A Composite Task Meta-Model as a Reference Model

Steve Goschnick, Liz Sonenberg, Sandrine Balbo

A Composite Task Meta-Model as a Reference Model

Steve Goschnick[1], Liz Sonenberg[1], Sandrine Balbo[2]

[1] University of Melbourne,VIC 3010, Australia
[2] Deakin University,VIC 3217, Australia
{stevenbg, l.sonenberg}@unimelb.edu.au | sandrine@acm.org

Abstract. In this paper we develop a comprehensive composite meta-model from Task Analysis models called the *Reference Task Meta-model* (*ReTaMeta model*) for the purpose of comparing numerous Agent-Oriented meta-models. The reference model needed to be derived from a field independent of the Agent-oriented paradigm, yet based on Psychology. To arrive at the ReTaMeta model we first extracted the meta-models from several well-known cognitive task models including GOMS, GOMSL, TKS, GTA and also the CTT and Diane+H Task Modeling notations for fine grain task detail, and then combined their respective concepts in a complementary and comprehensive way.

Keywords: Task Model, Meta-model, Task Analysis, Interaction model, reference meta-model.

1 Introduction

Milton and Kazmierczak [1] created a method called *The Method of Conceptual Evaluation and Comparison*, for comparing meta-models in one discipline, against a reference meta-model drawn from another discipline. Using it they successfully compared a number of data modeling languages against Chisholm's Ontology from Philosophy [1]. In this paper, we do not detail their method, but highlight its foundational requirements for *a reference model* that comes from an independent field of study while also having related concepts to the area under investigation.

Our larger goal was to evaluate competing meta-models in the Agent-Oriented (AO) paradigm, so, in order to use Milton and Kazmierczaks method, we needed such a *reference model*. This paper outlines the path we took in creating the *Reference Task Meta-model* (*ReTaMeta Model*), a comprehensive composite model, drawn from many existing meta-models from the Task Analysis (TA) paradigm.

The TA paradigm is an ideal source for a reference model independent of AO, since both paradigms cover many similar concepts, despite their different histories. Both cover mentalistic notions assumed to be represented in some way in the cognitive functioning of the human mind, each drawing upon Psychology.

We studied a number of cognitive task models and noted a progression of sophistication over time, as we moved from HTA [2] to GOMS [3], TKS [4], GTA/TWO [5, 6] and GOMSL [7], towards a cognitive model about 'how the mind' goes from goals to finished tasks. We also needed concepts that covered *temporal*

and relational sequencing of tasks, to achieve the level of detail needed in our reference model, so we abstracted a meta-model from the Diane+H notation [16,18]. We then merged concepts from all meta-models examined, into a comprehensive composite meta-model from the TA paradigm. **Note 1**: The models chosen were not done so for a comparison across TA models, nor to generate some TA model with new features. They were selected as a representative subset of models, to cover the progression of TA over time, to arrive at a single comprehensive reference meta-model, for the purpose outlined above. **Note 2**: UML is designed to allow the modeler to represent models at various stages of expressiveness, as needed. It has been used as such with meta-models here: sometimes presentation of attributes and methods detract from the discussion, and explanation of them can unnecessarily expand a paper for little value. **Note 3**: As with application modeling, when different Analysts abstract a model from written description, their models are invariably different. Other researchers have abstracted some of the models we investigated – we believe our work complements theirs by adding to a larger body of meta-model research.

2 Extraction of Task Meta-models

This section describes the meta-models of the various TA models considered: GOMS, GOMSL, TKS, GTA/TWO and then the Diane+H notation (with reference to CTT).

GOMS and GOMSL:

The GOMS (goals, operators, methods and selection rules) model by Card et al [3] was an attempt to represent human behaviour with regard to interfaces, with a high-level model that software engineers could apply without empirical testing. Beyond earlier approaches, GOMS models user *intention* and is mainly used to *predict* the time taken for an expert to complete given tasks, as a metric.

The entities and concepts in the left-side of figure 1 represent GOMS as follows:
Goal: represents a state that needs to be accomplished.
Methods: are required to achieve specified *Goals*, they consist of a series of steps, each are either *Operators* to perform, or s*ub-Goals* to achieve. Methods are like plans that the person has already learned.
Operator: *Operators* are basic or simple *actions* that a person can perform on the system or device in question. Card et al define them as elementary acts that are either: perceptual, motor or cognitive in nature.
Selection Rule, Condition: When alternative *Methods* exist to achieve a *Goal*, a *Selection Rule* uses *Conditions* to decide which to use. We included *Enactment* to allow a given *Method* to be used by multiple *Goals* – but not in parallel (a limitation of GOMS).

While GOMS was invented to evaluate interfaces by predicting task performance, it has been used to understand user activity, by choosing *Operators* at the right level of granularity. [7] portrays such a GOMS model as a description of the *procedural knowledge* required to do tasks on a device or system – a representation of '*how to do it*'. *Operators* are usually defined by the hardware (e.g. mouse, keyboard, button) with the analysis concentrated on defining the Goals and Methods.

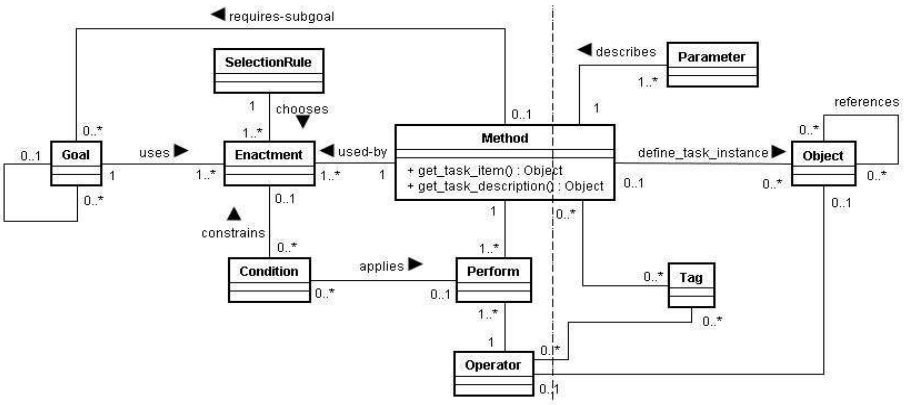

Fig. 1. On the left is the GOMS meta-model, while the superset meta-model is for GOMSL

GOMS is limited to the representation of pre-learned, error-free tasks - unsuitable for analysing *problem solving* tasks. An extension to GOMS named NGOMSL [9,10] - short for *Natural GOMS Language* – is a structured language used to capture *production-rules* that model the aspects of GOMS in analysis. The authors argue that the rendition of GOMS as production-rules, approximates the way people structure their task knowledge. When new UI features are added to an existing technology, the NGOMSL analyst is able to make estimations of the amount of learning required, based on the new production rules needed. A more recent executable version of NGOMSL called GOMSL – for *GOMS Language* [7,9] - is available in GLEAN, a programming environment. GOMSL retains all of the underlying cognitive aspects and direction of the original GOMS, but adds some additional functionality.

The language is partly declarative, with a few procedural branching constructs (*If Then* and *GoTo*), and a *Decide* construct - which is operationally equivalent to the *Switch* statement in C/Java. *Method* declarations have been extended to take a list of parameters, allowing the analyst to generalise methods – providing the ability to abstract procedural knowledge. The GOMSL meta-model is represented in figure 1. The entities *Goal, Method, Operator, SelectionRule, Condition, Perform* and *Enactment* – have the same meaning as they do in the earlier GOMS meta-model, while *Object, Tag* and *Parameter* are innovations introduced in GOMSL:

Object, Tag: Is basically a symbolic *name* with a list of *property/value* pairs. GOMSL can store them as lists of like-objects – called the Object Store (represented by Object in the meta-model). In cognitive terms, the objects in the store represent '*Long Term Memory'* (LTM). A value set in the properties of an object can be another object, hence the *self*-relationship in the model. Both Object and Tag are used to hold the *task data*. A *Tag* holds a value stored under a symbolic name like a *variable* in a programming language. All *Tags* in GOMSL are said to remain in '*Working Memory*' in cognitive terms. *Objects* can be retrieved from the LTM object store via a Tag, making their *property/value* pairs available to working memory.

Parameter: Represents each parameter used in a Method. The two functions shown in *Method* in the meta-model (which are not in GOMS), have the following purposes:

get_task_description(): retrieves the list of parameters of a Method;

get_task_item(): is then used to retrieved an instance of the parameter property/value pairs for all parameters, from the Object Store. This retrieved set of values are then be used in the Method instantiation.

Note: Using an Operator called *thinking_up* GOMSL can retrieve a *task_item* from the object store based on *search* using a single parameter value (i.e. like retrieving a record from an SQL database table of parameter values, based on the value of one attribute only). It simulates a person '*thinking up'* tasks they may need to achieve a Goal, recalling them from memory. *thinking_up* is an example of a *mental operator*, hypothetical or unobservable in the user, usually inferred by the analyst.

GOMSL retains the GOMS heritage through making Goals, Operators and Methods declarative. He has used an analogy to LTM (long term memory) as an Object Store, and Working Memory as an analogy for the *local memory* in a given simulation. Bringing objects '*into focus'* is his version of dynamic instantiation. Existing objects automatically going *out of focus* when a like-object is brought into focus, is a rudimentary garbage collection mechanism.

Although GOMSL is still focused upon the evaluation of user interfaces in a GOMS analysis, it has the ability to put timing measures (e.g. 50 ms) against various *methods* and *operators*, to simulate an experienced user. Executing code written in GOMSL leads to a GOMS model. The high level Methods with their set of parameters, are extracted from the Object Store and translated into the lower-level Methods needed to achieve the corresponding Goals, repeatedly, until they are all reduced to primitive Operators, each of which takes an assumed time.

The Task Knowledge Structure (TKS):

The cognitive ability humans bring to the interface predates the machines we apply them to, so the originators of some task models tried to make theirs more general, with application beyond the computer-human interface. The Task Knowledge Structure (TKS) model [4, 12] is and example. It takes GOMS as a starting point, then adds knowledge structures held in the mind, specifically, knowledge related to a *task.* They assert that people gather knowledge structures about tasks they have learned and performed, for application in future similar tasks. The TKS model represents this *task knowledge* held in memory then activated during task execution.

They thought that the four dimensions of GOMS with some extension, '*might be considered the basic structural components of any task knowledge that a person might recruit*' [4]. Not just in doing tasks, but structures outlining how particular knowledge is brought to the task. TKS represents *a summary* of the sorts of knowledge a person stores about doing a task. Figure 2 (left-side) is our meta-model of TKS, abstracted from their key publications, as follows:

Goal: Goals are represented in a goal substructure, which '*includes enabling and conditional states, if a goal is to prevail*', i.e. *Pre-conditions* and *post-conditions*. Goals and sub-goals represent states, and are not executable.

Plan: A method to achieve a goal in terms of state change, using tasks and sub-goals.

Task: A task is an activity to achieve a change of state. Task *procedures* (represented via the task *self*-relationship, and associated sub-goals) are one of the knowledge types, and are executable.

Object: They assume that knowledge is stored about both *physical* and *informational* objects. Johnson later added *conceptual* [12]. Objects are configured into taxonomies.

Agent: While they did not initially use the term 'agent', instead using 'person' and 'individual', P. Johnson later defined *agent* as either: person, animal or machine [12].
Role, Responsibility: They saw roles as '*heavily implicated'* in the way that people bring particular sources of knowledge to a task. A TKS role is defined by a particular *set of tasks* for which an individual is *responsible* – a many-to-many relationship here.
Similarity: Is a measure of the similarity between Tasks performed for different Roles. E.g. whether a person is organising a meeting at work or at home, will involve a different set of tasks, however, the skills in one, help in doing the other.

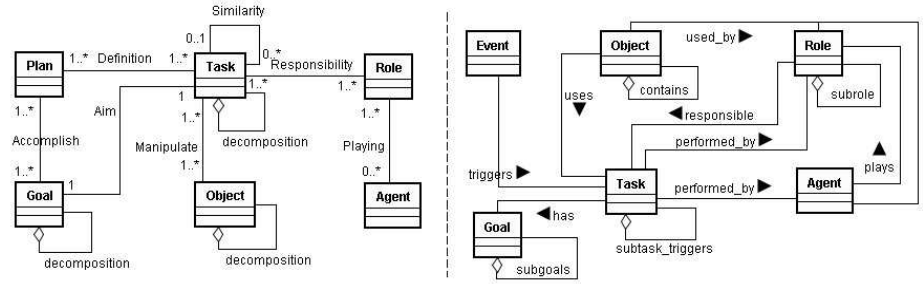

Fig. 2. Meta-model of TKS (left) abstracted from [4,11]; Meta-model of TWO (right)

The originators of TKS do have several other concepts that we have not singled out here from the task hierarchy in the meta-model in figure 2 above, namely:
Action, Procedure and Strategy: *Actions* are simply the terminal or leaf nodes of the *task hierarchy*, so they can be thought of as constituent parts of the task hierarchy. A TKS *Plan* varies within the paper [4]: at one point plans consist of '*goal substructures'*, at other times they consist of '*task substructures'*. In addition to that they have *procedures* described as a '*macro-action',* which a Task may call upon to achieve a goal. Elsewhere they refer to a procedure as '*an element of a subtask'*, elsewhere, as a '*task procedure'*. Their concept of a *Strategy* is simply one that separates procedures for achieving the same sub-goal. I.e. A logical OR in the task hierarchy caters for such alternatives. P. Johnson later removed Strategy from the TKS model himself [12]. In abstracting the TKS meta-model here, we have taken the more recent view of *a task hierarchy*, as defined in various *task notations*. These task model notations have *temporal* and *logical operators*, and can represent *selection* and *concurrency*. (E.g. Diane+H and CTT) As such, the Task *hierarchy* in the TKS meta-model in figure 2, is more than expressive enough to encompass *procedures*.

In several ways, the TKS model seems quite an advanced TM for its time, however, the variations in terminology used across the three defining publications, particularly between the terms: *task, procedure, macro action, task procedure*, and *plan* – impedes the abstraction of a meta-model from those descriptions. In comparison, the definitive descriptions of HTA, GOMS and GTA/TWO are very concise in the meaning of their inclusive concepts and terms.

We found TKS most interesting for its introduction of *Object, Role, Responsibility* and *Agent*, beyond the other entities it has in common with HTA and GOMS. The TKS authors cite research that backs up their use of Objects within knowledge structures held in the mind. P. Johnson later played down the *Role* and *Responsibility* concepts in TKS in his book [12].

Groupware Task Analysis (GTA), Task World Ontology (TWO):

Groupware Task Analysis is a generic task analysis model for which the creators present an ontology in later literature, called the *Task World Ontology* (TWO) [5, 6]. TWO is effectively the meta-model of GTA and it includes as its primary entities: *Event, Task, Goal, Agent, Role* and *Object*. GTA was an early task method that allowed for the analysis of people using *collaborative* technology, supported with tools, in the field of computer supported cooperative work (CSCW). In a comparison of TA models which included GTA/TWO [13], those authors characterise the changes in task models in order to deal with the cooperation, as: '*A role is defined by the task the role is responsible for. Roles are assigned according to organisational rules*'. Its support for *Agents, Roles* and *Responsibilities*, is not more than what the TKS model added to task modeling, unlike TKS, it is very clearly defined. The researchers around the TWO model extended support beyond conceptual modeling into tools for researchers and practitioners. As such, GTA/TWO gained a reputation as a TA and modeling framework used to support CSCW application development.

The TWO meta-model is rendered here in UML class notation in figure 2 (right-side). It includes an extra entity beyond the Task Models considered so far: it has ***Event*** as an entity within the model itself, which *triggers* Tasks to swing into action.

The concept of a *trigger* here leads to mandatory actions being fired via a Task hierarchy, without any deliberation. The TWO meta-model does have a Goal entity separate from the Event interaction, such that Goals can be related to Tasks and sub-tasks. However, TWO Goals are sub-ordinate to Tasks, in turn triggered by Events, rather than Goals being unrelated to external events, so the concept of goal-driven behaviour (as seen in software agent meta-models) is not represented within the conceptual framework built around TWO - but it goes conceptually very close.

GTA/TWO is coming from a more practical modelling and tools perspective. While it does have an entity for Object, TKS included the Object entity from a psychological basis, drawing from both theory and empirical studies. TWO includes the *Role* and *Agent* entities along with the *responsible* relationship, coming from the practical need to support the multiplicity of users involved in a CSCW application.

The Meta-model Behind Diane+H Notation, with a comparison to CTT:

Diane+ is a methodology with a task model *notation* [8,17]. Diane+H is a sub-set of Diane+ covering the formal notation, as implemented in software tools such as *Tamot* [16,18]. Diane+H has a significant amount of expressive power in portraying task compositions from high-level user goals to low-level actions. It allows for temporal and logical relationships between tasks. It differentiates tasks by *type* of either: interaction by the *user*; a task performed solely by the *application/system*; and *manual tasks* by the user not involving the system at all. Additionally, it can represent tasks that run in parallel, and tasks that need to follow on from one another in sequence.

Preconditions may *trigger* a task and *link-conditions* may be placed between sequential tasks. These and other capabilities make the Diane+H meta-model a complex one. Figure 3 portrays a meta-model of it we abstracted from [16,18].

Diane+H is included in our selected examination of meta-models as a representative of the graphical notations that have been developed in the TA discipline, which include CTT [19] and MAD. In a cross-comparison of six notations [14], Balbo et al rated Diane+ most highly overall, across 10 dimensions, including

aspects of the SDLC. That said, we still examined CTT and the related tool CTTe as it appears more flexible than Diane+H and we comment on some differences, below.

The range of applications it has been applied to demonstrates the flexibility of Diane+H. In addition to the two early stages of the SDLC (Requirements/Analysis; Design) that Diane+ was originally created, it has been used [14] to: automatically generate online help and user documentation; identify the place of new technologies in the work environment; in the later phases in the SDLC (Development and deployment); and more recently in displaying Task Models for web navigation paths and options, automatically generated from web sites [15].

A detailed description of the significant *entities*, *relationships* and most of the *attributes* needed to represent a Diane+H modeling tool such as Tamot, follow:

Task: may be performed either *manually* by the user, by the system (*automatically*), or via the user *interacting* with the system. They can be *optional* or *mandatory*, *elementary* or *composites*. Composites may appear visually expanded in the tool, or not – hence the boolean attribute *expanded*. Tasks may be performed repeatedly with a minimum and maximum number of *iterations* declared in the notation. A Task may be *terminating*, identified here with the boolean attribute named *terminal*.

LogicalOperator: Sub-tasks of a Task may be linked via a logical operator - OR, AND or XOR. Two *or more* sub-tasks may be grouped together with an AND. Interestingly, a Task ***hierarchy*** may start with a *LogicalOperator* at the top of the tree, so the cardinality on relationship *leads_to* is *zero-or-one* at both ends.

RelatedTask: Diane+H notation is most often used to represent ***hierarchies*** *of tasks*, but in some application models, a Task may be linked to more than one 'parent' Task, representing a ***network*** *graph* rather than a tree graph. The *RelatedTask* entity here, ***allows for both sorts of graphs***. In addition, it caters for the ability of Diane+H to represent Tasks that happen in *parallel* (for which the *sibling* attribute is set to *true*, or *false* – for parent-child relationships that happen *sequentially*). The *elder* relationship links it to a *parent Task* in the case of parent-child relationships, and it points to the *next oldest sibling* in the case of *parallel Tasks*. Note: In the notation, parallel Tasks are visually stacked in a vertical column without lines between them [18] – the business-rule used here to present such a stack, is to place the eldest sibling Task topmost in the stack, and so on down to the youngest sibling.

LinkCondition: Tasks that are linked together sequentially may encounter a condition that needs to be met before task execution continues. This *LinkCondition* is expressed in a string attribute called *expression* – which is set to *null* when defining a sequence with *no* conditions. In [19], Paterno portrays several task notations which have specific icons in CTTe which don't have equivalents in Diane+H. However, we find that '*LinkCondition*' could be broadened in meaning to enact several of them including: *concurrent communicating tasks*; *task independence*; and *suspend-resume*.

Precondition: In addition to LinkConditions, *Preconditions* may trigger a Task in the first place. The Precondition may *involve*: the progress of *other Tasks* elsewhere in the model; some *Event*; or else a change in a *Data* value.

Feedback: Diane+H has two attributes in a *Feedback* entity, one that describes the *purpose* of the Task, while the other represents a message to a user when the Task is complete. The description held in the *task_purpose* attribute is useful to the designer of an application when prototyping it via a tool such as Tamot, as it gives some context-sensitive information.

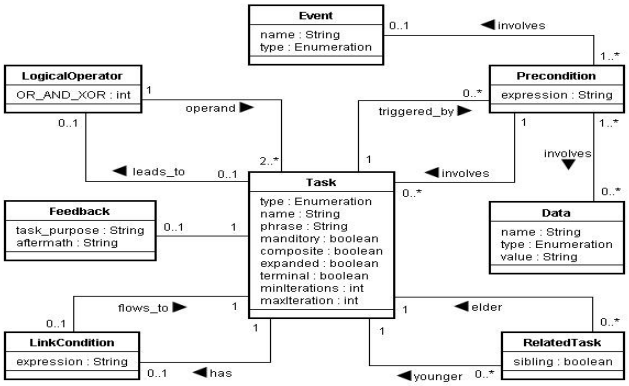

Fig. 3. Meta-model of the Diane+H Task Model Notation, abstracted from [16,18]

CTT has a good treatment of *Objects* which is absent in Diane+H (except via *Data*). CTT has a relationship between User and Objects called *Rights*, to cater for multiple users with differ access to Objects. CTT's *Task* also has an attribute called *Platform*, used to allow or disallow objects available to a Task, based on Platform.
Note: Several researchers have embedded Task Models in tools which facilitate much of the SDLC, notably: Tamot using Diane+H as their notation [16, 18]; and CTTe by Paterno [19]. Paris et al were able to automate the extraction of task models from standard text using WordNet, and from UML class and sequence diagrams. Inversely, they are also able to generate UML diagrams from Task models, visually represented in Diane+H and declaratively in an XML format. These extracted UML models are limited by the '*user-oriented nature of the task models*' they built [16].

3 A Composite Task Meta-model

In the introduction we touched upon an evaluation technique used to compare meta-models called *The Method of Conceptual Evaluation and Comparison*. It requires a *reference meta-model* from a field independent of the one being studied (the Agent-oriented paradigm, in our case). In this section we put forward our composite meta-model drawn from the TA paradigm as just such a reference model, one that we think is a sensible composition of concepts from the TA meta-models presented above.

There are several ways that such a reference model could be constructed from separate TA meta-models, all of them with some subjective reasoning, which is worth a brief discussion: Firstly, even if the models had clearly overlapping concepts, we could have taken the *intersection* of concepts between the models; however that would leave us with a minimalist model, not very helpful with the comparison of agent meta-models which often have an extensive range of concepts. E.g. some agent-meta-models have upwards of 30 entities and as many relationships. It was instructive to us, that in the comparison of *data modeling languages* done by Milton and Karzmierczak [1], they deemed a *subset* of the Chisholm ontology as a relevant reference model for their purposes (note: the Chisholm ontology from Philosophy is

considered a '*commonsense realism*' model of the world around us, that includes *states* and *events*), since data modeling languages are not intrinsically interested in *states* and *events* - other *process-oriented* languages are. What we needed was a *superset*, a comprehensive model representing a *union* of the constituent concepts in the representative TA meta-models we examined, to reach a comparatively complex meta-model to those in AO. When there were *similar* concepts in two different meta-models that did not perfectly line up, we took the more flexible one.

There was also some chronology to the TA models. The cognitive task models we examined were largely built upon the assumed knowledge structures in peoples' minds and together they represent an accumulating set of concepts, over the time period in which they were invented. From them we have *Tasks*, *Goals*, *Plans* (*Methods* + *Operators/Actions* + *Rules/Conditions*), *Objects (with ontological structure), Roles, responsibilities* and *Events* – with adequate connection to psychological theory and/or the backing of empirical studies, to place them somewhere within the cognitive architecture inside peoples' heads. To those, we further considered the concepts in the Diane+H and CTT notations, to adequately allow for the decomposition of tasks with sophisticated expressive power in order to cater for the temporal and logical relationships between tasks, and other features.

Figure 4 represents our *composite task meta-model*, which carefully mixes and matches concepts from the Task meta-models examined above, into a model that includes all key concepts in one meta-model. We claim that it is a *broad reference model*, but we certainly did not examine all task meta-models in our process, and so we do *not* claim that it is a *unified task model for all TA purposes*.

The definition of the concepts and terms as they are represented in figure 4 follow, with an explanation of the choices we made when alternatives were available:

Goal, Task: In a summary of his ConCurTaskTrees notation (CTT), Paterno defines a *goal* in a task model as follows: '*A goal is either a desired modification of a state or an enquiry to obtain information on the current state. Each task can be associated with a goal, which is the state change caused by its performance.*' [19]. Goals are represented in a substructure, which '*includes enabling and conditional states, if a goal is to prevail*'. That is a good general definition of a Goal, across the TA models examined. A TKS *Task* is an activity that results in a change of state in a given domain – and that aligns with a GOMS *method*. As in TKS and in Paterno's definition, there is a one-to-one mapping between Goal and Task, a *Task* sets about *satisfying* a *Goal* as portrayed in figure 4 - top-left. They could both be represented as predicates/method-signatures (as is the case in both GOMSL and GTA/TWO), but the Goals in the goal hierarchy will just list the parameter names, whereas the specific Task/method-signature, will have many if not all of the parameters set to *values*. I.e. The *Task* that comes via the *responsibility* relationship from *Role*, may calls upon *Goal*, for the list of sub-Goals that need to be answered/achieved.

RelatedTask: From the Diane+H meta-model, this is the most flexible approach to task decomposition of all the meta-models examined. As discussed above, it allows for serial and parallel execution of tasks, and for *networked* graphs (multiple parent tasks) in addition to *hierarchies*. As with the GOMS *Method*, the Task in this model consists of related sub-tasks and *Operators*. Operators are represented via the Perform entity (as the same Operator, may be used in many Tasks, e.g. a left *mouse-click*). The Goal associated with any given sub-task can be located via the *satisfying* relationship.

Plan: A Plan is a way to achieve a goal in terms of state changes. *Plan* here takes the GOMS meta-model entity *Enactment* as the better more detailed approach, with *SelectionRules* and *Conditions* that may be placed upon the elements of the Plan, e.g. upon the sub-tasks which the sub-goals. *SelectionRules* allows for the OR, AND and XOR logical operators that are in Diane+H, in addition to other possible rule constructs which may include *Conditions*. The Condition entity can also be used to represent the *LinkCondition* entity as seen in the Diane+H model, which represents a *PreCondition* for any sub-tasks a given Task model.

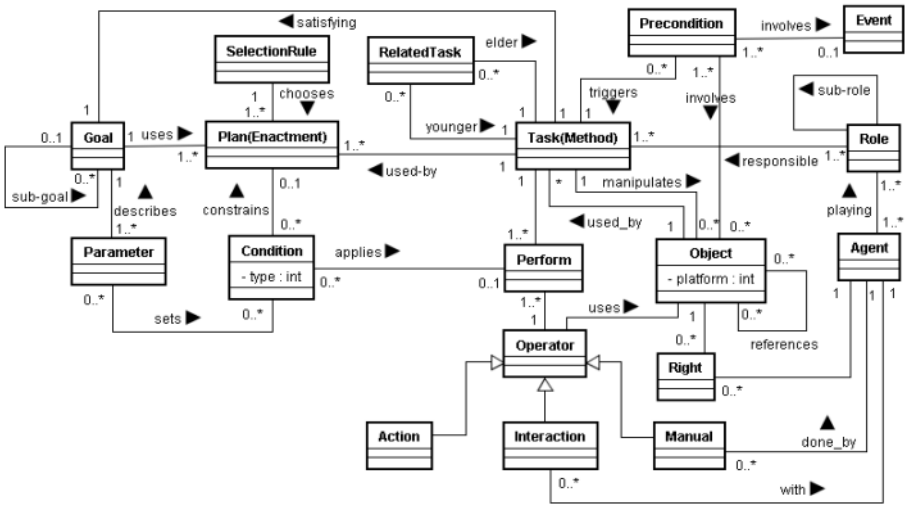

Fig. 4. The Composite Reference Task Meta-model (named the *ReTaMeta Model*)

Event, Precondition, Object, Right: The *Precondition* for the topmost Task to be set in motion, can be either an *Event* (as in GTA) or some change in state represented in the *Object* ontology, or both. What was the *Data* entity in Diane+H is represented here as *Object*, but not just involved in the preconditions of Task, since the other meta-models examined (i.e. TKS, GOMSL, GTA and CTT) use the concept Object to represent a structure of related domain objects, and so it is the case here too in the composite task meta-model. We also include *Right* here from CTT's *Rights*.

Operator, Action, Interaction, Manual, Agent: Diane+H and CTT can be used to model tasks that are performed by either: the user (*Agent*), the system, or both (via interaction between the two). While this variation in Task was allowed for via a *type* attribute in the Diane+H meta-model (and the *category* attribute in CTT), in our Composite meta-model it is refined by sub-classing the *Operator* entity three ways: *Action* (e.g. an internal basic system action, requiring no further fine-grained representation in the task Model); *Interaction,* from which a resulting action is returned via an interaction process not worth modeling in a specific task model – e.g. a standard GUI component such as a *FileChooser* dialog widget, used to locate a filename in a standard way; *Manual* (some basic manual task the user must do themselves, outside of the system being modeled – e.g. *stand-up*).

Agent, Role, responsible: The human user is represented in the *Agent* entity, which can represent either: person; system/machine; or animal (as in TKS). Both TKS and

GTA/TWO have *Role* as an entity that an Agent can play, such that an Agent can play one-or-more Roles and a Role can be played by one-or-more Agents – the Role is *responsible* for a set of tasks (one-or-more). While TKS has Role as a simple entity, GTA/TWO has it as a Role *hierarchy*, so in our composite meta-model, we include Role as a possible hierarchy of roles, with each *responsible* for a set of *Tasks*.
Parameter: is included here from the GOMSL meta-model. The use of *Parameters* in GOMSL is dynamic, to retrieve all sorts of *method signatures*, in a '*thinking'* like manner without a specific plan. In GOMSL, *Parameter* is attached to Method, but here we attach it to Goal. The reason is that a Goal is declarative, just as a *Method Signature* is declarative. So it makes good sense to attach *Parameters* to *Goal.* Any *Task* that needs such methods/*sub-goals*, can get to the definition via the *satisfying* relationship between *Task* and *Goal*. Also, once a Goal has been reached, the associated *Parameter/s* may be used to *Set* a *Condition* upon other tasks.

4 Usage and Potential of the ReTaMeta Model

Our composite meta-model was drawn from the TA paradigm to be completely independent of Agent meta-models. We have used it successfully to do a *Conceptual Evaluation and Comparison* of many AO meta-models, to be reported elsewhere. While that was the initial reason it was devised, the *ReTaMeta Model* has other possibilities, including in the service of designing interactive systems. A computational system based on the ReTaMeta Model would inherit several interesting properties from the meta-model:

- The Interaction sub-class of Operator allows for any number of pre-existing UI components/widgets to handle standard user-system interaction.
- The three forms of *Operator* - *Action* (the System), *Interaction* (User and the System), and *Manual* (just the human User) - means that it would be highly suitable for *mixed-initiative* systems, including *human-in-the-loop* agent systems.
- The *Action* entity could be mapped to Internet services (e.g. where an *operator* is outsourced to an Internet/web service).
- The *Goal* hierarchy lends itself to a *declarative* language approach (which in turn, allows for proactive behavior by the system), while the *SelectRule, Parameter* and *Condition* entities allow for *procedural* language constructs, in the one system: a more flexible yet concise approach to solving goals, than a purely declarative one.
- *Precondition* and *Event* facilitate reactive behavior - suitable for 24/7 systems.
- The *RelatedTask* entity allows for both *network* and *tree* graphs of task structures, and the *Object* hierarchy allows for ontologies of objects to be involved.

It would have some features comparable to agent systems (e.g. proactive goal-oriented behavior), but still clearly focused on the human-user from several directions, including: the origin of the meta-model itself (from the TA paradigm); the flexibility of the *Operator* entity, and via the *Role hierarchy* with related *responsibilities* in the form of Tasks to be undertaken and achieved. In short, it could be pressed into service in designing new interactive systems in the direction of People-Oriented Programming [21] – and that is where we intend to concentrate some future research effort.

References

1. Milton, S., Kazmierczak,E.: A Study Using a Common-sense Realistic Ontology. Journal of Database Management, 15(2), pp. 19--38 (2004)
2. Annet, J.: Hierarchical Task Analysis. In: [20], pp. 67--82 (2004)
3. Card, S., Morgan,T., Newell,A.: The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Hillsdale, HJ (1983)
4. Johnson, P., Johnson, H., Waddington, R., Shouls, A.: Task-Related Knowledge Structures: Analysis, Modelling and Application. In: Fourth Conference of the British Computer Society on People and Computers IV, p. 35--62, Manchester, UK (1988)
5. van der Veer, G., van Welie, M.: Groupware Task Analysis. In: Hollnagel, E.(ed.) Handbook of Cognitive Task Analysis Design. p. 808, Lawrence Erlbaum Inc., Maywah, NJ. (2003)
6. van Welie, M. and G. van der Veer. *An Ontology for Task World Models*. in *DVS-IS'98*. 1998. Adington, UK: Springer-Verlag, Wein.
7. Kieras, D.: GOMS Models for Task Analysis. In: [20], pp. 83-116 (2004)
8. Tarby, J.-C., Barthet. M.-C.: The Diane+ Method. In: The Second International Workshop on Computer-Aided Design of User Interfaces. University of Namur, Belgium (1996)
9. Kieras, D.: Towards a practical GOMS model methodology for user interface design. In: Handbook of Human-Computer Interaction, Helander, M. (ed.). pp. 135--158. North-Holland Amsterdam. (1988)
10. Kieras, D., Polson, P.G.: An approach to the formal analysis of user complexity. In: International Journal of Man-Machine Studies, Vol. 22, pp. 365--394 (1985)
11. Johnson, P., Johnson, H.: Knowledge Analysis of Task: Task analysis and specification for human-computer systems. In: Downton A. (ed.) Engineering the Human-Computer Interface, pp. 119--144. McGraw-Hill, London (1989)
12. Johnson, P.: Human Computer Interaction: Psychology, Task Analysis and Software Engineering. McGraw-Hill International Ltd, London (1992)
13. Limbourg, Q., Vanderdonckt, J.: Comparing Task Models for User Interface Design. In: [20], pp. 135--154 (2004)
14. Balbo, S., Ozkan, N., Paris, C.: Choosing the Right Task-Modelling Notation: A Taxonomy. In: [20], pp. 445--465 (2004)
15. Balbo, S., Goschnick, S., Tong, D., Paris, C.: Leading Web Usability Evaluations to WAUTER, In: Proceedings of the 11th AusWeb, Gold Coast, Australia (2005)
16. Lu, S., Paris, C., Vander Linden, K.: Tamot: Towards a flexible Task Modeling Tool. In: Proceedings of Human Factors, Nov., Melbourne, (2002)
17. Paris, C., Tarby, J.-C., Vander Linden, K.: A Flexible Environment for Building Task Models. In: People and Computer XV - Interaction without Frontiers, ICM-HCI-2001, pp. 313--330. Springer-Verlag, London (2001)
18. CSIRO: Diane+ Formalisms, http://www.ict.csiro.au/staff/cecile.paris/from-cmis/projects/isolde/tamot/onlinehelp/Diane+H.htm , last accessed: 15th February (2010)
19. Paterno, F.,: ConcurTaskTrees: An Engineered Notation for Task Models. In: [20], pp. 483-501 (2004)
20. Diaper, D., Stanton, N. (eds.): The Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates, Mahwah (2004)
21. Goschnick, S.: People-Oriented Programming: from Agent-Oriented Analysis to the Design of Interactive Systems. In: Jacko, J.A. (Ed.): Human-Computer Interaction, Part I, HCI International 2009, LNCS 5610, pp. 836-845, Springer-Verlag, Berlin (2009)