

Task-Related Knowledge Structures: Analysis, Modelling and Application

Peter Johnson, Hilary Johnson, Ray Waddington & Alan Shouls

Department of Computer Science, Queen Mary College, University of London, Mile End Road, London E1 4NS.

A theoretical and methodological approach to task modelling is described, with a worked example of the resultant model. The theory holds that task knowledge is represented in a person's memory and that this knowledge can be described by a Task Knowledge Structure (TKS). The method of analysis has been developed for carrying out analyses of real world tasks. The method uses a variety of techniques for collecting information about task knowledge. A second perspective of the paper shows how a developed TKS model can be decomposed into a design for a software system to support the identified tasks within the domain of the analysis. This decompositional method uses the structure of frames to provide consistency between different levels of design decomposition.

Keywords: Task modelling, knowledge representation, frames, design decomposition.

1. Introduction

Traditionally, task analysis (TA) has investigated what people do when they carry out one or more tasks and involves collecting information about how people perform those tasks. Various approaches to TA have been developed particularly in the context of human computer interaction (HCI). Probably the most often cited approach to TA in HCI is the GOMS approach of [2]. One central tenet of GOMS is that tasks can be thought of in terms of goal structures, operations, methods and selection rules. By extending the original ideas of [2] these four elements might be considered to be the basic structural components of any task knowledge that a person might recruit to perform a task. In other words, we might

suppose that people not only performed tasks, but developed knowledge structures that reflected the way they recruited particular sources of knowledge to a given task. A strong argument to make would be that people had something akin to goals, operations, methods and selection rules represented in long-term memory and processed in working memory. While not making this strong claim, [2] did identify an approximation of the psychological processors (in the form of the human information processor model) that might operate on some or all of these four elements. The work of [10] extends GOMS by arguing that production rules can be used to model goals, operations, methods and selection rules and that these production rules bear a close relationship to the way a person does (or would) structure their knowledge of the task. [10] then use these production rules to assess the complexity of a user interface by merely counting the number of production rules assumed to be required to perform the task; and assess the degree of learning required by counting the number of new production rules required in the new, as opposed to the old technological domain, in which the task is to be performed.

Neither [2] nor [10] explicitly claim that people actually possess these task-knowledge structures. In task-action grammars (TAG), [12] are explicit in their assumption that people do possess task-knowledge structures. However, they give little detail of what these structures might contain or how they might function. For [12], task knowledge appears to be decomposed into a dictionary of "simple tasks" and rule schema, which through the use of rewrite rules specify what knowledge is recruited for a given task. TAG also includes the notion of "family resemblances" for relating together similar task knowledge, and the notion of common knowledge which is task-independent knowledge that a person is assumed to already possess, (for example, pointing, typing, or the meaning of the word "up").

Both GOMS (including its extension by [10]) and TAG have assumed that people possess knowledge that is recruited and used in task performance. TAG further assumes that people possess task related knowledge structures. However, the detail of these knowledge structures and the theoretical or empirical evidence for their existence is weak. In the rest of this paper we will argue that task knowledge structures are functionally equivalent to the knowledge structures that people possess and use when performing a task. We then go on to describe a method for identifying, analysing and modelling task knowledge structures (TKS). The contents of these

TKS models can be usefully applied to the generation of design solutions. A method for applying these models to design is described. In describing TKS models and their application to design we will use examples taken from real tasks that have been analysed and real designs that have been proposed. However, because these examples are taken from complex domains we have selected them so that they will exemplify our approach rather than as complete models of the tasks within the domains.

2. Theoretical Basis for TKS

We will first outline a number of theoretical assumptions which we have made on the basis of existing theoretical and empirical work in cognitive psychology.

2.1. Task Knowledge as Conceptual Knowledge

Task knowledge is represented in conceptual or general knowledge structures in long term memory. This is akin to the theoretical position taken by [15] in assuming that knowledge of frequently occurring events is structured into meaningful units in memory. Empirical support for our assumption that people possess something akin to TKSs can be found in the work of [3] who conducted a series of experiments which show that people recognise and use structures of events, such as the order, the sequence and the importance of activities within the event sequence to understand, explain and make predictions about those events. Further support for our view that task knowledge is represented in long term memory comes from the work on text and story comprehension by [5] in which general knowledge structures relating goals to causal and enabling states, plans for achieving goals, intermediate states and alternative solutions or paths are all represented in a conceptual knowledge structure for the events of the story.

We assume that all the knowledge a person possesses about a task is contained within the task knowledge structure and that the TKS is activated in association with task performance.

2.2. Structure in Tasks

Task knowledge and therefore TKSs would be "unstructured" if the knowledge people had about the task world (or domain) was such that all possible elements of the knowledge used in those tasks could be associated, and was associated with equal probability to other

knowledge elements. This structure is not just an imposed structure but is a reflection of the structure found in tasks in the real world. All possible elements and attributes of a task do not co-occur with equal probability. Our view here is similar to that of [4] in describing how people represent physical objects based on the structure of those objects in the real world.

We argue that task behaviours do not occur independently of one another. Some pairs and n-tuples of task behaviours are quite probable while others which might be logically or physically possible, never occur together in reality. Within tasks, some behaviours are carried out together, precede or follow other behaviours, and in some cases prime, or are primed by other behaviours. One form of this structuring is in a **PLAN** which provides a feasible and acceptable structuring on the task behaviours. Empirical evidence for the representation of PLANS in long term memory can be found in the work on programming tasks [6]. Further evidence for structuring in tasks comes from the work of [1] on tasks in domains such as cookery and meal planning.

2.3. Representativeness of Task Elements

A TKS is composed of a number of task elements which differ in representativeness. This is similar to the notion of representativeness used by [13] and [14] to describe the relations between objects and their categorical representation in memory.

We assume that a TKS includes knowledge about objects (both physical and informational) and their associated actions. This is based on earlier work by [7]. These objects and their associated actions differ in how representative they are of the TKS of which they are a part. One implication of this is that the procedures containing these representative objects and actions are more *central* to the TKS than other procedures. Empirical evidence for the notions of procedural centrality and action/object representativeness in task behaviour can be found in [11] who found that for tasks such as borrowing a book from a library there were particular segments which were more central to the particular instantiation of going to the library and more representative of the class of similar tasks. It is interesting to note that frequency of occurrence alone was not a reliable indicator of object and associated action representativeness nor procedural centrality.

3. A theory of TKS

A TKS is a summary representation of the different types of knowledge that are recruited and used in task behaviour. A TKS is related to other TKSs by a number of possible relations.

3.1. *Within Role Relations*

One form of relation between TKSs is in terms of their association with a given role. A person may take on a number of roles, for example "author", "referee", "teacher" etc. There are tasks associated with each of these roles. For each task that a role may be required to perform there will be a TKS. Those tasks that are related because they are performed by the same role will have the role relation property associating their respective TKSs.

3.2. *Between Role Relations*

A second form of relation between TKSs is in terms of the similarity between tasks across roles. This occurs when a person is performing a similar task under different roles. For example, a person may carry out the task of "arranging meeting(s)" while assuming the role of "chairman", "client", "husband". Each task may be performed differently in one or other respect. However, a single person assuming all these roles would have a knowledge structure for each task and also knowledge (not necessarily explicit) of the relations between these tasks. Thus the TKSs for each of the "meeting arranging" tasks would have relational links to other "meeting arranging" tasks performed by that person when assuming a different role.

Within each TKS there are other knowledge representations. The **goal substructure** represents the goals and subgoals identified within the TKS. The goal substructure also includes the enabling and conditional states that must prevail if a goal or subgoal is to be achieved. In this way the goal structure represents the **plan** for carrying out the complete task. The plan defined in the goal substructure is carried out through a **procedural structure** which contains alternative procedures for achieving a particular subgoal. As there may be alternative sets of procedures for achieving a particular subgoal there are also conditional and contextual groupings of these procedures. In this way **strategies** are represented. Strategies are particular sequences of procedures. The procedures rely on other knowledge of the **objects** and

actions which when combined constitute a given procedure statement. This action and object knowledge is represented in a taxonomic substructure. Within the **taxonomic substructure** information about the properties of the action(s) and object(s) are represented. This includes, the **representativeness** of the object or action, the **class membership** and other attributes such as the procedures in which it is commonly used, its relation to other objects and actions and its features (such as what the object can do or possess). These are described in more detail in [8].

4. Identifying Knowledge for a TKS

A method of identifying the knowledge for a TKS has been developed [8] known as Knowledge Analysis of Tasks (KAT). The theory of TKS outlined above requires information about the following knowledge components for a given task; roles, goals, subgoals, subtasks, plans, procedures, strategies, actions and objects. In addition the analysis should identify the representativeness, class membership, and other features represented within the taxonomic substructure. We have chosen to define the more ambiguous task components in the following way:-

A role is assumed to be defined by the particular set of tasks that an individual is responsible for performing as part of their duty in a particular social context (a "job" is one form of social context). A person can and will take on more than one role and more than one person can and will take on a given role.

A goal is assumed to be the state of affairs that a particular task can produce and forms part of a person's conceptualisation of their task world while carrying out a given role.

A plan is the particular formulation and possible ordering of sub-tasks that are undertaken to achieve a particular goal.

A procedure is a particular element of behaviour that forms part of a subtask. One feature of procedures is that there can be alternative groupings and ordering of procedures for the same subtask, these alternatives are thought to reflect different strategies and as such contain conditional selection rules that enable the correct grouping of procedures to be selected under the appropriate conditions.

Actions and objects are the lowest level of elements of tasks and are the constituents of procedures.

A number of techniques for identifying the components of a TKS have been considered in [8]. For the sake of brevity, we will simply summarise the techniques in terms of their application to particular components of a TKS, in table 1. Most of the techniques need little in the way of explanation as they are standard techniques used in social surveys, however some guidelines are provided as to when to use the various techniques, since more than one technique can be used to identify the same TKS element. The techniques listed in table 1 vary in terms of their appropriateness and in terms of their ease of use, resource and time requirements. As a rule of thumb we recommend that the analyst should apply more than one technique to identify a given TKS component. This enables the information obtained under one technique to be confirmed, rejected or supplemented by the use of a further technique. In terms of effort, the analyst must consider the time requirements of the task performer as well as their own. With this in mind we recommend that where it is possible and appropriate for a particular TKS element, techniques which involve minimal or no direct time costs on the part of the task performer should be applied first (eg accessing manuals, texts or other documentation). The next techniques to be used where it is appropriate, should be questionnaires and interviews, since these are also least time consuming for the task performer. However, for some task elements it is necessary to have a more considerable time commitment from the task performer. Direct observation, concurrent and retrospective protocols are all expensive with respect to the task performer's time and they should be carried out in the order mentioned here. Finally techniques such as sorting, constructing tree diagrams and so on, should be carried out once the analyst has a relatively detailed understanding of the person's tasks.

In producing a TKS the analyst is required to identify the above knowledge elements. If the intention is to produce a composite model that is representative of a number of persons' knowledge of a given task then it is necessary to sample from the population of people who carry out the identified tasks. The extent of the sampling is again determined by the access and the time available to the analyst. However, it is rarely the case that only one person performing a given task will be analysed.

Table 1. Summary of techniques for identifying elements of a TKS

<u>TKS element</u>	<u>Identification technique</u>
actions & objects	selecting from texts or manuals; tutorial sessions; pilot study; analyst performing task; structured interview with task performer; questionnaire; direct observation of task performer; concurrent or retrospective protocol.
object representativeness	frequency counting; rating scales; sorting tasks (eg card sorting); list ordering.
procedures	direct observation; concurrent protocols.
plans	structured interview; retrospective protocols; direct observation.
goals and subtasks	questionnaires; structured interviews; constructing tree diagrams; retrospective protocols.

Table 2. Generification of TKS elements

<u>TKS element</u>	<u>Generification process</u>
Actions & objects	<ol style="list-style-type: none">1. Construct lists of actions and objects for a task across all task performers and instances of the same task.2. Note the frequency of each action and object in the list and remove all repetitions from the list.3. Using judges from task performers, group all like actions and objects.4. Identify appropriate labels for the identified groupings of actions and objects.5. Produce lists of the identified generic actions and generic objects.6. Validate the list with a different group of task performers.
Plans & procedures	<ol style="list-style-type: none">1. List all the procedures and the plans in which they are used, for each task performer and each task instance.2. Record the frequency of each procedure in a plan and across plans.3. Record the frequency of each plan across task instances and task performers.4. Select a frequency threshold level for procedures and for plans (the two need not be the same).5. Identify all procedures and plans which are above the previously specified threshold.6. Validate the identified generic procedures and plans with the task performers.

5 Identifying the Generic Properties of a TKS

Having collected adequate information about the individual TKS elements, the next step is to identify the generic elements of a TKS. Generic elements are included in a composite task model of the individual task performers and instances of the tasks which the TKS is meant to represent. This reduces the effect of variation in the TKS, for instance from non-relevant contextual effects on task behaviour (such as the equipment on which the task is performed). Clearly what is or is not relevant will be dependent upon the purpose of the analysis itself. However, the *method* of generification we have developed is independent of the purpose. The process of generification is summarised in table 2. and is reported in full in [8]. Each element of a TKS and its process of generification is shown. The process should be applied in an iterative manner such that if the generic properties identified on the first attempt are not validated by the task performers then the process should be reiterated with different threshold values (in the case of plans and procedures) and/or different groupings and labels (in the case of actions and objects).

6. Constructing a Complete TKS Model

The resultant task model, produced from the analysis comprises a number of summary representations and substructures which together provide a detailed model of the knowledge associated with a collection of tasks related together by roles.

The highest level of representation of the model is provided by the role/task relations. There are two types of role/task relations;

- a) The different roles that are required to perform a given task would be assumed to have a similar TKS for that task.
- b) The different TKSs that a person fulfilling a particular role may be expected to access and use.

The next level of representation of the model is the content of the TKS for a given task. This representation includes relations to lower level representations of the model containing the plan, procedures and taxonomic substructure (ie action and object property/feature lists). It details the knowledge associated with the specific task from the family of tasks performed by a given role or roles.

At a lower level of representation in the model is the plan and procedural knowledge associated with the specific task. This representation is referred to as a goal-oriented substructure. The plan represents the structure of the subgoals and the states which those subgoals satisfy. Each subgoal is related to one or more sets of procedures for carrying out that subgoal. Where there is more than one alternative set of procedures these constitute different strategies. Within a procedure the specific activity procedures are represented by production rules of the form "IF..... THEN". The procedures also have links to the taxonomic substructure which is the lowest level of representation in the model.

The taxonomic substructure contains the category structure for the object and its associated action. The category structure is divided into superordinate, basic and subordinate category levels. The basic level category contains the following information;

- is a member of.....
- is used in procedure(s).....
- is related to.....by.....
- is associated with actions.....
- has features.....
- typical instance.....

This concludes our description of the TKS model and its contents.

To help clarify and concretize the detail of a TKS we provide an example of a complete TKS for a task in the domain of office work, namely "arranging a meeting". We have shown this task model from the perspective of the role of an office secretary, however we also show relations between the secretary and manager role, and how both might possess corresponding TKSs. Also we show other TKSs which a person in the secretary role might possess. These are shown in figure 1 and figure 2 respectively. The summary contents of the TKS for "arrange a meeting" are shown in figure 3. Figure 4 shows the goal-oriented substructure which comprises the plan for the task and the procedures which are used to execute one of the subgoals (namely, "plan the meeting PPM"). This figure also shows the production rules that are used to execute one procedure in that procedure set, namely "consult a location for an information token of a project meeting" (Consult(LOC, ITs, Projectmeeting)). Figure 5 shows the taxonomic substructure for the object MESSAGE as it is used in the TKS for the "arrange meeting" task.

Figure 1. A subpart of a messaging task world

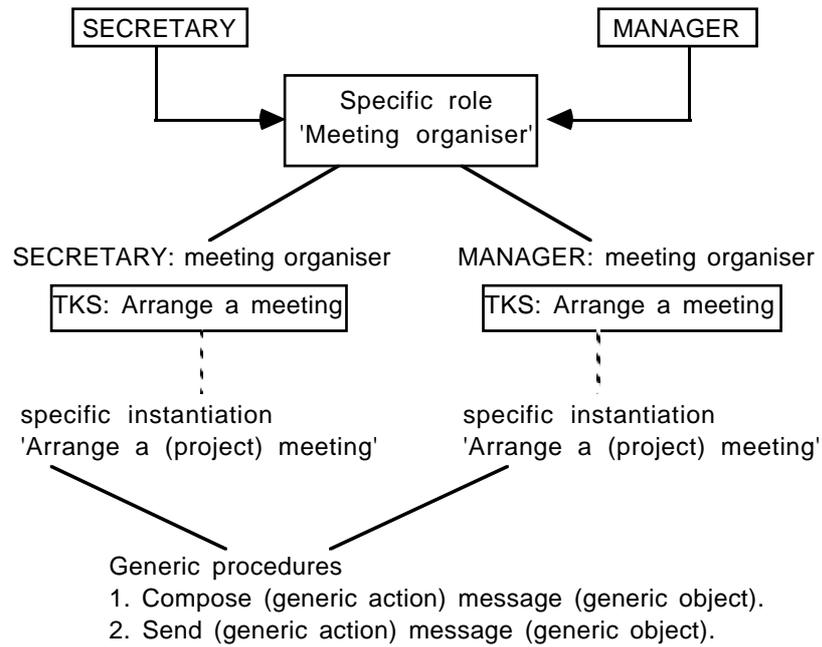
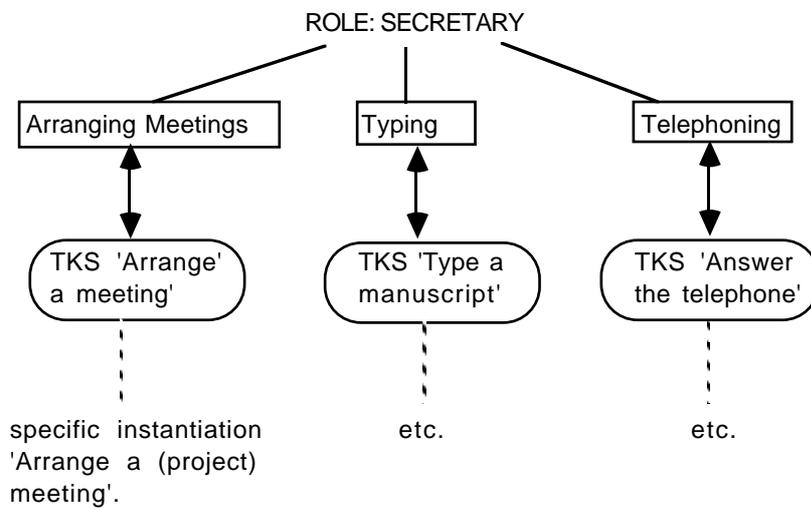


Figure 2. Some of the possible TKSs associated with a single ROLE



In the basic level category it is revealed that the MESSAGE object has the following properties;

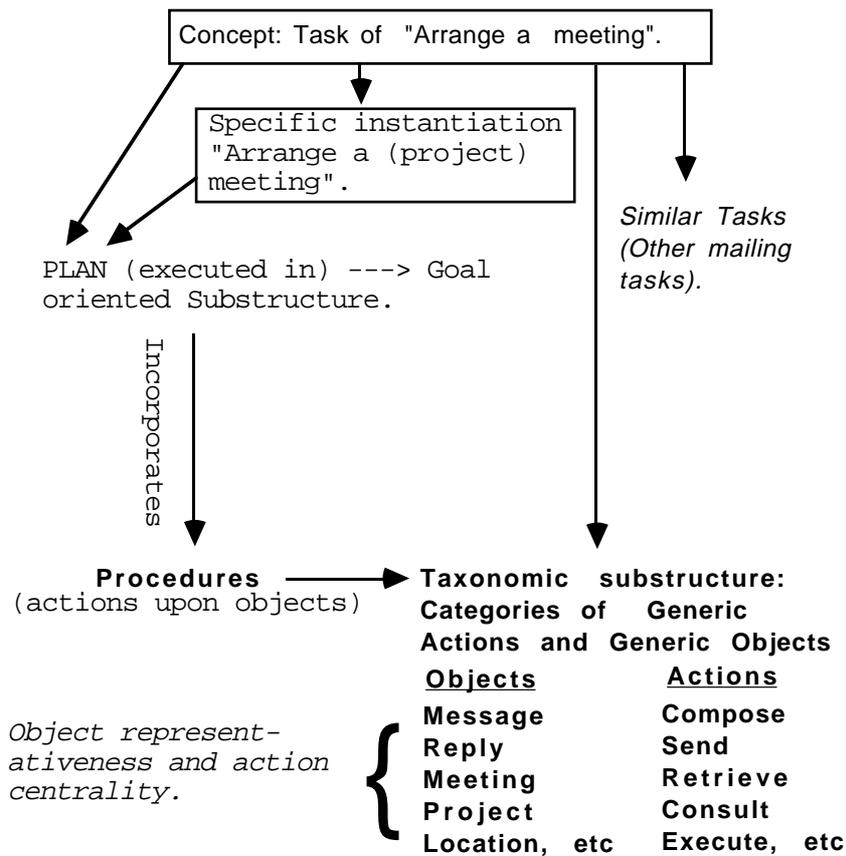
is a member of	"arrange a project meeting"
is used in procedures.....	"decide message:to"
	"decide message:body"
is related to objects.....	reply.....by.....precedes
is associated with actions....	decide, consult, storein, select etc.
has features.....	text, date, address, paper.
typical instance.....	message suggesting suitable dates for a meeting.

This simple example shows how the complete TKS model represents the different attributes of knowledge associated with the task of "arranging a meeting".

7. Making Design Recommendations from the TKS Model

The TKS model for "arranging a meeting" contains useful information that can be used to influence the design of a computer application program to support messaging by electronic mail. The way in which the TKS model can influence design relies upon the overriding assumption that a computer program which is designed to be used by a given user population to carry out an expected range of tasks, will be easier to use if the users are able to transfer some of their existing knowledge about those tasks to the new environment created by the computer program. This assumption underpins the use of "metaphors" such as "desk-tops" or "forms", in which the computer program attempts to retain some identifiable links with a user's assumed extant knowledge about real desk-tops and paper based forms. However, it is clear that a metaphor is only one mechanism by which transfer of extant knowledge might be facilitated; furthermore, the way that a metaphor might function is itself the subject of some debate. It should also be noted that not all aspects of a person's extant knowledge will be relevant or transferable to the new environment. For example, the knowledge of how to dial and use a telephone may have little relevance to support communication by a textual computer-based messaging system. Keeping with the same example, the knowledge a person has about asking questions, making requests, or providing answers would be applicable to both the old and new environments for communication and could (should) be supported in the new environment.

Figure 3. TASK KNOWLEDGE STRUCTURE



The TKS model identifies the knowledge structures that a person is assumed to access when carrying out any task. The method of analysis allows the analyst to identify the contents of those knowledge structures for particular tasks within a given domain. The selection of the domain and the focus of the tasks within the domain is outside the methodology. Having constructed a TKS model the analyst has identified a number of important properties of the users' knowledge about those tasks. At the object level the taxonomic substructure identifies the typical instances of objects within the domain and the features of those objects. The work on concept and object knowledge of [14] leads us to suggest that if the designer chooses to support this task and provide a visible representation of the objects, then the taxonomic substructure

Figure 4. Goal-oriented substructure of the "Arrange a (project) meeting" task

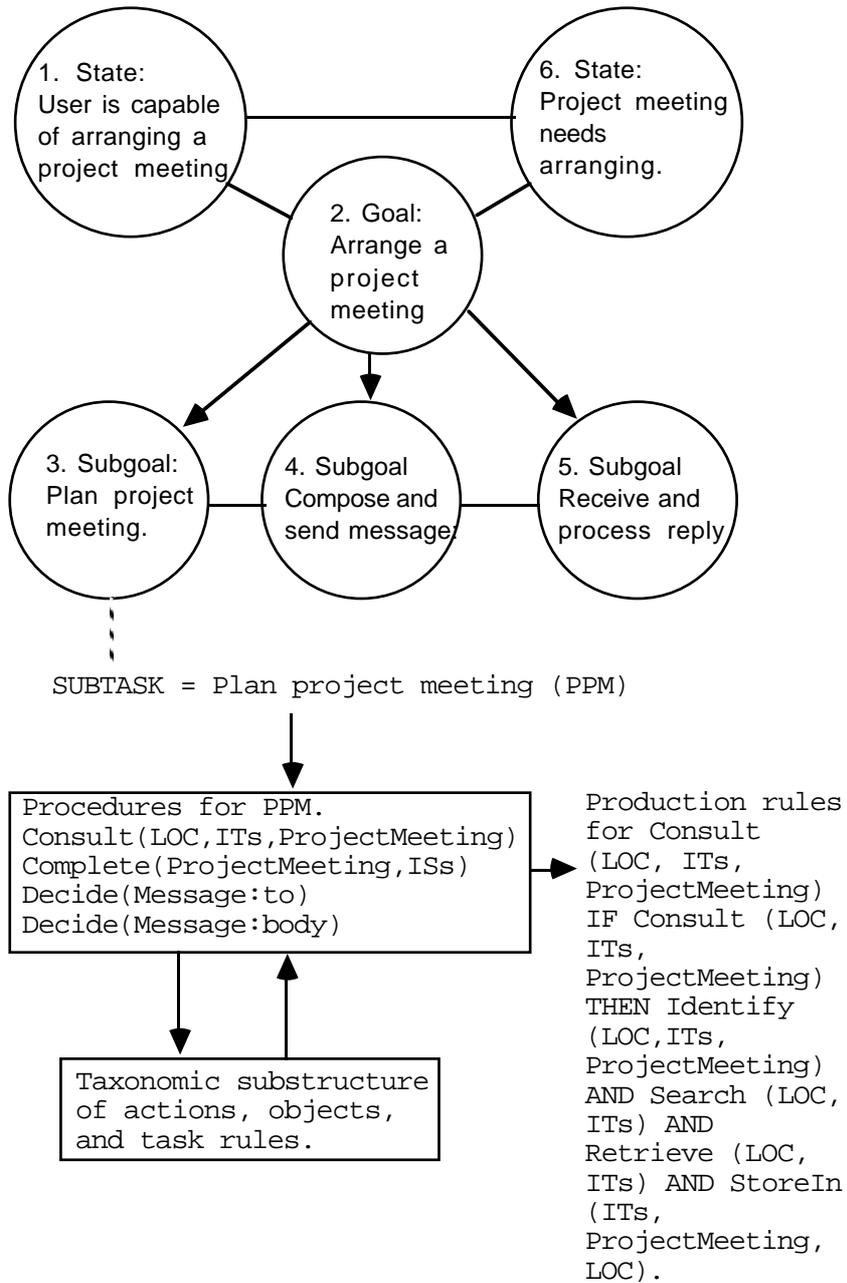
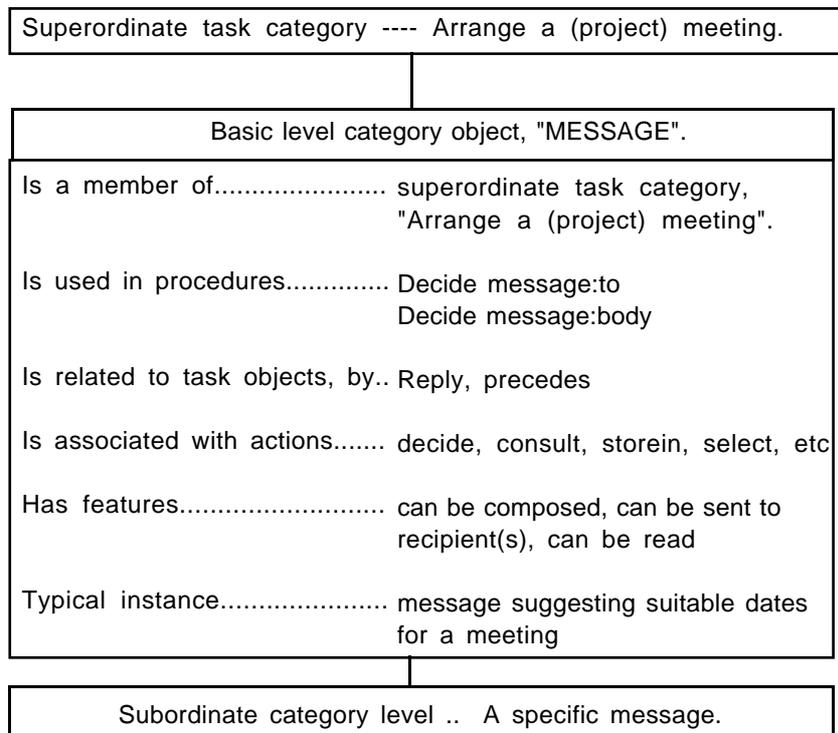


Figure 5. Taxonomic substructure for the "Arrange a (project) meeting" task illustrating the basic level object, "MESSAGE", and its relations to the superordinate and subordinate category levels



provides an informative and detailed description of the features a person will expect to associate with those objects. At the next level of representation the TKS model identifies the knowledge a person has about how to achieve their subgoals and the plans they construct and have for achieving the identified goal of the task. The procedures represented at this level of the TKS provide a description of the rules that people performing the task would expect to follow and the alternative procedures they would follow under particular conditions. This information can be used by the system designer to decide how the user will expect to make use of the objects and actions (functions). The TKS at this level also identifies what the most frequent or preferred procedure for achieving a subgoal is. This information can be used to set up default modes of operation in the program design. The next level of the TKS represents an overall summary of the plan, procedures and the object and actions people

associate with a particular task. This information may be of interest to the designer in so much as it provides an overview of particular contexts in which specific procedures might be used. It could also be used to provide the user with a summary representation of how the designer expects a task to be carried out. At the highest level the TKS shows the relations between tasks and roles. This information provides the designer with a view as to how different roles might expect to have access to the same task functions and to those task functions which are specific to particular roles. This task/role information is also of use to designers who may be concerned with configuring a system to suit the needs of a particular organisation, since it shows the task/role match of the organisation.

The information contained within a TKS is very rich and can be used as an information source for the design of computer programs to support those tasks. However, the TKS is not itself a design model. It is not decomposable into a design specification. It is simply a user knowledge model. However, it can be transformed into a design model and decomposed into a design. From initial ideas about the use of frame representations to describe analysis for design [9], we have developed a method of decomposing a task model into a design for a software system. This method of decomposition is described in the next section.

8. Decomposing a Task Model into a Design Model

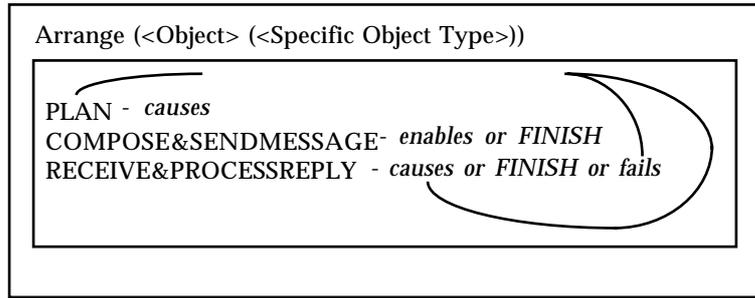
In decomposing a design the approach we have developed has three main objectives; first to maintain *consistency* between levels of decomposition, second to *add detail* at each level of decomposition, and third to take as its starting point a task knowledge model, whose structure is consistent with the outcome of the TKS model. The decompositional model we have developed uses the properties of frames to maintain consistency between the different levels of decomposition. Each frame *inherits* the properties of the previous level of design detail. In this way we are able to meet our first objective of consistency. At each level of the decompositional model further design detail is added until, at the lowest level, the visible features of the represented user interface are being designed. This meets our second objective of adding design detail at each level of decomposition. The decompositional model takes as its starting point a task knowledge model consistent with the TKS model, and transforms this into a frame representation, where the decomposition process may begin. This meets our third objective of starting with a structure consistent with the TKS model.

The method has three stages of decomposition. At the first stage the TKS model is transformed into a General Task Model (GTM). The GTM represents the ROLES, GOALS, ACTIONS, STRATEGIES and OBJECT DEFINITIONS of the TKS as a series of frames. At the second stage the GTM is transformed into a Specific Task Model (STM). The STM is different from the GTM in that it contains the additional information of program/user allocation by identifying what features of a task the computer program will support and how these fit into the broader description of the complete user/program definition of the system. Like the GTM the STM is represented in the form of frames and inherits the structure of the task from the GTM. The third stage is to transform the STM into a Specific Interface Model (SIM). The SIM describes the structure and the content of the user interface at a conceptual or semantic level. That is to say, the SIM describes what should be represented at the user interface and how it should support interaction, without describing the particular form of representation that the user interface might have. By defining the interface at this level of abstraction we are able to design independent of the particular machine on which the application might run. Thus, it is possible to implement the design on a standard VT100 type terminal or on a high-resolution, colour workstation with mouse and keyboard input. We propose also that the SIM could be further decomposed into machine-dependent designs. In fact, using this approach we have been able to describe the design of an extant user interface (to electronic mail software) as part of a research project concerned with early evaluation techniques. Thus the decompositional model meets each of our three objectives of consistency, adding design detail and taking input from a model consistent with TKS.

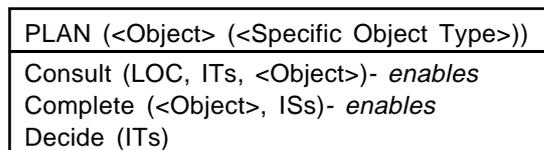
Each of the stages of the decompositional model are exemplified below. We have used the same example of the "arrange a meeting" task that was used to exemplify the TKS model above. Note that the models presented here show only the generic role of "meeting arranger" (which might be undertaken by a secretary). We acknowledge that such a task will have a *between role relation* with the generic role of "meeting participant" (which might be undertaken by a manager - see figure 1). The complete model would show *all* interacting roles in a task, but for the sake of simplicity we show only the former role here.

Figure 6. The GTM for the "arrange a meeting" task

General Goal Frame, Arrange



Instrumental Goal Frame, PLAN

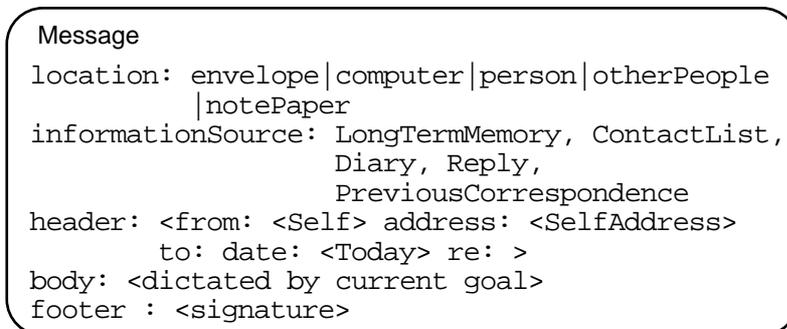


Macro-Action, Consult, decomposed into Micro-Actions

Consult (LOC, ITs, <Object>)

- identify (LOC, ITs, <Object>)- *enables*
- search (LOC, ITs) - *causes*
- retrieve (LOC, ITs)- *enables*
- storeIn (ITs, <Object>, LOC)

Object, Message



The first stage is the construction of the GTM. The full GTM for the "arrange a meeting" task is shown in Appendix A. An example of each of the components of the GTM is shown in figure 6 for convenience. The GTM contains General Goal Frames (GGFs), Instrumental Goal Frames (IGFs), Macro-Actions, Micro-Actions, Strategy Lists and Object Definitions.

The GGF represents the goal and subgoal structure of the TKS and shows the defined plan for the task. The form of the GGF is *General Goal (<Object> (<Specific Object Type>))*. In the example used, this would be *Arrange (Meeting (Managerial))*, assuming that the specific instance of the task concerns arranging this type of meeting. The GGF consists of IGFs - that is those goals which are instrumental to the achievement of the general goal - which are related to each other in that one may be sufficient for carrying out the next (enables), or indeed necessary and sufficient (causes).

IGFs show those elements of the task that are identified by the TKS procedures, and are decomposed into Macro-Actions. Macro-Actions are also related according to sufficient and necessary conditions. For example (see figure 6) PLAN consists of: Consulting a LOcation for Information Tokens associated with an Object; this enables one to Complete the details of that Object using any necessary Information Sources; this enables one to Decide on the precise nature of those Information Tokens.

The Macro- and Micro-Actions, and Strategy Lists represent production rules and strategies of the TKS. Micro-Actions are the decomposed form of Macro-Actions, and are the lowest-level of decomposition in the model. Again they bear the relations showing sufficient and necessary conditions. Figure 6 shows the decomposition of the Consult Macro-Action. (See Appendix A for the interpretation of individual Micro-Actions.) Some Macro-Actions do not get decomposed into Micro-Actions. Typically, these are ones for which analysis is unable to render compartmentalised sequences of behaviour, for example Decide. In these cases the decompositional approach models task behaviour using collections of production rules, called a Strategy List.

The Object definitions represent the objects of the TKS taxonomic substructure. Figure 6 shows the example of a Message Object. Objects are represented as frames, in which their structure is defined as Information Tokens that assume particular parameters (values). Thus a Message has a location, envelope, computer etc.,

and various information sources, Long Term Memory, Contact List etc., that may be used to obtain values for its other information tokens. One of the merits of adopting a frame representation lies in the notion of "default values." Many of the information tokens that define an Object may assume meaningful defaults. For example, a person normally (by default) sends a message from themselves; hence Message:header:from will have "Self" as a default value. The default values are identified in the TKS. This notion of default values is important when considering design. We shall return to it later.

The next stage requires the designer to define the user/program allocation of the system. This is modelled by the Specific Task Model (STM) in the decomposition method. The STM is shown in figure 7, it maintains the structure of the GTM but adds detail about the allocation of user/program functions.

This allocation is shown in the model by the use of different typeface. Those parts of the model shown in **bold text** are wholly supported by the program. Those that appear in *outline text* are partially supported by the program and those shown in plain text are not supported by the program. The concept of partial support comes about it two ways. The first is inherent in the decompositional nature of the model. For example (see figure 7) "Consult" is partially supported because some of its Micro-Actions are supported and others are not. The second concerns the kind of support given in terms of objects at the interface and actions performed upon them. For example (see figure 7) "search" is partially supported because it may occur in different contexts: in the context of Consult (ContactList, Message:to, Message) it is recommended in the STM that the interface support the user in performing search (ContactList, Message:to, Message), since the system has been designed to contain a representation of Message Objects, which includes a ContactList. However, in the context of Consult (LongTermMemory, Message:header:re, Message) a design decision has been made not to provide support for the user to perform search (LongTermMemory, Message:header:re, Message). Thus actions may be partially supported in that the support depends on the kind of objects that are supported at the interface.

The final stage is to model the interface and how it supports interaction. This is achieved by the Specific Interface Model (SIM) which is shown in figure 8. The same conventions of typeface are adopted here as in the STM. The SIM maintains the structure of the GTM and STM but adds the definition of interface objects, and shows

Figure 7. The STM for the "arrange a meeting" task

PLAN (<Object> (<Specific Object Type>)) - *causes*
COMPOSE&SENDMESSAGE (<Object> (<Specific Object Type>)) - *enables or FINISH*
 Consult (ISs, ITs, Message) - *enables*
 identify (ISs, ITs, Message) - *enables*
 search (ISs, ITs, Message) - *causes*
 retrieve (ISs, ITs, Message) - *enables*
 storeIn (ITs, Message, computer)
 Select (Medium, Message) - *enables*
 identify (LOC, Constraints, Message) - *enables*
 choose (Medium, Constraints)
 Complete (Message, ISs) - enables
 fillIn (ITs, Message, ISs)
 Execute (TransactionRequirements, Message, Medium)
RECEIVE&PROCESSREPLY (<Object> (<Specific Object Type>)) - *causes or FINISH or fails*
 Notice (Reply, Medium) - enables
 Execute (TransactionRequirements, Reply, Medium) - enables
 Consult (Reply, ITs, <Object>) - *enables*
 search (Reply, ITs, <Object>) - causes
 retrieve (Reply, ITs, <Object>) - enables
 storeIn (ITs, <Object>, LOC)
 Complete (<Object>, Reply) - *causes*
 fillIn (ITs, <Object>, Reply)
 Notice (NewInformation, Reply) - causes
 Record (ITs, NewInformation, Reply)
 establish (ITs, NewInformation, Reply) - *enables*
 fillIn (ITs, <Object>, NewInformation)

how task performance proceeds at the interface by showing the actions of both user and program on these objects.

Figure 8(a) is a summary representation of the STM, without the relations between components, and without the decomposition of Macro-Actions. The decomposition of Macro-Actions is shown in figures 8(b) and 8(c). Figure 8(b) shows the decomposition of the Macro-Actions from the IGF, COMPOSE&SENDMESSAGE. Note that the SIM does not show any decomposition for those Macro-Actions which the STM has shown as unsupported at the interface, in this

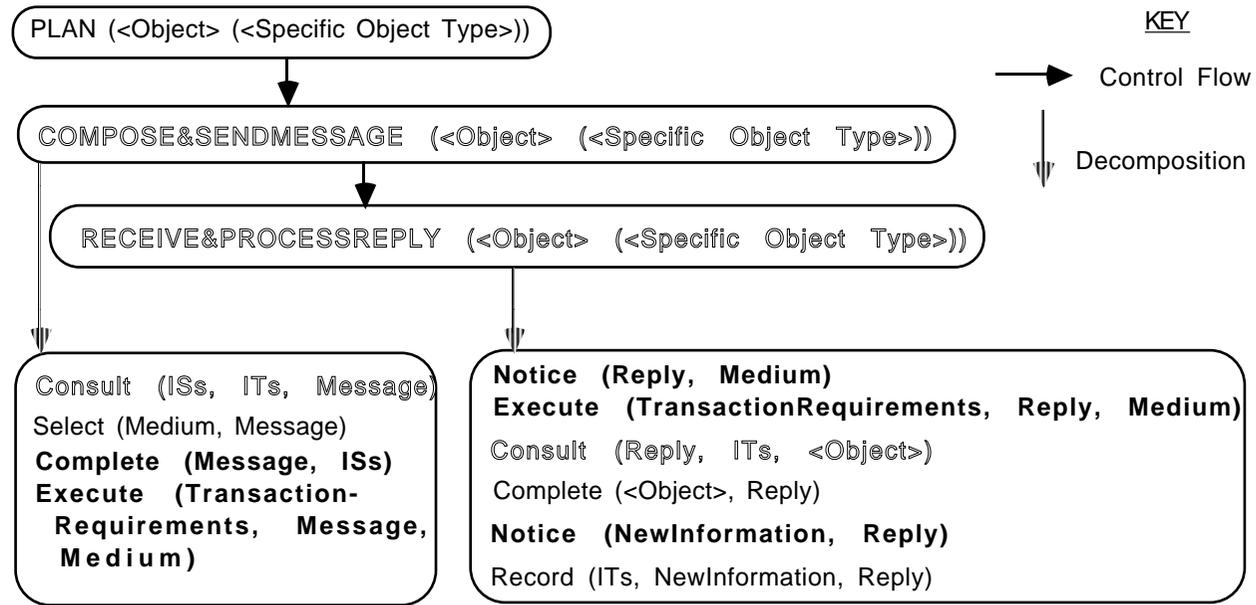


Figure 8(a). The SIM for the "arrange a meeting" task

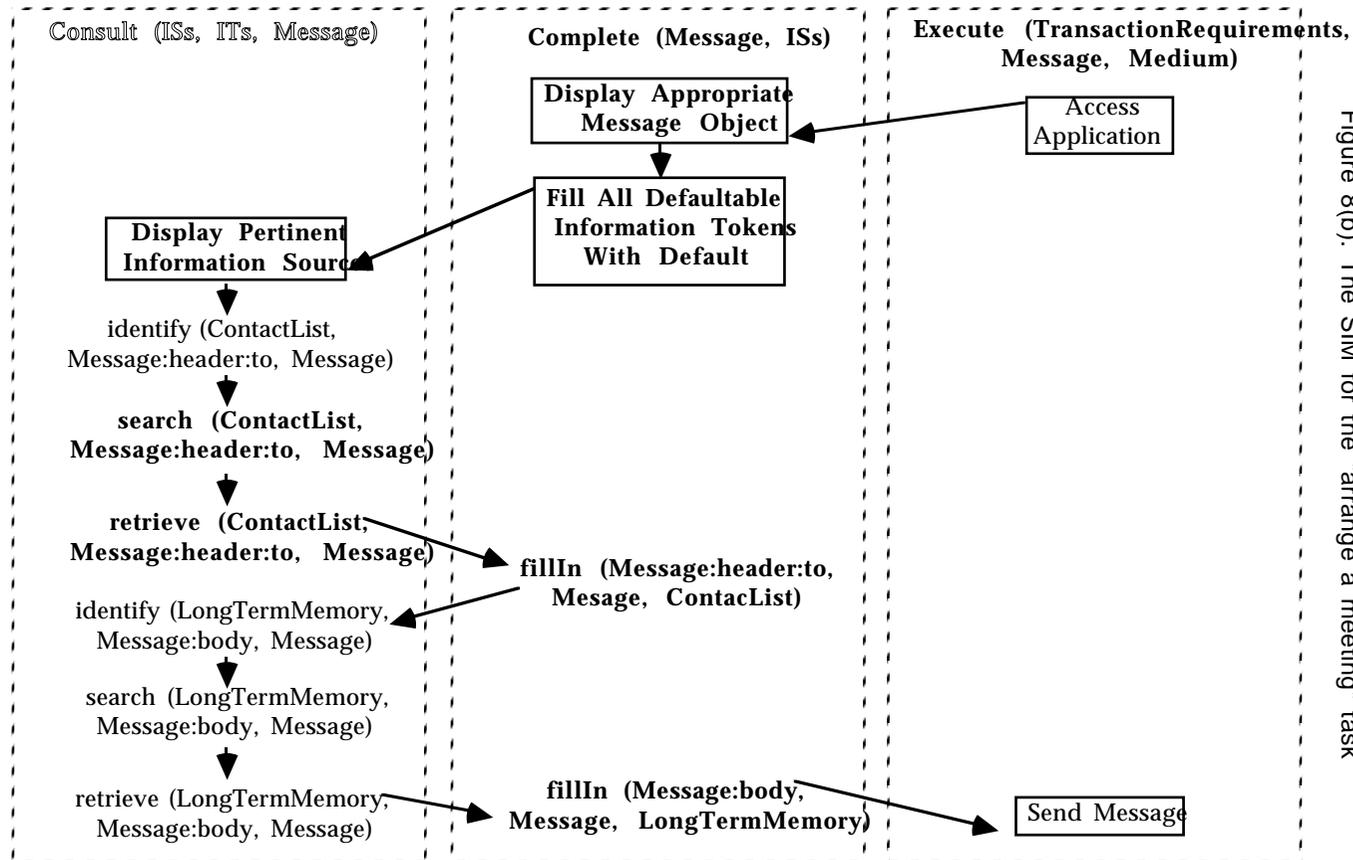
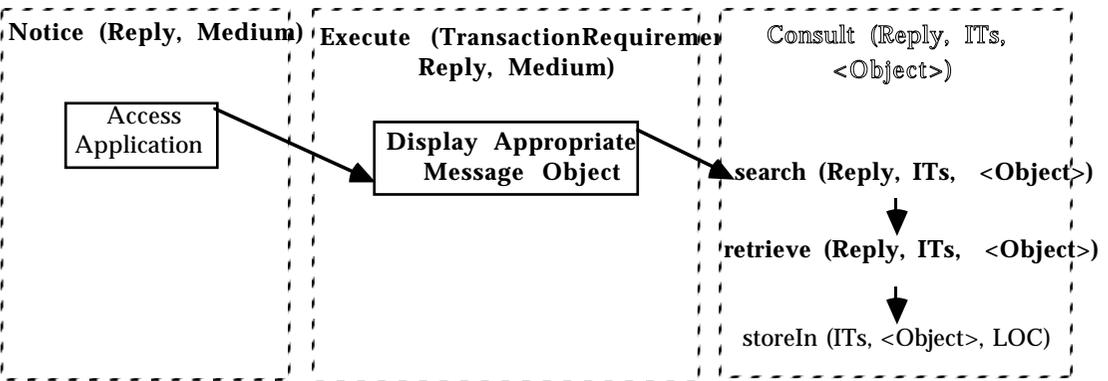


Figure 8(b). The SIM for the "arrange a meeting" task

case the Macro-Action, Select. The decomposition of each Macro-Action is enclosed in a dashed-line box. Any necessary steps in the interaction that do not map directly onto a Micro-Action are shown in a solid-line box. The interpretation of figure 8(b) is as follows. The user first accesses the application, which is part of the Macro-Action, Execute then displays the appropriate Message Object (e.g., memo, telephone message, reply etc.), which is part of the Macro-Action, Complete. At this stage the program responds by filling all defaultable information tokens of the Message Object with their default value. This corresponds to the notion of defaulting referred to above, and, if performed in line with a user's conceptual model, will presumably save effort by performing part of the task automatically. In line with this reasoning the program next automatically displays all information sources for which there is an interface representation and which are pertinent to the particular type of Message Object that the user has chosen to interact with. This is part of the Macro-Action, Consult. The user is then able to perform those parts of the task necessary for giving values to the non-defaultable information tokens of the Message Object, Message:header:to and Message:body. Finally the user must complete the interaction by sending the message, the final component of the Macro-Action, Execute.

Figure 8(c) shows the decomposition of the Macro-Actions from the IGF, RECEIVE&PROCESSREPLY. Again the decomposition is only shown for those Macro-Actions which are wholly or partially supported in the STM. The interpretation is as follows. The user accesses the application in order to perform the Macro-Action, Notice. In order to perform the Macro-Action, Execute, the user must display the appropriate Message Object (i.e., the reply). Finally, to perform the Macro-Action, Consult, the user performs the associated Micro-Actions. Note that there is no representation in this example of the Macro-Action Notice (NewInformation, Reply). This is because that part of the decomposition of the IGF comes from an analysis of tasks more complex than the meeting arranging example, in which potentially many messages are exchanged between roles before the general goal is likely to be completed. The Macro-Action, Notice (NewInformation, Reply), reflects situations where a reply contains information pertinent to the current state of the recipient, for example comparing a reply that contains an alteration to information sent previously. Since a simple meeting arrangement is not likely to include such sequences of behaviour it has been omitted from the SIM in this example.

Figure 8(c). The SIM for the "arrange a meeting" task



9. Summary

In this paper we have described a theory and method of modelling the knowledge people have of tasks and roles in a given domain. We then demonstrated how that model can be used as input to a design

decomposition method which has its own form of modelling. The TKS model is rich in its representation of task knowledge but weak with respect to its design decomposition. It provides an information source to which designers can be given access. It prevents the designer having to rely on their own intuitions about people and their tasks and provides a methodological approach to identifying and modelling task knowledge. The decompositional approach provides a method of applying the results of a TKS model to the design of a computer system by allowing the designer to consistently decompose the design adding detail at each stage of decomposition. Neither the TKS nor the decompositional model prevents the use of conventional software development methods, and the TKS can be used independently of the decompositional method.

Acknowledgements

The work reported here has resulted from research supported, in the case of the TKS by ICL URC funding and in the case of the decompositional method, by SERC/Alvey grant no. GR/D/75779MMI-122.

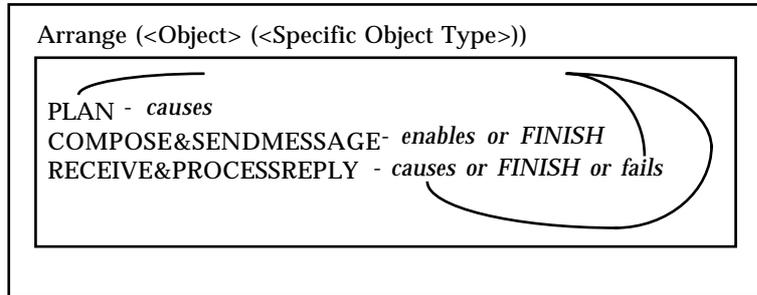
References

- [1] Byrne, R. (1977) Planning meals: Problem solving on a real data-base. Cognition (5), 287-332.
- [2] Card, S.K., Moran, T.P. & Newell, A. (1983) The psychology of human-computer interaction. Hillsdale, N.J.: Lawrence Erlbaum Associates.
- [3] Galambos, J.A. (1986) Knowledge structures for common activities. In Galambos, J.A., Abelson, R.P. & J.B. Black (eds) Knowledge structures. Hillsdale, New Jersey: LEA.
- [4] Garner, W.R. (1974) The processing of information and structure. New York: Wiley.
- [5] Graesser, A.C. & Clark, L.F. (1985) Structures and procedures of implicit knowledge. Norwood, New Jersey. Ablex Publishing Corpn.

- [6] Green, T.R.G. Bellamy, R.K.E. & Parker, J.M. (1987) Parsing and gnisrap: a model of device use. In Bullinger, H.-J. & Shackel, B. (eds) INTERACT 87 Proceedings of the Second IFIP Conference on Human-Computer Interaction, 1-4 September. Elsevier:North Holland.
- [7] Johnson, P. Diaper, D. & Long, J. (1984) Tasks, Skill & Knowledge; Task analysis for knowledge based descriptions. In B. Shackel (ed) Interact84. London North-Holland.
- [8] Johnson, P., Johnson, H., & Russell, A. (1988) Collecting and generalising knowledge descriptions from task analysis data. ICL Technical Journal, in press.
- [9] Keane, M. & Johnson, P. (1987) Preliminary Analysis for Design. In D. Diaper & R. Winder (eds). People and Computers. Cambridge: Cambridge University Press.
- [10] Kieras, D & Polson P.G. (1985) An approach to the formal analysis of user complexity. International Journal of Man-Machine Studies, 22, 365-394.
- [11] Leddo J. & Abelson R.P. (1986) The Nature of explanations. In Galambos, J.A., Abelson, R.P. & J.B. Black (eds) Knowledge structures. Hillsdale, New Jersey: LEA.
- [12] Payne, S. J. & Green, T. R. G. (1986) Task-Action Grammars: a model of the mental representation of task languages. Human Computer Interaction , 2, 93-133.
- [13] Rosch, E. (1985) Prototype classification and logical classification: The two systems. In E.K. Scholnick (ed) New trends in conceptual representation: Challenges to Piaget's theory? New Jersey: Lawrence Erlbaum.
- [14] Rosch, E., Mervis, C., Gray, W., Johnson, D., & Boyes-Braem, P. (1976) Basic objects in natural categories. Cognitive Psychology, 8, 382-439.
- [15] Schank, R.C. (1982) Dynamic Memory: A theory of reminding and learning in computers and people. New York: Cambridge University press.

Appendix A. The Full GTM for the "arrange a meeting" Task

General Goal Frame



Instrumental Goal Frames

PLAN (<Object> (<Specific Object Type>))
Consult (LOC, ITs, <Object>)- <i>enables</i>
Complete (<Object>, ISs)- <i>enables</i>
Decide (ITs)

COMPOSE&SENDMESSAGE (<Object> (<Specific Object Type>))
Consult (ISs, ITs, Message)- <i>enables</i>
Select (Medium, Message)- <i>enables</i>
Complete (Message, ISs)- <i>enables</i>
Execute (TransactionRequirements, Message, Medium)

RECEIVE&PROCESSREPLY (<Object> (<Specific Object Type>))
Notice (Reply, Medium)- <i>enables</i>
Execute (TransactionRequirements, Reply, Medium)- <i>enables</i>
Consult (Reply, ITs, <Object>)- <i>enables</i>
Complete (<Object>, Reply)- <i>causes</i>
Notice (NewInformation, Reply)- <i>causes</i>
Record (ITs, NewInformation, Reply)

Micro-Actions decomposed into Micro-Actions

Consult (LOC, ITs, <Object>)

identify (LOC, ITs, <Object>) - *enables*
search (LOC, ITs) - *causes*
retrieve (LOC, ITs) - *enables*
storeIn (ITs, <Object>, LOC)

Complete (<Object>, ISs)

fillIn (ITs, <Object>, ISs)

Decide (ITs)

(no micro-actions specified see strategy list)

Select (<Object1>, <Object2>)

identify (LOC, Constraints, <Object2>) - *enables*
choose (<Object1>, Constraints)

Execute (TransactionRequirements, <Object>, LOC)

(no micro-actions specified see strategy list)

Notice (<Object>, LOC)

(no micro-actions specified see strategy list)

Record (ITs, NewInformation, <Object>)

establish (ITs, NewInformation, <Object>) - *enables*
fillIn (ITs, <Object>, NewInformation)

Interpretation of Micro-Actions

choose (from alternative possibilities of <some object>, guided by <some object(s)>).

compare (<some object(s)>, derived from <some information source(s)>, with <some object(s)>).

establish (which <information token(s)> relate to <some object(s)> derived from <some information source(s)>).

fillIn (fill values for <some information token(s)>, associated with <some object(s)>, using <some information source(s)>).

identify (<some location(s)>, as the place to find <some information token(s)>, derived from <some object(s)>).

retrieve (from <some location(s)>, <some information token(s)>).

search (<some location(s)>, for <some information token(s)>).

storeIn (store <some information token(s)>, associated with <some object>, which is in <some location>).

Strategy Lists

Decide (ITs)

IF deciding Message:body or Reply:body
THEN default <all new/updated information pertinent to current goal,
requests/suggestions for missing Information Tokens pertinent to
current goal, replies to outstanding messages pertinent to current
goal>
IF replying to a message
THEN acknowledge original and give best reply
IF discrepancies/constraints arisen
THEN give best resolution
IF deciding Message:to or Reply:to
THEN default <all people affected or potentially affected by current
goal>
IF multiple copies AND recipients have an importance hierarchy
THEN prioritise participants and contact highest priority first
(default <no priorities>)
IF replying to a message
THEN reply as requested (default <Message:from>)
IF discrepancies/constraints arisen
THEN contact all people affected or potentially affected by
discrepancies/constraints (default <all people affected by current
goal>)

Execute (TransactionRequirements, <Object>, LOC)

IF sending a Message
THEN default <technology dictates precise TransactionRequirements>
IF Message:body refers to enclosures by a different Medium
THEN send enclosures in that Medium
IF receiving a Message/Reply
THEN default <technology dictates precise TransactionRequirements>
IF Reply:body refers to enclosures in a different Medium
THEN receive enclosures in that Medium

Notice (<Object>, LOC)

default <as part of everyday activities in this domain>
IF <Object> referred to in some previously encountered <Object>
THEN notice <Object> intentionally as part of current goal

Example of an Object

Message

```
location: envelope|computer|person|otherPeople
          |notePaper
informationSource: LongTermMemory, ContactList,
                  Diary, Reply,
                  PreviousCorrespondence
header: <from: <Self> address: <SelfAddress>
        to: date: <Today> re: >
body: <dictated by current goal>
footer : <signature>
```