Specification and Application of a Taxonomy for Task Models in Model-Based User Interface Development Environments

Gerrit Meixner
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Gerrit.Meixner@dfki.de

Marc Seissler
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Marc.Seissler@dfki.de

Marius Orfgen
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Marius.Orfgen@dfki.de

Abstract—**This paper presents a taxonomy allowing for the evaluation of task models with a focus on their applicability in model-based user interface development processes. Task models are explicit representations of all user tasks which can be achieved through a user interface. It further supports the verification and improvement of existing task models, and provides developers with a decision-making aid for the selection of the most suitable task model for their development process or project. The taxonomy is applied on the Useware Markup Language 1.0, the ConcurTaskTrees notation and the AMBOSS notation. The results of the application are briefly described in this paper which led to the identification of substantial improvement potentials for the Useware Markup Language.**

Keywords-Task model; Taxonomy; useML; CTT; AMBOSS; Model-based User Interface Development; MBUID.

I. INTRODUCTION

This contribution is a revised and extended version of our ACHI 2011 paper [22].

The improvement of human-machine-interaction is an important field of research reaching far back into the past [26]. Yet, for almost two decades, graphical user interfaces have dominated their interaction in most cases. In the future, a broader range of paradigms will emerge, allowing for multi-modal interaction incorporating e.g., visual, acoustic, and haptic input and output in parallel [46]. But also the growing number of heterogeneous platforms and devices utilized complementarily (e.g., PC's, smartphones, PDA) demand for the development of congeneric user interfaces for a plethora of target platforms; their consistency ensures their intuitive use and their users' satisfaction [18].

To meet the consistency requirement, factors such as reusability, flexibility, and platform-independence play an important role for the development of user interfaces [7]. Further, the recurring development effort for every single platform, single device or even use context solution is way too high, so that a model-based approach to the abstract development of user interfaces appears to be favorable [35].

The purpose of a model-based approach is to identify high-level models, which allow developers to specify and analyze interactive software applications from a more semantic oriented level rather than starting immediately to address the implementation level [20][40]. This allows them to concentrate on more important aspects without being immediately confused by many implementation details and then to have tools, which update the implementation in order to be consistent with high-level choices. Thus, by using models, which capture semantically meaningful aspects, developers can more easily manage the increasing complexity of interactive applications and analyze them both during their development and when they have to be modified [32]. After having identified relevant abstractions for models, the next issue is specifying them with suitable languages that enable integration within development environments.

The pivotal model of a user-centric model-based development process is the task model [21]. Task models—developed during a user and use context analysis—are explicit representations of all user tasks [34]. Recently, several task modeling languages have been developed, which differ, for example, in their degree of formalization, and their range of applications. To make the selection of a suitable task modeling language simpler, this paper introduces a task model taxonomy that enables all participants involved in an integrated model-based user interface development (MBUID) process, to evaluate and compare task modeling languages.

The rest of this paper is structured as follows: Section II explains the proposed taxonomy for task models in detail. Section III gives a short introduction on the Useware Markup Language (useML) 1.0 and shows the application of the taxonomy. Section IV evaluates the ConcurTaskTrees (CTT) notation, Section V the AMBOSS notation. The paper finishes with Section VI, which gives a brief summary and an outlook on future activities.

II. THE TAXONOMY AND ITS CRITERIA

The proposed taxonomy focuses on the integration of task models into architectures for model-based development of user interfaces allowing for consistent and intuitive user interfaces for different modalities and platforms. For the evaluation of different task models, criteria describing relevant properties of these task models are needed. The criteria employed herein are based on initial work of [1] and [43], and are extended by additional criteria for task models with their application in MBUID. A summary of the criteria and their values are given in TABLE I.

A. *Criterion 1: Mightiness*

The most important criterion in the taxonomy is the mightiness of the task model. Therefore it is divided into 8 sub criteria.

According to [30], a task model must help the developer to concentrate on tasks, activities, and actions. It must focus on the relevant aspects of task-oriented user interface specifications, without distracting by complexity. Yet, the granularity of the task definition is highly relevant. For the application of a task model in a MBUID process, the task model must comprise different levels of abstraction [17], describing the whole range of interactions from abstract top-level tasks to concrete low-level actions. According to [38], it is commonly accepted that every person has her own mental representations (mental models) of task hierarchies. The hierarchical structure thereby constitutes the human's intuitive approach to the solution of complex tasks and problems. Consequently, complex tasks are divided into less complex sub-tasks [11] until a level is reached where sub-tasks can be performed easily. Normally, task models are divided into two levels of abstraction. With abstract tasks the user is able to model more complex tasks, e.g., "Edit a file." On the other hand a concrete task is an elemental or atomic task, e.g., "Enter a value." Tasks should not be modeled too detailed, e.g., like in GOMS [8] at least at development time [10].

Tasks can also be modeled from different perspectives. A task model should differentiate at least between interactive user tasks and pure system tasks [4]. Pure system tasks encapsulate only tasks, which are executed by the computer (e.g., database queries). This differentiation is preferable, because it allows for deducting when to create a user interface for an interactive system, and when to let the system perform a task automatically.

A further aspect determining the mightiness of a task model is its degree of formalization. Oftentimes, task modeling relies on informal descriptions, e.g., use cases [10] or instructional text [9]. According to [31], however, these informal descriptions do rarely sufficiently specify the semantics of single operators as well as the concatenation of multiple operators (i.e., to model complex expressions). These task models therefore lack a formal basis [37], which impedes their seamless integration into the model-based development of user interfaces [29]. On the one hand, developers need a clear syntax for specifying user interfaces, and on the other hand, they need an expressive semantic. Furthermore, the specification of a task model should be checked for correctness, e.g., with a compiler. For these reasons a task model should rather employ at least semi-formal semantics [28].

By using temporal operators (sometimes called qualitative temporal operators [16]) tasks can be put into clearly defined temporal orders [12]. The temporal order of sub-tasks is essential for task modeling [31] and opens up the road to a completely model-based development of user interfaces [17].

The attribution of optionality to tasks is another important feature of a task modeling language [1]. By itemizing a task as either optional or required, the automatic generation of appropriate user interfaces can be simplified. Similarly, the specification of cardinalities for tasks [30] allows for the automatic generation of loops and iterations. Several types of conditions can further specify when exactly tasks can, must, or should be performed. For example, logical [36] or temporal [16] conditions can be applied. Temporal conditions are also called quantitative temporal operators [16].

B. *Criterion 2: Integratability*

Due to the purpose of this taxonomy, the ease of a task model's integration into a consistent (or even already given) development process, tool-chain or software architecture [17], is an important basic criterion. Therefore it is necessary to have a complete model-based view, e.g., to integrate different other models (dialog model, presentation model, etc.) in the development process [41]. Among others, the unambiguity of tasks is essential, because every task must be identified unequivocally, in order to match tasks with interaction objects, and to perform automatic model transformations [45].

C. *Criterion 3: Communicability*

Although task modeling languages were not explicitly developed for communicating within certain projects, they are suitable means for improving the communication within a development team, and towards the users [33]. Task models can be employed to formalize [1], evaluate [36], simulate [31] and interactively validate [3] user requirements. A task model should therefore be easily, preferably intuitively understandable, and a task modeling language must be easy to learn and interpret. Semi-formal notations have shown to be optimally communicable [28] in heterogeneous development teams.

D. *Criterion 4: Editability*

This criterion defines how easy or difficult the creation and manipulation of a task model appears to the developer [6]. In general, we can distinguish between plain-text descriptions like e.g., GOMS [8] and graphical notations like e.g., CTT [30] or GTA [43]. For the creation of task models, graphical notations are better utilizable than textual notations [12]. For example, graphical notations depict hierarchical structures more intuitively understandable. Here, one can further distinguish between top-down approaches like CTT, and left-right orders such as in GTA.

Although this fourth criterion is correlated to the third one (communicability), they put different emphases. For every graphical notation, obviously, dedicated task model editors are essential [31].

E. *Criterion 5: Adaptability*

This criterion quantifies how easily a task model can be adapted to new situations and domains of applications. This applies especially to the development of user interfaces for different platforms and modalities of interaction. The adaptability criterion is correlated to the mightiness criterion. Especially while using task models in the development process of user interfaces for ubiquitous computing applications [44], run-time adaptability is an important criterion [5], which must be considered.

F. *Criterion 6: Extensibility*

The extensibility of a task modeling language is correlated to its mightiness and adaptability. This criterion reveals the ease or complicacy of extending the semantics and the graphical notation of the task modeling language. This criterion is highly significant, because it is commonly agreed that there is no universal task modeling language, which can be applied to all domains and use cases [6]. In general, semi-formal notations are more easily extendable than fully formal ones. Formal notations are usually based on well-founded mathematical theories, which rarely allow for fast extensions.

G. *Criterion 7: Computability*

Computability quantifies the degree of automatable processing of task models. This criterion evaluates, among others, the data management, including the use of well-established and open standards like XML as data storage format. Proprietary formats should be avoided, because they significantly hinder the automatic processing of task models.

H. *Summary*

Some of the criteria are partly correlated, e.g., the Editability criterion is aiming in the same direction as the Communicability criterion, but their focus in terms of usability is quite different (see Figure 1). The Adaptability criterion is correlating with the Mightiness and the Extensibility criteria. Furthermore the Extensibility criterion is correlated to the Mightiness criterion.

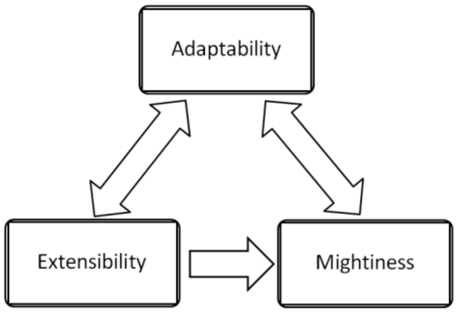

Figure 1: Correlating criteria

Table 1 shows all criteria and their possible values. All these possible values are more or less subjective. According to [6], the definition of more precise values is not possible, because there are no suitable metrics for value quantification.

TABLE I. CRITERIA AND VALUES

Criterion	Values
1. Mightiness	High, Medium, Low
a. Granularity	High, Medium, Low
b. Hierarchy	Yes, No
c. User- and system task	Yes, No
d. Degree of formalization	High, Medium, Low
e. Temporal operators	Yes, No
f. Optionality	Yes, No
g. Cardinality	Yes, No
h. Conditions	High, Medium, Low
2. Integratability	High, Medium, Low
3. Communicability	High, Medium, Low
4. Editability	High, Medium, Low
5. Adaptability	High, Medium, Low
6. Extensibility	High, Low
7. Computability	High, Low

III. EVALUATION OF USEWARE MARKUP LANGUAGE 1.0

This section gives a short introduction on the Useware Markup Language (useML) 1.0 and shows the application of the taxonomy.

A. *Overview of useML 1.0*

The Useware Markup Language (useML) 1.0 has been developed by Achim Reuther [36] to support the user- and task-oriented Useware Engineering Process [46] with a modeling language that could integrate, harmonize and represent the results of an initial analysis phase in one use model in the domain of production automation. Figure 2 visualizes the structure of useML 1.0. Accordingly, the use model abstracts platform-independent tasks, actions, activities, and operations into use objects that make up a hierarchically ordered structure. Each element of this structure can be annotated by attributes such as eligible user groups, access rights, importance. Use objects can be further structured into other use objects or elementary use objects. Elementary use objects represent the most basic, atomic activities of a user, such as entering a value or selecting an option. Currently, five types of elementary use objects exist [25]:

- Inform: the user gathers information from the user interface
- Trigger: starting, calling, or executing a certain function of the underlying technical device (e.g., a computer or field device)
- Select: choosing one or more items from a range of given ones
- Enter: entering an absolute value, overwriting previous values
- Change: making relative changes to an existing value or item

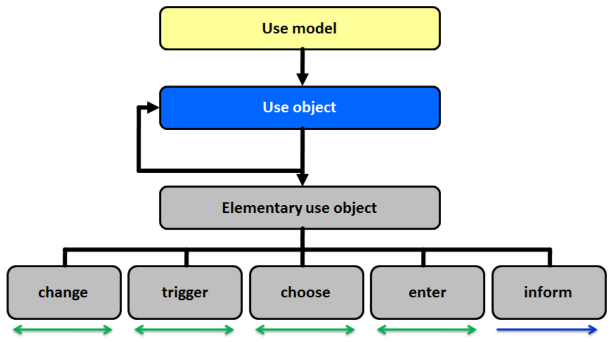

Figure 2: Schematic of useML 1.0

B. *Mightiness of useML 1.0*

a) Granularity

useML 1.0's differentiation between use objects and five types of elementary use objects is sufficiently granular. With the classification of these elementary use objects types, corresponding, abstract interaction objects can be determined [36]—which the rougher differentiation of task types in the de facto standard CTT does not allow [2] [18] [39].

b) Hierarchy

The hierarchical structure of the use model satisfies the Hierarchy sub-criterion of this taxonomy. Beside hierarchical structures, useML 1.0 also supports other structures, e.g., net structures.

c) User and System Task

The use model by [36] focuses on the users' tasks, while those tasks, which are fulfilled solely by the (computer) system, cannot be specified. Yet, for subsequently linking the use model to the application logic of a user interface, this task type is also required [2]. Querying a database might be such a pure system task, which however, might require that the query results are being presented to the user in an appropriate way. Pure system tasks can obviously be a part of a more complex, interactive action.

d) Degree of formalization

The use model or the useML 1.0 language can be categorized as semi-formal. Though useML 1.0 is not based on formal mathematical fundamentals as e.g., Petri Nets [13], its structure is clearly defined by its XML schema. It allows, among others, for syntax and consistency checks, which ensure that only valid and correct use models can be created.

e) Temporal Operators

For the current useML 1.0 specification, no temporal operators were specified, which constitutes a substantial limitation for the later integration of useML 1.0 into a fully model-based development process.

In [36], Reuther himself admits that useML 1.0 does not possess temporal interdependencies between tasks. Task interdependencies must therefore be specified with other notations such as, e.g., activity diagrams. Such a semantic break, however, impedes developers in modeling the dynamics of a system, because they need to learn and use different notations and tools, whose results must then be consolidated manually. This further broadens the gap between Software- and Useware Engineering [46].

f) Optionality

The current useML 1.0 version cannot indicate that certain use objects or elementary use objects are optional or required, respectively. Although there is a similar attribute, which can be set to a project-specific, relative value (between 1 and 10, for example), this is not an adequate mean for formally representing the optionality of a task.

g) Cardinality

There are no language elements in useML 1.0 that specify the cardinality (repetitiveness) of a task's execution.

h) Conditions

Although use models allow for specifying logical pre- and post-conditions, they don't support quantitative temporal conditions. Also, they lack means for specifying invariant conditions that must be fulfilled at any time during the accomplishment of the respective task.

C. *Integratability of useML 1.0*

Since no other models or modeling languages instead of use models or useML 1.0, respectively, have been applied and evaluated within projects pursuing the Useware Engineering Process, it is difficult to assess the applicability of use models into an integrated MBUID architecture. Luyten mainly criticized the lack of dialog and presentation models complementing useML 1.0 [18].

Further, no unambiguous identifiers exist in useML 1.0, which however, are required for linking (elementary) use objects to abstract or concrete interaction objects of a user interface—currently, use objects and elementary use objects can only be identified by their names that, of course, don't need to be unique. UseML 1.0 must therefore be extended to arrange for unique identifiers for (elementary) use objects, before it can be integrated into a complex architecture comprising multiple models representing relevant perspectives on the interaction between humans and machines. Until then, the integratability of useML 1.0 into such a model-based architecture must be rated low.

D. *Communicability of useML 1.0*

Since Useware Engineering demands for an interdisciplinary, cooperative approach [25], use models and useML 1.0 should be easily learnable and understandable. Being an XML dialect, in principal, useML 1.0 models can be viewed and edited with simple text or XML editors. Yet, these representations are difficult to read, understand, and validate. Readers with little knowledge in XML will have problems handling use models this way. Much better readability is achieved with the web-browser-like presentation of use models in the useML-Viewer by Reuther [36] (see Figure 3).

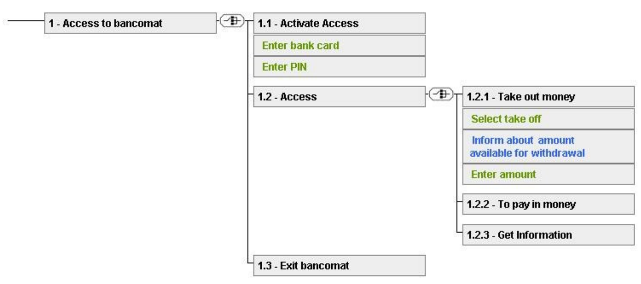

Figure 3: Excerpts of a use model as presented by the useML-Viewer

This HTML-based viewer allows for easily reading, understanding, and evaluating use models even without any knowledge in XML. It also prints use models using the web browsers' printer functions. However, the quality of the print is rather bad, among other reasons, because use models cannot be scaled to preferred paper sizes. Finally, the useML-Viewer can only display and print static use models, but does not provide means for interactive simulations or for the validation and evaluation of use models. Therefore, the communicability of useML 1.0 can only be rated medium.

E. *Editability of useML 1.0*

Though a simple editor may be sufficient for editing useML 1.0 models, XML editors are much more comfortable tools, especially those XML editors that run validity checks. Naturally, however, common versatile XML editors from third party developers are not explicitly adapted to the specific needs of useML 1.0. Therefore, they cannot provide adequate means to simply and intuitively edit use models. The editability criterion of useML 1.0 must be rated low.

F. *Adaptability of useML 1.0*

useML 1.0 had been developed with the goal of supporting the systematic development of user interfaces for machines in the field of production automation. It focuses on the data acquisition and processing during the early phases of the Useware Engineering Process. Tasks, actions, and activities of a user are modeled in an abstract and platform-independent way. Thereby, the use model can be created already before the target platform has been specified. useML 1.0 provides for the incorporation of the final users and customers during the whole process, by allowing for the automatic generation of structure prototypes.

The project-specific attributes (e.g., user groups, locations, device types) can be assigned as needed, which means that useML 1.0 can be employed for a huge variety of modalities, platforms, user groups, and projects. Among others, useML 1.0 has already been applied successfully, e.g., in the domain of clinical information system development [19]. In conclusion the adaptability criterion can be rated high.

G. *Extensibility of useML 1.0*

The fact that useML 1.0 is not strictly based upon well-grounded mathematical theories, actually simplifies its enhancement and semantic extension. This can simply be done by modifying the XML schema of useML 1.0.

In most cases, however, not even this is necessary, because useML 1.0 comprises a separate XML schema containing project-specific attributes (e.g., user groups, locations, device types), which can easily be adjusted without changing the useML 1.0's core schema. Since this allows for storing an unlimited number of use-case or domain-specific useML 1.0 schemes, the extensibility of useML 1.0 can be rated high.

H. *Computability of useML 1.0*

Since useML 1.0 is a XML dialect, use models can be further processed automatically. Employing dedicated transformations (e.g., XSLT style sheet transformations) prototypes can be generated directly from use models [25].

I. *Summary of the evaluation of useML 1.0*

The subsequently depicted table summarizes the evaluation of useML 1.0. Those criteria that were rated "No" or "Low", highlight severe deficits of the language. Figure 4 visualizes the results of the evaluation in a radar chart that reveals these deficits: They identify starting points for the upcoming, and for future improvements of the useML 1.0.

TABLE II. CRITERIA AND VALUES OF USEML 1.0

Criterion	Values
1. Mightiness	**Low**
a. Granularity	High
b. Hierarchy	Yes
c. User- and system task	**No**
d. Degree of formalization	Medium
e. Temporal operators	**No**
f. Optionality	**No**
g. Cardinality	**No**
h. Conditions	Medium
2. Integratability	**Low**
3. Communicability	Medium
4. Editability	**Low**
5. Adaptability	High
6. Extensibility	High
7. Computability	High

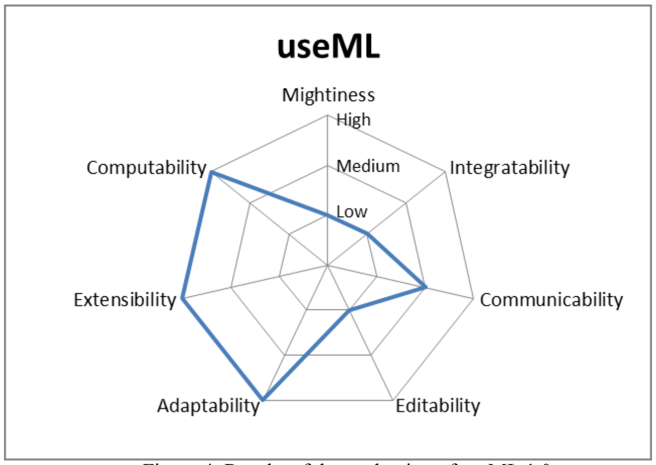

Figure 4: Results of the evaluation of useML 1.0

IV. Evaluation of CTT

This section gives a short introduction on CTT and shows the application of the taxonomy.

A. *Overview of CTT*

The notation of CTT was developed by Fabio Paternò in 1995. CTT can be seen as an extension of notations like LOTOS [15] with a graphical syntax. In difference to other graphical notations like GTA [42] it features the temporal operators Interruption and Optionality. It is used in the description of task models and one of the most common notations, which is aided further through support with different tools, e.g. CTTE (ConcurTaskTree Environment).

CTTs form a hierarchic tree structure and provide several operations to model temporal relationships in tasks. It focuses on "the activities that the users aim to perform" [31], thereby abstracting from low-level application tasks.

B. *Mightiness of CTT*

a) Granularity

CTT differentiates between four task categories: User tasks, which are performed by the user alone, "usually [...] important cognitive activities" [30], application tasks, which are "completely executed by the application" [30], interaction tasks, where user and application interact and abstract tasks, "which require complex activities whose performance cannot be universally allocated, for example, a user session with a system." [30]. The categories can be used at every abstraction level, from very high-level tasks to very concrete ones.

Although the differentiation into four types of tasks and further information in form of task relations has been provided, it is not sufficient to specify all user tasks clearly and efficiently. For example, the abstract task type can contain tasks like using an application (which involves physical movement, cognitive activities, interaction and application tasks). The whole structure has a higher granularity than other task models, which might provide a possibility of defining the structure with better classification for tasks enabling easier identification.

b) Hierarchy

CTT has a hierarchical tree structure. "It provides a wide range of granularity, allowing large and small task structures to be reused, and it enables reusable task structures to be defined at both low and high semantic levels." [30].

In contrast to useML 1.0, CTT has no explicit concept of primitive or atomic tasks.

c) User and System Task

With CTT it is possible to specify interactive user tasks as well as system tasks. It further differentiates between user tasks that involve interaction and those that do not (e.g. mental processes) [30].

d) Degree of formalization

The CTTE tool can save CTTs as "… XML format. To this end, the DTD format for task models specified by CTTs has been developed. Its purpose is to indicate the syntax for XML expressions that correctly represent task models." [31].

However, the task descriptions are given as informal text, which can only be further processed manually.

Therefore the degree of formalization can be rated as semi-formal.

e) Temporal Operators

CTT supports several temporal operators [31] (see TABLE III.).

TABLE III. Temporal operators of CTT

Operator	**Description**
Hierarchy	This operator is used for decomposing tasks into less abstract subtasks. A subset of the subtasks has to be performed to perform the decomposed task.
Enabling	Specifies that a "second task cannot begin until [the] first task has been performed." [31]
Enabling with information passing	Like Enabling, but information that is produced in the first task is provided as input to the second one.
Choice	With this operator, starting one task disables the other.
Concurrent tasks	This operator specifies that two tasks "can be performed in any order, or at the same time, including the possibility of starting a task before the other one has been completed" [31].
Concurrent Communicating Tasks	With this operator, it is possible to specify concurrent tasks that also exchange information while being performed.
Task independence	Specifies that "Tasks can be performed in any order, but when one starts then it has to finish before the other can start" [31]
Disabling	With the "disabling" operator, a "task is completely interrupted by the

	second task" [31]. The interrupted task cannot be resumed.
Suspend-Resume	This operator extends the "disabling" operator by allowing to resume the interrupted task after the interrupting task has been finished. After completion of the second task, the first "can be reactivated from the state reached before." [31]

f) Optionality

The operator 'Optional tasks' allows choosing whether the mentioned task is optional [24]. In [30] it is noted that optionality can only be used with concurrent or sequential operators.

g) Cardinality

CTT supports cardinality with the operator 'Iteration', which indicates "that the tasks are performed repetitively […] until the task is deactivated by another task." [24] .

Another operator "Finite Iteration" is used to define a fixed number of iterations [31].

h) Conditions

It is possible to specify preconditions with CTT: "For each single task, it is possible to directly specify a number of attributes and related information. […] General information includes […] indication of possible preconditions." [31]

C. Integratability of CTT

CTT models created with CTTE can be imported into the tool MARIAE [48]. MARIAE allows mapping of system tasks to web service descriptions, which can then be used to derive web-based user interfaces. It can also be used to generate Abstract User Interfaces (AUI) from CTT models.

Integration into another tool therefore exists and the Integratability can therefore be rated as Medium.

D. Communicatability of CTT

The CTTE tool allows exporting and viewing of task models based on a graphical notation as well as graphical comparison of task models and simulation of the dynamic behavior. The communicability can be rated as High.

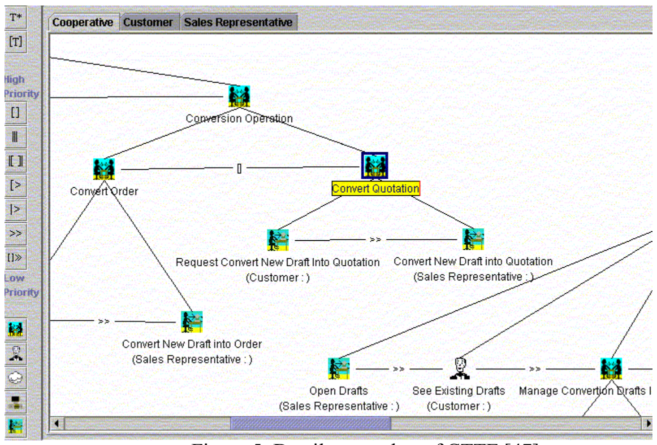
Cooperative
Customer
Sales Representative
High Priority
Low Priority
Conversion Operation
Convert Order
Convert Quotation
Request Convert New Draft Into Quotation (Customer :)
Convert New Draft into Quotation (Sales Representative :)
Convert New Draft into Order (Sales Representative :)
Open Drafts (Sales Representative :)
See Existing Drafts (Customer :)
Manage Convertion Drafts I

Figure 5: Detail screenshot of CTTE [47]

E. Editability of CTT

The CTTE tool allows editing of task models based on a graphical notation (see Figure 5) and annotation with informal descriptions.

Models can be checked for completeness and compared graphically. CTTE allows simulation of the dynamic behavior. The editability can be rated as High.

F. Adaptability of CTT

CTT is a general-purpose model for describing tasks. Since user tasks, application tasks as well as interactions can be described, it can be used to specify interfaces from a user perspective or by taking into account internal behavior of the application. The annotation of tasks with roles further helps to specify models for a wide range of domains. The Adaptability has therefore to be set to High.

G. Extensibility of CTT

CTT is integrated into several graphical environments like CTTE or MARIAE. While this improves the editability and communicatability of CTT, it has the drawback of making changes to the notation difficult, since it requires updating the environments as well, with substantial development effort. Extensibility is therefore also set to Low.

H. Computability of CTT

CTTE saves CTTs as an XML format. A "… DTD format for task models specified by CTTs has been developed. Its purpose is to indicate the syntax for XML expressions that correctly represent task models.

This can be useful to facilitate the possibility of analyzing its information from other environments or to build rendering systems able to generate user interfaces for specific platforms using the task model as abstract specification." [24]. The Computability can therefore be set to High.

I. Summary of the evaluation of CTT

While CTT supports every subcriterion of mightiness, properties like granularity or the degree of formalization is only moderately supported, leading to the overall medium rating for mightiness.

TABLE IV. CRITERIA AND VALUES OF CTT

Criterion			Values
1.		Mightiness	Medium
	a.	Granularity	Medium
	b.	Hierarchy	High
	c.	User and System task	Yes
	d.	Degree of formalization	Medium
	e.	Temporal operators	Yes
	f.	Optionality	Yes
	g.	Cardinality	Yes
	h.	Conditions	Medium
2.		Integratability	Medium
3.		Communicability	High
4.		Editability	High
5.		Adaptability	High
6.		Extensibility	**Low**
7.		Computability	High

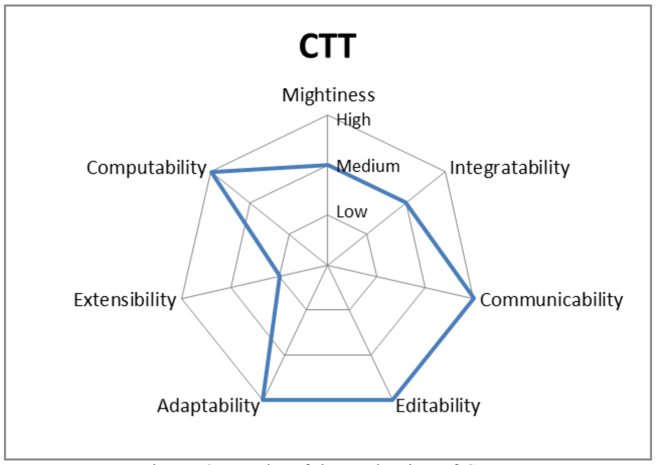

Figure 6: Results of the evaluation of CTTE

V. Evaluation of AMBOSS

This section gives a short introduction on AMBOSS and shows the application of the taxonomy.

A. Overview of AMBOSS

AMBOSS [14] is a graphical editing tool for the task model approach of the same name. The model and the environment are tightly integrated, resulting in good editability but drawbacks on the extensibility.

The tool was developed from 2005 to 2006 at the University of Paderborn. While it can be used for general-purpose task modeling, its focus lies in the support for modeling of properties for safety-critical systems. Additionally to task objects and roles, it supports barriers and risk factors.

B. Mightiness of AMBOSS

a) Granularity

With AMBOSS, tasks of different abstraction levels can be defined. The tool allows high flexibility for creating task models and consistency checkers for validating the correct structure afterwards. Its granularity can therefore be set to high.

b) Hierarchy

Tasks can be abstract or concrete and can be refined into more concrete subtasks. It therefore supports hierarchy.

c) User- and system task

AMBOSS has three basic types of actors: human, system and abstract [14]. Abstract tasks are "tasks performed in co-operation between"[14] human and system. Human tasks are performed just by the human, similar to CTT. System tasks are also similar to CTT. AMBOSS therefore supports user- and system tasks.

d) Degree of formalization

The AMBOSS environment contains checkers that test for cycles other constraints. The resulting task model is formal enough so it can be simulated. Its formalization is therefore high.

e) Temporal operators

"AMBOSS contains six different temporal relations" [14] (see TABLE V.).

TABLE V. AMBOSS TEMPORAL RELATIONS

Operator	**Description**
Fixed Sequence	Subtasks have to be performed in a fixed sequence.
Sequence with arbitrary order	Subtasks can be performed in any order.
Parallel	Subtask can be started and stopped independently.
Simultaneous	"All subtasks have to start before any subtask may stop." [14]
Alternative	"Exactly one subtask is performed." [14]
Atomic	This task has no further subtasks.

f) Optionality

AMBOSS allows a temporal relationship called "ALT" (for alternative), which means that exactly one subtask is being performed. There exist different temporal relationships for defining, which tasks can or must run parallel or separate. It therefore supports optionality.

g) Cardinality

There are no language elements to define how many times a task has to be executed.

h) Conditions

AMBOSS supports "two different types of preconditions. Message preconditions" [14] and barriers. There seems to be no support for postconditions or invariants. Conditions are therefore rated as medium.

C. Integratability of AMBOSS

The AMBOSS environment provides an API for linking other analysis tools to it. It also uses a XML-based storage format. While a plug-in mechanism allows extension of AMBOSS with new functionality, the storage format is not standardized, resulting only in a medium integratability.

D. Communicatability of AMBOSS

The language features the refinement of tasks into subtasks as well as temporal operators. Task models are created and viewed using the AMBOSS environment, which is a graphical application based on the Eclipse Rich Client Platform [50]. Besides creation and viewing, the environment allows simulation and validation of models. Communicability can therefore be set to High.

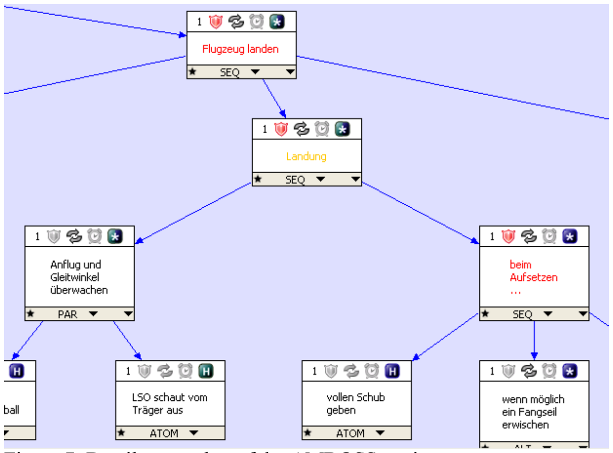
Flugzeug landen
SEQ
Landung
SEQ
Anflug und Gleitwinkel überwachen
PAR
beim Aufsetzen ...
SEQ
ball
LSO schaut vom Träger aus
ATOM
vollen Schub geben
ATOM
wenn möglich ein Fangseil erwischen
ALT

Figure 7: Detail screenshot of the AMBOSS environment

E. *Editability of AMBOSS*

The AMBOSS environment [49] allows tree-based and free-form editing. Nodes can be placed on arbitrary positions and connected later.

AMBOSS implements structural constraints that check for design errors (e.g. cycles). The integrated simulator allows testing and evaluating AMBOSS task models.

While the focus of the simulation is the checking of safety-criticality of a given task model, it can be used to generally simulate the temporal behavior of the modeled tasks.

The editability has therefore been rated as High.

F. *Adaptability of AMBOSS*

The focus of AMBOSS is the safety-criticality of systems. Other than that, there is no specific domain for this language and it is therefore adaptable for different platforms and modalities. Therefore the Adaptability is set to high.

G. *Extensibility of AMBOSS*

Since the AMBOSS approach is tightly integrated with the AMBOSS modeling environment, extensions of the model require also adapting the environment, which requires investing development effort. Therefore the Extensibility has to be set to low.

H. *Computability of AMBOSS*

AMBOSS imports and exports are files in a custom, but XML-based format. Therefore, tools can be created that parse or convert the format, but since the format is not standardized, it might change in future versions. The computability can therefore be set to Medium.

I. *Summary of the evaluation of AMBOSS*

Based on the subcriterion of mightiness, it can be rated as high. While the cardinality can be an important factor (and a possible improvement for AMBOSS), the other subcriteria support this rating.

TABLE VI. CRITERIA AND VALUES OF AMBOSS

		Criterion	Values
1.		Mightiness	High
	a.	Granularity	High
	b.	Hierarchy	High
	c.	User and System task	Yes
	d.	Degree of formalization	High
	e.	Temporal operators	Yes
	f.	Optionality	Yes
	g.	Cardinality	**No**
	h.	Conditions	Medium
2.		Integratability	Medium
3.		Communicability	High
4.		Editability	High
5.		Adaptability	High
6.		Extensibility	**Low**
7.		Computability	Medium

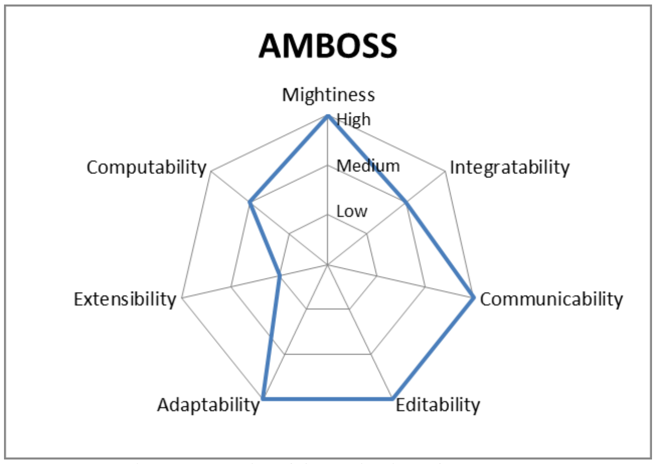

Figure 8: Results of the evaluation of AMBOSS

VI. CONCLUSION AND OUTLOOK

In this paper, a taxonomy for task models has been proposed to simplify the selection of the most suitable task model for projects employing model-based development processes for user interfaces.

Furthermore, to show the feasibility of the task model taxonomy, it has been applied to the task model notations useML 1.0, CTT and AMBOSS.

The application of the taxonomy on useML 1.0 showed the need for enhancing useML 1.0 semantically, while the specific strengths and weaknesses of CTT [31] and AMBOSS [14] as shown in the analysis can be used to improve task models that lack these strengths. The analysis further showed a general inverse correlation between editability and extensibility.

Based on the evaluations, the existing models should be extended to provide the properties that they currently lack. Also, the criteria should be evaluated in the context of model-based user interface development projects to refine their individual importance and impact on the modeling of tasks.

REFERENCES

[1] S. Balbo, N. Ozkan, and C. Paris, "Choosing the right task modelling notation: A Taxonomy" in the Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. Stanton, Eds., Lawrence Erlbaum Associates, pp. 445–466, 2003.

[2] M. Baron and P. Girard, "SUIDT: A task model based GUI-Builder", Proc. of the 1st International Workshop on Task Models and Diagrams for User Interface Design, 2002.

[3] M. Biere, B. Bomsdorf, and G. Szwillus, „Specification and Simulation of Task Models with VTMB", Proc. of the 17th Annual CHI Conference on Human Factors in Computing Systems, ACM Press, New York, pp. 1–2, 1999.

[4] B. Bomsdorf and G. Szwillus, "From task to dialogue: Task based user interface design", SIGCHI Bulletin, vol. 30, nr. 4, pp. 40–42, 1998.

[5] K. Breiner, O. Maschino, D. Görlich, G. Meixner, and D. Zühlke, "Run-Time Adaptation of a Universal User Interface for Ambient Intelligent Production Environments", Proc. of the 13th International Conference on Human-Computer Interaction (HCII) 2009, LNCS 5613, pp. 663–672, 2009.

[6] P. Brun and M. Beaudouin-Lafon, "A taxonomy and evaluation of formalism for the specification of interactive systems", Proc. of the Conference on People and Computers, 1995.

[7] G. Calvary, J. Coutaz, J., and D. Thevenin, "A Unifying Reference Framework for the Development of Plastic User Interfaces", Proc. of the Eng. Human-Computer-Interaction Conference, pp. 173-191, 2001.

[8] S. K. Card, T. P. Moran, and A. Newell, "The psychology of human-computer interaction", Lawrence Erlbaum Associates, 1983.

[9] J. Carroll, "The Nurnberg Funnel: Designing Mini-malist Instruction for Practical Computer Skill", MIT Press, 1990.

[10] L. Constantine and L. Lockwood, "Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design". Addison-Wesley, 1999.

[11] A. Dittmar, "More precise descriptions of temporal relations within task models", Proc. of the 7th International Workshop on Interactive Systems: Design, Specification and Verification, pp. 151–168, 2000.

[12] A. Dix, J. Finlay, G. D. Abowd, and R. Beale, "Human-Computer Interaction, 3rd ed., Prentice Hall, 2003.

[13] C. Girault and R. Valk, "Petri Nets for Systems Engineering", Springer, 2003.

[14] M. Giese, T. Mistrzyk, A. Pfau, G. Szwillus, and M. Detten, „AMBOSS: A Task Modeling Approach for Safety-Critical Systems", Proc. of the 2nd Conference on Human-Centered Software Engineering and 7th international Workshop on Task Models and Diagrams, Pisa, Italy, pp. 98–109, 2008.

[15] ISO/IS 8807: LOTOS – A Formal Description Based on Temporal Ordering of Observational Behaviour

[16] X. Lacaze and P. Palanque, "Comprehensive Handling of Temporal Issues in Task Models: What is needed and How to Support it?", Proc. of the 22th Annual CHI Conference on Human Factors in Computing Systems, 2004.

[17] Q. Limbourg, C. Pribeanu, and J. Vanderdonckt, "Towards Uniformed Task Models in a Model-Based Approach", Proc. of the 8th International Workshop on Interactive Systems: Design, Specification and Verification, pp. 164–182, 2001.

[18] K. Luyten, "Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development", PhD thesis, Transnationale Universiteit Limburg, 2004.

[19] G. Meixner, N. Thiels, and U. Klein, "SmartTransplantation – Allogeneic Stem Cell Transplantation as a Model for a Medical Expert System", Proc. of Usability & HCI for Medicine and Health Care, Graz, Austria, pp. 306–317, 2007.

[20] G. Meixner, D. Görlich, K. Breiner, H. Hußmann, A. Pleuß, S. Sauer, and J. Van den Bergh, "4th International Workshop on Model Driven Development of Advanced User Interfaces", CEUR Workshop Proceedings, Vol-439, 2009.

[21] G. Meixner, "Model-based Useware Engineering", W3C Workshop on Future Standards for Model-Based User Interfaces, Rome, Italy, 2010.

[22] G. Meixner, M. Seissler, "Selecting the Right Task Model for Model-based User Interface Development", Proc. of the 4th International Conference on Advances in Computer-Human Interactions, pp. 5–11, 2011.

[23] T. Mistrzyk and G. Szwillus: Modellierung sicherheitskritischer Kommunikation in Aufgabenmodellen, i-com, vol. 7, nr. 1, pp. 39–42, 2008.

[24] G. Mori, F. Paternó , and C. Santoro: CTTE: Support for Developing and Analyzing Task Models for Interactive System Design, IEEE Transactions on Software Engineering, vol. 28, nr. 8, pp. 797–813, 2002.

[25] K. S. Mukasa and A. Reuther, "The Useware Markup Language (useML) - Development of User-Centered Interface Using XML", Proc. Of the 9th IFAC Symposium on Analysis, Design and Evaluation of Human-Machine-Systems, Atlanta, USA, 2004.

[26] B. Myers, "A brief history of human-computer interaction technology", interactions, vol. 5, nr. 2, pp. 44–54, 1998.

[27] H. Oberquelle, "Useware Design and Evolution: Bridging Social Thinking and Software Construction", in Social Thinking – Software Practice, Y. Dittrich, C. Floyd, and R. Klischewski Eds., MIT-Press, Cambridge, London, pp. 391–408, 2002.

[28] N. Ozkan, C. Paris, and S. Balbo, "Understanding a Task Model: An Experiment", Proc. of HCI on People and Computers, pp. 123–137, 1998.

[29] P. Palanque, R. Bastide, and V. Sengès, "Validating interactive system design through the verification of formal task and system models", Proc. of the IFIP Working Conference on Engineering for Human-Computer Interaction, pp. 189–212, 1995.

[30] F. Paternò, "Model-based design and evaluation of interactive applications", Springer, 1999.

[31] F. Paternò, "ConcurTaskTrees: An Engineered Notation for Task Models" in the Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. Stanton, Eds., Lawrence Erlbaum Associates, pp. 483–501, 2003.

[32] F. Paternò, "*Model-based Tools for Pervasive Usability"*, Interacting with Computers, Elsevier, vol. 17, nr. 3, pp. 291–315, 2005.

[33] C. Paris, S. Balbo, and N. Ozkan, "Novel use of task models: Two case studies", in Cognitive task analysis, J. M. Schraagen, S. F. Chipmann and V. L. Shalin, Eds., Lawrence Erlbaum Associates, pp. 261–274, 2000.

[34] C. Paris, S. Lu, and K. Vander Linden, "Environments for the Construction and Use of Task Models" in the Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. Stanton, Eds., Lawrence Erlbaum Associates, pp. 467–482, 2003.

[35] A. Puerta, "A Model-Based Interface Development Environment", IEEE Software, vol. 14, nr. 4, pp. 40–47, 1997.

[36] A. Reuther, "useML – systematische Entwicklung von Maschinenbediensystemen mit XML", Fortschritt-Berichte pak, nr. 8, Kaiserslautern, TU Kaiserslautern, PhD thesis, 2003.

[37] D. Scapin and C. Pierret-Golbreich, "Towards a method for task description: MAD", Proc. of the Conference on Work with DisplayUnits, pp. 27–34, 1989.

[38] S. Sebillotte, "Hierarchical planning as a method for task analysis: The example of office task analysis", Behavior and Information Technology, vol. 7, nr. 3, pp. 275–293, 1988.

[39] J. Tarby, "One Goal, Many Tasks, Many Devices: From Abstract User Task Specification to User Interfaces" in the Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. Stanton, Eds., Lawrence Erlbaum Associates, pp. 531–550, 2003.

[40] J. Van den Bergh, G. Meixner, K. Breiner, A. Pleuß, S. Sauer, and H. Hußmann, "5th International Workshop on Model Driven

Development of Advanced User Interfaces", CEUR Workshop Proceedings, Vol-617, 2010.

[41] J. Van den Bergh, G. Meixner, and S. Sauer, „MDDAUI 2010 workshop report", Proc. of the 5th International Workshop on Model Driven Development of Advanced User Interfaces, 2010.

[42] G. Van der Veer, B. Lenting, and B. Bergevoet: GTA: Groupware task analysis - modeling complexity. In: Acta Psychologica, Heft 91, S. 297-322, 1996

[43] M. Van Welie, G. Van der Veer, and A. Eliens, "An ontology for task world models", Proc. of the 5th International Workshop on Interactive Systems: Design, Specification and Verification, pp. 57–70, 1998.

[44] M. Weiser, "The computer for the 21st century", Scientific American, vol. 265, nr. 3, pp. 94–104, 1991.

[45] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, „Linking GUI Elements to Tasks – Supporting an Evolutionary Design Process", Proc. of the 4th International Workshop on Task Models and Diagrams for User Interface Design, pp. 27–34, 2005.

[46] D. Zuehlke and N. Thiels, „Useware engineering: a methodology for the development of user-friendly interfaces", Library Hi Tech, vol. 26, nr. 1, pp. 126–140, 2008.

[47] http://giove.isti.cnr.it/ctte.html, Retrieved at January 13, 2012.

[48] http://giove.isti.cnr.it/tools/MARIAE/home, Retrieved at January 13, 2012.

[49] http://mci.cs.uni-paderborn.de/pg/amboss/, Retrieved at January 13, 2012.

[50] http://www.eclipse.org/home/categories/rcp.php, Retrieved at January 13, 2012.