Selecting the Right Task Model for Model-based User Interface Development

Gerrit Meixner
German Research Center for Artificial Intelligence
Innovative Factory Systems
Kaiserslautern, Germany
Gerrit.Meixner@dfki.de

Marc Seissler
University of Kaiserslautern
Center for Human-Machine-Interaction
Kaiserslautern, Germany
Marc.Seissler@mv.uni-kl.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. 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. Furthermore the taxonomy is applied on the Useware Markup Language 1.0. The results of the application are briefly described in this paper which led to the identification of substantial improvement potentials.**

Keywords - Task model, Taxonomy, Useware Markup Language, Model-based User Interface Development, MBUID.

I. INTRODUCTION

The improvement of human-machine-interaction is an important field of research reaching far back into the past [22]. 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 [41]. 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 [16].

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 perseverative development effort for every single platform, single platform or even single use context solution is way too high, so that a model-based approach to the abstract development of user interfaces appears to be favorable [31].

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 [18][36]. 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 [28]. After having identified relevant abstractions for models, the next issue is specifying them through suitable languages that enable integration within development environments.

The pivotal model of a user-centric model-based development process is the task model [19]. Task models—developed during a user and use context analysis—are explicit representations of all user tasks [30]. 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 MBUID (Model-based User Interface Development) 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 followed by Section IV, which shows the application of the taxonomy on useML 1.0. The paper finishes with Section V, 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 [38], and are amended by additional criteria for task models with their application in MBUID. Following, the taxonomy and its criteria are described in detail.

A. Criterion 1: Mightiness

According to [26], 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 [15],

 ISBN: 978-1-61208-117-5

describing the whole bandwidth of interactions from abstract top-level tasks to concrete low-level actions. According to [34], 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 [27], 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 [33], which impedes their seamless integration into the model-based development of user interfaces [25]. 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 [24].

Using, for example, temporal operators (sometimes called qualitative temporal operators [14]) tasks can be put into clearly defined temporal orders [12]. The temporal order of sub-tasks is essential for task modeling [27] and opens up the road to a completely model-based development of user interfaces [15].

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 [26] 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 [32] or temporal [14] conditions can be applied. Temporal conditions are also called quantitative temporal operators [14].

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 [15], 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 [37]. 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 [40].

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 [29]. Task models can be employed to formalize [1], evaluate [32], simulate [27] 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 [24] 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 [26] or GTA [38]. 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 [27].

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 [39], 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.

 ISBN: 978-1-61208-117-5

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.

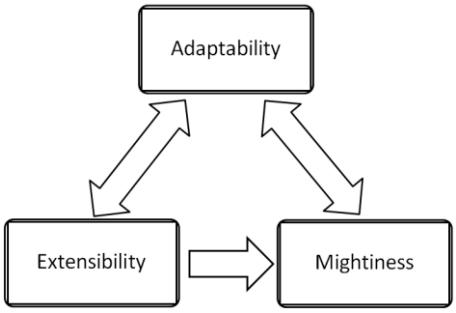

Figure 1: Correlating criteria

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. USEWARE MARKUP LANGUAGE 1.0

The Useware Markup Language (useML) 1.0 had been developed by Reuther [32] to support the user- and task-oriented Useware Engineering Process [41] with a modeling language that could integrate, harmonize and represent the results of an initial analysis phase in one common, so-called use model in the domain of production automation. 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 [21]:

- 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 visualizes the structure of useML 1.0.

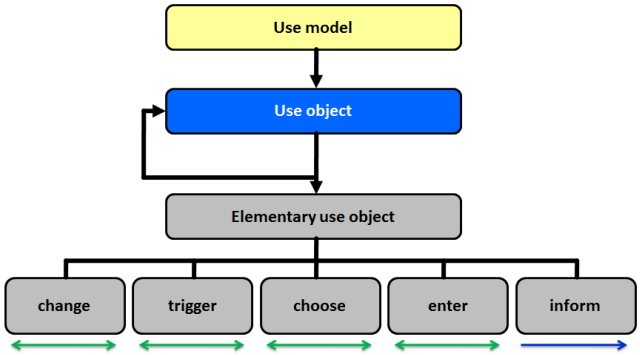

Figure 2: Schematic of useML 1.0

IV. APPLICATION OF THE TAXONOMY ON USEML 1.0

In the following subsections the application of the taxonomy on useML 1.0 is briefly described.

A. *Mightiness of useML 1.0*

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 [32]—which, the rougher differentiation of task types in the de facto standard CTT does not allow [2] [16] [35].

The use model or the useML 1.0 language, respectively, 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.

The use model by [32] focuses on the users' tasks, while those tasks which, are fulfilled solely by the (computer) system, can't 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.

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. For the current useML 1.0 specification, however, 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 [32] 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 [41].

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. Finally, the current useML 1.0 version cannot indicate that certain use objects or elementary use objects are optional or required ones, 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. Accordingly, there are no language elements in useML 1.0 that specify the cardinality (repetitiveness) of a task's execution. The value of the Mightiness Criterion is based on the values of its sub criteria. Taking into account all the sub criteria, the value of the Mightiness criterion must be evaluated low.

B. *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 [16]. 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.

C. *Communicability of useML 1.0*

Since Useware Engineering demands for an interdisciplinary, cooperative approach [21], 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 [32] (see Figure 3).

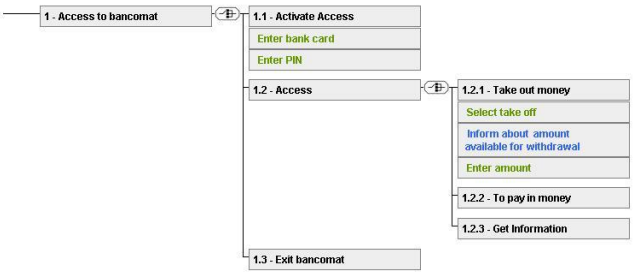

Figure 3: Excerpts of a use model as presented by the useML-Viewer

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.

D. *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

 ISBN: 978-1-61208-117-5

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.

E. 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 adjusted 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 [17]. In conclusion the adaptability criterion can be rated high.

F. 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.

G. 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 [21].

H. 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

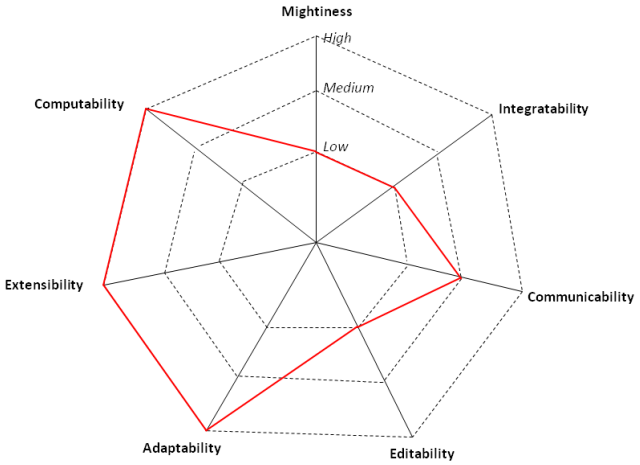

Figure 4: Results of the evaluation of useML 1.0

V. 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 on useML 1.0. Also the application of the taxonomy on useML 1.0 showed the need for enhancing useML 1.0 semantically.

Currently we're enhancing useML 1.0 in different aspects, according to the initial results of the application of the taxonomy. Additionally, we would like to improve the refinement of the criteria and apply this taxonomy to a selection of further task models, such as CTT [26] or AMBOSS [20] to proof the usefulness of this taxonomy.

REFERENCES

[1] S. Balbo, N. Ozkan, and C. Paris, "Choosing the right task modelling notation: A Taxonomy" in the Handbook of Task Analysis for

 ISBN: 978-1-61208-117-5

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] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

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

[23] 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.

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

[25] 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.

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

[27] 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.

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

[29] 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.

[30] 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.

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

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

[41] 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.