Using profiled ontologies to leverage model driven user interface generation

Werner Gaulke
University of Duisburg-Essen
Duisburg, Germany
werner.gaulke@uni-due.de

Jürgen Ziegler
University of Duisburg-Essen
Duisburg, Germany
juergen.ziegler@uni-due.de

ABSTRACT
Mobile computing and new input methods have increased the need to create multiple interfaces for one functional core. Automatic generation of user interfaces attempts a solution for this problem. Existing approaches either generate interfaces on the base of a detailed task model or use domain models in conjunction with interface specific annotations and transformation rules. While task models are very time consuming to create and cannot easily be reused domain models lack the flexibility for use cases which are not covered or in conflict with used transformation rules. Based on an overview of existing approaches this paper sets out a conceptual framework which combines both task model and ontology based concepts. It is shown that the proposed combination leads to more abstract and reusable task models.

Author Keywords
Model-Driven Development; User-Interfaces; Ontologies; Task-Modeling

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous;

INTRODUCTION
Model-based development and generation of user interfaces have for a long time been proposed as approaches that can make user interface development more systematic and effective and that can help to adapt UIs to different platforms and device characteristics. The uptake of these methods in practical system development, however, has been slow and has often lost out against advances in toolkits and programming environments. One of the reasons for this can be seen in the effort needed for specifying UI-related models such as task models and the unfamiliarity of developers with these methods. In most cases, several models have to be developed in a coordinated fashion, for example, domain and task models. In this process redundancies are often created. In particular, the task models that are frequently advocated as being central to model-based UI development, while useful for obtaining an understanding of the user's tasks, typically duplicate information that may already be contained in other models. For example, specifying the object of an operation together with the operation itself as a task repeats information – the object – that has already been described in the domain model. These issues create the need to avoid redundancies in the model specification and to provide abstractions that can decrease the overall effort needed to specify the models.

A possible approach to solving this problem is to leverage the information already contained in (essential) domain models and enrich it with information that can be used to deduce relevant features of the resulting UI.

In this paper, we present an approach that attaches user interface relevant metadata to an ontological domain model in order to reuse the added information in the interface generation process. Two major contributions are made: First, a model combing the workflow of model driven user interface development (MBUID) with domain model based user interface generation is introduced. Second, it is shown how the combination can lead to more lean and abstract task-models in terms of reduced redundancies and higher level task definitions.

The paper begins with an overview of previous research in the area of MBUID, as well as ontology driven interface generation. Findings are used to derive a combined model of both approaches. Based on the model, it is explained how domain-model information may be leveraged in task models. Finally, the concluding section outlines objectives and plans for consecutive research.

RELATED WORK
Model driven development of user interfaces is a well-established research area and traces back to the late 1960s. An extensive overview of past and present research, divided into four main generations, is given in [14]. Approaches of the current fourth generation can be characterized on the basis of the unified reference framework by Calvary et. al. [3]. The reference model is divided into four main steps. In the first step a *concepts and task model* is created to define types and actions for the intended application. The second step uses an *abstract interface* model to describe views and interactions without specifying actual user interface elements. Determination of real user interface elements is

EICS'15, June 23 - 26, 2015, Duisburg, Germany

http://dx.doi.org/10.1145/2774225.2775070

done in the *concrete interface* model during the third step. For example, an abstract 1:N selection could be transformed to a concrete, single-selection, dropdown widget. Finally, the instantiation of the *final interface* is part of step four. The drop-down widget could be transformed to HTML markup usable by any web browser. To conclude, the reference model defines a path starting with a task model leading to a user interface refined by intermediate steps.

In practice, realizations of the introduced reference model can be differentiated in regard to used modeling languages as well as the degree of automation transforming between model layers. There are many task modeling languages available, suited for user interface development [18]. Concurrent task trees (CTT) [3], Business Process Model and Notation (BPMN) [1], UsiXML [21] as well as Useware Markup Language (UseML) [15] are possible candidates to create the task-model. An attempt to further standardize task-models is undertaken in [17]. Although all approaches have a hierarchical composition of tasks and subtasks in common, technical realization and supported modeling features differ. Advanced features like events, conditions and repetitions are not common across all variants. In particular, the integration of background knowledge as described in [13] and connection of tasks with domain model objects varies across existing languages. Either objects are specified within the task model or existing objects are referenced. Details about used objects and types is crucial to transform task models into an abstract interface.

Domain model based approaches rely on contained information to generate user interfaces. A domain model describes concepts and relations of a limited field of interest in a formal way. In order to be universal and reusable, application or interface specific data is usually not integrated. However, an examination of existing domain model based research revealed that either the domain model is extended by user interface specific annotations like in [8],[10],[11] and [12] or additional rules are added [2], [5], [6] to generate the user interface. Typically, semantic languages like the resource description framework (RDF) or the web ontology language (OWL) are used to create domain models as they provide the necessary expressiveness for even complex relationships.

To conclude, this section has shown that task model as well as domain model based approaches are established methods to generate user interfaces. The following sections introduce a combination of both methods, aiming to reduce redundancies as well as provide a way to create more abstract task models.

OVERVIEW - DOMAIN MODEL BASED UI GENERATION

The following concept emphasizes the role of semantic domain models by putting it at the start of the generation process while the task-model is deferred to a subsequent step. In its entirety, the process is based on the unified reference model [3] given that the last three steps are nearly identical. Figure 1 depicts an overview of the process. The next

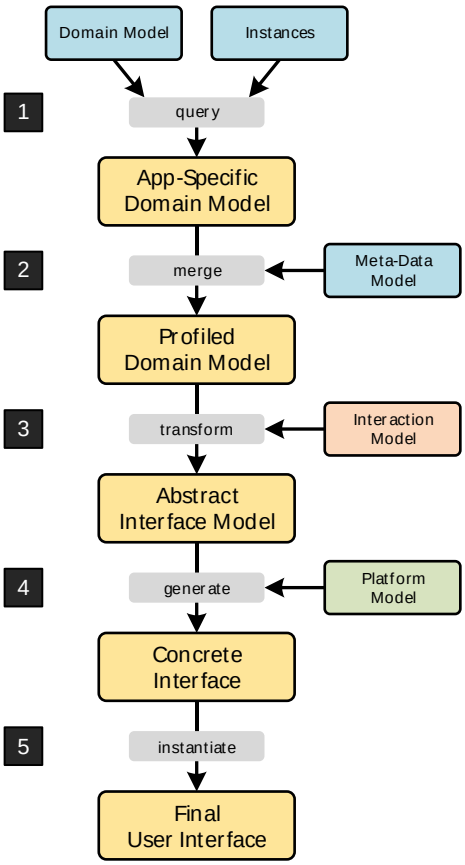

Figure 1. Domain model based interface generation process. The domain model is extended with interface specific metadata and an interaction model to derive an abstract interface. The abstract interface is finally transformed to a concrete interface adhering a given platform configuration.

sections describe all steps in more detail, accompanied by examples taken from the area of e-commerce.

First, an existing or specially created domain-model as well as available instance data is selected. Given that domain models can be very large, a query could be used to limit the data to only necessary concepts. SPARQL is a standardized query language for semantic models. It provides the capability to express CONSTRUCT queries which create new, usually smaller, models out of existing ones.

Domain models in a typical e-commerce scenario would contain a categorization and characterization of products and their attributes. A camera ontology for instance could differentiate between compact and single lens reflex camera. Additionally, auxiliary concepts like vendors, shopping cart and payment methods would be added to enable an e-commerce frontend.

Second, a metadata model is added, extending the domain model to a profiled domain model containing user interface specific information. The metadata model can be split into two major parts. The first part provides a set of user interface specific annotations which are attachable to existing domain model concepts. These annotations add information like

orders, priorities, or groups that are usually not part of an application independent domain model. The second part contains new concepts or relations targeted towards the use case but are not found in the initial model.

The retailer in this scenario wants to create an e-commerce store with non-professional consumers in mind. Priority and grouping is used to emphasize camera properties of particular interest for the target group. In this case, advanced technical details are annotated with a low priority whereas commonly known features like 'megapixels' or 'levels-of-zoom' get a high priority. Additionally a new concept 'luxury cameras' is added that was not part of the technically focused domain model.

Third, user interaction is aggregated in an interaction model that specifies possible tasks within the prospective user interface. The interaction model contains a hierarchical structure of tasks connected with concepts and relations from the underlying domain model. SPARQL queries are used to connect a task with concepts. Query creation is simplified by metadata annotations added in the last step. Both, interaction model and profiled domain model are transformed into an abstract interface model.

While the profiled domain-model contains products and properties, the interaction model defines customers' interaction using these elements. An explore task is defined to enable browsing of products containing a query to select top level product classes. The recently defined 'luxury camera' class would also be a part of this result set. Product properties could be ordered by priority. A checkout task is defined, involving the concepts of products, shopping cart and payment methods.

Fourth, the concrete user interface model is generated out of the abstract interface from the last step. An additional platform model defines specifics about targeted environments like screen size or input device. Available information about the user interface is utilized to select suitable widgets, input/output elements and layouts.

The e-commerce use case is targeted towards desktop users and uses a widget toolkit like Java Server Faces (JSF) to create concrete interface models. The task of exploring products could lead to a simple two column layout where the selection of products classes is done in a tree widget and the listing of products in a data table. Available information like priorities could be used to decide whether certain properties are displayed or not depending on available screen space.

Fifth, generated code from the fourth step is instantiated, deployed and made accessible to the user.

Generated code of the e-commerce platform is instantiated. In the case of JSF widgets a server side web application would be generated accessible by a regular web browser.

To summarize, the introduced concept in this section depicts a model driven workflow for user interface generation. Unlike task driven approaches, this workflow starts with a domain model containing all concepts and relations of the use case. A separate metadata model adds interface specific information, keeping the original model application-independent, whilst supporting tasks definition in an interaction model. Combined information is used to generate an abstract interface which leads to the final interface. On its own, this workflow is a rather small deviation from existing approaches. Hence the next section clarifies *which* metadata is added and *how* interaction models are defined.

EXTENDING THE DOMAIN MODEL WITH UI METADATA

In order to be application independent and reusable domain models tend to express concepts and relations decoupled from a concrete use case. Examples for such ontologies would be large ontologies like dbpedia, but also comparatively small domain models describing just one or more product classes for instance. All aforementioned domain model based approaches add interface specific metadata to enable and support user interface generation.

The following table contributes a unified collection of annotations extracted from related research. Information added by the annotations are essential for the approach of this paper but can be used by any approach that utilizes domain models for interface generation.

Annotation	Description
Priority [8],[16]	Defines the priority of a concept/ property (c/p). Priorities can be used by queries as a criterion for selection or as a sorting indicator for the interface.
Custom Label [4]	Overrides the given label of a c/p with a new label. The new label may be formally wrong but more suitable for the targeted user group.
Order [4],[9],[16]	Sets the order how instances of the c/p should be sorted. Overrides default natural ordering (e.g. ascending alphabetical).
Group [4],[9],[16]	Adds a group identifier to a c/p. The identifier is used to pool concepts or properties in queries or visual interface elements.
Key Element [6],[8]	Used as an indicator to mark c/p. Marked elements could be used as starting points for navigation or filters.
Widget Type [4],[9],[16]	Sets a fixed widget type for the user-interface while overriding default / derived widgets based on datatypes.
Access Type	Sets access control list (ACL) permission for c/p usage. By default all elements are read only.

Table 1. Collection of annotations to enhance the domain model with user interface specific information.

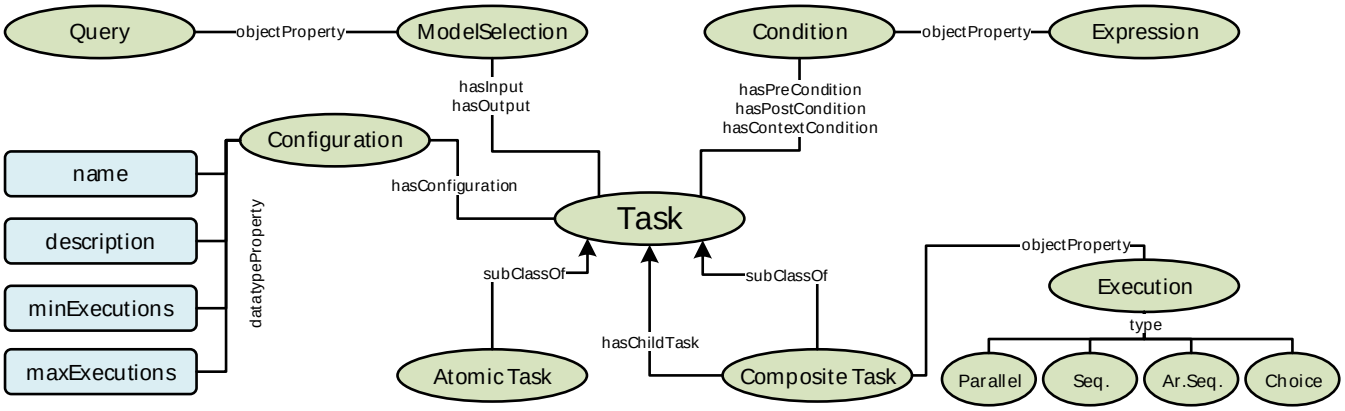

Figure 2. Interaction metamodel derived from OWL-T [20] **and the DEMISA task-Ontology** [19]**. A task can either be atomic if it has no subtask or a composite task if it combines multiple subtasks. A task can be made conditional by adding an expression as pre/post/context condition. Input/Output-connection with the domain model is done by SPARQL queries.**

Adding the introduced annotations to an ontology aligns it towards a specific use case and influences the user interface generation process. Technically OWL2 annotations are used to add the specified information. All annotations, as well as tentative, added new concepts and relations are grouped together in the *metadata model*. One domain ontology can be combined with multiple metadata models, each one targeting another use case. For example, multiple metadata models could be used to target different consumer groups by setting group specific priorities, labels and key elements.

In addition to adjusting the domain model towards the desired use-case, added metadata information can be used to leverage creation of task models which is shown in the next section.

MODELING INTERACTION

In comparison to task model based approaches the profiled domain model already provides plenty of interface specific information at this stage. This additional knowledge is used to create leaner and more abstract task definitions consolidated in a central interaction model.

The interaction model contains hierarchical task definitions made in a semantic format derived from the task ontology OWL-T [22] and the DEMISA task ontology [21]. Using a semantic format for both, the domain model, as well as the interaction model, eases the integration of domain concepts within defined tasks. Figure 2 shows the interaction metamodel in detail specifying available concepts.

The *Task* concept is the central element in the metamodel. A task can either be non-separable and is therefore an *Atomic task* or it is composed of multiple subtasks, grouping them to a *Composite Task*. Composite tasks can further be configured to influence the *Execution* of subtasks. Execution order can be determined by selecting either *parallel* for simultaneous, *sequential* for linear, *arbitrary sequential* for random or *choice* for single subtask execution. Furthermore, an additional *Configuration* can be attached to tasks, providing a *name*, *description* as well as setting a number of minimal and maximal executions by defining *minExecutions* and *maxExecutions*. Given these basic concepts, creation of task models is already possible. Similar to comparative task-modeling languages, tasks and task hierarchies can be built, even though the connection with domain model concepts is still missing at this point.

Information from the profiled domain model is incorporated in two ways. First, *input* and *output* elements of a task are determined by the *ModelSelection* element. A *Query* is used to select concepts and relations from the domain model in order to use them within the task. The *input* query selects concepts from the domain model that will be used during task processing. Tasks that manipulate data use *output* queries to indicate the type of processed concepts. Second, domain data can be used to create a *pre-*, *post-* or *context-condition*. Evaluation of conditions influences whether a task is executed or not. Pre-conditions are checked before a task is started, preventing the execution if the condition has not been met. Context-conditions are tested during task execution, preventing non valid actions based on a contextual state or property. Post-conditions are validated after the task has been processed but before it is set as finished. Conditions are defined by an *Expression*, containing a query that either evaluates to true or false when executed.

To conclude, a metamodel suited to build semantic task models has been introduced. Multiple hierarchical structured task definitions are combined in an interaction model. Tasks are linked to domain models using SPARQL queries selecting used concepts. Linking both models is the key to reuse and benefit from available domain model knowledge.

Leveraging domain model semantics

Interactions, constructed using the introduced metamodel, can further benefit from information of the profiled domain model. Information can be integrated in tasks as well as used in the user interface generation process. The next sections discuss three possible adoptions of available information.

First, the profiled domain model allows to formulate model queries more easily by using metamodel annotations as query conditions. For example, instead of manually listing all

chosen concepts, the *keyElement* annotation could be used as a query condition to select items for a top level navigation. *Priority* or *group* annotation can also be used to limit or order query results. The *accessType* can be employed to determine allowed operations for the current concepts. Metamodel annotations integrated as conditions result in queries that can be adapted to other use cases with less effort. Furthermore, queries limited just to annotations and abstract concepts like generic classes can be reused without modifications making them candidates for a query catalogue.

Second, the process of transforming profiled domain model and interaction model to the abstract user interface can benefit from available annotations. *Order* and *Priority* annotations are used to determine sorting of instances in an interface. The g*roup* is used to create visual segments which can be arranged fitting available screen sizes. *KeyElements* may be highlighted to emphasize important elements. If and how certain concepts are shown is determined by the *accessType* annotation. *Widget type* and *custom label* override inferred widget types or existing values.

The third and most advanced benefit is seen by utilizing domain model relations within the generated concrete user interface. A simple case would be the selection of appropriate input and output widgets based on specified datatypes, ranges and restrictions. Expressiveness of semantic modeling languages allows characterization of complex datatypes beyond well-known types found in programming languages or databases making calculated and composed datatypes possible. Alongside selecting appropriate widgets, a more advanced case would be the utilization of domain model relations to derive interactions and flows. For example, relations between concepts can be used to create dynamic forms with conditional input elements, depending on available subtypes without having to model each possible step in the corresponding interaction model.

Figure 3 shows a small extract of an interaction model for the aforementioned e-commerce scenario. The example shows the process of product exploration, detail viewing and finally adding them to the shopping cart. The main composite task *Choose class* marks the starting point. The task is used to create a navigation out of domain model concepts selected by an *input query*. The input query selects all classes annotated with the *KeyElement* annotation. In this particular case, a list of product classes would be returned. A query selecting top level classes could be used to achieve similar results, but especially in the area of e-commerce custom navigation hierarchies often do not reflect formal categorization. Achieving the same result without an annotated metamodel would result in far more complex queries because conditions would have to be more detailed and specific. Once the user chooses a menu item and thus selects a product category he enters subtasks of the *Choose class* task. Three subtasks, namely *Explore instances, view instance* and *Add to cart* are available. The subtasks are not tied to be executed in a particular order, however

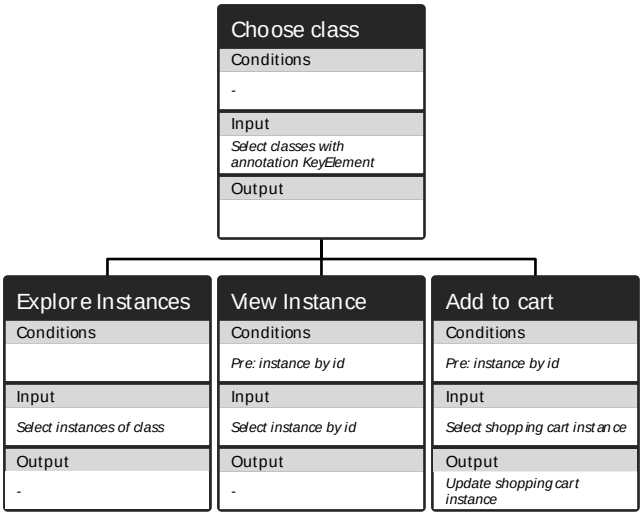

Figure 3. Extract of the e-commerce task model showing typical actions to browse products, view product and add products to a shopping cart for checkout (not shown here). Depicted queries are simplified.

preconditions can influence how interaction is done in practice. Browsing of all instances is done in the *Explore Instances* task. Based on the previously selected product class, all instances are shown. Product features are ordered according to their priority value. Additionally, filter widgets are created for the most important properties. In the domain of cameras, zoom level, megapixels and vendor could be such filters. If a product instance is selected, the *View Instance* task is executed. Without a selected instance the task would not be started as indicated by the *precondition*. Appearance of the detail view is influenced by group annotations used to create visual blocks. Relations within the domain-model, connecting product instances with other similar products or accessories, are used and displayed. Finally, the *Add to cart* task enables the user to put a product in his cart. In contrast to previously used concepts, the shopping cart instance has to be manipulated in order to be able to add the product. The *accessType* annotation is used to allow write access. As a result an updated cart instance is produced, indicated by the *output query*.

In conclusion, it has been shown how (profiled) domain models may be used to leverage model driven user interface generation. Three possible ways have been explored and disccussed: creating simplified domain queries to connect tasks with domain models, during transformation to abstract interfaces and in generated concrete interfaces.

OUTLOOK & CONCLUSION

This paper introduced a concept, combining task based and domain model based user interface generation approaches in a unified model. Goal of this combination was to remove redundancies between domain models and task models as well as integrating the domain model deeper into interface generation processes. For the latter case, a set of interface specific domain model annotations was composed. The

annotations are used to demonstrate how enriched domain models can ease the creation of task models, domain queries and the interface generation process itself.

Considerably more work is needed to clarify technical details and concepts introduced in this paper. Furthermore, a library of reusable abstract task models will be created to further reduce necessary modeling efforts. To demonstrate a real world application of the described concepts, a modeling environment, targeted towards non-technical users, is currently implemented based on previous work described in [7]. An evaluation will focus on ease of use and overall modeling efficiency compared with competitive solutions.

REFERENCES

1. Bacha, F., Oliveira, K., and Abed, M.A model driven architecture approach for user interface generation focused on content personalization. *2011 Fifth International Conference on Research Challenges in Information Science*, (2011), 1–6.
2. Butt, A.S., Haller, A., Liu, S., and Xie, L.ActiveRaUL: Automatically Generated Web Interfaces for Creating RDF Data. *semantic-web-journal.net 0*, (2013).
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J.A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers 15*, 3 (2003), 289–308.
4. Canadas, J., Palma, J., and Túnez, S.Model-Driven Rich User Interface Generation from Ontologies for Data-Intensive Web Applications. *Proceedings of the 7th Workshop on Knowledge Engineering and Software Engineering at the 14th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2011)*, (2011).
5. Correa, M.C., Deus, H.F., Vasconcelos, A.T., et al.AGUIA: autonomous graphical user interface assembly for clinical trials semantic data services. *BMC medical informatics and decision making 10*, 1 (2010), 65.
6. Eriksson, H., Fergerson, R.W., Yuval, S., and Musen, M.A.Automatic generation of ontology editors. *Proceedings of the 12th International Workshop on Knowledge Acquisition, Modelling and Mangement (KAW'99)*, (1999).
7. Gaulke, W. and Ziegler, J.Entwicklung semantischer Produktdatenmodelle durch Domänenexperten: Fehleranalyse und Werkzeugunterstützung. *Mensch & Computer 2014 - Tagungsband*, De Gruyter Oldenbourg (2014), 225–234.
8. Hildebrand, M., Ossenbruggen, J. Van, and Hardman, L./facet: A Browser for Heterogeneous Semantic Web Repositories. *The Semantic Web - ISWC 2006*, Springer Berlin Heidelberg (2006), 272–285.
9. Khushraj, D. and Lassila, O.Ontological approach to generating personalized user interfaces for web services. *The Semantic Web–ISWC 2005*, (2005), 916–927.
10. Liu, B., Chen, H., and He, W.Deriving user interface from ontologies: a model-based approach. *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, (2005), 6 pp.–259.
11. Lohmann, S., Kaltz, J., and Ziegler, J.Model-driven dynamic generation of context-adaptive web user interfaces. *Models in Software Engineering*, (2007), 116–125.
12. Macías, J. a. and Castells, P.Providing end-user facilities to simplify ontology-driven web application authoring. *Interacting with Computers 19*, 4 (2007), 563–585.
13. Martinie, C., Palanque, P., Cedex, T., and Fahssi, R.Extending Procedural Task Models by Systematic Explicit Integration of Objects , Knowledge and Information. (2013).
14. Meixner, G., Paternò, F., and Vanderdonckt, J.Past, Present, and Future of Model-Based User Interface Development. *i-com 10*, 3 (2011), 2–11.
15. Meixner, G., Seissler, M., and Breiner, K.Model-driven useware engineering. *Studies in Computational Intelligence 340*, (2011), 1–26.
16. Mori, G., Paterno, F., and Santoro, C.Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering 30*, 8 (2004), 507–520.
17. Paternò, F., Santoro, C., Raggett, D., and Davide, S.L.MBUI - Task Models. 2014. http://www.w3.org/TR/task-models/.
18. Szwillus, G.Task Models in the Context of User Interface Development. *Studies in Computational Intelligence 340*, (2011), 277–302.
19. Tietz, V., Rümpel, A., Voigt, M., Siekmann, P., and Meißner, K.Tool support for semantic task modeling. *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, (2013), 40:1–40:12.
20. Tran, V.X. and Tsuji, H.OWL-T: A task ontology language for automatic service composition. *Proceedings - 2007 IEEE International Conference on Web Services, ICWS 2007*, Icws (2007), 1164–1167.
21. Vanderdonckt, J.*A MDA-Compliant Environment for Developing User Interfaces of Information Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.