6

Comparing Task Models for User Interface Design

Quentin Limbourg
Université catholique de Louvain

Jean Vanderdonckt
Université catholique de Louvain

Many task models, task analysis methods, and supporting tools have been introduced in the literature and are widely used in practice. With this comes need to understand their scopes and their differences. This chapter provides a thorough review of selected, significant task models along with their method and supporting tools. For this purpose, a meta-model of each task model is expressed as an Entity-Relationship-Attribute schema (ERA) and discussed. This leads to a comparative analysis of task models according to aims and goals, discipline, concepts and relationships, expressiveness of static and dynamic structures. Following is discussion of the model with respect to developing life cycle steps, tool support, advantages, and shortcomings. This comparative analysis provides a reference framework against which task models can be understood with respect to each other. The appreciation of the similarities and the differences allows practioners to identify a task model that fits a situation's given requirements. It shows how a similar concept or relationship translates in different task usage models.

6.1 INTRODUCTION

User-Centered Design (UCD) has yielded many forms of design practices in which various characteristics of the context of use are considered. Among these, task analysis is widely recognized as one fundamental way not only to ensure some user-centered design (Hackos & Redish, 1998) but to improve the understanding of how a user may interact with a user interface to accomplish a given interactive task. A task model is often defined as a description of an interactive task to be performed by the user of an application through the application's user interface. Individual elements in a task model represent specific actions that the user may undertake. Information on subtask ordering as well as conditions on task execution is also included in the model.

Task analysis methods have been introduced from disciplines with different backgrounds, different concerns, and different focuses on task. The disciplines include:

Cognitive psychology or *ergonomics* (Stanton & Young, 1998). Task models are used to improve the understanding of how users may interact with a given user interface for carrying out a particular interactive task. Task analysis is useful for identifying the cognitive processes (e.g., data manipulation, thinking, problem solving) and structures (e.g., the intellectual skills and knowledge of a task) exploited by a user when carrying out a task and for showing how a user may dynamically change them as the task proceeds (Johnson, Diaper, & Long, 1984). It can also be used to predict cognitive load and rectify usability flaws.
Task planning and allocation. Task models are used to assess task workload, to plan and allocate tasks to users in a particular organization, and to provide indicators to redesign work allocation to fit time, space, and other available resources (Kirwan & Ainsworth, 1992).
Software engineering. Task models can capture relevant task information in an operational form that is machine understandable (see chap. 1). This is especially useful where a system needs to maintain an internal task representation for dynamic purposes, such as to enable adaptation to variations in the context of use (Lewis & Rieman, 1994; Smith & O'Neill, 1996).
Ethnography. Task models can focus on how humans interact with a particular user interface in a given context of use, possibly interacting with other users at the same time (see chap. 14).

Existing task models show a great diversity in terms of formalism and depth of analysis. They also are used to achieve a range of objectives (Bomsdorf & Szwillus, 1998, 1999):

- *To inform* designers about potential usability problems, as in HTA (Annett & Duncan, 1967; chap. 3).
- *To evaluate* human performance, as in GOMS (Card, Moran, & Newell, 1983; chap. 4).
- *To support design* by providing a detailed task model describing task hierarchy, objects used, and knowledge structures, as in TKS (Johnson, 1992; chap. 15) or CTT (Paternò, 1999; chap. 24).
- *To generate* a prototype of a user interface, as in the Adept approach (Wilson & Johnson, 1996; chap. 24).

These different objectives give rise to many interesting concepts. For example, task analysis methods used in cognitive analysis go beyond the goal level to analyze the cognitive workload, execution time, or knowledge required to carry out a set of tasks. In this respect, they are similar to user modeling. On the other hand, methods intended to support cooperative work have developed formalisms to represent how tasks are assigned to different roles, broadening the scope of task analysis with organizational concepts. This situation leads to a series of shortcomings (see also chap. 30):

Lack of understanding of the basic contents of each individual task model, including the rationale behind the analysis method, the concepts, their relationships, their vocabularies, and the intellectual operations involved.
Heterogeneity of the contents of different models due to the variety of methods employed by different disciplines. The heterogeneity encompasses different focuses, different vocabularies, and different definitions of what a task model may be.
Difficulty in matching the contents of two or more analyses done using different task analysis models. Sometimes no matching between contents can be established at all. As each method

PT: FCH/SFH P2: FCH
LE071-06 LE071-Diaper & Stanton LE071-Diaper-v2.cls July 30, 2003 11:20 Char Count= 0

has its own vocabulary, it is difficult to relate the vocabularies of different methods to see what they cover and what they do not.

Lack of interoperability of systems. Since task-modeling tools do not share a common format, they are restricted to those task models expressed according to their accepted formats.

Reduced communication among task analysts. Owing to the lack of interoperability, a task analyst may experience trouble in communicating the results of a task analysis to another analyst or any other stakeholder. In addition, any transition between two persons may generate inconsistencies, errors, misunderstandings, or inappropriate modeling.

Difficulty in identifying frontiers of task analysis. Some methods are more focused on some specific aspects of the task to be represented whereas others tend to go beyond the limits of the task by encompassing parameters external to the task yet relevant to the context of use.

To address the above shortcomings, this chapter pursues two major goals. The first, *a theoretical goal*, is to provide a deep conceptual and methodological understanding of each individual task model and its related approach. The second, *an ontological goal*, is to establish semantic mappings between the different individual task models so as to create a transverse understanding of their underlying concepts independently of their particularities. This goal involves many activities such as vocabulary translation, expressiveness analysis, degree of details, identification of concepts, emergence of transversal concepts, and task structuring identification.

6.2 A REVIEW OF TASK MODELS

In general, any task analysis method may involve three related poles:

1. *Models*. A model captures some facets of the problem and translates them into specifications.
2. *A stepwise approach*. An approach in which a sequence of steps is used to work on models.
3. *Software tools*. A software tool supports the approach by manipulating the appropriate models.

Any task analysis method may contain representative parts that fall within each of these three poles. This comparative study is focused on the first pole, that is, on models. It is assumed that the structuring of a method's steps for modeling tasks should remain independent of the task model's contents. Therefore, the methodological part of each task model was taken to fall outside the scope of our analysis. A tool clearly facilitates the task modeling activity in hiding the model notation from the analyst and helping him or her capture it, edit it for any modification, and exploit it for future use (e.g., task simulation, user interface derivation). Most models presented below are software supported.

Lots of task models have been proposed in the literature, and some are described in this handbook: CTA (chap. 15 and 16), CTT (chap. 24; Paternò, 1999), Diane+ (chaps. 22 and 26; Barthet & Tarby, 1996; Lu, Paris, & Vander Linden, 1998), GOMS (chap. 4; John & Kieras, 1996), GTA (chaps. 7 and 24; van Welie & van der Veer, 1998; van Welie, van der Veer, & Eliëns, 1998), HTA (chap. 3; Annett & Duncan, 1967; Shepherd, 1989, 1995), MAD* (chaps. 22 and 24; Scapin & Pierret-Golbreich, 1989; Gamboa & Scapin, 1997; Scapin & Bastien, 2001), MUSE (Lim & Long, 1994), TAG (Payne & Green, 1986), TAKD (Diaper, 1989a),

TKS (Johnson, 1992; see also Chapter 15), TOOD (chap. 25; Mahfoudhi, Abed, & Tabary, 2001).

A subset of task models were selected to reflect the disciplinary variety. Each significant discipline is represented by one member. The set is also intended to reflect the geographical range of the task models. In addition, each selected task model is integrated in a development methodology as a core or side element, and each has been submitted to experimental studies to assess its validity.

After selecting the models, we analyzed the foundation papers on each. We then decomposed each model into constituent concepts using an entity-relationship-attribute (ERA) method of analysis so as to obtain a task metamodel (see the appendix at the end of this chapter). Task models invoke concepts at different levels of importance. A concept that is similar across methods can be modeled as an entity, a relationship, or even as an attribute. We decided that these concepts should be represented in a consistent manner. For example, a temporal operator was always represented as an entity to recognize an equal importance of this concept throughout the different task metamodels. Each pertinent concept was then precisely defined and commented on. The initial terminology of the originating papers was kept for naming identified concepts.

6.2.1 Hierarchical Task Analysis (HTA)

Hierarchical Task Analysis (HTA; Annett & Duncan, 1967; chap. 3) was a pioneering method of task analysis. It was primarily aimed at training users to perform particular tasks. On the basis of interviews, user observation, and analysis of existing documents (e.g., manuals, documentation), HTA describes tasks in terms of three main concepts (Fig. 6.1): tasks, task hierarchy, and plans. Tasks are recursively decomposed into subtasks to a point where subtasks are allocated either to the user or the user interface, thus becoming observable. The task hierarchy statically represents this task decomposition. The decomposition stopping criterion is a rule of thumb referred to the $p \times c$ rule. This criterion takes into account the probability of a nonsatisfactory performance and the cost of a nonsatisfactory performance (i.e., the consequences it might produce).

Since the task hierarchy does not contain any task ordering, any task should be accomplished according to a plan describable in terms of rules, skills, and knowledge. A plan specifies an ordering in which subtasks of a given task could be carried on, thus acting as a constraint on task performance.

A plan is provided for each hierarchic level. Although the plan is an informal description of temporal relationships between tasks, it is one of the most attractive features of HTA, as

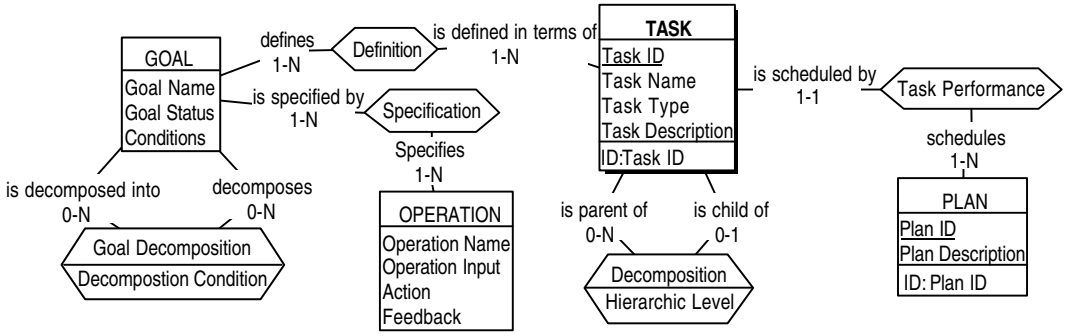

FIG. 6.1. The HTA task meta-model.

it is both simple and expressive. Plans are very close to textual description or to the activity list (chap. 1) of traditional task analysis. One advantage of plans is that they do not create any artificial tasks, as some formal notations force analysts' to do to avoid ambiguous specification. On the other hand, because plans are informal, it is not possible to apply automatic checking of properties such as consistency and reachability.

Any task can be expressed in terms of goals that are reached when the corresponding task is accomplished. Each goal has a status (i.e., latent or active) and conditions to be satisfied. The advantage here in HTA is that goals are independent of the concrete means of reaching them. Therefore, for each goal at any level of decomposition, For each goal, several different operations for reaching the goal can be imagined and specified. Each operation is consequently related to a goal (or goals) and is further specified by the circumstances in which the goal is activated (the input), the activities (action) that contribute to goal attainment, and the conditions indicating the goal has been attained (feedback).

HTA provides a graphical representation of labeled tasks and a plan for each hierarchic level explaining the possible sequences of tasks and the conditions under which each sequence is executed. HTA also supports task analysis for teamwork, as described in Annett, Cunningham, and Mathias-Jones (2000; see also chap. 3).

6.2.2 Goals, Operators, Methods, and Selection rules (GOMS)

Card et al. (1983) developed GOMS as an engineering model for human performance to enable quantitative predictions (chap. 4). By incorporating tables of parameter values that rely on a cognitive architecture, GOMS can be used as an engineering approach to task design (Beard, Smith, & Denelsbeck, 1996). The original GOMS model, referred as CMN-GOMS (Card et al., 1983), is the root of a family of models that were elaborated later (John & Kieras, 1996), such as GOMSL (GOMS language) and CPM-GOMS (Critical Path Method GOMS). Although the first uses a "mental programming language" and is based on a parallel cognitive architecture, the second uses a PERT chart to identify the critical path for computing execution time (Baumeister, John, & Byrne, 2000).

In GOMS, the concept of a *method* is essential, as methods are used to describe how tasks are actually carried out (Fig. 6.2). A method is a sequence of operators that describes task performance. Tasks are triggered by goals and can be further decomposed into subtasks corresponding to intermediary goals. When several methods compete for the same goal, a selection rule is used to choose the proper one.

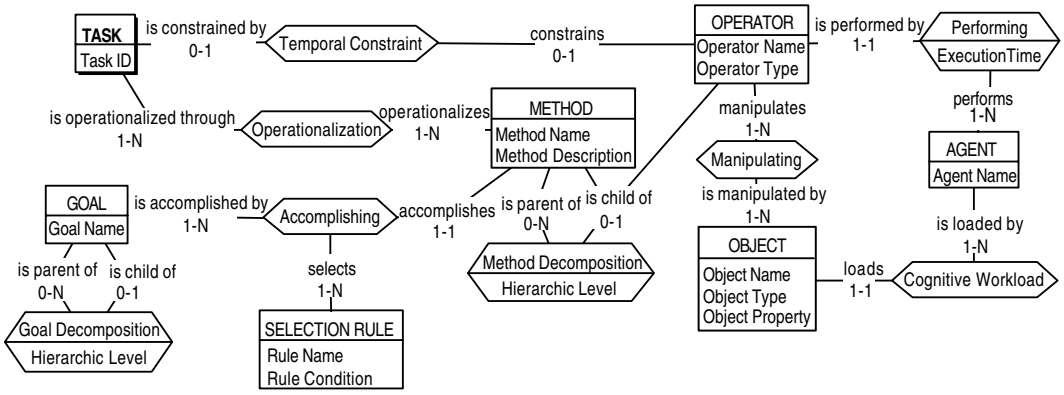

FIG. 6.2. The GOMS task meta-model.

Methods describe how goals are actually accomplished. Higher level methods describe task performance in terms of lower level methods, operators, and selection rules. The lowest level of decomposition in GOMS is the unit task, defined by Card et al. (1983) as a task the user really (consciously) wants to perform. Higher level methods use task flow operators that act as constructors controlling task execution.

GOMS makes a clear distinction between tasks and actions. First, task decomposition stops at unit tasks. Second, actions that in GOMS are termed operators are specified by the methods associated with unit tasks. Action modeling varies depending on the GOMS model and the method specification. Operators are cognitive and physical actions the user has to perform in order to accomplish the task goal. Since each operator has an associated execution time (determined experimentally), a GOMS model can help in predicting the time needed to perform a task.

Actions undertaken by the user are specified using external and mental operators. Some special mental operators are flow-control operators that are used to constrain the execution flow. Although the granularity varies according to the purpose of the analysis, it is clear that GOMS is mainly useful when decomposition is done at operational level (i.e., under the unit task level).

6.2.3 Groupware Task Analysis (GTA)

GroupWare Task Analysis (GTA) was developed by van der Veer, Lenting, and Bergevoet (1996) as a means of modeling the complexity of tasks in a cooperative environment. GTA has roots in ethnography; It is applied to design cooperative systems, and is based on activity theory. It adopts a clear distinction between tasks and actions.

An ontology describing the concepts and relations of the method has been proposed by van Welie, van der Veer, and Eliens (1998). The five central concepts are task, role, object, agent, and event (Fig. 6.3). In GTA, complex tasks are decomposed into unit tasks (Card et al., 1983) and basic tasks (Tauber, 1990). There is no indication how this relates to user interface design.

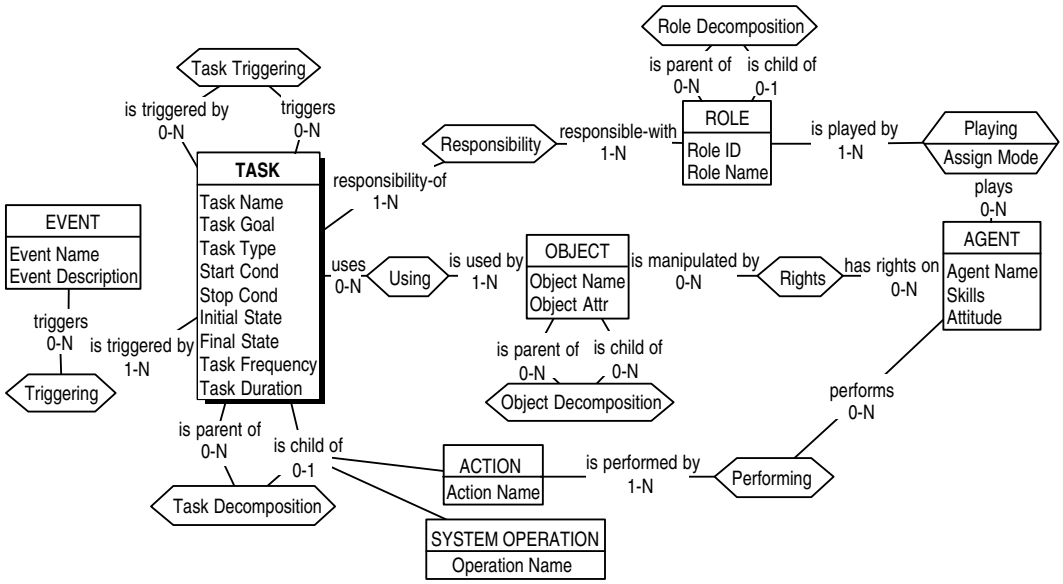

FIG. 6.3. The GTA task meta-model.

More recently, techniques have been added (van Welie, van der Veer, & Eliens, 1998), including a user action notation for the decomposition of basic tasks. An attractive feature of GTA is its capability of representing cooperative tasks (groupware). The representation is done by integrating the role concept in the task world and enabling representation of tasks sets for which a role is responsible of organizational aspects, such as how a role is assigned to different agents.

Although the ontology defined for GTA improves the conceptualization of the task world, the representation is not based on an adequate formalism. For instance, goals and actions are represented as attributes of the task and not as concepts. This is somewhat inconsistent with the fact that GTA allows for a goal to be reached in many ways. Also, since the same action can be used in many tasks, actions are better represented as a main concept. This way, object manipulation is represented as a relationship between actions and objects rather than between tasks and objects. Since actions depend on operational conditions, when different objects are used, different actions may be needed. Any task can also trigger another one. Euterpe, the software tool supporting GTA, allows the specification of constructors in a way similar to MAD*. Like in MAD*, parent-child constructors may lead to artificial tasks to satisfy the temporal constraints.

6.2.4 ConcurTaskTrees (CTT)

ConcurTaskTrees (CTT), a model developed by Paternò (1999; chap. 24), is based on five concepts: tasks, objects, actions, operators, and roles (Fig. 6.4). CTT constructors, termed *operators*, are used to link sibling tasks on the same level of decomposition. In this respect, CTT differs from previously described models, where operators act on parent-children relationships. It is also important to note that CTT has a formal definition for its temporal operators. In addition, CTT provides the means to describe cooperative tasks. To describe such a task, the task model is composed of different task trees, one for the cooperative part and one for each role that is involved in the task. Tasks are further decomposed up to the level of basic tasks, which are defined as tasks that could not be further decomposed. Actions and objects are specified for each basic task. Objects could be perceivable objects or application objects. Application objects are mapped onto perceivable objects in order to be presented to the user.

An interesting feature of CTT is that both input actions and output actions associated with an object are specified. Object specification is mainly directed toward the specification of interaction objects (interactors). The last modification made to CTT was to add the concept of platform in order to support multiplatform user interface development. A task can be associated

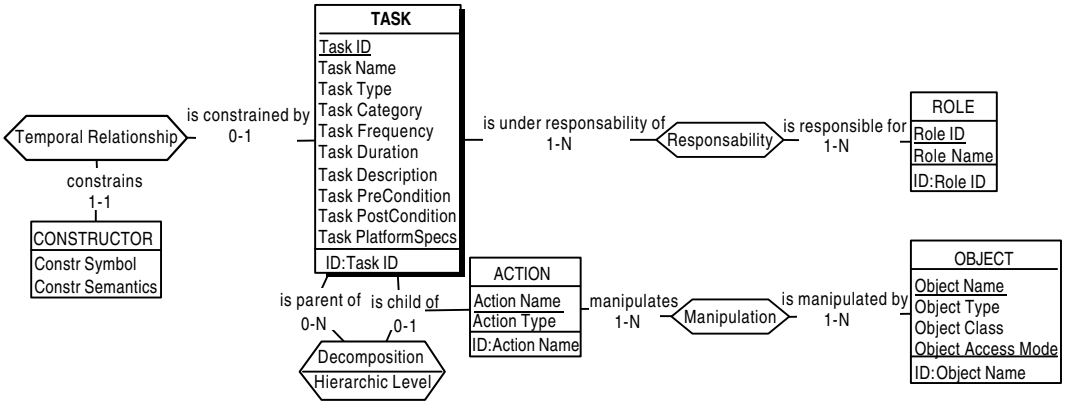

FIG. 6.4. The CTT task meta-model.

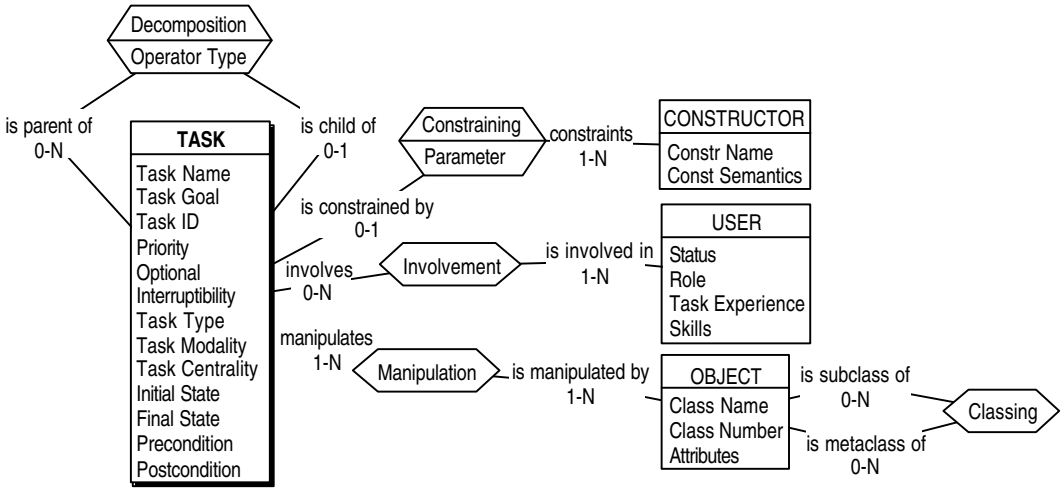

FIG. 6.5. The MAD* task meta-model.

with one or several previously defined platform descriptions. Views on the task model are obtained by filtering the model with respect to one or several platforms. CTT uses a tool for building the task model that specifies tasks, roles, and objects as well as creates a task hierarchy with temporal operators.

6.2.5 Méthode Analytique de Description de tâches (MAD*)

*M*éthode *A*nalytique de *D*escription de tâches (MAD*; Scapin & Bastien, 2001; Scapin & Pierret-Goldbreich, 1989; see also chaps. 22 and 24) provides object-oriented task specifications to support design phases. The main concepts of MAD* are task, constructor, user, and objects (Fig. 6.5). The task structure concept is implicitly represented in the decomposition relationship and the relationship with the constructor entity. As in GTA, tasks are divided into two broad categories: elementary tasks and composite tasks.

Elementary tasks are tasks that cannot be further decomposed. An elementary task contains direct reference to one or many domain objects that are manipulated by the task. A composite task is decomposed into children tasks by the use of different operators belonging to four categories: synchronization operators (i.e., sequence, parallelism, and simultaneity), ordering operators (i.e., AND, OR, and XOR), temporal operators (i.e., begin, end, and duration), and auxiliary operators (i.e., elementary or unknown). An elementary task is specified by an elementary "auxiliary" operator, and a task whose decomposition is not yet terminated is specified by an "unknown" operator.

In MAD*, a task, either composite or elementary, is characterized by several attributes: a name, a goal, an identifier (e.g., "Task 2.5.3 means third subsubtask of fifth subtask of second task"), a priority indicating preemptivity over other tasks, an optional character, a specification indicating whether the task can be interrupted, a type (i.e., sensorimotor, cognitive), a modality (i.e., manual, automatic, interactive), and a degree of centrality (i.e., important, somewhat important, secondary). The initial state specifies a state of the world prior to the execution of the task. The final state specifies a state of the world after the task execution. The goal consists of the object modifications that an agent wants to achieve. Although the goal is attached to the task as an attribute, the same goal can be reached by accomplishing different tasks. This fact motivated putting the goal concept into an entity type.

The precondition is a set of assertions constraining the initial state. Those assertions must be satisfied prior to the execution of the chosen task. Preconditions are classified as either sufficient conditions or as necessary and sufficient conditions. The postcondition is a set of assertions constraining the final state. The postcondition must be satisfied after the execution of the task. Any task can be linked to the user responsible for carrying out the task and to objects manipulated by the task. MAD* is supported by ALACIE, a task editor that allows analysts to input and manipulate their model (Gamboa & Scapin, 1997).

6.2.6 Task Knowledge Structure (TKS)

In the model known as the Task Knowledge Structure (TKS) method (Johnson & Johnson, 1989; chap. 15), the analysts manipulate a TKS, which is a conceptual representation of the knowledge a person has stored in his or her memory about a particular task (Fig. 6.6). A TKS is associated with each task that an agent (i.e., a user) performs. Tasks that an agent is in charge of are determined by the role the agent is presumed to assume. A role is defined as the particular set of tasks the agent is responsible for performing as part of his or her duty in a particular social context. An agent can take on several roles, and a role can be taken on by several agents. Even if tasks or TKSs may seem similar across different roles (e.g., typing a letter for a secretary and a manager), they will be considered as different. The "similarity" relationship is aimed to represent this situation.

The TKS for a task holds information about the task's goal, which is the state of affairs the task is intended to produce. A particular goal is accomplished by a particular task. A goal is decomposed into a goal substructure, which contains all intermediate subgoals needed to achieve it. Goal and task hierarchies are represented in Fig. 6.6 as two overlapping substructures, the first decomposing the goal into subgoals, the second decomposing tasks into subtasks. Each subgoal in the goal structure has a corresponding subtask in the task structure, and vice versa. The structure is composed either by task decomposition or by temporal or causal relationship mechanisms (*constructors*).

Constructors operate on tasks and by association on goals. The same goal can be reached by different subgoal sequencing. This leads to the concept of a plan. A plan is a particular arrangement of a set of subgoals and procedures for achieving a particular goal. As in other models, actions and objects are found at the lowest level of the task analysis. They are the

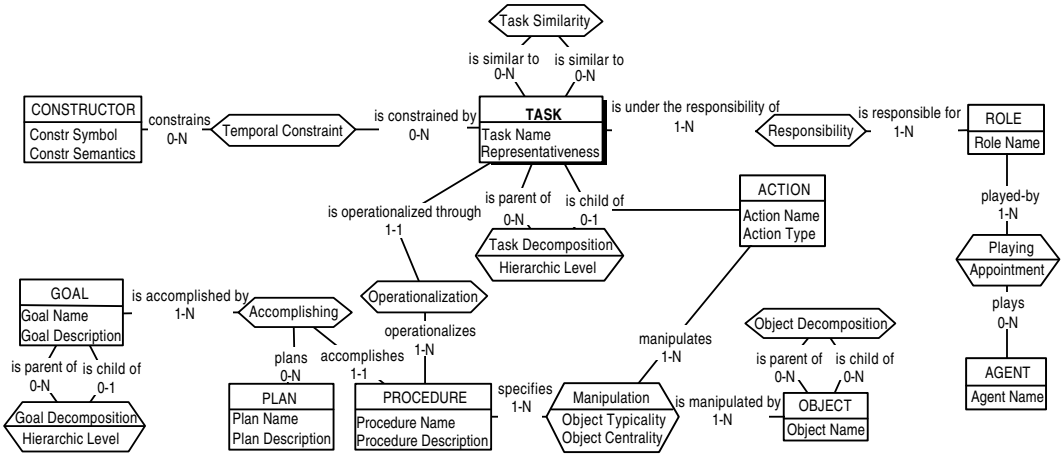

FIG. 6.6. The TKS task meta-model.

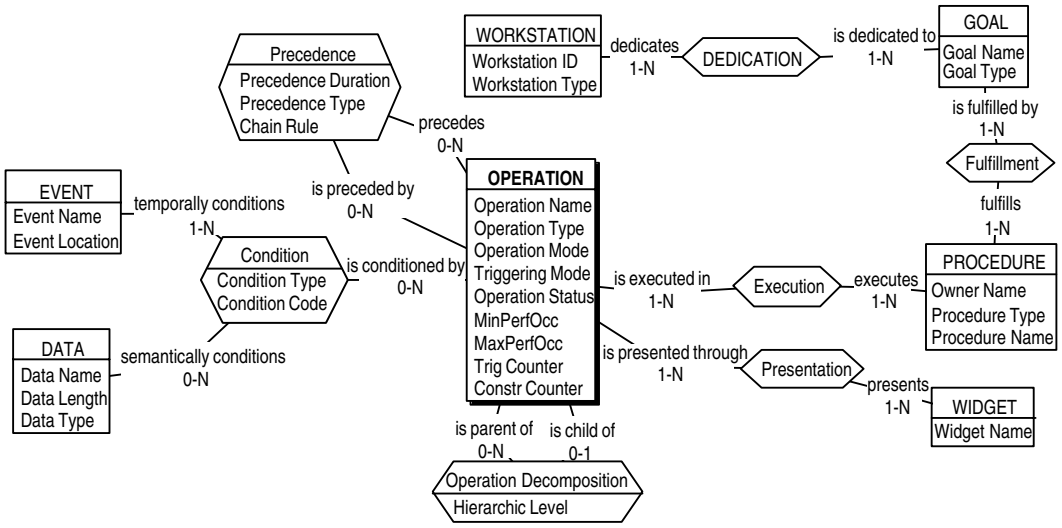

FIG. 6.7. The Diane+ task meta-model.

constituents of procedures that operationalize subtasks. One or several procedures can be linked to the same subtask. The TKS method proposes a production rule system for choosing the appropriate procedures for each context. Actions are directly related to a task tree and form its leaf level. Actions and objects have properties of interest. They can be central to the execution of a task and have typical instances. For example, the centrality and typicality of an object or action are always expressed with respect to a task or procedure that operationalizes the object or action. Note that objects are structured into a decomposition hierarchy.

6.2.7 Diane+

There are two important points to be made about the way in which Diane+ (Fig. 6.7) models a task (Barthet & Tarby, 1996; chap. 26):

1. The procedures describe only the characteristics specific to an application and do include the standard actions common to most applications, such as quit, cancel, and so on. This assumes that the supposed standard actions, previously defined, really apply to the application of interest. (If a standard action does not apply, this would be indicated.)
2. The described procedures are not mandatory; what is not forbidden is allowed.

We note that Diane+ can represent all the constraints of the above specifications. All the algorithmic structures do exist in Diane+, such as ordered sequence, unordered sequence, loop, required choice, free choice, parallelism, default operations, and so on.

6.2.8 Method for USability Engineering (MUSE)

The *M*ethod for *US*ability *E*ngineering (MUSE) is a structured human factors method aimed at helping designers consider human factors in the development of interactive software (Lim & Long, 1994). MUSE consists of three major phases: information elicitation and analysis, design synthesis, and design specification. The method is initiated by the analysis of the extant system, which results into an extant task model. This model is progressively transformed and augmented until eventually an interface model and display design is reached in the last phase.

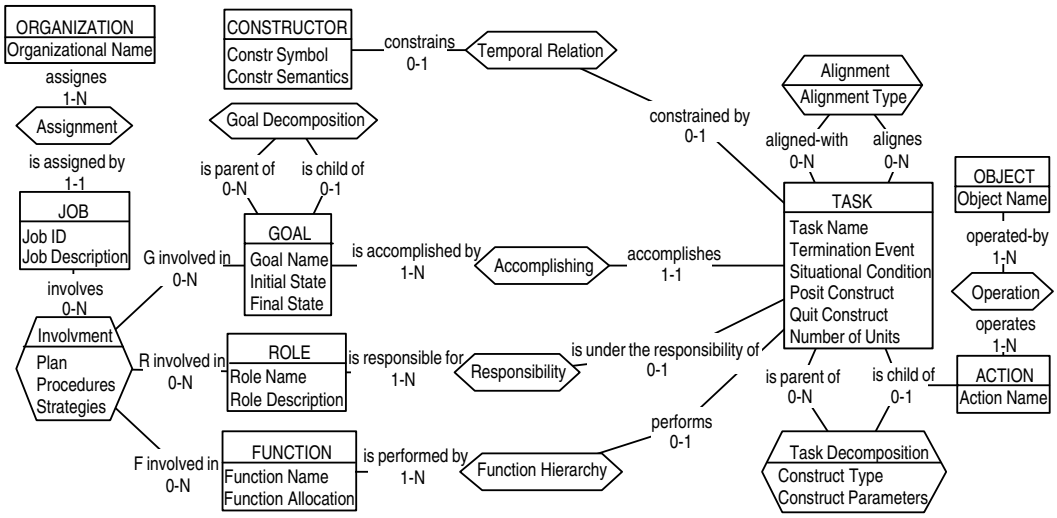

FIG. 6.8. The MUSE task meta-model.

The task model is described using task hierarchies covering several concepts. At the top of the hierarchy is the *organization*, which may be conceptualized as a supersystem to be decomposed into subsystems, called *jobs* (Fig. 6.8). Each job may involve three components:

1. A *goal hierarchy*, in which the main goal assigned to a job is recursively decomposed into subgoals, subgoals into subsubgoals, and so on. The main goal of a job describes the related subsystem as a transformation from its initial state to its final state.
2. A *role list*, in which roles assigned to a job are enumerated.
3. A *function list*, in which the specific functions required to perform the role are gathered. A function may be allocated to a human, a system, or both.

Each role and function appearing in the function lists is associated with a task, which is again hierarchically decomposed into subtasks, subsubtasks, and so on, down to the level of actions and objects. Each task is detailed with a termination event, a situational condition. The detection of unacceptable situational conditions may serve as a shortcut for terminating a path in the task tree. These constructs are used to describe uncertain events that may occur while the task is carried out. A number of units describes when a task is multiple (i.e., a instance of the same task carried out many times). Since the graphical representation of a MUSE task model is purely hierarchical, a single action composing multiple tasks or a single object operated by multiple actions may be reproduced. Other constructs related to the task decomposition and potentially requiring parameters are sequence, hierarchy, selection, iteration, concurrency, interleaving, multiplicity, posit, and quit. Figure 6.8 presents more concepts than are involved in the task modeling; the right part is the core component leading to more refined task models (Lim & Long, 1996).

6.2.9 Task Object-Oriented Description (TOOD)

Task Object-Oriented Description (TOOD) consists of an object-oriented method for modeling tasks in the domain of control processes and complex interactive systems, such as those used in air traffic control (Mahfoudhi, Abed, & Tabary, 2001; chap. 25). The method consists of

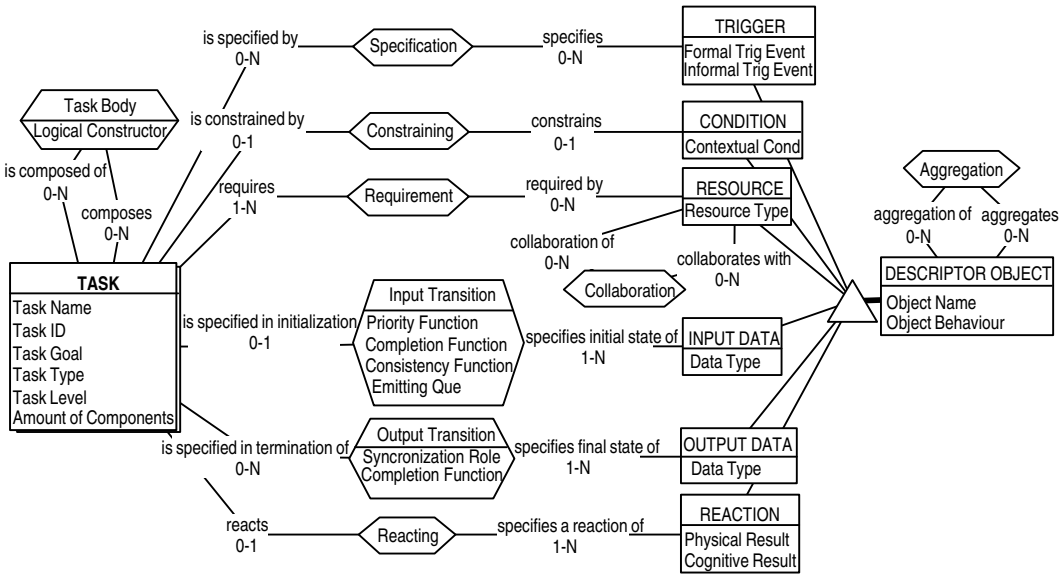

FIG. 6.9. The TOOD task meta-model.

four steps: hierarchical decomposition of tasks, identification of descriptor objects and world objects, definition of elementary and control tasks, and integration of concurrency (Fig. 6.9). Each task is treated as an instance of a task class identified by a name and an identifier and characterized by a goal, a type (i.e., human, automatic, interactive, and cooperative), the level in the hierarchy, and the total amount of task components. The task body represents a task hierarchy organized using three logical constructors (i.e., AND, OR, and XOR). Each task is then associated with a task control structure (TCS) made up of six classes of descriptor objects that are consumed when the task is carried out and they are aggregated:

1. The *triggering* class has four types of events: formal and informal events, events occuring outside and inside the system.
2. The *condition* class contains contextual conditions governing the performance of the task.
3. The *resource* class describes resources (human or system) required for the task to be performed.
4. The *input data* class specifies information items required for performance of the task. To initialize a task, an input transition expresses logical conditions on these data by sending rules and benefits from various checking functions to ensure that all conditions required to perform the task are fulfilled. For instance, the completeness function checks that all input data are available and satisfy related constraints.
5. The *output data* class specifies information items produced by the task performance. To terminate a task, an output transition expresses logical conditions on these data through synchronization rules and benefits from various checking functions.
6. The *reaction* class describes physical and cognitive results resulting from the task performance.

The combination of TOOD descriptor objects covers task hierarchy and temporal ordering. TOOD is supported by a graphical editor allowing analysts to specify instances of task classes as well as instances of their related classes.

P1: FCH/SPH P2: FCH
LE071-06 LE071-Diaper & Stanton LE071-Diaper-v2.cls July 30, 2003 11:20 Char Count= 0

6.3 CONCLUSION

The task models analysed in previous sections exhibit a variety of concepts and relationships. The differences between concepts are both syntactic and semantic.

6.3.1 Syntactic Differences

Syntactic differences include differences of vocabulary used for the same concept across models. The most notable syntactic differences are summed up in Table 6.1. The comparison is based on four model features: task planning (how a task is related to high-level goal to be planned), operationalization capacity (the extent to which the method provides ways to map high-level goals to low-level ways of reaching them), lowest task decomposition level (the leaf node in the task decomposition), and operational level (the task decomposition level where actions take place).

It can be observed from Table 6.1 that similar or different terms can be used to refer to the same concepts. For example, *plan, operator, constructor*, and *goal* are often used for discussing high-level task planning. Although most models do provide for task decomposition, both structurally and temporally, they do not necessarily describe how decomposition can be effectively carried out. For instance, a scenario is frequently considered to be a particular instantiation of a general task model that depends on particular circumstances within the context of use.

The two last rows describe how a task is recursively decomposed into subtasks to end up with leaf nodes. Some models do not make terminological distinction between different levels of decomposition. On the other hand, GOMS, GTA, CTT, and MUSE do separate any nonterminal level of decomposition from leaf nodes, which usually bear a special name (often, *action*).

6.3.2 Semantic Differences

Semantic differences are related to conceptual variation across models. Semantic differences can be of major or minor importance. Of major importance are the differences in entity or relationship definitions and coverage, (e.g., cases where the same concept is not defined in the same way across models). Less consequential is the variation in how an entity or a relationship is expressed. For example, constructors in GTA, MAD*, or TKS express the temporal relationship between a task and its subtasks (although the set of constructors is not identical in all models), whereas operators in CTT are used between sibling tasks. Operators used in GOMS have a dual semantics: They specify actions (cognitive and motor) performed by the user, and they are also used as syntactic constructions of the language to control task flow, similar to a programming language. Table 6.2 compares task models along the following dimensions:

Discipline of origin. The discipline of origin has an impact on the concepts that will be found in a particular model. In particular, HTA and GOMS are deeply rooted in cognitive analysis. Models rooted in cognitive psychology or related disciplines focus on cognitive concepts and avoid software artifacts, whereas software engineering models do the reverse.
Formalization. This dimension specifies whether a model is based on a formal system or not. For instance, CTT temporal operators (chap. 24) are defined in process algebra, TOOD (chap. 25) contains mathematical definitions of transitions, and Task Layer Maps (chap. 11) has a formal basis.
Collaborative aspects. In order to describe cooperative aspects, models use a role concept. A role is defined by the tasks the role is responsible for. Roles are played by agents and are

TABLE 6.1
Main Task Model Features

	HTA	*GOMS*	*GTA*	*CTT*	*MAD**	*TKS*	*Diane+*	*TOOD*	*MUSE*
Task planning	Plans	Operators	Constructors	Operators	Constructors	Plans/constructors	Goals	Input/output transitions	Goals and constructors
Operationalization		Methods/ selection rules		Scenarios	Pre- and postconditions	Procedures	Procedures		
Task tree leaves		Unit tasks	Basic tasks	Basic tasks		Actions			Actions
Operational level	Tasks	Operators	Actions/system operations	Actions	Tasks		Operations	Task	Task

TABLE 6.2
Main Semantic Variations Between Models

	Origin	*Formalisation*	*Collaborative aspects*	*Context of use variation*	*Cognitive aspects*	*System Response*	*Scope of constructors*	*Manipulated objects*
HTA	Cognitive analysis	Ø	√	Ø	Usability problems	√	Parent	Reference/task object
GOMS	Cognitive analysis	Ø	Ø	Ø	User performance	Ø	Multiple levels	Reference/task object
GTA	Ethnography, cognitive analysis, & CSCW		√ (roles/agents)	Ø	User performance	√	Parent	Object
CTT	Software engineering	√	√ (roles & cooperative parts)	√ (platforms)	Ø	√	Sibling	Detailed/task objects
TKS	Cognitive analysis & software engineering	Ø	√ (roles/agents)	√ (user)	Knowledge structures	Ø	Multiple levels	Detailed/task objects
MAD*	Psychology	√	Ø	Ø	Ø	√	Parent	Detailed/task objects
Diane+	Process control & software engineering	√	Ø	√ (devices)	Ø	√	Sibling	Object
TOOD	Process control	√	Can be expressed in transitions	Ø	Cognitive result in a reaction	√	Sibling	Resource, input data, and output data
MUSE	Software engineering & human factors	√	Ø	Ø	Ø	√	Parent	Object

Note. √ = supported, Ø = unsupported.

assigned according to organizational rules. They are represented by task trees constructed by performing several task decompositions. In the case of a cooperative role, the task hierarchy represents only the cooperative part of the activity. Since a user interface is designed for a given role, this distinction in task decomposition is useful both for user interface design and for coordinating the computer-supported cooperative work.

Context of use variation. Because the user community and the number of computing platforms and working environments are increasing, context-sensitive user interfaces are becoming more important to support task variations resulting from differences in the contexts of use. The importance given to various external elements varies from one system to another. Some approaches place an emphasis on the context of use (e.g., TKS, GTA, and, to a degree, CTT). For instance, the users' characteristics, the organization, the computing platform, and the physical environment are taken into consideration to develop a usable system. It is worth noting that several task models have been modified to enable them to characterize aspects relevant to context sensitivity. For instance, the Unified Design Method (Savidis, Akoumianakis, & Stephanidis, 2001) introduced the mechanism of polymorphic task models to integrate variations that resulted from the consideration of different design alternatives—alternatives that resulted from addressing the needs of several categories of users in different contexts of use. Similarly, CTT has been extended where particular subtasks are selected based on conditions resulting from context variations (Paternò & Santoro, 2002; Thevenin, 2001). The idea of conditional subtrees also leads to the identification and formal definition of context-dependent, context-independent, and partially context-independent decompositions, as oulined in Souchon, Limbourg, and Vanderdonckt (2002).

Cognitive aspects. This dimension concerns the incorporation and/or support of cognitive aspects in modeling activities.

System response. This dimension determines whether semantic functions from the technical system can be identified and embodied n the modeling. Some models remain open by not distinguishing any type of (sub)task (e.g., HTA), whereas others effectively pursue this goal (e.g., Diane+).

Scope of constructors. This dimension expresses the scope of the task elements on which the temporal operators work. The scope can be the *parent* or the *sibling* when any temporal operator constraint affects the ordering, respectively, between a father node in the task decomposition and its children (as in HTA) or between siblings of the same father (as in CTT). Sometimes the scope goes beyond two levels in the task hierarchy, in which case multiple levels may be required to expressed a temporal ordering (such as in Dittmar, 2000).

Manipulated objects. Although the *task model* is intended to represent how a particular task can be carried out by a particular user stereotype in a certain context of use, a *domain model* is frequently used to refer to the domain objects (or *task objects*) that are manipulated by any (sub)task at any level of decomposition. Some task models explicitly embody this information, whereas others prefer to establish a link between the task model and the domain model (*reference*). This reference may cover the entities of interest only (as in CTT), perhaps along with their relationships (as in TOOD).

6.3.3 Common Properties

The following concepts are critical for a task-based design of a user interface. First, goal and task hierarchies are essential. Second, operators must express temporal constraints between tasks. Third, a minimal requirement for dealing with cooperative aspects is role specification

in terms of tasks. Fourth, objects and the actions that are performed on them make possible the detailed modeling of presentation and dialog of the user interface. More details are provided in (Limbourg, Pribeanu, & Vanderdonckt, 2001).

6.3.4 Model Usage

HTA is mainly intended as a means of training people in the use of a given interface. Although plans are attractive and precise, they are informal and are not subject to proof verification or model checking. Moreover, plans typically describe temporal constraints in a procedural way. Adding new tasks means rewriting the plans associated with the next higher level task, which may be tedious and lead to errors. Although plans are suitable for early task analysis and when information is elicited in an informal but unambiguous way, they do not provide the kind of representation able to support a (computer-aided) derivation of a user interface model.

GOMS models also assume a given user interface. There is no explicit task decomposition but only a hierarchical organization of methods used to operationalize tasks. Cognitive analysis in GOMS is done at unit task level, and it requires different levels of specification depending on the level of detail desired by the analyst. GOMS is built for the lowest level of task decomposition and thus provides no support for user interface modeling. Rather, the objective is to optimize user performance and evaluate execution time, memory workload, and learning time early in the design process. The concepts are more closely related to user modeling than to task modeling.

GTA is especially good at specifying delegation mechanisms between roles. TKS uses both plans and procedures similar to the way HTA uses plans and GOMS uses selection rules and methods. Like methods, procedures are useful for achieving a detailed specification of the elementary task when there is a need to describe actions performed on objects. Models should be declarative rather than procedural in order to support successive transforms and to be suitable for the use of computer tools (Eisenstein, Vanderdonckt, & Puerta, 2001). Task models that are primarily intended as support for evaluation and user training, like HTA and GOMS, are not suitable for supporting user interface modeling. Rather, these models require an initial design of the interface whose usability they focus on improving.

6.3.5 Expressiveness Versus Complexity

Fig. 6.10 shows that task models become increasingly complex as they become progressively expressive (e.g., they can express many different facets of task modeling and not only the task decomposition). HTA and MAD* are located at the left end of the continuum because they

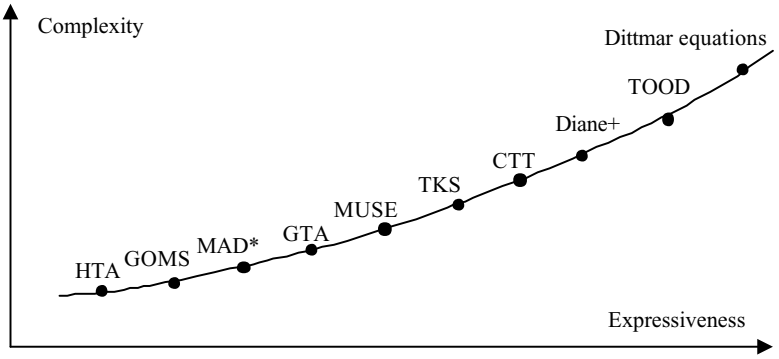

FIG. 6.10. Expressiveness versus complexity of task models.

are basically restricted to decomposing tasks into subtasks and temporal operators. Successive models refine the temporal relationships using pre- and postconditions (MUSE), abstract data types and axioms (TKS), LOTOS operators (CTT), first-order logical predicates (Diane+), mathematical functions (TOOD), and mathematical equations (Dittmar, 2000). The right side of the continuum ends with Dittmar's task model, which regulates temporal ordering by means of a set of mathematical equations, linear or nonlinear. This method is considered the most expressive (mathematical equations are very general) yet the most complex (solving a set of nonlinear, possibly conflicting, equations is a nontrivial problem).

ACKNOWLEDGMENTS

We would like to greatly thank those who helped us in this comparison of task models: Cristina Chisalita, Hilary Johnson, Peter Johnson, Kim Young Lim, John Long, Cécile Paris, Fabio Paternò, Costin Pribeanu, Dominique Scapin, Neville Stanton, Dimitri Tabary, Jean-Claude Tarby, and Gerritt van der Veer.

APPENDIX: OVERVIEW OF THE ENTITY-RELATIONSHIP-ATTRIBUTE MODEL

The figures of metamodels compared in this chapter use the notation of the Entity-Relationship-Attribute (ERA) model. This model relies on understanding the relationship between one entity and another. For example, the model in Fig. 6.11 graphically depicts the ownership of vehicles by persons. Reading the model from left to right represents the statement "An owner may own no, one, or many vehicles," whereas reading from right to left represents the statement "Any vehicle is owned by one or many owners."

ERA uses a rectangle to graphically depict an entity and a hexagon for any relationship between entities. A relationship is said to *cyclic* if the relationship has the same entity for both source and destination. Any entity or relationship can have one or many attributes. For instance, an owner may be characterized by the following attributes: the number of her identity card, her first name, her last name, and her address. If an attribute is underlined, it is an *identifier*, that is, an attribute whose values remain unique for every instance of the entity. The connectivity specifies how many instances of that entity can be associated with a single instance of the other entity via the relationship. In this example, an owner may have many vehicles, even simultaneously, and a vehicle should be owned by one or many owners. To facilitate reading, a role is added to each part of the relationship.

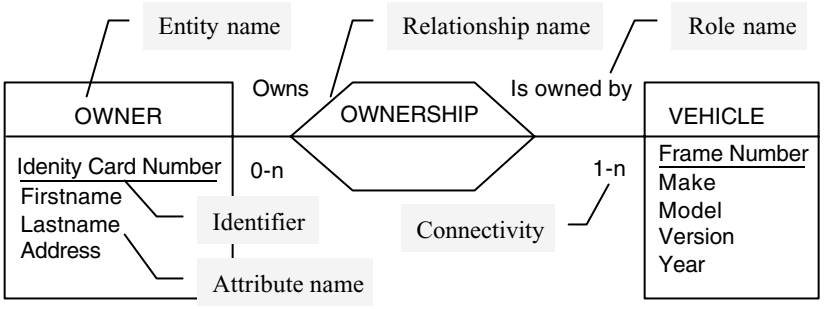

FIG. 6.11. Overview of the entity-relationship-attribute model used.

REFERENCES

Annett, J., Cunningham, D., & Mathias-Jones, P. (2000). A method for measuring team skills. *Ergonomics, 43*, 1076–1094.

Annett, J., & Duncan, K. (1967). Task analysis and training design. *Occupational Psychology, 41*, 211–227.

Barthet, M.-F., & Tarby, J.-C. (1996). *The Diane+ method*. In J. Vanderdonckt, (Ed.), *Computer-aided design of user interfaces* (pp. 95–120). Namur, Belgium: Presses Universitaires de Namur.

Baumeister, L. K., John, B. E., & Byrne, M. D. (2000). A comparison of tools for building GOMS models: Tools for design. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI 2000)* (Vol. 1; pp. 502–509). New York: ACM Press.

Beard, D. V., Smith, D. K., & Denelsbeck, K. M. (1996). QGOMS: A direct-manipulation tool for simple GOMS models. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '96)* (Vol. 2; pp. 25–26). New York: ACM Press.

Bomsdorf, B., & Szwillus, G. (1998). From task to dialogue: Task-based user interface design. *SIGCHI Bulletin, 30*(4), 40–42.

Bomsdorf, B., & Szwillus, G. (1999). Tool support for task-based user interface design. A CHI '99 workshop. *SIGCHI Bulletin, 31*(4), 27–29. Available: http://www.uni-paderborn.de/cs/ag-szwillus/chi99/ws/

Card, S. K., Moran, T. P., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Diaper, D. (1989a). Task analysis for human-computer interaction. Chichester, England: Ellis Horwood.

Diaper, D. (1989b). Task Analysis for Knowledge Descriptions (TAKD): The method and examples. In D. Diaper (Ed.), *Task analysis for human-computer interaction* (pp. 108–159). Chichester, England: Ellis Horwood.

Dittmar, A. (2000). More precise descriptions of temporal relations within task models. In P. Palanque & F. Paternò (Eds.), *Proceedings of Seventh International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS 2000)* (Lecture Notes in Computer Science, Vol. 1946; pp. 151–168). Berlin: Springer-Verlag.

Eisenstein, J., Vanderdonckt, J., & Puerta, A. (2001). Model-based user-interface development techniques for mobile computing. In J. Lester (Ed.), *Proceedings of ACM International Conference on Intelligent User Interfaces (IUI 2001)* (pp. 69–76). New York: ACM Press.

Gamboa, F., Scapin D. L. (1997) Editing MAD* task descriptions for specifying user interfaces, at both semantic and presentation levels. In M. D. Harrison & J. C. Torres, (Eds.), *Proceedings of Fourth International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS '97)* (pp. 193–208). Berlin: Springer-Verlag.

Hackos, J. T., & Redish, J. C. (1998). *User and task analysis for interface design*. New York: Wiley.

John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction, 3*, 320–351.

Johnson, P. (1992). *Human-computer interaction: Psychology, task analysis and software engineering*. London: McGraw-Hill.

Johnson, P., Diaper, D., & Long, J. (1984). Tasks, skill and knowledge: Task analysis for knowledge based descriptions. In *Proceedings of First IFIP Conference on Human-Computer Interaction (Interact '84)* (Vol. 1; pp. 23–28). North-Holland: Elsevier Science Publishers.

Johnson, P., & Johnson, H. (1989). Knowledge analysis of task: Task analysis and specification for human-computer systems. In A. Downton, (Ed.), *Engineering the human-computer interface* (pp. 119–144). London: McGraw-Hill.

Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '95)* (pp. 91–100). New York: ACM Press.

Kirwan, B., & Ainsworth, L. K. (1992). *A guide to task analysis*. London: Taylor & Francis.

Lewis, C., & Rieman, J. (1994). *Task-centered user interface design: A practical introduction*. Available at http://www.acm.org/∼perlman/uidesign.html#chap0

Lim, K. Y., & Long, J. (1994). *The MUSE method for usability engineering*. Cambridge: Cambridge University Press.

Lim, K. Y., & Long, J. (1996). Structured task analysis: An instantiation of the MUSE method for usability engineering. *interacting With Computers, 8*(1), 31–50.

Limbourg, Q., Pribeanu, C., & Vanderdonckt, J. (2001). Towards uniformised task models in a model based approach. In C. Johnson (Ed.), *Proceedings of Eighth International Workshop on Design, Specification, Verification of Interactive Systems (DSV-IS 2001)* (Vol. 2220; pp. 164–182). Berlin: Springser-Verlag.

Lu, S., Paris, C., & Vander Linden, K. (1998). Towards the automatic generation of task models from object oriented diagrams. In *Proceedings of Seventh IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI '98)* (pp. xxx). Dorolrecht, Netherlands: Kluwer Academic.

Mahfoudhi, A., Abed, M., & Tabary, D. (2001). From the formal specifications of user tasks to the automatic generation of the HCI specifications. In A. Blandford, J. Vanderdonckt, & P. Gray, (Eds.), *People and computers XV* (pp. 331–347). London: Springer.

Paternò, F. (1999). *Model based design and evaluation of interactive applications*. Berlin: Springer-Verlag.

Paternò, F., & Santoro, C. (2002). One model, many interfaces. In C. Kolski & J. Vanderdonckt (Eds.), *Proceedings of Fourth International Conference on Computer-Aided Design of User Interfaces (CADUI 2002)* (pp. 143–154). Dordrecht: Kluwer Academics.

Payne, S. J., & Green, T. R. G. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction, 2*(2), 93–133.

Savidis, A., Akoumianakis, D., & Stephanidis, C. (2001). *The unified user interface design method*. In C. Stephanidis (Ed.), *User interfaces for all* (pp. 417–440). Mahwah (NJ): Lawrence Erlbaum Associates.

Scapin, D. L., & Bastien, J. M. C. (2001). *Analyse des tâches et aide ergonomique à la conception : l'approche MAD*.* In C. Kolski (Ed.), *Analyse et conception de l'IHM: Interaction homme-machine pour les systèmes d'information* (Vol. 1; pp. 85–116). Paris: Edition Hermès.

Scapin, D., & Pierret-Golbreich, C. (1989). Towards a method for task description: MAD. In L. Berlinguet & D. Berthelette (Eds.), *Proceedings of Work with Display Units (WWU '89)* (pp. 27–34). North-Holland: Elsevier Science.

Shepherd, A. (1989). Analysis and training in information technology Tasks. In D. Diaper (Ed.), *Task analysis for human-computer interaction* (pp. 15–55). Chichester, England: Ellis Horwood.

Shepherd, A. (1995). Task analysis as a framework for examining HCI Tasks. In A. Monk & N. Gilbert (Eds.), *Perspectives on HCI: Diverse approaches* (pp. 145–174). London: Academic Press.

Smith, M. J., & O'Neill, J. E. (1996). Beyond task analysis: Exploiting task models in application implementation. In *Proceedings of ACM Conference on Human Aspects in Computing Systems (CHI '96)* (Vol. 2; pp. 263–264). New York: ACM Press.

Souchon, N., Limbourg, Q., & Vanderdonckt, J. (2002). Task modelling in multiple contexts of use. In P. Forbrig, Q. Limbourg, B. Urban, & J. Vanderdonckt (Eds.), *Proceedings of Ninth International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS '2002)* (Vol. 2545; pp. 59–73). Berlin: Springer-Verlag.

Stanton, N., & Young, M. (1998). Is utility in the eye of the beholder? A study of ergonomics methods. *Applied Ergonomics, 29*(1), 41–54.

Sutcliffe, A. (1989). Task analysis, systems analysis and design: Symbiosis or synthesis? *Interacting With Computers, 1*(1), 6–12.

Tauber, M. J. (1990). ETAG: Extended task action grammar, a language for the description of the user's task language. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel (Eds.), Proceedings of the 3rd IFIP TC 13 Conference On Human-Computer Interaction Interact 90, pp. 163–168, Amsterdam, 1990, Elsevier.

Thévenin, D. (2001). *Adaptation en interaction homme-machine: Le cas de la plasticité*. Unpublished doctoral dissertation, Université Joseph Fourier, Grenoble, France.

van der Veer, G. C., Van der Lenting, B. F., & Bergevoet, B. A. J. (1996). GTA: Groupware task analysis-modeling completity, *Acta Psychological*, 91: 297–322, 1996.

van Welie, M., & van der Veer, G. C. (1998). EUTERPE: Tools support for analyzing cooperative environments. In T. R. G. Green, L. Bannon, C. P. Warren, & J. Buckkleys (Eds.), *Cognition and co-operation* (pp. 25–30). Roquencourt INRIA.

van Welie, M., van der Veer, G. C., & Eliëns, A. (1998). An ontology for task world models. In P. Markopoulos & P. Johnson (Eds.), *Proceedings of the Fifth International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS '98)* (pp. 57–70). Vienna: Springer-Verlag.

Wilson, S., & Johnson, P. (1996). Bridging the generation gap: From work tasks to user interface designs. In J. Vanderdonckt (Ed.), *Computer-aided design of user interfaces* (pp. 77–94). Namur, Belgium: Presses Universitaires de Namur.