A Concerted Model-driven and Pattern-based Framework for Developing User Interfaces of Interactive Ubiquitous Applications

Jürgen Engel
Augsburg University of Applied Sciences
Augsburg, Germany
juergen.engel@hs-augsburg.de

Christian Märtin
Augsburg University of Applied Sciences
Augsburg, Germany
christian.maertin@hs-augsburg

Peter Forbrig
University of Rostock
Rostock, Germany
peter.forbrig@informatik.uni-rostock.de

ABSTRACT
Modeling and building interactive user interfaces (UI) typically requires the skills of software developers and HCI experts who cooperate with platform and marketing experts in order to arrive at solutions with the required software quality, usability, and user experience. The combination of model-driven user interface development practices with pattern-based approaches that specify HCI- and software-patterns in a formalized way and respect emerging standards offers potentialities to facilitate and at least partially automate the user interface development process, therefore reduce the time-to-market and development costs, and lead to solutions that can easily be adapted to varying contexts and target devices. Such pattern-aided UI adaptation is not limited to design time decisions but can also be applied during runtime. This paper highlights the architecture and capabilities of the Pattern-Based Modeling and Generation of Interactive Systems (PaMGIS) framework to broadly support the construction and adaptation of user interface models. It is discussed, how pattern descriptions that capture important parts of the design knowledge should be organized in order to be automatically processed during the modeling process.

Author Keywords
User interfaces; interactive systems; model-driven development; pattern-based development.

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

INTRODUCTION
Highly interactive software has become a crucial ingredient of modern human life. Independent of time and location, people are used to interact with products built around interactive software components, such as web applications, telecommunication devices, car navigation systems, smart home appliances, wearables, or other electronic equipment. Nowadays users expect that software products run on a variety of heterogeneous devices with a consistent look and feel, invariable high usability, and an extremely high degree of appealing user experience. Additionally, users tend to be impatient and want to have the software available for their different devices at the same point in time. Therefore, time-to-market is vital for software suppliers.

It is nearly impossible to meet all requirements simultaneously when exercising traditional software engineering and development processes. A promising way out of this dilemma is the application of a model-driven approach that allows for describing the particular aspects of the intended user interface by means of distinct models at different abstraction levels which can be created - at least partially - by automatic transformations.

We have combined model-driven user interface development practices with pattern-based approaches that specify HCI-patterns in a formalized way [6]. Result is the fundamentally renovated PaMGIS 2.0 framework which is presented in the following sections.

RELATED WORK
Model-based user interface development environments (MB-UIDE) introduce models to the development process of interactive applications. A variety of existing MB-UIDE and model-driven approaches for facilitating the development process of interactive systems can be found in the literature. Related recapitulations and discussions are provided in [3] [5] [9] [12]. The models used by these approaches are usually task-based or object-oriented and contain functional domain and data requirements at different abstraction levels for the interactive system under development. Typically, the models are also used for mapping and linking the functional requirements of the business logic to the different abstract and concrete representations of the user interface with the intent to achieve high user interface quality, usability, and good user experience for the user of the final interactive application. Possible solutions to avoid practical problems and discrepancies between the automatic derivation of user interfaces and their usability are discussed in [11]. Benefits from using model-based user interface development and meaningful use cases are provided in [8].

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

The role of the various models used in MB-UID environments varies with respect to the modeling purpose. Typically more than one model is exploited during the development process to construct the desired solution interactively or (semi-) automatically. Some degree of standardization was brought into the diversity of MB-UIDEs by the CAMELEON Reference Framework (CRF) [1]. CRF proposes the use of domain, context-of-use, and adaptation models. Here, the domain model combines task and concepts sub-models, the context-of-use model consists of user, platform, and environment models, and the adaptation model is separated into evolution and transition models. With regard to model abstraction levels, CRF distinguishes task-oriented specification, abstract user interface, concrete user interface, and final user interface.

HCI patterns are a means to document design decisions based on established design solutions or best practice work and therefore capture fundamental principles for good design. In general, patterns represent a relation between a certain design problem and a solution in a given context. In addition, they are simple and easily readable for designers, developers and researchers, and they alleviate the collaboration between the involved people. In order to ensure a certain standard, patterns are organized in so-called pattern catalogs [1]. A catalog of related patterns that belongs to a common domain is called a pattern language [14]. Since many pattern authors pick their own formal description styles and formats with often different understanding of pattern attributes, several standardization approaches have been introduced, e.g. the Pattern Language Markup Language (PLML) version 1.1. PLML unifies the description schemes of different authors with the help of XML tags which represent the particular characteristics of the patterns. According to PLML 1.1 the documentation of a pattern should consist of the following elements: a pattern identifier, name, alias, illustration, descriptions of the respective problem, context and solution, forces, synopsis, diagram, evidence, confidence, literature, implementation, related patterns, pattern links, and management information [7]. A recapitulation and discussion of existing pattern description standardization approaches is provided in [4].

PAMGIS FRAMEWORK

Basic Concepts

The intention of the PaMGIS framework is to assist and support its users in the process of developing highly interactive user interfaces. As illustrated in Figure 1, the basic concept is to combine both model-driven and pattern-based development methods and techniques.

Hence, descriptions of HCI patterns are equipped with model fragments that on one hand can be used as building blocks for the diverse models and on the other hand allow influencing model transformations. In addition, usability evaluation results can be fed back in order to draw conclusions and improve the patterns, models, and the resulting user interfaces.

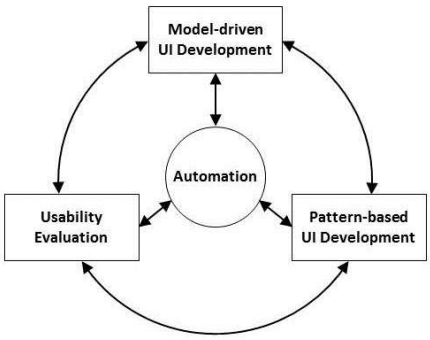

Figure 1. Basic concepts of the PaMGIS Framework

The framework supports our research with respect to the potentials and limits of automated UI development. In order to enable automatic processing, all model entities as well as pattern specifications are expressed and stored in an XML-compliant format.

Model-driven Aspects

The model-driven part of the framework as illustrated in Figure 2 is designed in the style of the CAMELEON Reference Framework. Particularly, the ontological domain and context-of-use models are used as proposed by the CRF. However, we decided to split the CRF platform model into a device model and a UI implementation model. While the former comprises all relevant characteristics of the respective end device the latter holds information about the UI elements that are available on the respective underlying software platform. This avoids redundancies especially in cases where the same software basis supports significantly different devices, e.g. Android on smartphones and tablet computer.

The framework is organized in six abstraction levels, i.e, domain, context of use, abstract user interface (AUI), concrete user interface (CUI), final user interface (FUI), and runtime level. As the most abstract representation, the domain level embodies the domain model which in turn consists of the task and concept sub-models. The task model provides information of domain-specific user goals and the entirety of process steps and actions which must be executed in order to attain these goals. The concepts model can be understood as a type of data model describing all UI-relevant data elements and artifacts which are required in the course of task completion. Hence, these two models are closely interrelated. In the context of PaMGIS, the task model is represented in a ConcurTaskTrees (CTT) notation [10] with some specific adaptations and enhancements which primarily refer to the specification of relationships between certain tasks and the data elements that are required for the execution of these tasks. The concept model is specified on the basis of XML Schema Definition (XSD). The context-of-use model consists of the user, environment, and the already mentioned device and UI implementation sub-models. While the user model holds information about particular characteristics of individual users or clusters of users, e.g. preferences or possible disabilities, the

environment model describes environmental influence factors, e.g. lighting conditions, noise, or air pollution.

The knowledge captured within the domain model is used to construct an abstract user interface model which is a canonical representation of the rendering of the domain concepts which is independent from the actually available UI elements as specified within the UI implementation model. At this juncture, the concepts model indicates which AUI objects are required while the task model's hierarchical structure and inherent temporal dependencies enforce the definition of the relationships between these objects.

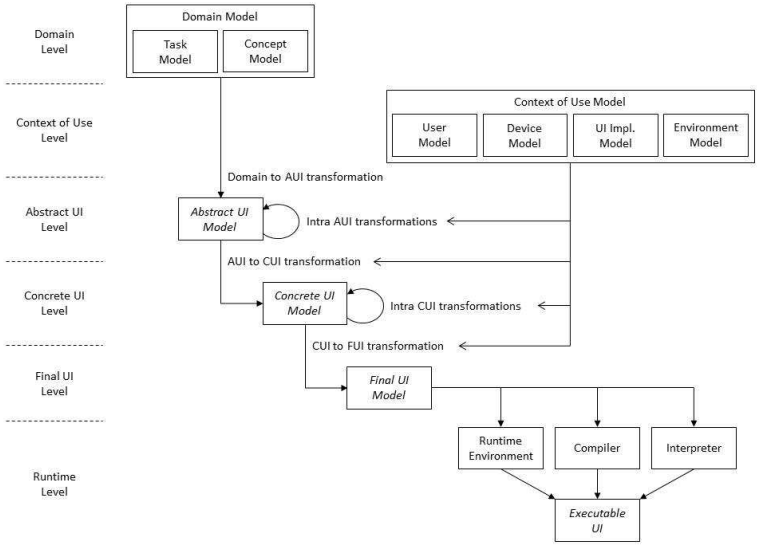

Figure 2. Overview of utilized models and abstraction levels.

A list of feasible AUI objects is provided Table 1.

Abstract UI Object	Description
Activator	Activates another object or initiates a call of a business logic function
Navigator	Facilitates the navigation to another dialog
Output	Displays (read-only) objects of diverse data types
Editor	Similar to Output, but manipulable by the user
SingleChoice	Selection of exactly one item out of several
MultiChoice	Selection of none, one, or more items out of several

Table 1. Examples of supported abstract user interface objects.

The information contained within the context-of-use model is used to control the subsequent transformations of the diverse UI models and to substantiate deliberate design decisions. For instance, some tasks or sub-tasks might be undesired, impractical or impossible to be carried out within a certain context of use due to user-, device-, and/or environment-related restrictions. In this case, the corresponding parts of the AUI have to be eliminated. Furthermore the design of the dialog structure is defined in consideration of the given context of use by means of dialog graphs [13].

Once the AUI model is completed, it can be transformed into a concrete user interface model. For this purpose, the abstract user interface objects are replaced by appropriate concrete ones. In this sense, the most appropriate CUI object is the one that fits best to both the requirements and restrictions which result from the various aspects of the context-of-use model.

Further, a first impression of the final look-and-feel is created by roughly determining the layout, i.e., positioning, and the appearance, e.g. color, font, and size, of the CUI objects. In a last step, the final user interface can be automatically generated from the CUI model.

Figure 3 recapitulates the necessary transformation steps between the four different levels of abstraction as specified in the CAMELEON reference framework. The process starts with the domain model followed by the abstract and concrete model levels and finally arrives at the final user interface. Please note, that the framework user may perform manual adjustments at any step of the development process.

From a runtime perspective, there are three general options how to deal with FUIs. Firstly, the FUI is available as source code that can be transformed into an executable format by means of a compiler. Secondly, the FUI has the format of a script that can be executed by an interpreter. Thirdly, the FUI can be executed by a runtime engine provided with the development framework. The advantage of such a runtime engine is that it is not necessarily bound to the FUI level, but

can also create at least executable UI prototypes from higher abstraction levels, i.e., CUI and AUI models, and therefore enables framework users to identify design problems in early stages of the development process.

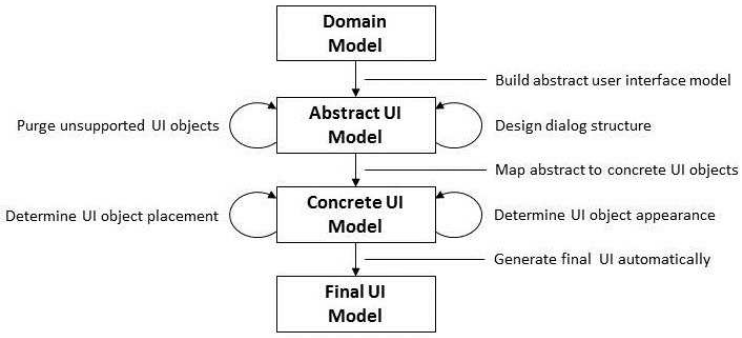

Figure 3. Overview of PaMGIS model transformations.

The utilization of default values within the respective model allows executing UI prototypes on the basis of not yet finalized models. In addition, the use of a runtime engine also allows for implementing model-based responsive designs and runtime adaptive behavior of the user interface.

Pattern-based Aspects

Within our combined development approach, patterns are used as means to alleviate the complexity of the model-driven processing. The patterns provide pre-assembled building blocks which can be used for domain and UI model construction. In addition, certain patterns provide valuable input to the various model transformation steps shown in Figure 3.

For this purpose it is essential to specify the patterns in a uniform and machine-readable manner and equip them with the required information. Further, it must be possible to compose pattern languages, i.e., to define the interrelationships between the patterns.

Hence, we developed the PaMGIS Pattern Description Language (PPSL) which is suitable to fulfil the aforementioned requirements.

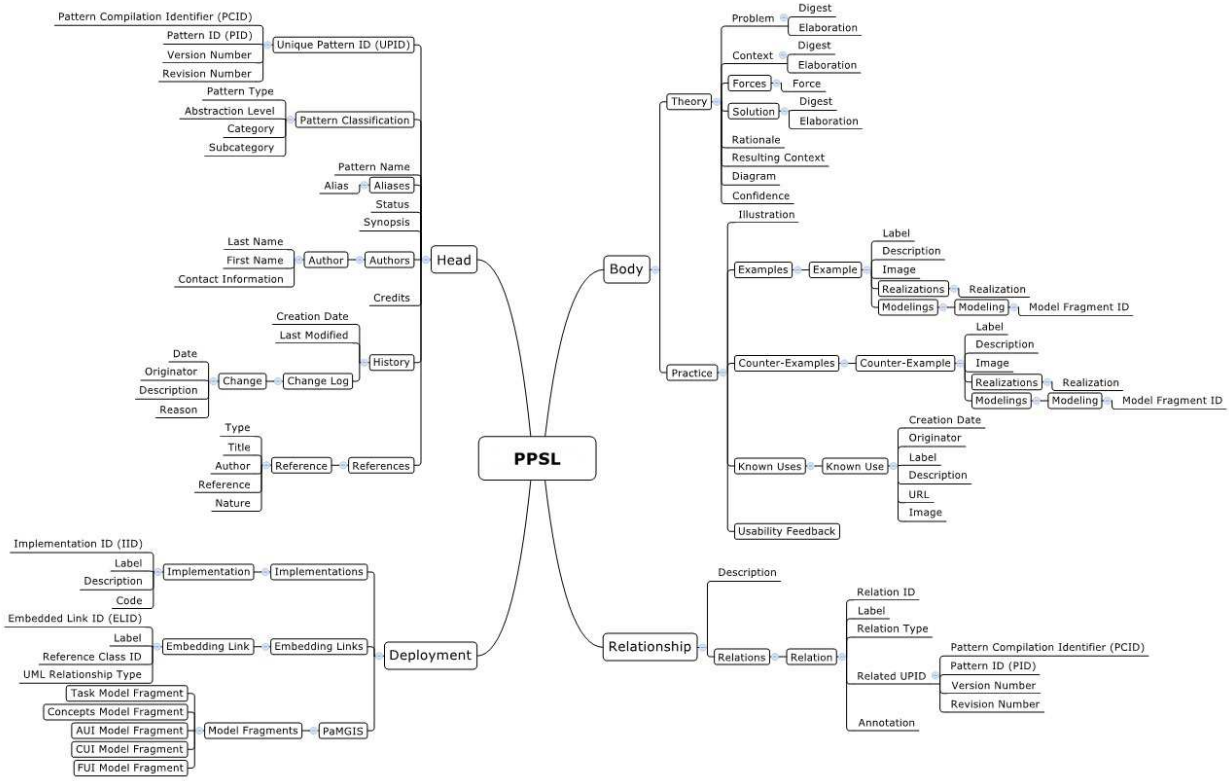

Figure 4. Overview of the PaMGIS Pattern Specification Language.

We reviewed existing pattern description standardization approaches as well as pattern tools in order to define PPSL in a way that patterns which are specified in the related formats can be transformed to PPSL. Thus, the entirety of all PLML 1.1 description elements is covered in PPSL, where required in a restructured or modified form. The only exception is the PLML element *Evidence* which is not directly included, but whose two sub-elements *Example* and *Rationale* are part of PPSL. Further, the PLML description element *Literature* can be mapped to *References* and *Related-Patterns* to *Relations*.

In addition, we introduced new description attributes for storing the supplementary information required by PaMGIS. An overview of the description elements of PPSL is provided in Figure 4. Pattern specifications are organized in four top level elements, i.e., *Head*, *Body*, *Relationships*, and *Deployment*.

The *Head* element incorporates metadata such as unique pattern identification, pattern classification, pattern name and aliases, information about pattern authors, credits, pattern evolution, and references to further sources and literature. The *Body* element is split into the two sub-elements *Theory* and *Practice*. The former provides theoretical background, including – amongst others – descriptions of the underlying problem, the context in which the pattern can be applied, and the proposed solution of the given problem. The latter demonstrates how the pattern was applied in practice by means of illustrations, examples, and counter-examples. The *Relationships* element serves as resource for the specification of the relationships between the various patterns and therefore allows the construction of pattern languages. Finally, the *Deployment* element contains – amongst others – model fragments of different types and abstraction levels as usability feedback. The model fragments are used as building blocks for the domain and the diverse UI models.

The model fragments are stored within the *Deployment/PaMGIS/ModelFragments* element and provide ready-to-use modelings of the pattern's inherent solution. During the process of constructing the domain model, the framework user can search, select, and apply patterns, i.e., automatically insert the respective task and concept model fragments into the domain model. It is also possible to store prefabricated AUI, CUI, or FUI model fragments with the pattern which can be directly embedded into the UI models of the corresponding abstraction levels. While patterns typically contain only one task and one concept model fragment, they might possess multiple AUI, CUI, and FUI model fragments for different contexts of use. This allows both applying different UI design solutions during design time and even during runtime, i.e., substituting one model fragment by another one. This mechanism is not limited to model fragments of the selfsame pattern. In fact, it is even possible to substitute whole patterns by alternative ones.

Regarding the process of finding appropriate patterns the framework offers multiple methods: pattern browsing, keyword search, free text search, exploiting pattern relations, or evaluating formal context descriptions which are stored as logical expressions within the *Body/Theory/Context/Digest* element.

Usability Evaluation Aspects

Running user interfaces – either on the basis of a complete FUI or in form of a prototype based on more abstract UI models – can be evaluated in terms of their usability and user experience using pertinent techniques and methods. The evaluation itself is not in the scope of PaMGIS. Hence, the framework does not offer any support for evaluation preparation, execution, and post-processing. But it is possible to document relevant insights within the system. Since the origin of model elements is captured inside the PaMGIS domain model and the various UI models, it is possible to locate the respective pattern and post the evaluation results to the pattern definition. For this purpose we introduced the pattern description element named *Body/Practice/UsabilityFeedback*.

A second, more automated option is to specify and utilize special usability evaluation (UE) patterns. They can be integrated in the domain model where they add some measuring instrumentation. For instance, the *Textual User Usability Feedback Dialog* pattern ensures, that an appropriate dialog is available allowing the user to record and send his or her opinion about certain aspects of the user interface at hand back to the PaMGIS framework. In the simplest case, this dialog is composed at least of an *Output* object providing some textual explanations for the user, an *Editor* object for the acquisition of the actual textual user feedback, and two *Activators* for either submitting the feedback or canceling the action. The aforementioned pattern includes the required task and concept model fragments as well as AUI and optionally less abstract UI model fragments. In this sense, the underlying domain-specific pattern language can be enriched by such UE patterns in order to capture usability feedback. At least in the case that the user interface is executed by means of the runtime environment, it is possible to automatically attach the user feedback directly to the respective pattern. Otherwise the information can be temporarily stored in a log file outside the scope of PaMGIS and fed back manually or in a semi-automatic way at a later point in time.

The collected usability feedback can be used to improve the quality of the patterns, the diverse models, and therefore of the final user interface.

Functional Framework Architecture

The PaMGIS framework consists of several logical function units, each supporting the various users in different fields of activities. An overview of the functional framework architecture is provided within Figure 5.

The core components are the two repositories, the *Pattern Repository* for storing the pattern specifications and the *Model Repository* to accommodate the diverse models as shown in Figure 2.

Access control is managed by means of the *User Database* which is administered via the *Framework User Administration* component. PaMGIS distinguishes several general types of users, i.e., unregistered users, registered users, pattern authors, power users, and administrators. Unregistered users are allowed to access a restricted part of the pattern specifications solely in read-only mode. In addition, they may register themselves to the framework. Registered users gain more insight into pattern details, have very limited write permissions, and may use certain collaboration functions, such as sending messages to pattern authors. The pattern authors have full access to the pattern descriptions and may create new patterns and modify existing ones. Power users can read entire pattern specifications and are allowed to make copies to which they have full read and write access. In addition, they can use and control the model-driven part of the framework. Finally, administrators take over the responsibility of managing the framework, e.g. creating, modifying, and deleting users, granting and withdrawing access rights, and maintaining the PaMGIS meta-models via the *Pattern Meta Model Administration* and the *Model Meta Model Administration* components.

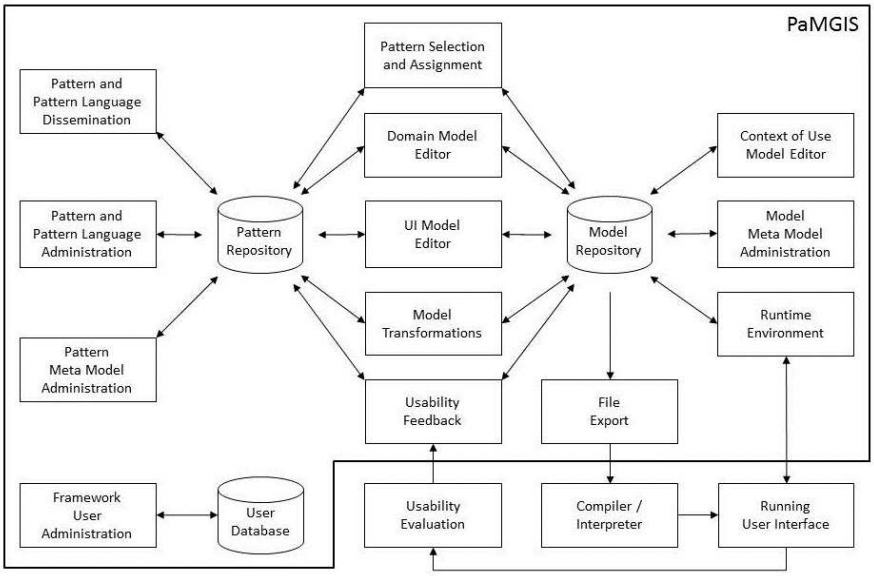

Figure 5. Overview of the functional PaMGIS architecture.

On the one hand, the *Pattern and Pattern Language Administration* unit supports pattern authors in creating and modifying patterns. On the other hand, power users can copy particular patterns to a private workspace where they can modify them according to their needs and build up pattern languages by specifying interrelationships between patterns.

The *Pattern and Pattern Language Dissemination* tool can be used by unregistered users to browse, search, and display certain aspects of the pattern descriptions which are released for this purpose. Additionally, it allows registered users to view more pattern details, send feedback and comments to pattern authors, and attach information about existing implementations to the pattern specifications. For this purpose we introduced the *Body/Practice/KnownUses* description element.

The *Domain Model Editor*, *Context-of-Use Model Editor*, and *UI Model Editor* allow power users to create and modify the respective PaMGIS models manually. In contrast, the *Pattern Selection and Assignment* component helps power users to search and find adequate patterns which can be selected and applied, i.e., insert the attached model fragments automatically into the domain, context-of-use, and/or different UI models. The *Model Transformations* unit supports the execution of the model transformations summarized in Figure 3 and can be configured to a certain extent.

The *Runtime Environment* is a means to execute user interfaces in the form of final UIs or prototypes as described above. The *File Export* component is used for exporting models in the form of text files for further external processing or documentation purposes.

Finally, the *Usability Feedback* unit offers support regarding the import of usability evaluation results into the framework and write it back to the respective patterns and/or models.

CONCLUSION

In this paper we presented our concerted pattern-based and model-driven approach for the development of interactive ubiquitous systems and provided an overview of the functional architecture of the related PaMGIS framework.

We strongly believe that the mélange of model- and pattern-related methods and techniques has the potential to alleviate weaknesses of the individual approaches and can create benefits in terms of reducing complexity and realizing reuse of already existing design knowledge.

The implementation of the framework is indeed work in progress, but major components already exist at least in prototypical form. Many patterns and several pattern languages have been developed, amongst others a pattern language for the domain of public transportation ticket selling.

The framework is a cornerstone for our further research on the potentials and limits of automated UI development. Moreover, we will intensify our work on supporting wearable computers with the PaMGIS framework.

REFERENCES

1. C. Alexander, S. Ishikawa, and M. Silverstein. 1977. A Pattern Language.Oxford University Press.
2. G. Calvary, J. Coutaz, L. Bouillon, M. Florins, Q. Limbourg, L. Marucci, F. Paternò, C. Santoro, N. Souchon, D. Thevenin, and J. Vanderdonckt. 2002. The CAMELEON Reference Framework. Retrieved April 15, 2015 from http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf.
3. Paulo Pinheiro da Silva. 2001. User Interface Declarative Models and Development Environments: A Survey. In *DSV-IS'00 Proceedings of the 7th International Conference on Design, Specification, and Verification of Interactive Systems*, 207-226.
4. J. Engel, C. Herdin, and C. Märtin. 2012. Exploiting HCI Pattern Collections for User Interface Generation. In *Proceedings of PATTERNS 2012*, 36-44.
5. J. Engel, C. Herdin, and C. Märtin. 2014. Evaluation of Model-based User Interface Development Approaches. In *Proceedings of HCII 2014*. 295-307.
6. J. Engel and C. Märtin. 2009. PaMGIS: A Framework for Pattern-based Modeling and Generation of Interactive Systems. In *Proceedings of HCI International '09*. San Diego, USA, 826-835.
7. S. Fincher and J. Finlay. 2003. Perspectives on HCI Patterns: Concepts and Tools (Introducing PLML). *Interfaces, Vol. 56*, 26-28.
8. G. Meixner, G. Calvary, and J. Coutaz. 2014. Introduction to Model-Based User Interfaces. *W3C Working Group Note 07 January 2014*. Retrieved May 27, 2015 from http://www.w3.org/TR/mbui-intero/.
9. Brad A. Myers. 1992. State of the Art in User Interface Software Tools. *Advances in Human-Computer Interaction*, Vol. 4, Ablex Publishing.
10. F. Paternò. 2000. The ConcurTaskTrees Notation. In *Model-Based Design and Evaluation of Interactive Applications*, Springer Berlin Heidelberg, 39-66.
11. A. Pleuss, B. Hauptmann, D. Dhungana, and G. Botterweck. 2012. User Interface Engineering for Software Product Lines: The Dilemma Between Automation and Usability. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark,. 25-34.
12. Egbert Schlungbaum. 1996. Model-based User Interface Software Tools - Current State of Declarative Models. *GVU TECH REPORT*. Graphics, Visualization and Usability Centre, Georgia Institute of Technology.
13. E. Schlungbaum and T. Elwert. 1996. Dialogue Graphs: A Formal and Visual Specification Technique for Dialogue Modelling. In *Proceedings of the 1996 BCS-FACS Conference on Formal Aspects of the Human Computer Interface FAC-FA'96*, Sheffield, UK.
14. A. Seffah. 2010. The evolution of design patterns in HCI: from pattern langauges to pattern-oriented design. In *Proceedings of the 1st Interational Workshop on Pattern-Driven Engineering of Interactive Computing Systems* (PEICS'10), 4-9.