles are different which implies different approaches to the tasks and interfaces.

VI.1. Dealing with Different User Classes

A designer faced with a user population consisting of several distinct user classes must create a design that addresses the needs of all user classes. TMM incorporates support for different user classes into a design in several ways:

- switch among several interfaces depending on the user's class,
- provide an intelligent interface that anticipates needs and addresses them, or
- create a single composite interface for all the user classes.

The first method, switch among several interfaces, is the simplest to design. In this case, several interfaces are created. TMM analysis is performed for each interface for each salient user task, and a specific user class profile is coupled with the corresponding interface for the analyses. Also, an analysis of the user task of switching from one interface to another may be important.

The second method, an intelligent interface, is far more complicated, and a complete discussion is beyond the scope of this guide Basically, a meta-theory of user knowledge based on the different user classes is necessary for the interface to anticipate the users' class and actual knowledge requirements.

The third and final method, a composite interface that provides support for all the various user classes, involves a TMM analysis of the interface with an intersection of all user class profiles. The intersection gives the analysts and designers a description of the "lowest-common users'" knowledge.

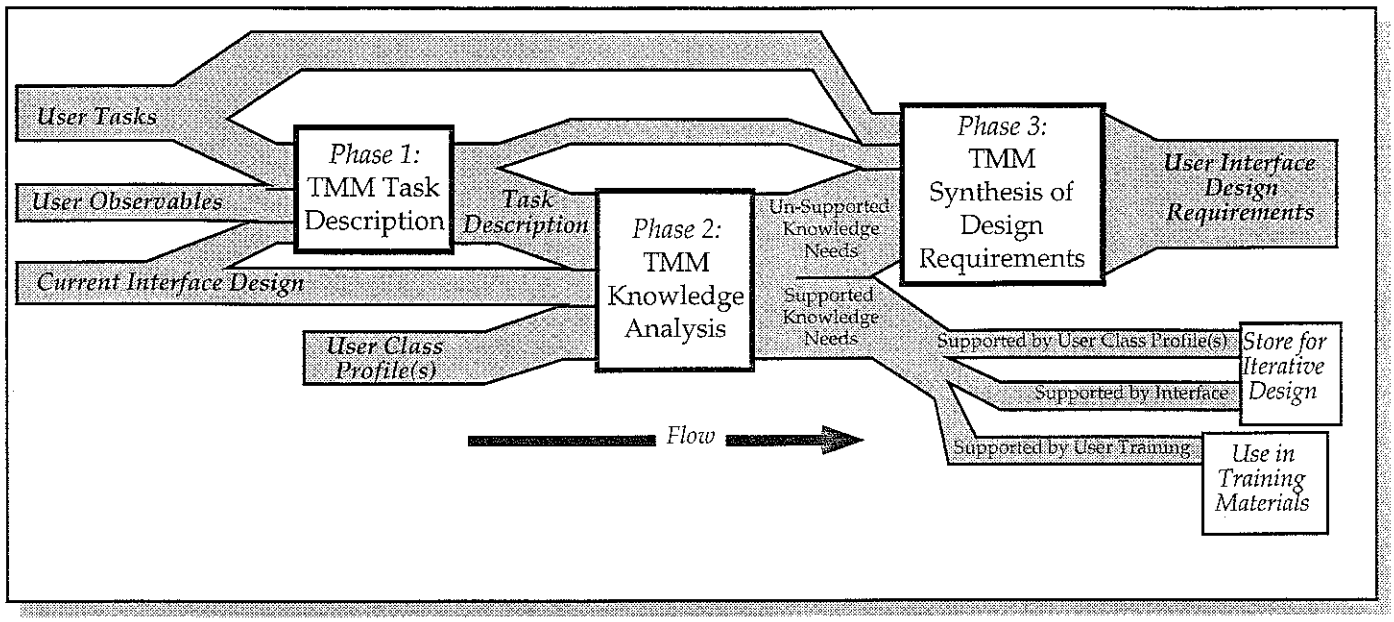
VI.2. User Class Profiles

Currently, this research does not provide a mechanism for generating user class profiles. However, research in knowledge elicitation in both cognitive psychology and artificial intelligence provides methods that can be used. There are several different methods that can be employed in generating the user class profile, they include: *experimental methods to infer cognitive processes, interviewing/self-reporting techniques, repertory grids,,* and *verbal protocol.* A critique of these methods can be found in (Evans, 1988).

Once a method is selected to gather user class profile information, it is important that this information be organized to aid TMM analyses. Mappings are analyzed by examining items being mapped and the domains in which they reside. Because of this, it is beneficial to isolate user profile information into groups identified by TMM domain-pairs (e.g., problem/computer semantic domain pair). This grouping will reduce the analysts time searching the profile definition.

VII. Using TMM in User Interface Design

The basic TMM analysis life cycle has three stages. The first stage generates TMM descriptions for user problematic tasks identified within formative evaluation. These candidate tasks are broken down hierarchically into sub-tasks. The task description is based on the task, sub-tasks, current interface (if any), and user observables (both user requirements and task performance). In stage two, the descriptions are analyzed with the user class profile(s) to produce a list of unsupported-knowledge elements. And in the final stage, a list of new user interface design requirements are synthesized from the unsupported-knowledge elements. If the interface is redesigned then this process can be repeated. The following figure represents the TMM process life cycle:



TMM Analysis Life Cycle

This section describes methods used within the TMM life cycle. Also, this section discusses metrics collectable over TMM descriptions and analyses.

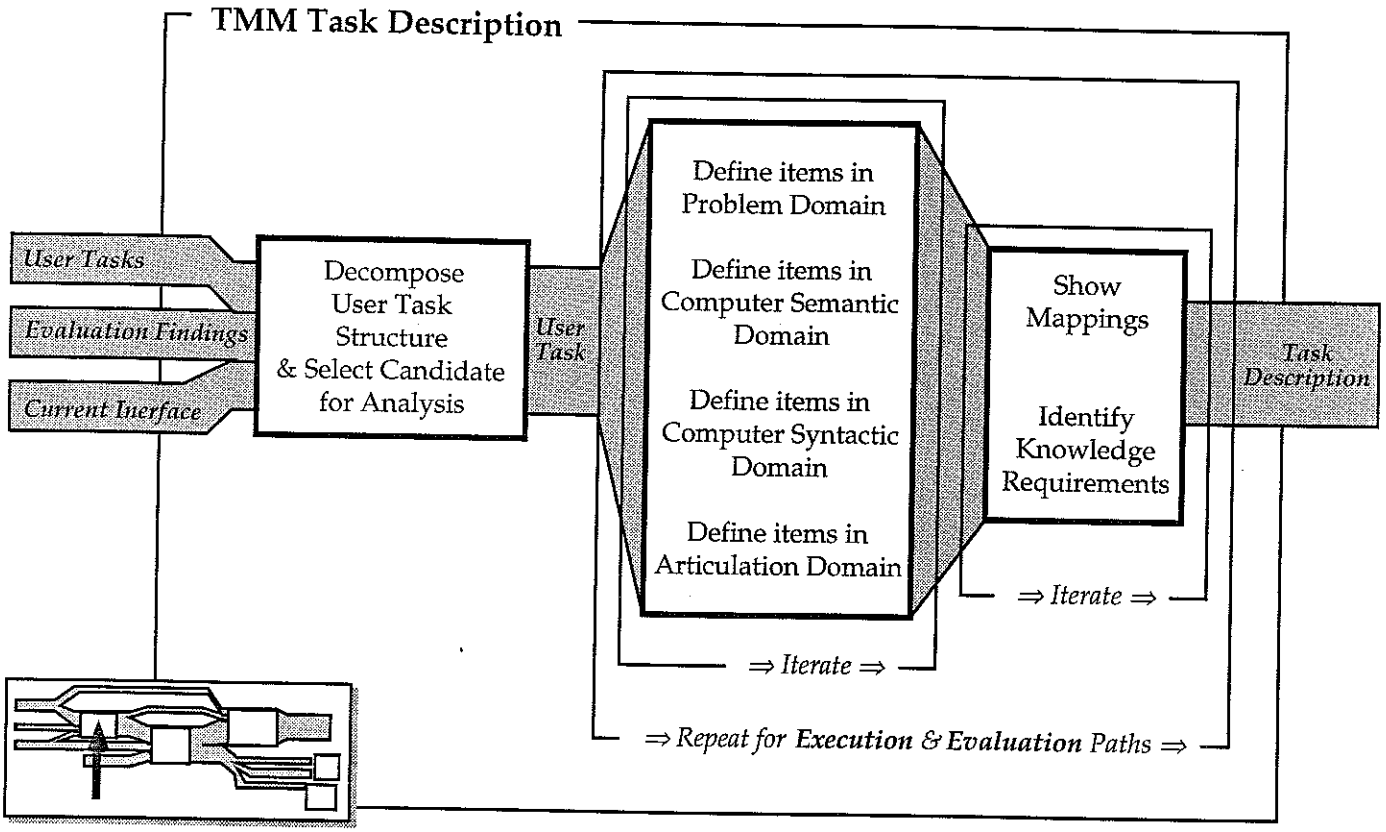
VII.1. Generating TMM Task Descriptions (Methodology)

The first, and most important, step in performing any analysis with TMM is generating a task description. Task descriptions can be generated for tasks with or without existing computer interfaces—both of which are related to the design processes (initial design or redesign). It is wise to examine, through description and analysis, the most common user tasks and highly complex user tasks. Usability of an interface can be affected most when these classes of tasks are examined.

When generating a task description for a task that is not yet automated (a computer interface does not exist), it is important for analysts to have a good understanding of the problem domain. The analyst must examine the problem domain and user requirements, gleaning the most important (common and complex) tasks for description and analysis. After an initial design, further analysis, prototyping, and usability evaluation can drive the redesign process.

Generating task descriptions for already existing interfaces is more straightforward. Qualitative and quantitative formative evaluation results give analysts basic structure of user tasks. From this analysts extrapolate domain items and mappings in order to create full task descriptions.

Following is a general outline for creating task descriptions using the **TMM Task Description Form** (see Analysis Templates section). This does not represent a hard-and-fast dictum, but only one possible approach to generating TMM task descriptions.



First Stage in TMM Analysis Life-Cycle

Generating Task Descriptions using TMM

Select the task

1. Perform task decomposition to generate suitable candidates for task description.
2. Select candidate for description.
3. With a new **TMM Task Description Form**—decompose task within problem domain (if necessary) and for each of the sub-tasks perform steps 4through 10.

Define the domain items

4. For the problem domain items define all related computer semantic domain items. Decompose tasks within computer semantic domain if necessary. *Don't be concerned with their relationships at this point—just define all necessary items.*
5. For the derived computer semantic domain items (step 4) define all related computer syntactic domain items. Decompose tasks within computer syntactic domain if necessary. *Don't be concerned with their relationships at this point—just define all necessary items.*
6. For the derived computer syntactic domain items (step 5) define input sequences and user articulatory actions within the articulation domain. *Use UAN (or any other articulation notation) in the articulation domain.*
7. Iterate steps 4 through 6 until a comfortable degree of certainty is attained as to the completion of the domain item identifications.

Identify the mappings

8. For each domain item create **mappings** (arcs) to the corresponding adjacent domain items.
9. For each mapping list all the **knowledge requirements.**
10. Iterate steps 8 through 9 until a comfortable degree of certainty is attained as to the completion of the translation and mapping identifications.

Repeat process for the evaluation paths

11. Each system feedback creates an evaluation path. Perform the same process in 4-7 **in reverse to capture the users' needs** for each evaluation path. And then, perform the same process in 8-10 for each evaluation path.

Steps 8-11 are critical for the success of TMM analysis. These steps identify the mappings and translations among domain items. A finer task description granularity produces a better identification and analysis of user knowledge requirements.

Step 10 begs the question: What is 'A COMFORTABLE DEGREE OF CERTAINTY'? Here the analysts must examine the description's mappings and knowledge requirements determining if the they adequately capture of the underlying user needs. The analyst gathers task mappings and knowledge requirements by brainstorming, asking colleagues, and observing user performance. Analysts' experience will guide this process, failing that, observing users is the best method for capturing user needs. As with other task analysis methods, the TMM analyst is warned against analyzing arcane task methods—methods that *very few* users employ.

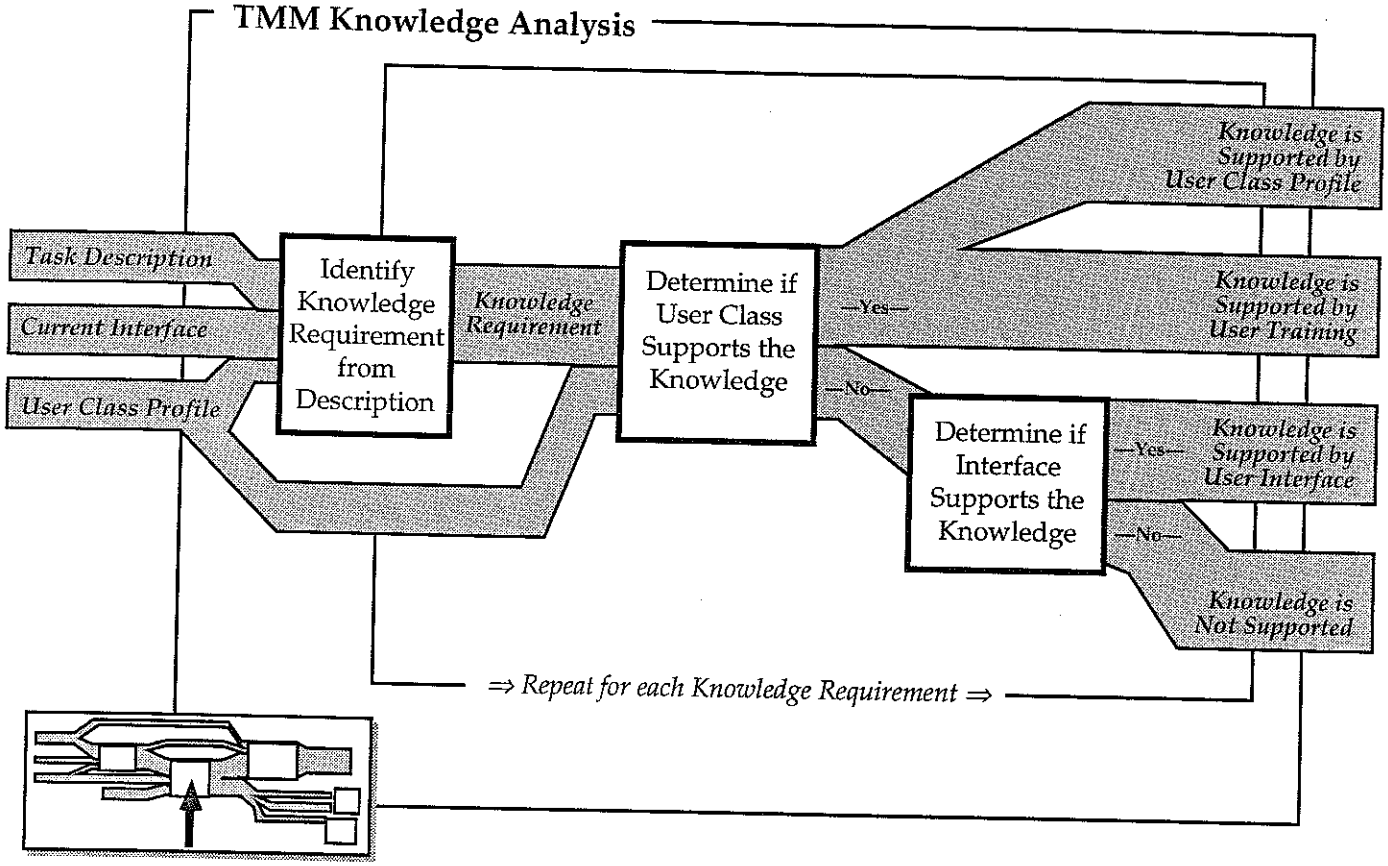
VII.2. Unsupported-Knowledge Analysis (Methodology)

At this point in the TMM analysis life-cycle, the analyst has a description of a task is within the TMM framework. This description is replete with mappings and necessary user knowledge requirements. An analysis is performed on each of the knowledge requirements within the mappings to ascertain whether the knowledge requirement is supported. User knowledge requirements are **supported** if:

- the user possess the knowledge (i.e., the user class profile identifies the knowledge as possessed or understood by the user class) , or
- the interface models or gives reasonable access to this knowledge (i.e., the interface provides the knowledge in a symbolic/metaphor/iconic representation or as textual prompts).

If either of these two conditions hold, the knowledge requirement is supported, otherwise it is an **unsupported** user knowledge requirement. This stage of analysis focuses on generating a list of unsupported user needs in terms of knowledge. Generating this list is referred to as an **unsupported knowledge analysis.**

Following is a general outline for generating an unsupported knowledge list using the completed **TMM Task Description Form** and filling in the **TMM Identified Knowledge Element Form** (see Analysis Templates section). This does not represent a hard-and-fast dictum, but only one possible approach to analyzing a task description for unsupported knowledge requirements.



Second Stage in TMM Analysis Life-Cycle

Performing a TMM Knowledge Analysis

1. Generate a **task description** for the task in question.
2. Gather **user class profile(s).**

3. From mappings within the task description, compile a list of all **knowledge requirements** separated into three lists: factual, conceptual, and procedural knowledge. Use the **TMM Identified Knowledge requirements Form.**

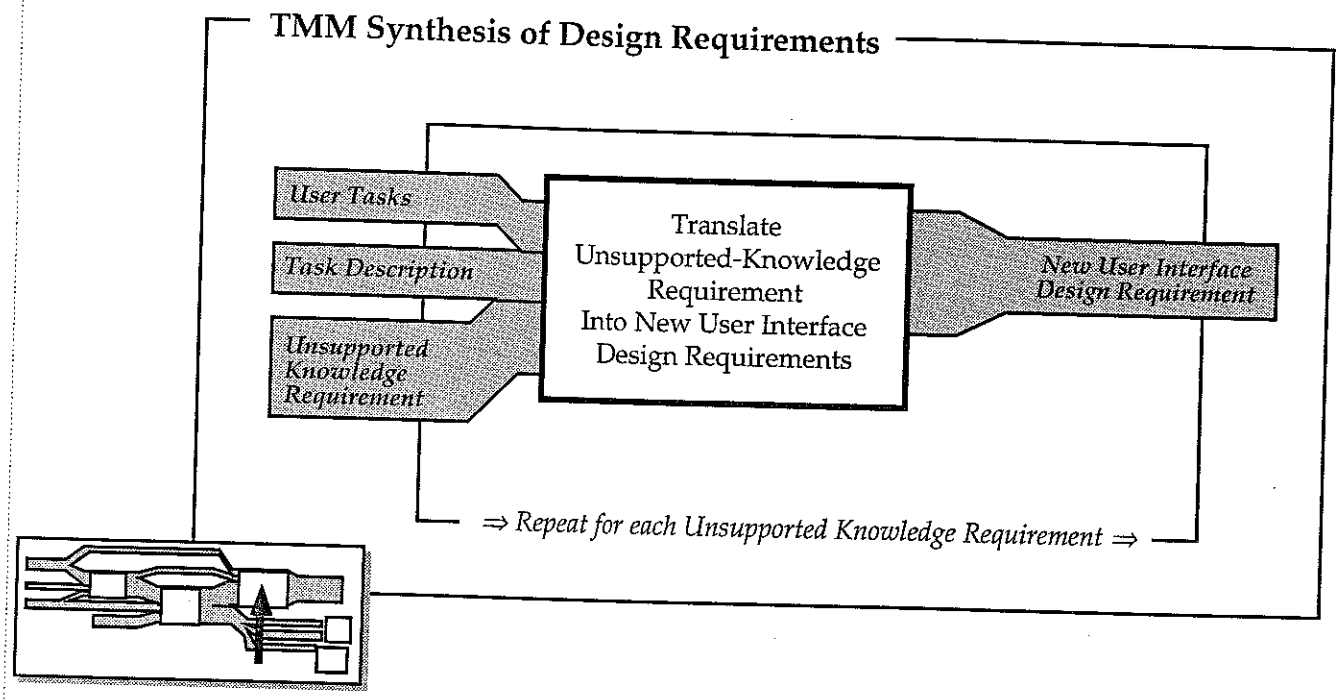
4. For each of the three lists (from step 3) do steps 5-9.
5. For each knowledge requirement on the list do steps 6-9.
6. Determine if user possesses the knowledge requirement though comparison of the knowledge requirement and user class profile.
   - If the user has the knowledge then mark the knowledge requirement as a **supported knowledge requirement** and proceed to next item on the list (goto step 5).
   - If the user does not possess this knowledge requirement continue (goto step 7).
7. Determine if interface adequately provides access to the necessary knowledge.
   - If the interface provides the knowledge then mark the knowledge requirement as a **supported knowledge requirement** and proceed to next item on the list (goto step 5).
   - If the interface does not provide this knowledge continue (goto step 8).
8. At this point, the knowledge requirement is unknown to the user and the interface does not provide access to necessary information. Therefore, that knowledge requirement is marked as an **unsupported knowledge requirement**.

9. Goto step 5.

Determining if knowledge requirements are supported, steps 6 and 7, is the crux of this analysis stage. Often there are ambiguities as to knowledge support, and in these cases, the analysts' experience and knowledge play an integral part. For example, consider an interface where a knowledge element is supported from the analysts' view (it is displayed), but not from the user's view (no logical connection can be made). However in this case, the results of usability testing would indicate the flaw in the designers' assumption of knowledge requirement support.

VII.3. Interface Design Support Analysis (Methodology)

Finally, once a TMM task description and a list of unsupported knowledge requirements are generated, a final analysis turns this information into design requirements. Following is a general outline for generating new user interface design requirements using the **TMM Translation Analysis Form** (see Analysis Templates section). This does not represent a hard-and-fast dictum, but only a possible approach.



Third Stage in TMM Analysis Life-Cycle

Interface Design Support Analysis

1. Perform a **unsupported knowledge analysis** for the task in question.
2. For each **unsupported knowledge requirements** identified in step 1, do steps 3-8.
3. Transfer the unsupported knowledge requirement(s), mappings, and associated related domain items to a new **TMM Translation Analysis Form.**
4. With relation to the unsupported knowledge requirement, examine the **current interface** (if appropriate) and distinguish and record related characteristics .
5. With relation to the unsupported-knowledge element, examine the **user class profile(s)** and distinguish and record related deductions and inferences .
6. With regards to steps 4 and 5, define the **user interface design requirement(s).**
7. ***(Optional)*** Translate the User Interface design requirement into an actual **user interface design specification.**
8. Goto step 2.

The quality of this analysis relies on the abilities and experience of analysts and interface designers—steps 5-7 in particular. However, the following templates of user interface design requirements can aid analysts and designers when faced with unsupported knowledge elements and user needs.

Supporting Factual Knowledge

An example user interface design requirement for an unsupported factual knowledge, <STATEMENT>, could be:

```
UIDR:   COMMUNICATE <STATEMENT> TO USER
        (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).
```

For example, suppose a factual knowledge requirement of THIRD COLUMN IS DEBT. If this requirement is unsupported, then both the user must be unaware and the interface doesn't give any cue that the third column represents debt. This requirement is cast into a user interface design require like:

```
UIDR:   COMMUNICATE <THIRD COLUMN IS DEBT> TO USER
        (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).
```

Of course, the user interface design specification could outline an interface solution for this requirement. Within interfaces, requirements for factual knowledge can be dealt with by displaying the knowledge requirement either textually or graphically, and for a short duration or continuous. So a possible user interface design specification could be:

```
UIDS:   ALL DISPLAYED COLUMNAR DATA IS TO BE CLEARLY LABELED.
```

Supporting Conceptual Knowledge

With conceptual knowledge, <CONCEPT>, the UIDR could be:

```
UIDR:   PROVIDE UNDERSTANDING OF <CONCEPT> TO USER
        (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).
```

For example, Apple Macintosh-OS helps users conceptualize files by using icons. These file representations can be used with other icons that represent actions (e.g., trashcan represents deletion). The use of icons and software widgets to support conceptual information is not new; however, it is very complicated.

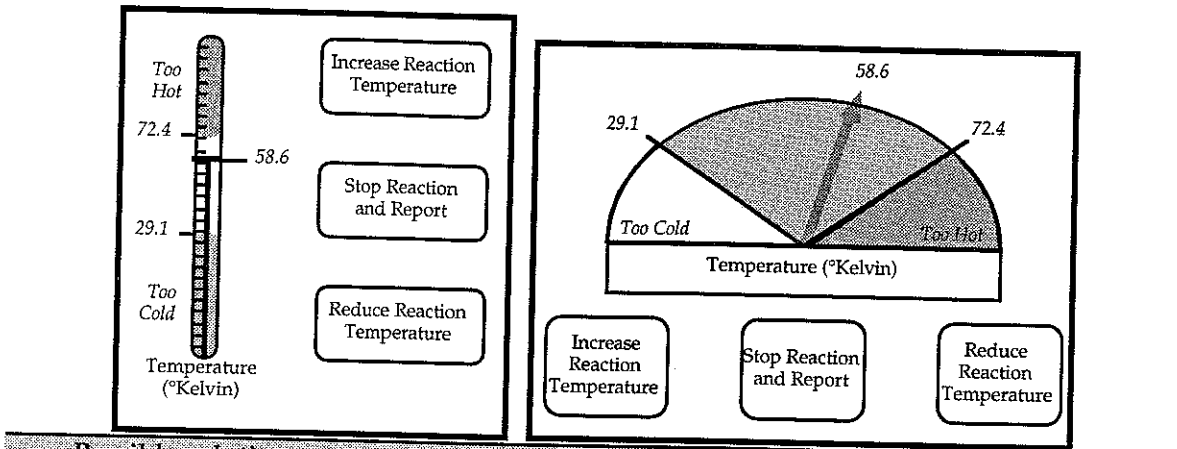
For example, consider the user conceptual knowledge requirement of CHEMICAL PROCESS TEMPERATURE CONTROL within an interface that controls some chemical reaction. Temperature is common physical phenomena; however, in this case the temperature has direct bearing on a chemical process, where too hot or too cold can corrupt the process. If the user class profile does not indicate that the user class is intimately familiar with the ranges, settings, and outcomes based on temperature, and the interface is poor at conveying this information (found through usability testing)—then a UIDR could be cast as:

```
UIDR    PROVIDE UNDERSTANDING OF <CHEMICAL PROCESS TEMPERATURE
        CONTROL> TO USER
        (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).
```

The redesign associated with this UIDR is not as straightforward as that of the factual knowledge example. In this case, an entire concept must be conveyed. A user interface specification defines a possible solution:

```
UIDS:   CONTINUOUSLY DISPLAY A WIDGET THAT ENCAPSULATES ALL
        NECESSARY OPERATOR INFORMATION BASED ON A 'SCALE' METAPHOR.
```

There are many possible design solutions for this interface specification, including:



Possible solutions to UNDERSTANDING OF <CHEMICAL PROCESS TEMPERATURE CONTROL>

In this example, the critical temperatures are identified for the user, as well as, buttons for changing the reaction temperature (either increasing or decreasing the temperature).

Supporting Procedural Knowledge

Procedural knowledge of actions or courses of actions could generate the following UIDR:

```
UIDR:   IDENTIFY POSSIBLE PATHS OF EXECUTION TO USER
        (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).
```

Informing users about possible courses of action is difficult. Sometimes performing a task requires the user be aware of several sub-tasks and their outcomes. Also, alternative task methods may be available to the user. Yet, with the increasing complexity of user interfaces and user tasks, identification of procedural knowledge becomes more important.

For example, consider a bank teller's task of DEPOSIT MONEY INTO BANK ACCOUNT using the bank's computer system. This task is made up of several different tasks: IDENTIFY CURRENCY TYPE (cash, check, or another account), LOCATE DEPOSIT ACCOUNT, PERFORM TRANSACTION, and VERIFY TRANSACTION. These tasks may also have sub-tasks (e.g., identify currency type for a check may involve verifying the check against an account, etc.) So the procedural knowledge necessary for the user (bank teller) would generate the following UIDR:

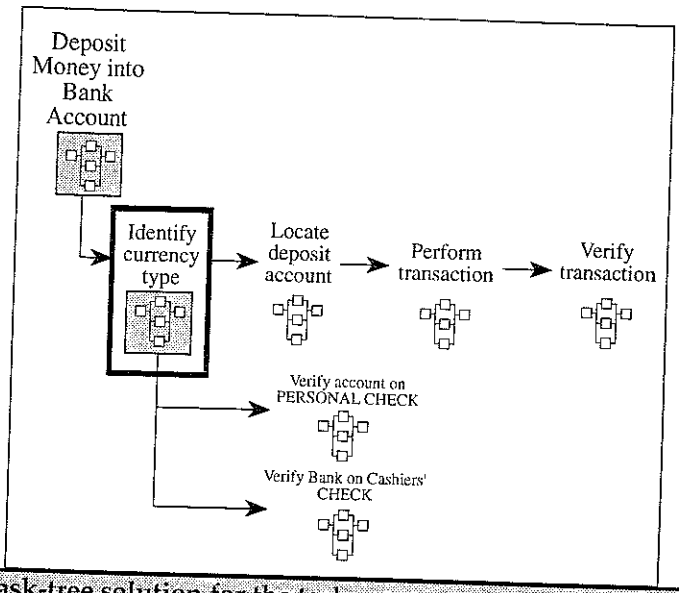
```
UIDR:   INFORM THE USERS THAT <DEPOSIT MONEY INTO BANK ACCOUNT> TASK
        REQUIRES THE FOLLOWING TASKS TO BE PERFORMED:  IDENTIFY
        CURRENCY TYPE (cash, check, or another account), LOCATE
        DEPOSIT ACCOUNT, PERFORM TRANSACTION, and VERIFY TRANSACTION.
```

Each sub-task would also have a UIDR associated with it to outline the tasks required. However, for this example we will focus on this UIDR.

The question comes up, how can this UIDR be converted into an UIDS or user interface design solution. Specifications that try to communicate procedural knowledge to users include: task tree (a flow diagram of possible courses of actions), user prompting, etc. In our example the UIDS could be:

```
UIDS:   PROVIDE A TASK-TREE TO THE USER IDENTIFYING SUB-TASKS AND
        ORDERING FOR THE TASK: DEPOSIT MONEY INTO BANK ACCOUNT
```

with the possible solution:



Possible task-tree solution for the task: `DEPOSIT MONEY INTO BANK ACCOUNT`

In this example, a task-tree is implemented so that users (bank tellers) know at all times their location within the task (identified here as rectangular boarder). Users also know what they have done and what is left to do. For this example the tasks are expanded as necessary to show possible alternative methods and task timing. This can be very useful in training novice users, but may be an impediment to expert users.

TMM and design requirements

It is worth restating that TMM is used to synthesize new user interface design requirement, and not new interface designs. The examples provided here show the utility of transferring TMM design requirements into specifications and redesign; however, TMM does not directly support the designer's creative problem solving process.

VII.4. TMM Metrics

Another style of analysis that can be performed over TMM task descriptions and analyses is metrics collection. Metrics are measurements and functions of measurements that can indicate a strength or weakness within the measured object.

Currently two different metrics exist: **goodness of fit** and **task stacking**. TMM metrics are still growing and need validation.

VII.4.i. Goodness of Fit

The goodness of fit (GOF) metric measures the 'fit' of specific interface to a task. The 'fit' is defined as the ratio of number of supported knowledge requirements to the total number of knowledge requirements derived from TMM analysis. Logically, the goodness of fit metric is bounded between zero and one—worst and best respectively. A high ratio indicates that a large portion of the knowledge requirements were supported either by the users or the interface.

$$\frac{\text{Number of Supported Knowledge Requirements}}{\left(\text{Number of Unsupported Knowledge Requirements} + \text{Number of Supported Knowledge Requirements}\right)}$$

Goodness of Fit Metric

We feel that these metrics can provide an empirical way to compare different tasks over alternate interface designs; however, currently this metric needs validation.

VII.4.ii. Task Stacking

Another metric that can be gathered over a TMM task description is task stacking. Task stacking is the user process of suspending the current task, performing another task to completion, and then returning to the previous task (that had been stacked). The user must maintain a stack of the current tasks that s/he is performing and that can be counted by examination of a TMM task description. Task stacking has been explored in other models to determine user cognitive workload, and our approach is similar. (Card, Moran, & Newell, 1983; Kieras & Polson, 1985; Lewis & Polson, 1992)

While it is useful to count the number of suspended tasks at any task level, this method does not attempt to access the *complexity of each task stacked*. Therefore, this metric can only served to indicate the number of suspended tasks, but not the overall effect of suspending a particular task. If, however, you are examining short duration, low complexity, sequences of tasks then this metric can help to identify location of cognitive overload—where there are too many tasks stacked.

VII.5. TMM as a Teaching/Documentation Device

TMM task descriptions can also be used for documentation and teaching purposes. TMM is based on a domain structure where natural language specifies items, also the clarity of UAN at the articulation level helps. (Hartson, et al., 1990) The straightforward nature of the descriptions lends itself to non-expert interpretation, and thus, a description can, to a degree, be understood by users. If users can follow a task description and identify necessary knowledge, then their training needs are met.

In the arena of teaching, TMM task descriptions can support, through guidance, task performance. TMM descriptions outline necessary information that users must have—hence, users only need to parrot the actions within TMM descriptions.

VIII. Common Questions

Will a tool be implemented to support TMM analyses?

Yes, eventually. For any theoretical endeavor to make its way to the practical world there must be some physical manifestation. Currently, there are preliminary thoughts towards creating a shareware C/X11 application that will provide ability to create and analyze task descriptions using the TMM framework and methods.

Should TMM be used for global analysis?

If time and money permitted than a full analysis could be done; however, in most cases the time to analyze an entire system is too great. We feel that a strength of TMM is its ability to be used in *situational analysis*—examination of a particular situation or task. This focuses the designer on real problems instead of allowing them to wander through other unfruitful analyses. Using situational analysis, TMM can be more easily incorporated into the iterative design process, during initial design as well as between formative evaluation and redesign.

Why all this work for design requirements?

Often, the interface design requirements gathered from users are incomplete and vague. TMM task modeling and analysis can help further define the requirements, and these interface design requirements are then used in interface specification and high level design. By providing a systematic method to derive new design requirements, as opposed to *ad hoc* methods, TMM can help reduce design time.

Does TMM tie the interface designers' hands?

No. TMM modeling and analysis only derive interface design requirements and possibly interface design specifications. Using these requirements and specifications to create a usable interface is still the job of interface designers.

Should we always redesign based on TMM analyses?

As with most endeavors, the interface design process is limited by finances and time. Obviously, the goal of all interface designers is to produce a highly usable interface design, however, decisions must be made when confronted by deadlines. The decision to commit to a redesign, based on analyses, is a function of time, cost, and user impact. It is a context-dependent decision that cannot be addressed in this forum.

IX. Example

This section outlines an example of using TMM to derive/synthesize new design requirements. This example shows TMM analyses for a genealogy project prototype on a particular user task.

The first subsection outlines the prototype, while the second subsection provides a TMM situational analysis with analyst commentary. Each stage of the TMM analysis life-cycle is demonstrated, albeit condensed for space reasons.

X.1 Genealogy Project Description

Recently several users were asked to compile a list of requirements for a 'simple' genealogy interactive system. They produced three different types of requirements: data requirements, functional requirements, and interface requirements. Data requirements outline the project's responsibilities with respect to data and data links (what to store and relationships among data). Functional requirements specify the basic functionality of the project—these form a minimum set of commands and actions available to users. Lastly, interface requirements were users' requests concerning the project's human-computer interface. An abbreviated list of these user defined requirements is presented to define characteristics of the genealogy project.

User Defined Data Requirements

The interactive system must store information (personal datasets, PD) indexed by person—information includes, but is not limited to: Date, Time, and Place of Birth; Lifetime Achievements or Awards, short history, and any other free-format information.

The interactive system must store relationships among PDs—these relationship links include: husband, wife, child, sibling, parent.

User Defined Functional Requirements

The interactive system must generate reports of the following formats (format given under separate cover). These formats are not important to this discussion, so they are not included.

The interactive system must have the following functionality:

Add/Remove/Edit a personal dataset
Add/Remove/Edit a relationship link
Ability to search the family tree
Ablity to traverse a tree 'Cut-Away' (section)
Ability to get help

User Defined Interface Requirements

Point-and-Click Interaction (as much direct manipulation as possible)

Easy to use. *Designer translates into 'high usability'.*

All functions must be present and indicated on the screen at the appropriate time.

The designer examined the three sets of requirements, performed a high-level task analysis and functional decomposition, created an essential (logical) model of the system, developed initial interface designs, and then developed a working prototype.

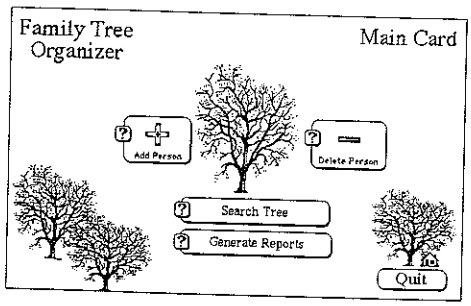
Functional decomposition and task analysis provides a list of important tasks that users commonly perform while using interactive systems. These tasks are generic in form, so they are translated into specific benchmark tasks for use during formative evaluation. A sub-set of tasks are chosen for this project design's formative evaluation, and they are:

Generic Tasks	Benchmark Tasks
Task: Add a Personal Dataset to the database.	Task: Add George Alfred Mayo to the database.
Task: Change Relationship within a Personal Dataset. *(This includes the tasks of adding/removing children, spouses, and parents.)*	Task: Change George Alfred Mayo's wife to Rebecca Rennick Johnson Task: Add Gregory Thomas Mayo as a child of George Alfred Mayo Task: Change George Alfred Mayo's mother to Inda Inskeep
Task: Traverse Tree Structure through Ancestor Chart	Task: Pull-up Ancestor Chart for George Alfred Mayo Task: Pull-up Ancestor Chart for Gregory Thomas Mayo
Task: Pull-up Personal Dataset for a given Person from the Ancestor Chart	Task: Pull-up Personal Dataset of Rebecca Rennick Johnson

Generic/Specific Tasks used in Formative Evaluation

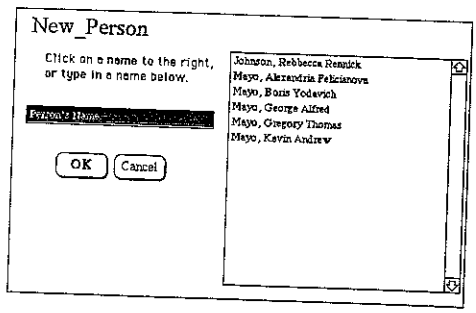
A prototype was created to evaluate the interface design. This prototype was implemented in Hypercard™ and consisted of several (>10) 'cards' over various 'stacks'. There are several problems with this interface design. The problems range from simple consistency (e.g., card name position) to more detailed conceptual difficulties (e.g., how to navigate through the interface family tree structure). The following prototype screens show the basic user interface design.

Project Start-Up & Main Screen



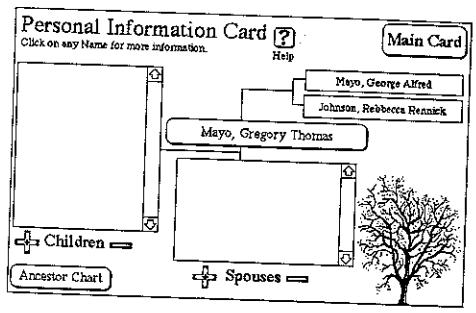
This is the 'start-up' screen seen by users when the project software is launched. This provides a rough button-menu driven approach to system functionality. At this level the user may add/delete personal datasets from the database, search the database, or produce database reports. Help is also available for all commands.

Name Query Screen



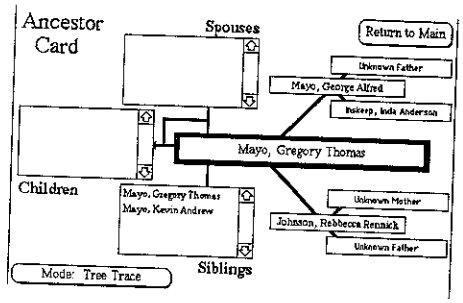
This screen represents the general 'name query' sub-task common to many of the system's functions. In this case, the query screen interacts with the user to gain a new name to add to the database. One of the users' data requirements specifies the database be indexed by names, and this screen lists all names in the database helping users in name specification.

Personal Dataset Display Screen



This screen shows the general form for displaying personal datasets, with an example dataset for 'MAYO, GREGORY THOMAS'. The user may interact with this interface design to add/delete children, spouses, or parents. Any name may be selected to display the corresponding personal dataset (this functionality traverses the family tree structure). The user may also 'pull-up' an ancestor chart by selection the 'Ancestor Chart' button.

Ancestor Chart Screen



This final screen example shows an ancestor chart (card). The chart shown focuses on one person at a time—'MAYO, GREGORY THOMAS' for this example screen. The chart displays all spouses, children, and siblings; there is also a partial display of father and mother lineage—only two levels deep (i.e., only to grandparents). There are two modes of searching for traversing ancestry charts, and name selection on this chart will invoke a search.

The formative evaluation on the prototype serves several functions, it (1) provides testing for functional completeness, (2) furnishes users access to working prototype for task analysis, and (3) grants users input into the user interface design process.

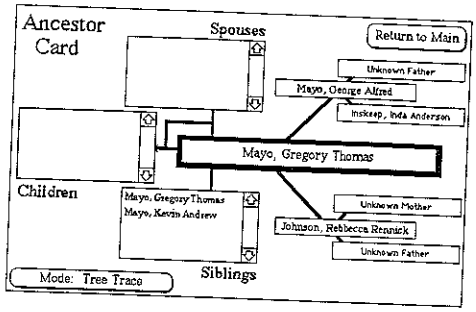
X.2 Situational Analysis of Genealogy Task

The discussion, to remain short, concentrates on a single problem that several novice users encountered while traversing the ancestor chart.[1] To reiterate, the ancestor chart is traversed by name selection with two different search modes: Tree Trace *(displays another ancestor chart for selected name)*, and Information Search *(displays personal dataset for selected name)*. Users found it difficult to understand these modes, and to traverse the ancestor chart.

To explore this user problem, an analyst examined a task highlighting the use of search modes—the user must use the ancestor chart to locate a person and display the associated personal dataset. The benchmark task was: FROM THE ANCESTOR CARD (FOCUSED ON MAYO, GREGORY THOMAS) PULL-UP THE PERSONAL INFORMATION CARD FOR JOHNSON, REBECCA RENNICK.

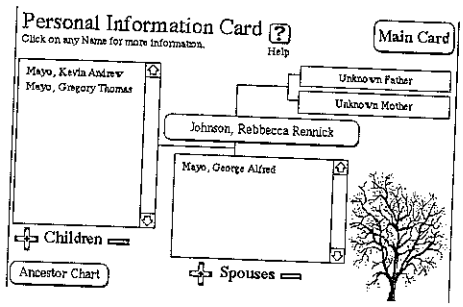
[1] Novice users are novice to the genealogy project software, not to direct manipulation computer system interfaces.

Problem Task: Starting Screen



The user starts the task at this screen—ancestor chart focused on MAYO, GREGORY THOMAS.

Problem Task: Ending Screen

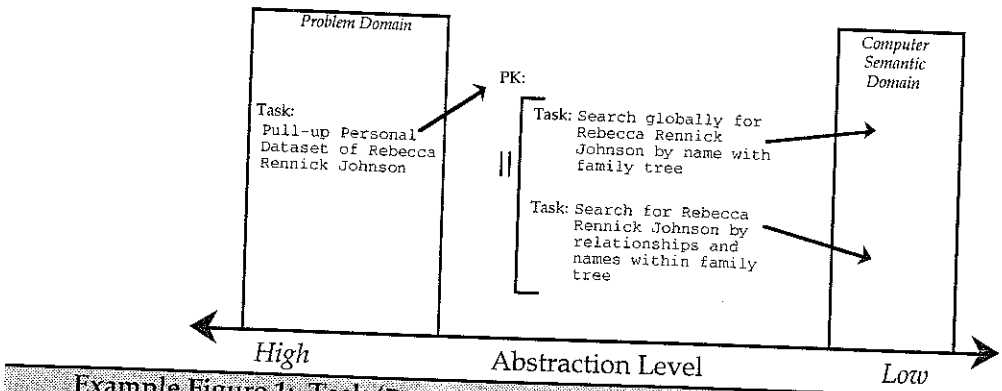


This is the final screen. Getting to this screen indicates that the user successfully completed the task.

These two screens show the user's dilemma. The user must interact with the interface in the first screen to get to the personal dataset shown in the second screen.

Following is an example analysis scenario with analyst/designer quotes in highlighted-italic text.

> ***Analyst/Designer:*** *"Okay, I've just finished some user testing, and the prototype interface has some problems. I will analyze the task:* PULL-UP PERSONAL DATASET OF REBECCA RENNICK JOHNSON. *I know the user's performed two different methods to accomplish this task, either the user will traverse the ancestor card to find the necessary information, or the user will return to the main card and start a search from there. These are the representative task methods"*



Example Figure 1: Task (PULL-UP PERSONAL DATASET OF REBECCA RENNICK JOHNSON) Decomposition

Thinking through these two different methods, it becomes apparent that they represent two different styles of searches: the first method, if chosen, was accomplished easily with little error. In this case the user returned to the main screen and initiated a global search for REBECCA RENNICK JOHNSON.

The second method, using the ancestor chart to traverse the family tree structure is based on a sub-task of locating a family member position with the family tree. This is a very difficult sub-task, and the location within the task that users had the most problems. See Example Figure 3 for a description of this sub-task. (Example Figures 3, 4, and 5 are at the end of this section due to their size.)

This task description gives an outline of the task `Locate Family Member Position in Family Tree (LFMP-FT)`. In order to accomplish this task, the user must first examine the tree and change it's focus (possibly more than once). This represents the user traversing through the family tree within the '`Information Trace`' search mode. Here the user must conceptualize personal pedigree charts as tree structures and that proper names index this tree structure. Decomposing this task further, to change the ancestor chart focus, the user must confirm the search mode and select a key name to search on. The knowledge required here includes information about the identifying and changing search modes, as well as, search outcomes based on these modes.

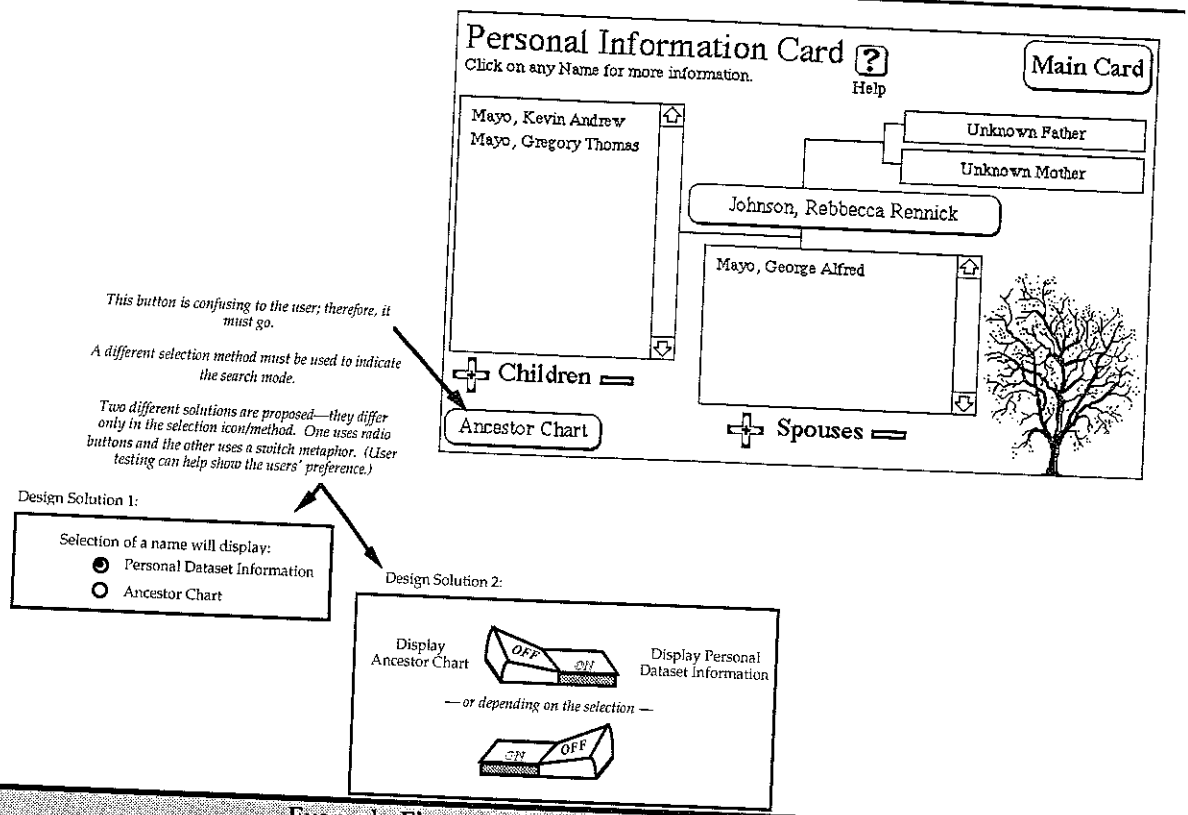
> ***Analyst/Designer:*** *"I feel pretty good about these descriptions. Of course, I realize that there are other ways a user might perform this task—but, these represent the methods that most users performed. I know (through user testing) users were having problems with search modes, but let's see what the analysis turns up."*

Next the designer performed an Unsupported-Knowledge Analysis. Listing and examining the knowledge elements yielded several knowledge elements that were unsupported. (See Example Figure 4.)

> ***Analyst/Designer:*** *"Humm. The users complained about modes, and the TMM analysis showed modes were not supported! I wish I had done this analysis before paying for all that user testing! I hope I'm not fired for this."*

The final analysis examined the unsupported-knowledge elements to synthesize design requirements—see Example Figure 5. This example uses the basic user interface design requirement (UIDR) template from the user's guide, and synthesizes four new UIDRs for the specified task. From these UIDRs the designers can create new interface design solutions.

In fact, from these UIDRs several different solutions can be posed, two are shown in Example Figure 2.

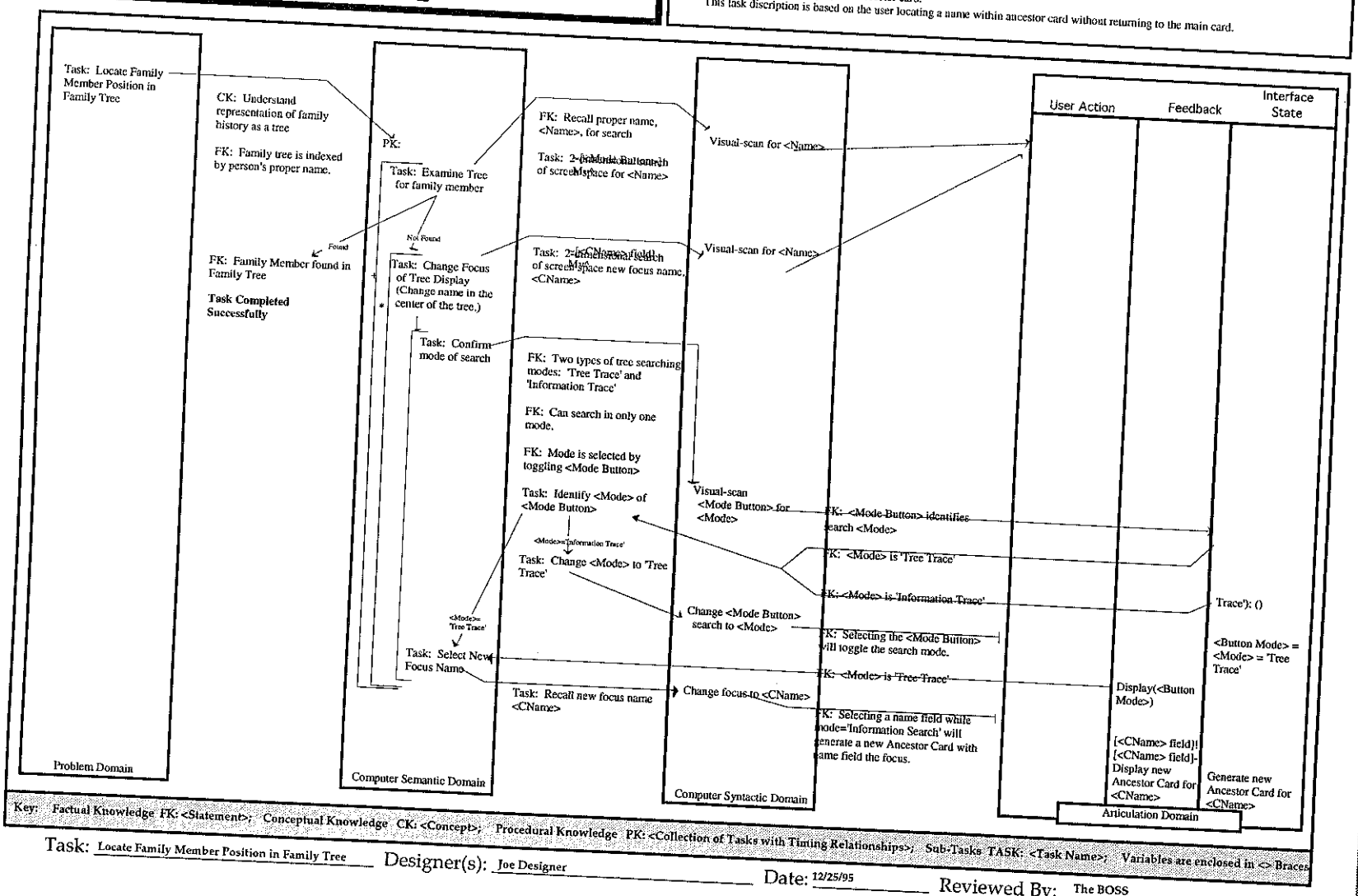


Example Figure 2: Possible Design Solutions

This scenario shows how TMM can be used in formative evaluation to support interface development. The new design requirements, once implemented, should go far to help support the user in this task.. This example also shows how TMM can be used for situational analysis (analysis of a specific user problematic task) to derive interface design requirements.

TMM Task Description Form

Task Context/Notes:
Currently the screen displays an ancestor card.
This task discription is based on the user locating a name within ancestor card without returning to the main card.



Key: Factual Knowledge FK: <Statement>; Conceptual Knowledge CK: <Concept>; Procedural Knowledge PK: <Collection of Tasks with Timing Relationships>; Sub-Tasks TASK: <Task Name>; Variables are enclosed in <> Braces

Task: Locate Family Member Position in Family Tree Designer(s): Joe Designer Date: 12/25/95 Reviewed By: The BOSS

Example Figure 3. Task Description for: Locate Family Member Position in Family Tree (LFMP-FT)

TMM Identified Knowledge Element Form

Task/Design: Locate Family Member Position in Family Tree
Designer(s): Joe Designer
Date: 12/25/95
Reviewed By: The Boss

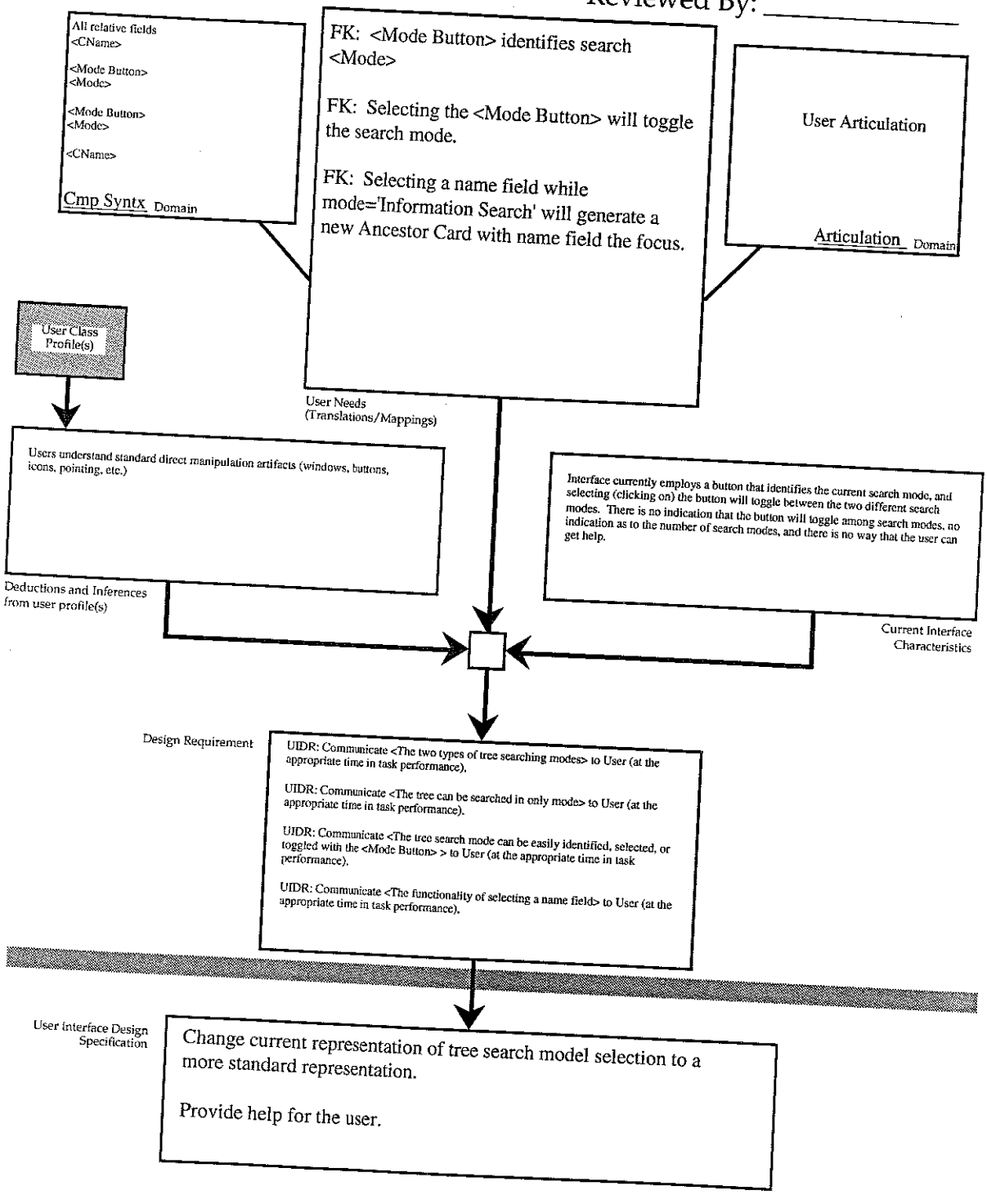
Identified User Knowledge Needs

	Notes (e.g., justification of support or non-support)	Supported by User Profile	Supported within Interface	Not Supported
Factual Knowledge Elements				
Family tree is indexed by person's proper name	This is supported by the user profile. A user defined data requirement is that the tree is indexed by the person's name.	✓		
Two types of tree searching modes: 'Tree Trace' and 'Information Trace'	There is no indication as to what 'Tree Trace' or 'Information Trace' means. Also, there is no indication as to the number of searching modes. This information is not in the user profile.			✓
Can search in only one mode	The interface does not provide this infomation, and it is not in the user profile.			✓
Search mode is selected by toggling <Mode Button>	There is no indication that the mode button toggles between search types.			✓
<Mode Button> identifies search <Mode>	The mode button identifies the search type.		✓	
<Mode> is 'Tree Trace'	The mode button identifies the search type.		✓	
<Mode> is 'Information Trace'	There is no indication that selecting a field will generate any feedback based on search mode.		✓	
Selecting a name field while mode = 'Information Search' will generate a new Ancestor Card with name field the focus				✓
Conceptual Knowledge Elements				
Understand representation of family history as a tree	This information is in the user profile.	✓		
Procedural Knowledge Elements				
Decomposition/Timing of Computer Semantic Domain representation	The user knows to search this tree and if not found to change tree focus.	✓		
Decomposition/Timing of Task: Change Focus of Tree Display	The interface supplies visual clues to changing the focus of the tree. However, because of unsupported factual knowledge, the user is unable to uses these clues.		✓	
Understand task/outcomes of Task: Examine Tree for Family Member	User knows how to visually search tree, and when to stop.	✓		
Understand task/outcomes of Task: Identify <Mode> of <Mode Button>	This is supported in the interface.		✓	
Task: Change <Mode> to 'Tree Trace'	There is no indication in the interface, and the user does not know how to perform this task.			✓
Recall & Visual Search Tasks	The user knows how to perform visual searches, and perform memory recall.	✓		

Example Figure 4: Unsupported-Knowledge Element Analysis for LFMP-FT Task

TMM Transaction Analysis Form

Task/Design: _____
Designer(s): _____
Date: _____
Reviewed By: _____



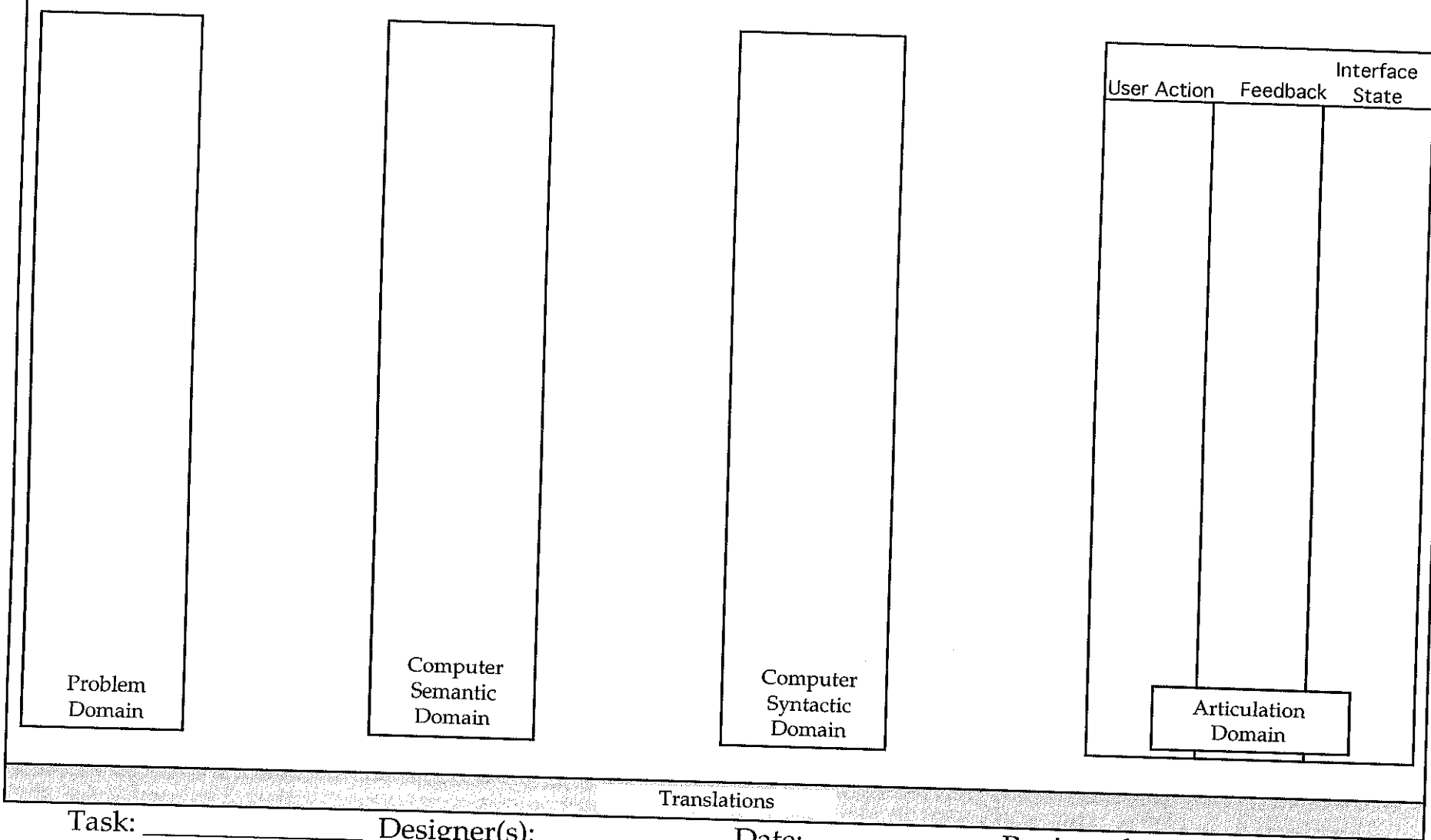
Example Figure 5: UIDR Synthesis for LFMP-FT Task.

Analysis Templates

This section provides several blank templates (Forms) for TMM users.

TMM Task Description Form

Key:



Task: _____ Designer(s): _____ Date: _____ Reviewed By: _____

TMM Task Description Form

TMM Identified Knowledge Requirement Form

TMM Identified Knowledge Element Form

Task/Design: ____________________

Designer(s): ____________________

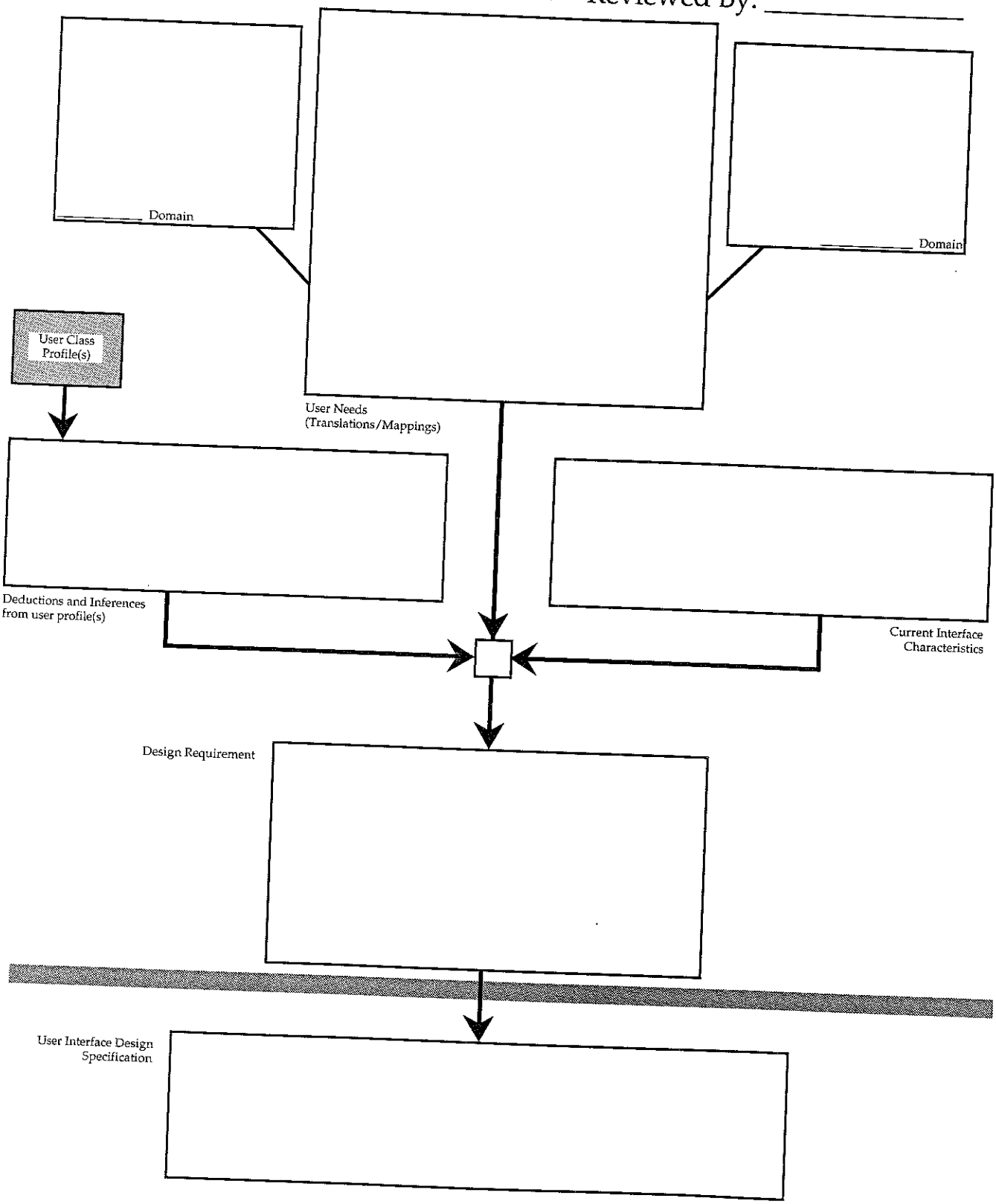
Date: ____________________

Reviewed By: ____________________

	Identified User Knowledge Needs	Notes (e.g., justification of support or non-support)	Supported by User Profile	Supported within Interface	Not Supported
Factual Knowledge Elements					
Conceptual Knowledge Elements					
Procedural Knowledge Elements					

TMM Transaction Analysis Form

Task/Design: _____
Designer(s): _____
Date: _____
Reviewed By: _____



Glossary

This section contains definitions of commonly used terms within this document and also within the task mapping model.

Analysts: A person concerned with the description and analysis of user task s for performance and usability improvements of human-computer interfaces.

Analysis Orientation Modeling Characteristic: Models with this orientation generally attempt to analyze tasks and performance for currently existing interfaces. These analyses can be used within formative evaluation with limitations (because of size and complexity); these models are more suitable for final analysis within the summative evaluation phase.

Articulation Domain: The lowest level of abstraction in the domain structure of TMM. The actual physical user actions are recorded within this domain.

Closed-Loop Modeling Characteristic: Closed-loop modeling views tasks as cyclical involving a user action-feedback-reaction processes. Models that approach tasks in this manner have this characteristic.

Computer Semantic Domain: A domain of the TMM domain structure. This domain contains generic computer artifacts translated from the problem and computer syntactic domains.

Computer Syntactic Domain: A domain of the TMM domain structure. This domain contains abstract grammatical components for conveyance to the computer.

Conceptual Knowledge: the understanding of relationships between collections of factual knowledge, ideas, and other conceptual knowledge.

Concurrent Tasks: Tasks that are executed at the same time are concurrent.

Design Support Modeling Characteristic: This characteristic is represented in models whose results are directly applicable to designs or the design process. These models feed results directly into the iterative user interface design process.

Domain: A region populated with items characterized by a single feature. (E.g., all items are defined within the problem scope = problem domain.)

Domain Items: Any objects, operations, or sub-tasks defined within a specific domain.

Error-Free Performance Modeling Characteristic: Models with this characteristic can perform analysis with the assumption of error-free user behavior. This assumption is very common in prediction and interface models.

Error Performance Modeling Characteristic: Models that can analyze user behavior with errors have this characteristic. This characteristic is much more problematic for task analysis techniques, and requires an extended view of task performance and analysis.

Essentially Analytical Modeling Characteristic: Models that examine and manipulate representations of tasks and interfaces apart from empirical measurements/observations are analytical. This is a common characteristic among several models.

Essentially Empirical Modeling Characteristic: Models based on and incorporating empirical evidence have this characteristic. Empirical evidence could be verbal protocol, timing values, user satisfaction surveys, and critical incidents among others.

Evaluation Path: A sequence of translations and associated domain items from lower to higher

levels of abstraction. These paths represent a reflective mode of the user, often spawned by system output.

Execution Path: A sequence of translations and associated domain items from higher to lower levels of abstraction. These paths represent the user taking higher level domain items into lower level items, often all the way to articulation.

Expert Performance Modeling Characteristic: The ability to model and analyze expert behavior and performance is represented by this characteristic. Many models have this characteristic.

Factual Knowledge: single or collections of declarative facts; something said to be true.

Global Modeling Characteristic: This characteristic is present in models that analyze entire representations of the interface and task. Performance models are often of this type because the predictions need the entire system described to be accurate.

Interface: The collection of devices (hardware and software) that allows the communication between two different systems. (In this case, the communication is between humans and computers.)

Interface Design: The developed requirements and specifications of an interface.

Interface Designer: A person who performs the necessary chores for interface design. This could, under loose interpretation, also include implementers of the interface design.

Interface Design Requirement: A design statement of a goal that must be present in the interface. E.g., `The interface must be Motif compliant.`

Interface Design Specification: A design requirement translated into hardware or software terminology. E.g., `There will be four menus: File, Edit, View, and Special.`

Interleaved Tasks: A set of tasks of which only one can execute at a time, but they may be suspended and resumed at anytime and in any order.

Knowledge: Facts, concepts, and procedural knowledge. *Also see conceptual knowledge, factual knowledge, and procedural knowledge.*

Open-Loop Modeling Characteristic: Open-loop modeling views a task as performing actions to a desired result without benefit of feedback, or that feedback is irrelevant. Models with expert behaviors assumptions often ignore feedback.

Performance Prediction Modeling Characteristic: Models that calculate performance predictions are most commonly used in summative evaluation for comparisons against usability specifications. A complete (or greatly extended) design is necessary for these models to approach accuracy.

Problem Domain: The highest abstracted domain of the TMM domain structure. This domain contains items defined in, and only in, the user task (problem) domain.

Procedural Knowledge: Knowledge of a particular action, or course of action.

Mapping: A knowledge element crucial to a translation. One or more mappings constitute a translation.

Method: A plan or course of action.

Method Selection: The determination of which method from a list of alternative methods.

Methodology: A system of methods, principles and rules.

Metric: A measurement taken on an object. In this case, measurements taken to increase usability or performance from a software interface.

Multiple User Classes Modeling Characteristic: Users are no homogenous. Methods that allow analysis and modeling of multiple user classes per task have this characteristic. Models that center on expert error-free performance often cannot have this important characteristic.

Non-Sequential Non-Concurrent Tasks: Tasks that are not ordered in time and each task is atomic (non divisible) in nature.

Novice Performance Modeling Characteristic: This characteristic is present in models that can model and analyze novice performance. Novice users comprise a growing class of users that must be addressed, yet, many models only partially support them.

Repeated Tasks: A set of tasks that are repeated zero or more times.

Sequential Tasks: A set of tasks where the ordering is linearly based on time.

Situational Modeling Characteristic: Models that allow analysis of single tasks or interface features are situational and have this characteristic. Often these models are employed during formative evaluation due to their utility.

Sub-task: A task subordinate to another task. Its completion or failure impacts the primary task(s).

Synthesis Orientation Modeling Characteristic: Models that derive new designs or requirements are synthesis oriented. The new design or requirements can be incorporated into a design revision within the iterative design process, and thus, aid the designer.

Task: "A task is an arrangement of behaviors (perceptual, cognitive, motor) related to each other in time and organized to satisfy both an immediate and a longer-range purpose." (Meister, 1985)

Task Abstraction: To examine and redefine a task form and function away from the actual manifestation; to reconceptualize in high-level ideas and concepts.

Task Analysis: "Task analysis is a process of identifying and describing units of work, and analyzing the resources necessary for successful work performance. Resources in this context are both those brought by the worker (e.g., skills, knowledge, physical capabilities) and those which may be provided in the work environment (e.g., controls, displays, tools, procedures/aids)." (Drury, Paramore, van Cott, Grey, & Corlett, 1987)

Task Description: To examine and reiterate a task form and function within a pre-defined task description notation.

Task Mapping Model: A task-/user-centered approach to synthesizing user interface design requirements from task analysis.

TMM: Task Mapping Model.

TMM Identified Knowledge Need Form: Template of a analysis form that helps designers examine task descriptions to produce a list of unsupported knowledge needs.

TMM Task Description Form: Template of a user interface task design form that is crafted to work with TMM.

TMM Translation Analysis Form: Template of a analysis form that helps designers examine unsupported knowledge elements in order to produce user interface design requirements (UIDR).

UAN: See User Action Notation.

UIDR: User Interface Design Requirement. *See design requirement.*

User: Any person that performs a task on a computer.

User Action Notation: A symbology and notation for describing user physical actions during human-computer interaction. **(Hartson, et al., 1990)**

User Class: A pre-defined set of similar users.

User Class Population: The set of all salient user classes defined for a particular application.

User Class Profile: A description of knowledge, skills, and attributes of a particular user class.

User Knowledge Modeling Characteristic: Models that have this characteristic attempt to capture necessary user knowledge. This differs from user mental modeling in that this characteristic does not in any way represent the mental state of user, but only necessary user knowledge.

User Mental Modeling Characteristic: Models that rely on a basis user mental model have this characteristic. A user mental model, as this research defines it, is a collection of user mental states, translations among the states, and cognitive operators to effect the state changes. This could include a representation of an interface or user knowledge within a context. Interface modeling techniques have an underlying user mental model of the system (interface).

References

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

Drury, C. G., Paramore, B., van Cott, H. P., Grey, S. M., & Corlett, E. N. (1987). Task Analysis. In G. Salvendy (Ed.), *Handbook of Human Factors* (pp. 370-401, Chap. 3.4). New York: John Wiley & Sons.

Evans, J. S. B. T. (1988). The Knowledge Elicitation Problem: A Psychological Perspective. *Behaviour and Information Technology, 7*(2), 111-130.

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

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

Lenorovitz, D. R., Phillips, M. D., Ardrey, R. S., & Kloster, G. V. (1984). A Taxonomic Approach to Characterizing Human-Computer Interfaces. In G. Salvendy (Ed.), *Human—Computer Interaction* (pp. 111-116). Amsterdam: Elsevier Science Publishers B. V.

Lewis, C. & Polson, P. (1992). Cognitive Walkthroughs: A Method for Theory Based Evaluation of User Interfaces (Tutorial). *Proceedings of Human Factors in Computing Systems, CHI '90 Conference*, Seattle, Washington, April 1-5: ACM, pp.

Mayo, K. A. & Hartson, H. R. (1993). Synthesis-Oriented Situational Analysis in User Interface Design. *Proceedings of The 1993 East-West International Conference on Human-Computer Interaction*, Moscow, Russia, August 3-6: ACM, pp. (To Appear).

Meister, D. (1985). *Behavioral Analysis and Measurement Methods*. New York: John Wiley & Sons.

Moran, T. P. (1981). The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems. *International Journal of Man-Machine Studies, 15*, 3-50.

Nielsen, J. (1986). A Virtual Protocol for Computer-Human Interaction. *International Journal of Man-Machine Studies, 24*, 301-312.

Shneiderman, B. (1979). Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. *International Journal of Computer and Information Science, 8*(3),

Shneiderman, B. (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, Massachusetts: Addison-Wesley Publishing Company.