A Factor Model Capturing Requirements for Generative User Interface Patterns

Stefan Wendler, Danny Ammon, Ilka Philippow, Detlef Streitferdt
Software Systems / Process Informatics Department
Ilmenau University of Technology
Ilmenau, Germany
{stefan.wendler, danny.ammon, ilka.philippow, detlef.streitferdt}@tu-ilmenau.de

***Abstract* — The lowering of efforts for the adaptation of GUI dialogs to changing business processes is still a worthwhile goal. In this context, user interface patterns (UIPs) have been introduced in the development of user interfaces to increase both usability and reusability. Originally derived from human computer interaction patterns, UIPs are generative and thus have to be formalized. Recent approaches for model-based GUI development employ UIPs with specific notations. These UIP concepts have not yet been evaluated on the basis of a stringent set of criteria. We elaborate detailed requirements for generative UIPs. The resulting influence factor model is used to assess recent UIP approaches and identify open issues.**

Keywords* — *user interface patterns; model-based user interface development; HCI patterns; graphical user interface.

I. INTRODUCTION

A. Motivation

Domain. Nowadays, companies heavily rely on systems that offer a vast support for the activities defined in business processes. To serve their purpose effectively, those business information systems provide graphical interaction dialogs to various users. Besides the enormous effort to be taken into the specification of the business processes and related requirements, there are also considerable high costs involved in the development of GUI dialogs the user interacts with in order to process certain activities of the business process. In addition, those dialogs need to be matched with their currently assigned workflow derived from the respective business processes. As business information systems must be changed over time, the need to keep business processes, application kernel functions and the GUI, that provides the dialogs based thereupon, in correspondence [1] has arisen.

Problem. Approaches have been proposed that aim at raising both efficiency and reuse by applying model- and pattern-based concepts for the development of GUIs and dialog structures derived from task models. The different concepts have not been verified yet. Currently, there is no detailed set of requirements, which can be used as foundation to assess the pattern concepts employed for GUI generation.

B. Objectives

We review the state of the art of model-based development processes employing generative user interface patterns (UIPs) and present answers to following questions:

- What requirements have to be addressed by a general definition for generative UIPs applied for GUIs?
- What are the capabilities and limitations of current generative UIP concepts concerning reusability and variability?

Our main focus lies on the last question, so we formulate requirements for a UIP definition on the presentation level. After we analyze current UIP issues, we apply a customized Global Analysis [2] to derive the factors, which bear the great impacts on the definition and application of generative UIPs. As a result, we continue and detail our previous work [3][4] on initial requirements associated to generative UIPs.

C. Structure of the Paper

The next section provides a brief overview to GUI development and UIPs. In Section III, we establish a factor model that captures major requirements for UIPs. The model-based development approaches are described in Section IV and are analyzed on the basis of the factor model. In Section V, we express our findings and conclusions.

II. RELATED WORK

A. Graphical User Interface Development

GUI architecture patterns. Besides the requirements and software architecture to be changed harmonically, a new implementation of dialogs induces high costs as reuse of existing code is hardly possible. More precisely, the GUI system may be composed of proven architecture patterns that enable separation of concerns and reduced dependencies like the Quasar client architecture [8], but these kinds of patterns are restricted to non-visual aspects of the GUI like event and data processing or the communication with a workflow system. As far as visual and closely associated interaction design aspects are concerned, the common patterns do not posses the means to offer the desired aspects of reuse. Moreover, the usability is crucial for dialogs, as it affects how quickly users are able to learn to use new features of the GUI and how efficiently they will perform reoccurring tasks. Usability also is not covered by architectural patterns.

GUI-generators. Generators have been applied for a longer period now and could not fill the gap, since they can only cover dialogs that allow a realization based on fixed layout and interaction definitions. Besides the visual and interactive aspect, GUI-generators often were based on information provided by the domain model, so that task models or other process definitions could not be sourced for the generation of dialogs with acceptable usability.

B. User Interface Patterns

To overcome the high efforts and permit higher reusability along with proven usability, patterns of human computer interaction (HCI) have been integrated as model artifacts in model-based development. In that environment HCI patterns had to be formalized in order to obtain a

 ISBN: 978-1-61208-276-9

machine processable format. Called generative patterns by Vanderdonckt and Simarro [6], a new form of pattern has emerged based on descriptive HCI patterns. Commonly, these patterns are named User Interface Patterns (UIPs).

Reusability. UIPs as generative patterns are to be deployed as reusable entities in GUI development. By specifying dialog parts abstractly (visual parts and interaction) as well as parameters for variability, UIPs should facilitate the reuse and automated generation of GUI dialogs. Configured accordingly, the UIPs would be instantiated to target contexts. This way, a GUI system should be compiled by the selection and combination of chosen UIPs. Key features of this approach shall be the variable application of UIPs to any appropriate context and their ability to form hierarchies of further cascading UIP instances. The latter could form a context-specific composition of already specified appearance and behavior qualities, which would be quantitatively adapted to the context when instantiated.

Issues. The application of UIPs for GUI generation has successfully been probed by past research [1][11][14][19][24]. As HCI patterns need to be augmented for automatic deployment, the main issue of finding a suitable formalization format, which offers a feasible definition of generative UIPs, has arisen. Current approaches propose different UIP concepts combined with tools, which propagate the instantiation of the abstract UIP entity for various contexts and thus an increase in reuse. Nevertheless, reusability is still restricted to a limited set of UIPs, which can be deployed without having to consider all variability aspects [3]. The potential variations for view, interaction and in particular the control aspect are so extensive that they need to be further detailed by a set of comprehensive criteria.

III. REQUIREMENTS FRAMEWORK

UIP definition. The specification of UIPs is impaired by a fundamental problem that persists in the lack of a dedicated definition for this generative artifact. Many sources have been published on HCI or GUI related patterns, but these either presented no or did not converge towards a unified definition. We stick to our drafted definition in [3] and use the term User Interface Pattern (UIP) that addresses the generative form ready to be instantiated to a certain GUI context. So, a UIP is settled in close proximity to architecture and code artifacts assuming presentation responsibilities.

Approach. To overcome the disunity concerning the definition and features of UIPs, we develop a system of requirements that is able to express the conception of UIPs independently from any employment in modeling frameworks and tools. We apply the Global Analysis [2], as requirements for UIPs are rather general. So, we refine them according to their impact on the generative UIP artifact definition. The background and an initial factor model have been developed in [4], which is detailed in the following.

A. Criteria for User Interface Patterns

As outlined in [3], sufficient solutions for pattern-based GUI development have to meet basic criteria. Firstly, they must enable reusability in the context of vast variability of stored patterns. Secondly, facilities must permit to compose several patterns to form a hierarchy of GUI components - an attribute that is not common for all kinds of software patterns. Lastly, the instantiation into varying user interface paradigms, platforms and types should be possible.

The first two criteria are relevant for our scope and we will decompose them in our factor model as we progress towards Section III.E. For now, the factor “Reusability of UIPs” is defined, which is composed by the three factors “Structural composition ability”, “Behavioral composition ability” and “Variability of UIP instances”. The split nested factors are motivated by the following distinction. A single UIP may be reused for many contexts and for that purpose, certain variability concerns have to be met that are covered in the next section. Besides, a combination of more than one UIP may be reused. In that case, both the structural and behavioral definitions should be adaptable to the desired context. Section III.C treats these composition ability factors.

B. Variability of User Interface Patterns

MVC analogy. If one UIP is variably instantiated, implementations of given architecture components evolve and eventually differ in certain aspects. For this reason, the architectural pattern of model-view-controller (MVC) is used to describe the UIP adaptability for different contexts [3]. An UIP adaptation changes the actual view structure, data types for the view parts and the control serving visual and application event handling of a certain architecture instance.

Variability factor. The above mentioned variability concerns affect various contents of an instance of a certain adaptable UIP. The content is materialized by the two aspects *view* and *interaction* in the factor model. Each sub-factor of *variability* is operationalized by an aspect. Besides, the *variability* factor influences a second dimension, which describes the moment in time, when the UIP adaptation takes place. Thus, the *configuration* factor details *variability*.

C. Aspects of User Interface Patterns

Purpose. Originally, we described three aspects of UIPs to detail our definition in [3]. We pointed out the differences between a concrete specification of a GUI unit, the abstracted formalization of a UIP and its instances. Here, we summarize the aspects to further evolve the factor model.

View. By the stereotype but abstract *view* of a UIP, selection, arrangement and types of user interface controls (UI-Controls) are defined. With its abstract definition the “view aspect” preserves the applicability of a UIP to various contexts and should not rely on certain GUI frameworks, hence a UIP must be able to be transformed to desired platforms. Through the “view aspect”, UIPs can be categorized into simple and composite patterns. Simple UIPs, like a simple search [10], consist of a fixed set of UI-Controls, while composite UIPs, like an advanced search [10], contain even other UIPs. Therefore, the “Structural composition ability” is operationalized by the “view aspect”. To define the *visual element structure* of a UIP, a developer may source both UI-Controls and already defined UIPs.

Interaction. A user always perceives and performs interactions with instances of a certain UIP in the same way. Combined with the *view*, the *interaction* forms the general purpose of a UIP and so, both aspects constitute the reusable entity and distinguish UIPs from mere UI-Control compositions. With *interaction* states, data handling and

 ISBN: 978-1-61208-276-9

presentation related events are defined by referring to *view* contents. Moreover, a UIP may demand for structural *view* states that are determined at run-time by user inputs.

Control. Composite UIPs, as defined above, actuate in- and outputs depending on the defined selection, instantiation, and configuration of their child UIPs. Sections III.D and III.E treat how *control* operationalizes "Behavioral composition ability" and in this regard details the *interaction* of several UIPs in one *view structure unit*. Depending on the *variability configuration* dimension, a dynamic *control* may be needed where child UIPs are selected and instantiated at runtime. The following section covers this case.

Reusability factor. *View*, *interaction* and *control* aspects operationalize the before-mentioned *reusability* factor. All three factors ensure either the *composition abilities* or *variability* of UIPs. The reuse of single UIPs for different contexts is achieved by abstraction in both the structure of the *view* and the dynamics of the *interaction* as well as *parameters* that provide instance-specific information.

D. Architecture Experiments

Architecture. For the GUI architecture, we assume a structure to be established in analogy to Figure 1, which was derived from [9] and altered for our scope. Notably is the distinction of three controllers for presentation, dialog and task. The *PresentationController* queries data from technical *GUI Framework* objects, receives technical events from them, adapts the *DialogVisuals* accordingly and finally forwards events relevant for the application state to the *DialogController*. The responsibility of the latter is to implement application logic, query data from and send data to the *ApplicationKernel* after selection based on the *Model* data. Additionally, the *DialogController* decides on the lifecycle of the *Dialog*, as it evaluates the state of the *Model* and events received from the *View*. Acting as a factory, the *DialogConfiguration* builds the *Dialog* composition unit, and for that purpose, communicates with the *TaskController*, which initiates the creation or deletion of dialogs.

The architecture is detailed, since a *Dialog* can be based on composite UIPs. A child UIP affects the *View* component only, while the superior one triggers *DialogController* actions, when new sub-dialogs or data must be loaded. Thus, the factor model lists presentation and dialog action-binding.

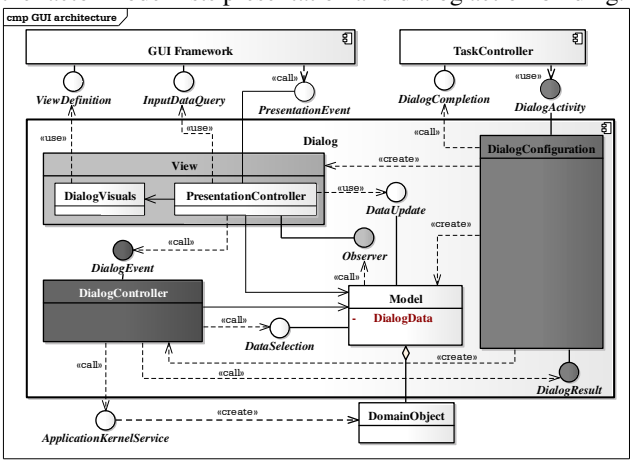

Figure 1. GUI architecture reference model

Experiments. We presented two architecture concepts to implement UIPs specified with UIML (User Interface Markup Language) in [5]. The main findings of our experiments were that UIPs supporting the criteria in III.A could not be formalized as single artifacts with UIML. This was due to the complex example, which required for a "control aspect" with dynamic *configuration*.

The advanced search UIP [3] holds a certain number of search criteria, each demanding a certain UIP type, e.g., a price range and a date represent two different search criteria. The states such a composite UIP can adopt cannot be enumerated by a static specification as they depend on user input or another context not known at design-time. Figure 2 illustrates an example of an advanced search UIP instance.

Firstly, the object to be searched is selected and secondly, attributes are offered for search criteria depending on the choice. The architecture is affected, as new UIPs and UI-Controls are instantiated for the *DialogVisuals.* Additionally, the *PresentationController* actions and scope are altered.

In [24] and [25] run-time awareness of UIPs is mentioned, but not further outlined. As outlined in Section III.B, respective impacts of UIP *configuration* were included. Finally, we discovered two possible workarounds for composite UIPs, which govern the lifecycle of other sub-ordinate UIPs and thus demand for the "control aspect".

UIP context parameters. Firstly, the UIP specification language should permit parameters essential for an instantiation to varying contexts. This decoupling of UIPs from concrete GUI definitions has already been considered by the model-based approaches, which are assessed in Section IV. Without such parameters only invariant but most UIPs simply could not be formalized at design time [4][5].

Virtual user interface. Secondly, UIPs could be split into several atomic UIPs, which would compose a dialog on demand of the dynamic *control* aspect behavior. The atomic UIPs, being mostly invariant and mainly variable concerning data types and the number of structure elements, could be instantiated during run-time by a virtual user interface architecture [7]. This option would demand for manual realization or a DSL for *DialogController* and *View* creation.

E. Influence Factor Model

The influence factor model continuously has been supplemented during the previous sections and its final shape is depicted by Figure 3. The method applied is described in [4]. We cut-out factors not to be considered here.

UIP definition. The main *definition* factor is decomposed by the three aspects derived from Sections III.B and III.C. These are intended to identify, group and separate the impacts with respect to architecture responsibilities.

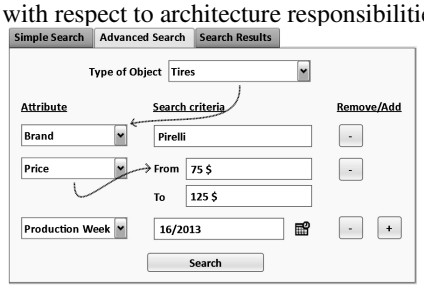

Figure 2. Exemplary advanced search [10] [4]dialog

View impacts. The impacts of the "view aspect" are concentrated on the *DialogVisuals* component. They are refined by two factors. "View definition" demands for the creation of stereotype visual structures composed of UI-Controls or even UIPs. As these impacts resemble static elements of a UIP definition, the second factor requests for parameters to be defined for them. In detail, they need to be named, enumerated and ordered, arranged in layout and customized by style in order to enable *variability* of single UIP instances.

Interaction impacts. The *interaction* impacts seem to be primarily focused on the *PresentationController*. In fact, the "Visual element structure states definition" impact depends on actual *View* structure composition and so, a point of view. Hence, the following distinction is made:

Firstly, when UI-Controls are the only components contained in the visual structure of an individual UIP, several states may have to be defined, which describe alternative *Views*. So, the first impact requires the definition of states a *PresentationController* has to ensure. For example, a UIP may formalize the choice of just two options out of many available, as it is sketched in Figure 4. Consequently, the possible states, e.g., activations, deactivations or toggling collapsible panels [10] of the *visual element structure* have to be specified by the UIP. Moreover, the defined *view structure elements* need to be bound to presentation related actions that trigger changes in states or data to be displayed. The "Presentation action-binding" foresees this binding. In detail, a certain UI-Control has to be configured to trigger a change in state of already defined visual elements of the same scope, e.g., deactivate a delivery address (when it is the same as billing address), assumed that the toggle button or checkbox belongs to the same UIP specification unit.

Secondly, superior UIPs of a composition need to specify an outside view on the sub-ordinate UIPs in order to change or instantiate new sub-UIPs dynamically. For instance, this is required when the user triggers the attribute combobox or buttons on the right hand side of Figure 2, which change states of criteria rows. Accordingly, when a UIP defines a composition of UIPs, then the lower situated UIPs constitute the *view structure elements*. Therefore, their outside view states have to be governed by the superior UIP. In this case, the *control* related impacts become relevant.

Control impacts. The impacts associated with *control* mainly apply to UIP compositions and affect both the *PresentationController* and *DialogController*. Several UIPs may define the *DialogVisuals* altogether. In that case the actions of the *PresentationController* are scattered among the individual UIP specifications as each one governs its own part of *View* separately. One UIP is to be defined as a supreme entity to control the other UIPs visual states or lifecycles. This way, a *hierarchical control flow* for presentation is to be established.

The UIP formalization has to enable the combination of various UIPs with the option to reuse their individual *view state* and *structure definition*. Thus, the *encapsulation of UIPs* demands for the autonomy of each UIP unit. As a consequence, UIPs need to define an interface to report their changes in state to superior UIPs. For this purpose, UIP *intercommunication events* need to be defined that allow for plugging in UIPs in a flexible way. However, UIPs still need to be isolated from each other in order to maintain a flexible composition and exchange options. According to events, they have to be distinguished as the architecture differentiates *PresentationController* and *DialogController*. Since the UIPs principally may be combined in any fashion to build composite UIPs, it is essential that one can define a differentiated perception for UIP originated events. On the one hand, one must specify, which UIPs events will trigger a change in sub-ordinate UIPs *view structure*. On the other hand, one has to define the UIP, which provokes application relevant events that are to be forwarded to the *DialogController*. For instance, a button of an online shopping dialog may trigger to copy billing address data to delivery address data fields of that dialog. Another button in a button bar may confirm the entire shopping process, so that data is validated and delivery address is checked. So, there is a need for "Dialog action-binding". The latter could also be associated to *interaction*, but we decided that this impact has a stronger relation to UIP compositions, when the superior UIP has to filter events from sub-ordinate UIPs and respectively forward them to the *DialogController*.

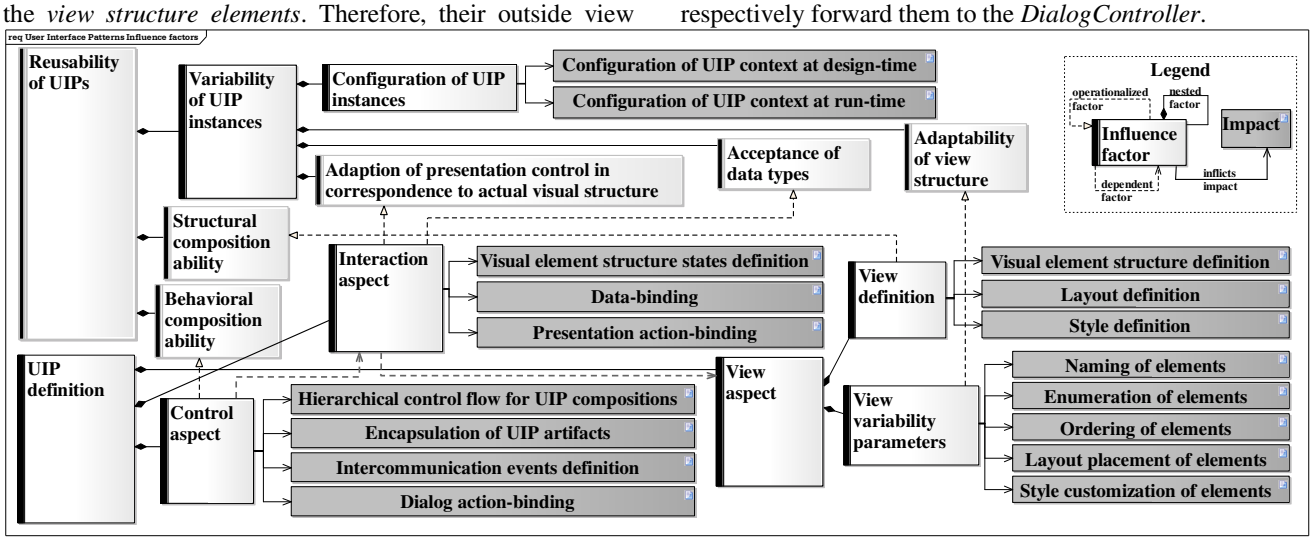

Figure 3. Influence factors identified for the UIP analysis

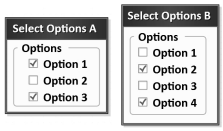

Figure 4. Checkboxes for the choice of two options

IV. EVALUATION OF MODEL-BASED DEVELOPMENT PROCESSES

Recently, model-based development processes for GUIs employing UIPs or similar artifacts have been proposed. Based on available sources, we investigate what generative UIP concept they have incorporated. Afterwards, we review the capabilities and limitations of the respective concepts. More precisely, we consider what impacts of the factor model in Section III.E are supported or inspired.

A. Annotated Task Models - Queen's University Kingston

Harmonic evolution. To restrain the disharmonic evolution of business processes, application kernels and finally user interfaces, Zhao et al. [1] proposed the generation of GUI dialogs on the basis of task model specifications. They applied "usability practices and UI design principles to guide transformations" in order to ensure a better usability of generated solutions. So, task activities were annotated with information about roles, data in- and output. By parsing the augmented task model and applying rules, tasks were automatically segmented. For each segment, windows of a dialog model were derived, so that tasks handling the same data were kept together. This way the dialog structure and its transitions were created.

Task patterns. A fixed set of HCI patterns - called task patterns and based on collections like [10] - was mapped to specific task type segments on the basis of similar naming between both. During the transformation phases, a set of rules was applied for task- and dialog-modeling. For the presentation model, each occurring data type within the respective task was mapped to a certain UI-Control. Thus, a harmonic balance between grouped tasks, stereotype HCI-pattern assignments and windows with a reduced UI-Controls was propagated. On this basis, consistency between changed task models and GUI should be achieved by re-performing the transformation steps.

Factor support. Analyzing the relations to our factors, we found out that the only impacts to be mentioned were the following: For "Visual element structure definition", the UIPs were implicitly and strictly assigned by window or dialog rules according to the information provided in the mapped task-segment. Only a limited set of "task-patterns" was introduced so far. A free composition of UIPs was not possible or even aimed at. In addition, no fancy UI-Controls like separators, progress bars, sliders etc. were to be included for *view structure* definition. Thus, the *DialogVisuals* were statically dictated by a limited set of model dependencies.

Concerning "Layout definition", the general layout already was determined by the dialog model rules as there were Editor, Viewer and Dialog windows. Therefore, the meta-model for presentation was limited to very basic abstract UI-Controls and did not allow for custom UI-Controls arrangement like the separation of mandatory from optional data or a user specific grouping of data.

As far as *variability* and thus "Configuration of UIP context at design-time" are affected, the *DialogVisuals*, *PresentationController* and *Model* (data) were generated on the basis of GUI-generators. In sum, there hardly was any variability for the patterns aside from "data-binding" and the strict automated rules. In this regard, the definition of own presentation related patterns and usability principles was considered as future work.

Questions. The formalization of abstract UI-Controls or task patterns and their instantiation for certain contexts has not been outlined yet. How the mapping of task-patterns and tasks is done also remains as a question.

Summary. From our point of view, the approach of annotated task models combined with a mapping to task-patterns resembles a pure GUI-generator solution. However, this generator has much enhanced capabilities compared to single GUI-generators as it supports a much greater requirement basis: The task names drive the selection of a matching task pattern that is composed by certain usability rules. More important, the process does not need manual intervention and can be repeated when business processes have changed. Starting with the task model, the developer is able to initiate the update for both dialog and presentation model. Although, the solution promises great automation, it is not as flexible as the other UIP-based solutions. Its suitability for a wider range of task types, the customization of the uniform look & feel and the proposed future work of integration of own task patterns and UI design principles should be considered.

B. Patterns in Modeling - University of Rostock

PIC introduction. Forbrig et al. presented their development environment that employed UIPs in many consecutive sources. They described an approach also being based on task-models [11]. Dialog graphs were manually created with the DiaTask tool performing the steps of defining views, assigning tasks to those views and finally creating transitions between views. Then views were translated to windows for a WIMP paradigm [11] platform deriving the abstract user interface (AUI). For each interactive task defined, buttons were created as UI-Controls of a view within the AUI. The transition of the AUI to CUI (concrete user interface) was performed by manual refinement with the XUL-E tool. In this step, buttons from AUI could be replaced by other UI-Controls or even Pattern Instance Components (PICs). This way, the abstract windows with their buttons served as UIP-placeholders for manual replacement. In this regard, the tool-chain achieved to maintain the initial connection between UI-Controls and the task of the original task-model. Both AUI and CUI were formally described with an enhanced XUL format as the final output. Hence, PICs drafted an early approach to formalized HCI patterns, as they only supported a set of five patterns [12]. Moreover, they allowed for the simultaneous replacement of more than one button and had task control data for specific tasks attached, which enabled a more customized processing of the respective task or domain data.

PIM. Continuing the work on pattern integration into model-based processes, a new modeling framework was presented in [13] along with the PIM (Patterns In Modeling)

 ISBN: 978-1-61208-276-9

tool. The pattern application areas were expressed as "UI Multi Model" (task, dialog, presentation and layout). The PIM was intended to apply and manage patterns for the four defined models. Mainly, the focus was laid on the task model and an approach for pattern integration therein. Other models were not yet supported, but the steps for pattern instantiation using a wizard were drafted. Firstly, instance structures had to be specified, describing how often model fragments are duplicated. Secondly, the variables defined by model fragments had to be assigned. To express model fragments, a research into formalization options for the model patterns - especially presentation and layout - was conducted.

PIC revised. After two years, an overview of earlier work was provided in [16]. The focus once more lay on task-patterns to derive dialog navigation structures. It was outlined that the pattern instantiation process could not be fully automated. Therefore a "combination of automatisms and human designer" was propagated. So, a semi-automatic approach to creation of dialog graphs on the basis of task models and respective pattern application was introduced. Thus, the AUI was generated from dialog graphs containing views as placeholders for UIP integration. For comparison, the approach by Zhao et al. [1] fully relied on automated derivation of dialog structures. As before, the CUI was manually refined by replacement of UI-Controls. XUL-E as a tool would permit the refinement of view structures within generated navigation dialogs and their correspondence to the tasks. Manual customization and instantiation of UIPs was suggested by relying on PICs as formal HCI pattern representations, which already resembled context-specific instances of the respective patterns [17].

Factor support. Although it was proven that the instantiation of invariant UIPs was possible, this step was restricted to the replacement pre-determined UI-Controls. Concerning **"**Visual element structure definition", the presentation model included UIPs implicitly as they were intended to be changed and adapted manually. For instance, the content area of each of the wizard dialog windows [11] had to be customized once more via the replacement mechanism. The generator created an initial abstract design like the buttons in the mail client example. Thus, the wizard pattern was strictly defined and could be adapted only in limited ranges to the context as the textfields could be replaced by other PICs or UI-Controls. Additional manual adjustments were necessary, e.g., to remove the next button in the last dialog "Apply". Thus, *variability* depended on manual rework. Lastly, not all kinds of UIPs were supported.

In sum, the "Configuration of UIP context at design-time" relied on the PICs "pre-arranged as components." [11]. From the wizard-example, we assume that there existed an explicit dependency to the task information serving implicitly as UIP-instance parameters. Thus, one could freely decide on what parameters to be used for particular UIP-instances, as this was the case for the approach by Zhao et al.

The "Layout definition" was determined by the PICs, which probably consisted of a strict layout (content area of a dialog, wizard button bar) and always instantiated the same button configuration (Prev, Next, Finish).

For "View variability parameters" the PIM-Tool approach [13] suggested an instantiation process, which could have inspired our parameter impacts: The *view structure definition* and variable assignments were introduced. Also, a hierarchical refinement of an entity by structured patterns concentrated on one of the four models or a mixed selection of them could be learned from this approach. Therefore, the following requirements were also inspired by Forbrig et al.:

The "Hierarchical control flow for UIP compositions" and the "Dialog action-binding" had been drafted. Based on published work, we support the assumption, that UIPs must not interfere with application states since those are to be determined by tasks. According to the "Intercommunication events definition" and after following the vision established by Forbrig et al., one could come to the conclusion that standard-events were quite relevant to plug-In UIPs for altering tasks or to allow UIPs for various task-combinations. After all, the idea of replacement is also important since UIPs should be exchangeable in dialog placeholders in order to enable a change in *view structure* but not in application workflow. Therefore, UIPs need to be replaceable and universal in shape and the impact "Encapsulation of UIP artifacts" may be inspired as well. We vote that the ability to build a cascade of UIPs is important because artifact details or their modules and matching project requirements are hardly the same in different projects. Hence, specific interpretations and instances are of the essence.

Questions. The main emphasis was put on task modeling and the application of patterns on that context. How PICs would be instantiated and applied to contexts is not clearly outlined. It is also questionable how a PIC was successfully shaped to be abstract and universally deployable.

Summary. The approach with rich tool support investigated on the feasibility of "patterns in modeling" [13] and backed or could have inspired some of our factors impacts. The PIM-Tool voted for a combination of model-based and pattern-based approach. This implicated and required a UIP base model to increase reuse and lessen efforts for linking and model integration.

Both Zhao and Forbrig et al. followed a similar approach as they progressed towards the combination of model- and pattern-based development to ensure cost-effectiveness and the application of patterns for the sake of good usability. Forbrig et al. put more emphasis on the pattern aspect. In this respect, they developed tools and customized formal languages for the individual models. However, besides tool support automation could not be increased to the desired level and manual refinements in interaction with the tools had to be performed. For instance, task models had to be shaped to accommodate PICs after the derivation of dialog graphs. Finally, specific variants of arbitrary UIPs could be modeled with the tools and thus greater variability could be achieved compared to Zhao et al.

C. UsiPXML - University of Rostock

Following the former PIM approach, another pattern application framework for UIPs was presented in [14]. The models were further elaborated here, as layout and presentation were intended to refine the AUI and thus enable the transition to a CUI by instantiation of common solutions encapsulated by respective patterns. To organize the patterns

of all four models, the "User Interface Modeling Pattern Language" was introduced as a pattern language. The CUI, where UIP instances were to be integrated next, still needed manual adaptation work.

Continuing towards formalization of UIPs, they presented UsiPXML (User Interface Pattern Extensible Markup Language) as an enhanced UsiXML (USer Interface eXtensible Markup Language) pattern specification language for all four models. To provide both context information for proper usage of a pattern and "implementational information" [15] for automated processing, UsiPXML incorporates PLML (Pattern Language Markup Language) for description and UsiXML as generative part. The PIC concept of older sources is not mentioned here. In contrast, the UsiXML enhancements are further elaborated in [15]. The new pattern notation followed the PIM pattern instantiation steps and thus featured structure attributes, which would determine how many times (min, max) an element within the pattern is instantiated. In addition, variables were incorporated to define mandatory placeholders for values, which could be governed via assignments and applied for various purposes. The former defined how variables would be evaluated. Pattern references, a third feature, would specify sub-pattern-relationships for refinement.

However, UsiPXML is no longer mentioned in subsequent sources again focusing on PICs. Finally, the goals to be achieved with UsiPXML were relativized in [17] as they stated PIC "is called instance component, since we consider the template to be already an instance of the pattern that is described through this component. We are aware of the fact that, due to their nature, not all known HCI patterns can be treated as or translated into an algorithm or a PIC."

Factor support. For the "Visual element structure definition" presentation patterns were applied to define *view structures*. Concerning UIP compositions, the patterns were always presented in isolation and never in entirety, so the real capabilities cannot be judged.

Separate patterns were dedicated to the "Layout definition" impact. As it was not clearly outlined, where they could be included in the hierarchy of pattern instances, the flexibility of the solution cannot be assessed as well.

As far as "View variability parameters" are concerned, structure attributes as well as variables and assignments were invented. Those parameters would permit the deactivation of certain pattern structure parts [15] by "set" assignments.

The "Data-binding" was also realized by the "set" assignment, so that an implicit mapping of data types to abstract UI-Controls was possible. This way the developer did not have to decide for each domain object attribute what kind of UI-Control or UIP to instantiate in a form.

A hierarchical structure of patterns was employed, so patterns could be combined via a pattern interface. More precisely, the variables of higher order patterns could be passed to the pattern interface of lower patterns in order to allocate their variable definition. For vast flexibility in *pattern composition ability*, such a pattern interface could have been arranged for potential reusable patterns, but this is not further mentioned.

The pattern interface, variables and assignment facilities might have been useful to empower "Hierarchical control flow for UIP compositions" and the "Encapsulation of UIP artifacts", but due to missing examples and language specifications, these cannot fully be judged. However, the variables were not standardized for certain pattern types, so they depend on the individual pattern model fragment and their evaluation by the assignments. So, a superior UIP needs to know about implementation details of sub-ordinate UIPs. That is why the *encapsulation* eventually might be broken.

Inspired by the realization of the "Unambiguous Format" [15] pattern, the advanced search criteria rows of Figure 2 could be defined in an abstract manner by UsiPXML, but it has to be answered how they could be requested during run-time. Eventually, the realization of "Configuration of UIP context at run-time" remains unsolved.

Questions. At first we ask, how presentation and layout patterns are merged in a generated window. Both are "CUI Model Fragments" [15] and in that source the patterns are only shown separately but not integrated. As far as UsiXML is reused here, UsiPXML should have inherited some of its weaknesses [3][4]. For instance, how could UI-Control types be platform independently described when UsiXML uses a strict set of types for UI-Controls? How did UsiPXML allow for the description of all four 4 models when UsiXML cannot describe presentation and especially layout models separately? For a better assessment of these issues, we miss code examples of UsiPXML.

Summary. This solution may be a great enhancement concerning the expression ability of generative UIPs. Yet, it is overshadowed by many open issues concerning impact details, which have not been presented yet. So, this approach could not accurately be assessed by us. Moreover, this approach is limited to UIPs being able to be specified at design time. A UIP dynamically morphing during run-time as in Figure 2 most likely cannot be defined with known UsiPXML facilities. Lastly, the occurrence of sub-patterns was the only considered relationship so far. "Inter-Model" [24] patterns have not been considered yet.

D. PaMGIS - University of Augsburg

HCI Pattern language. Engel et al. [20][21][22] state that current UIP-collections do not reflect the need to structure the UIPs to certain aspects, which would enable to select and judge them independently from domain or their relationships. They express that the abstraction and organization criteria are not satisfying. Starting with advice, how to structure a UIP language properly [18], Märtin et al. gradually advanced to their own concepts for UIP instantiation. The rules of a global entry point, allowed and not allowed links within the pattern hierarchy, should guide the user of the pattern collection, so that he would have to start with a rather "abstract pattern for the general problem class" [18] and consequently follow the same abstractions searching the pattern hierarchy for a solution. It should be avoided to oversee a potential useful pattern and isolate individual patterns. The concept was applied in later sources.

PaMGIS. An entirely new modeling architecture was presented in [19], named "pattern-based modeling and generation of interactive systems (PaMGIS)" and neglecting

the very recent work of PIM, UsiPXML and respective four pattern types of the University of Rostock. A central pattern repository would hold patterns for the following modeling stages. Firstly, an "abstract application model is generated" (AAM) by interpreting a set of potential input models (task, user, device, context). Secondly, "a semi-abstract application model" (SAAM) is generated. During this step, the patterns might be instantiated. For this purpose, patterns were composed of both descriptive and generative information. In detail, the generative part introduced an <automation> XML tag allowing the parameterization of the respective <element>, which served as the container and layout unit of the UIP. The <children> tag referenced child UIPs or UI-Controls, governed their number, ordering and position in relation to the parent UIP. This mechanism was based on the <element> tag and the therein defined attributes of the respective sub-UIP or UI-Control. The superior UIP could select from the lower specified attributes.

The approach of PaMGIS was further outlined in [22] by Engel. He stated that the process was based on the enhancement of information derived from fully-fledged task-models and unique pattern models. Patterns would be applied for both the extended AAM and the SAAM. Furthermore, he mentioned that the framework contained a repository for UI-Controls as lowest units in the UIP hierarchy, which would be mapped to target platforms.

Joined by Forbrig, Engel and Märtin presented further information about the PaMGIS framework and the DTD applied for the generative <automation> tag of UIPs in [23]. By example, they outlined the unique way of structuring UIPs based on [18]. Therein, main categories resembled technically shaped patterns appropriate for the current GUI structure element, e.g., a panel or button-bar as sub-patterns.

Factor support. The XML specification defined by the <automation> DTD is closely related to "Visual element structure definition". In general, UIPs are supported as composites. They always define the inclusion of child patterns, since even UI-Controls are regarded as patterns. Their ordering is explicitly determined and constraints are allowed as well as optionals. However, the composite patterns only approach is unfavorable, since one cannot decide on what are composite and what are atomic units of reuse. For instance, a panel is often to be used as an atomic unit in Figure 2. The advanced search UIP defines its own tree of elements or reuses entire UIPs. Not the included panel should decide on that. Anyway, one cannot use a panel without children definition in PaMGIS, since this would result in an empty panel as well as a breach in layout definition hierarchy. The UIP hierarchy is designed in a way, that UIP definitions cannot traverse more than one level at once. So the structure parameters would be limited to a certain levels scope. Single UIPs were too strictly bound to the hierarchy, as they always would have to determine about sub-ordinate UIPs. The leveling would be too strict and one-dimensional, so that one can only include a certain UIP with its respective children and not without them. For Figure 2 this would implicate, a specialized set of panels had to be formalized. Many specialized versions of a panel would have to be created, because the children hardly would be reusable in other contexts. In sum, the visual options are detailed, but high efforts for formalization are needed, as there would be a high amount of UIPs and branches in hierarchy.

As there is no dedicated layout pattern, "Visual element structure definition" and "Layout definition" are merged in UIP definitions. The Layout is governed by the superior UIP, which refers to parameters provided by the children and provides values for them. Therefore, layout attributes are explicitly maintained by children. This may be a drawback compared to layout patterns used in UsiPXML, hence for changes in layout each single UIP instance has to be touched.

Concerning "View variability parameters", there are no dedicated parameters for the view structure, as each pattern instance has to be declared explicitly to be included. For layout, naming and ordering, the respective attributes have to be assigned with certain values.

Questions. Consequently, each pattern, that reuses others, needs to define them as children. As Seissler et al. [24] have found out, the UIP hierarchy may be inflexible or does not permit all possible combinations of UIPs to form new UIP compositions. So the UIPs may indeed be very statically linked among each other. For instance, the panel in [19] can only be instantiated with the two buttons, since this pattern has declared them as children. It is questionable whether for each pattern instance the <automation> has to be defined over and over, or if one is assisted by a tool. Since the pattern instance configuration was not described, it is not clear, how the occurrences of children (min and max) are configured and how this impacts their order in layout.

In addition, the concepts for *data and action binding* have not been presented yet. Moreover, the intended realization of *control aspect* impacts is not clear. This is of the essence for the fine grained pattern structure and so, each UIP instance is composite.

Summary. Due to above issues, the *variability* of this approach can hardly be assessed. Along with missing concepts for the *control aspect*, the generic and fine-grained UIP categorization approach is arguable and has to be proved. Both framework and process of PaMGIS were only drafted by available sources. Therefore, the scope for AAM and SAAM model generation stages were not outlined as the application of patterns was only mentioned for the SAAM.

E. Encapsulated UIML - University of Kaiserslautern

Reflection of recent approaches. Seissler et al. shortly reflect previous approaches and present their rather new pattern application framework in [24]. Concerning PaMGIS, they claim, separation of concerns was compromised, since layout information was implicitly included in the generative part of <automation> (anchor attribute) and this way, layout and presentation structure were mixed up. In addition, the pattern language suggested was "very fine-grained (and complex)" and thus contradicted the idea that patterns would cover a broader view on the problem. Regarding UsiPXML, it is described as "one of the more mature approaches", but also has a weak spot, since links between individual patterns were rated as rather static. Finally, it was implied that UIP compositions could not be built flexibly.

Process. Within their process, they suggest the "Use Model" for tasks, "Dialog Model" for the states of view and finally a "Presentation Model" to express certain interaction

 ISBN: 978-1-61208-276-9

objects and their layout. For each model patterns could be defined. The patterns were classified according to their relationships on the model layer and to each other. Single patterns do stand alone; "intra-model" patterns reference sub-patterns of the same model and "inter-model" patterns reference patterns of other models and may include both other kinds. In contrast to UsiPXML, separate notations were being used for every model. The presentation used UIML and both held structure (UI-Controls) and layout. Rather than deriving dialog graphs from tasks, they defined infinite state machines for dialogs to be interpreted at run-time.

Pattern instantiation. A "generative pattern solution" consisted of the three parts "Pattern Specification Interface" (PSI), "Pattern Interface Implementation" (PII) and "Model Fragments" [24]. The PSI offered instance parameters of two types, as there were variables and constraints (data type, min, max and default) to be defined for each model fragment. A selection acted as a special variable to enable a choice out of more than one data option. Furthermore, model fragments constituted the core solution (e.g., UIML for presentation) and thereby a non-altered notation. The enhancements were limited to the PSI and PII. The latter is realized via XSLT and allowed the specification of four basic operations. It put the parameters to effect on the core part: The structure of a model fragment might be altered by add, remove and replace operations. The assign operation passed parameters to the corresponding model fragment attributes in order to assign data to defined variables. After selection and instantiation, patterns were integrated to be finally interpreted.

For future work the tool-chain has to be developed, the pattern notation is to be tested according to its formalization capabilities and lastly, a refinement of inter-pattern relationships is to be sought after.

In a more recent source, Breiner et al. [25] once more introduce their model framework, but add the conclusion that HCI patterns are difficult to integrate in model-based processes, since they missed a "lingua franca or modeling standard". They outline the process of pattern formalization and add that a pattern commonly features both fixed and adaptable content. In the future, the automation of pattern instantiation and integration shall be investigated. Another aspect, aimed at in future, focuses on how to determine and consider user capabilities during GUI creation at run-time.

Review of criticism. To begin with, we consider their way of argumentation for criticism on other approaches. In principle, Seissler et al. do not provide information on requirements allowing for a comparison with the other approaches. According to their valuation, UsiXML has least weaknesses. We wonder, what a direct comparison between their and the UsiPXML approach would result in.

According to PaMGIS, they regard the mix-up of layout and presentation patterns as unfavorable. A separation might be irrelevant, since layout patterns in PaMGIS would always serve as a container in the final hierarchy. The UsiPXML separation may eventually be mixed up in the same model as it seems (both are rooted as "CUI Model Fragment"). It is arguable, whether layout patterns are an aspect and thus can be applied almost anywhere at a certain stage in PaMGIS pattern language. It might be no help keeping layout separately in this kind of pattern hierarchy. In this respect, the fine grained structuring of patterns for PaMGIS has been criticized, too. There might be too many levels of decomposition, but Märtin and Roski suggested starting to search in the highest hierarchy in order to preserve all options. However, from the statements by Seissler et al. about UsiXML keeping core models encapsulated an indirect critic about PaMGIS can be uttered: PaMGIS merges model information to create AAM and subsequently the SAAM. Thereby, it was not mentioned if and how backwards links, as Forbrig et al. have propagated, are established.

Factor support. Seissler et al. have drafted their thoughts on "View variability parameters". They follow the idea to incorporate parameters on a very general level, as those are not categorized as structure, layout related information. Instead, they define parameters individually and ad-hoc for each model fragment. This way, parameters are clearly bound to the core pattern contents and are dependent on tool algorithms, like this is the case for UsiPXML and PaMGIS. Using the four operations supported by the PII, versatile modifications on the model fragment similar to the capabilities of UIML 4.0 template handling can be achieved. In addition, they augment the UIML features with parameters, since they state "PICs might be interpreted as attributed templates that can be instantiated" [24]. Therefore, they may have realized all impacts of the "view aspect".

As far as "Configuration of UIP instances" is concerned, this may be realized for design time only. They are aware of the need to configure UIPs at run-time [25] and thus support the respective impact. Nevertheless, they did not present a concept, how parameters could be changed at run-time.

Questions. Since Seissler et al. propose the PSI, the "Encapsulation of UIP artifacts" seem to be realized. As the parameters were also not standardized in analogy to UsiXML, this impact finally might not be met. A superior UIP requires information about the variables roles in the actual "Model Fragment" and their handling by the PII. In addition, it was not presented how UIPs may be composed.

Summary. This approach is very promising, but not easy to valuate, since no full UIP has been presented as working example. In addition, they fail to argue deeply for thoughts on the instantiation mechanism and pattern notation. Facing UsiPXML, their approach seems not to be backed entirely by their criticism. Despite this, their pattern categories may trade off, since they are more oriented towards pattern inter-relationships. However, it is arguable if categories by Märtin et al. will work in complex examples as well or will prove to be too atomic for a high usability in pattern composition. Seissler et al. strive for many goals such as dialog transitions, presentation model UIML fragments to be interpreted at run-time and maybe re-configurable at run-time. Up to now, a comprehensive proof of concept has not been given.

V. CONCLUSION AND FUTURE WORK

Results. We presented an overview of recent approaches to generative UIP deployment within model-based development. Different researchers proposed their own model frameworks and UIP formalization techniques. Our analysis revealed that they either could not cover every factor (especially the UIP configuration at run-time) or have significant issues to be solved. A reason for that may be

 ISBN: 978-1-61208-276-9

found in missing criteria to guide the applied concepts and UIP representations. In our opinion, a sufficient notation for UIPs has yet to be developed or refined based on available approaches. Until now, there have been no efforts for standardization concerning a unified UIP specification. In contrast, UIML and UsiXML both have emerged as strong options for GUI specification. Whether they can serve as a basis to develop a language dedicated to the specification of generative UIPs, remains an open research question.

Achievements. We refined our earlier work [3][4][5] and elaborated a detailed requirements model for the analysis of UIP formalization and instantiation aspects. As we found strong support or inspiration by the other approaches, the established factor model can be used for their verification.

Limitations. We did not consider devices, environments [21] or user skills [25] for UIPs. The categorization of UIPs [18], their descriptive relationships [20], their mapping to tasks [16], as well as their instantiation for paradigms different than WIMP also were not covered.

Future work. We strive to communicate the requirements for UIPs more deeply and in more detail. Other researchers involved in UIP related topics may reassess their aims and capabilities on the basis of the presented factors. They are sincerely invited to suggest improvements. Our analysis solely is based on the sources included in references. A deeper comparison of the approaches could be initiated by contacting the respective authors to honestly ask for current tools or UIP notations to be evaluated in a practical study.

REFERENCES

[1] X. Zhao, Y. Zou, J. Hawkins, and B. Madapusi, "A Business-Process-Driven Approach for Generating E-commerce User Interfaces," Proc. 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS 07), 2007, Springer LNCS 4735, pp. 256-270.

[2] C. Hofmeister, R. Nord, and D. Soni, Applied Software Architecture. Boston, Addison-Wesley, 2000.

[3] D. Ammon, S. Wendler, T. Kikova, and I. Philippow, "Specification of Formalized Software Patterns for the Development of User Interfaces," Proc. 7th International Conference on Software Engineering Advances (ICSEA 12), Nov. 2012, Xpert Publishing Services, pp. 296-303.

[4] S. Wendler, I. Philippow, "Requirements for a Definition of generative User Interface Patterns," Proc. 15th International Conference on Human-Computer Interaction (HCII 13), July 2013, in press.

[5] S. Wendler, D. Ammon, T. Kikova, and I. Philippow, "Development of Graphical User Interfaces based on User Interface Patterns," Proc. 4th International Conferences on Pervasive Patterns and Applications (PATTERNS 12), July 2012, Xpert Publishing Services, pp. 57-66.

[6] J. Vanderdonckt and F. M. Simarro, "Generative pattern-based Design of User Interfaces," Proc. 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS 10), June 2010, ACM, pp. 12-19.

[7] E. Denert and J. Siedersleben, "Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur," Informatik Spektrum, 23(4), Aug. 2000, Springer, pp. 247-257.

[8] M. Haft, B. Olleck, "Komponentenbasierte Client-Architektur," Informatik Spektrum, 30(3), June 2007, Springer, pp. 143-158.

[9] J. Siedersleben, Moderne Softwarearchitektur - Umsichtig planen, robust bauen mit Quasar. 1st ed. 2004, corrected reprint, Heidelberg, dpunkt, 2006.

[10] M. van Welie, "A pattern library for interaction design," http://www.welie.com 20.01.2013.

[11] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, "Tool Support for an Evolutionary Design Process using Patterns," Proc. Workshop: Multi-channel Adaptive Context-sensitive Systems (MAC 06), May 2006, pp. 71-80.

[12] R. Rathsack, A. Wolf, and P. Forbrig, "Using HCI-Patterns with Model-based Generation of Advanced User-Interfaces," Proc. MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 06), Oct. 2006, CEUR Workshop Proc. Vol-214.

[13] F. Radeke, P. Forbrig, A. Seffah, and D. Sinnig, "PIM Tool: Support for Pattern-driven and Model-based UI development," Proc. 5th International Workshop on Task Models and Diagrams for Users Interface Design (TAMODIA 06), Oct. 2006, Springer LNCS 4385, pp. 82-96.

[14] F. Radeke and P. Forbrig, "Patterns in Task-based Modeling of User Interfaces," Proc. 6th International Workshop on Task Models and Diagrams for Users Interface Design (TAMODIA 07), Nov. 2007, Springer LNCS 4849, pp. 184-197.

[15] F. Radeke, "Pattern-driven Model-based User-Interface Development", Diploma Thesis in Department of Computer Science, University of Rostock.

[16] A. Wolff and P. Forbrig, "Deriving User Interfaces from Task Models," Proc. Workshop: Model Driven Development of Advanced User Interfaces (MDDAUI 09), Feb. 2009, CEUR Workshop Proc. Vol-439.

[17] A. Wolff and P. Forbrig, "Pattern Catalogs using the Pattern Language Meta Language," Electronic Communication of the European Association of Software Science and Technology, vol. 25, 2010.

[18] C. Märtin and A. Roski, "Structurally Supported Design of HCI Pattern Languages," Proc. 12th International Conference on Human-Computer Interaction (HCII 07), July 2007, Springer LNCS 4550, pp. 1159-1167.

[19] J. Engel and C. Märtin, "PaMGIS: A Framework for Pattern-Based Modeling and Generation of Interactive Systems," Proc. 13th International Conference on Human-Computer Interaction (HCII 09), July 2009, Springer LNCS 5610, pp. 826-835.

[20] J. Engel, C. Herdin, and C. Märtin, "Exploiting HCI Pattern Collections for User Interface Generation," Proc. 4th International Conferences on Pervasive Patterns and Applications (PATTERNS 12), July 2012, Xpert Publishing Services, pp. 36-44.

[21] J. Engel, C. Märtin, and P. Forbrig, "HCI Patterns as a Means to Transform Interactive User Interfaces to Diverse Contexts of Use," Proc. 14th International Conference on Human-Computer Interaction (HCII 11), July 2011, Springer LNCS 6761, pp. 204-213.

[22] J. Engel, "A Model- and Pattern-based Approach for Development of User Interfaces of Interactive Systems", EICS 2010 Proc. ACM SIGCHI symposium, June 2010, ACM, pp. 337-340.

[23] J. Engel, C. Märtin, and P. Forbrig, "Tool-support for Pattern-based Generation of User Interfaces," Proc. 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS 10), June 2010, ACM, pp. 24-27.

[24] M. Seissler, K. Breiner, and G. Meixner, "Towards Pattern-Driven Engineering of Run-Time Adaptive User Interfaces for Smart Production Environments," Proc. 14th International Conference on Human-Computer Interaction (HCII 11), July 2011, Springer LNCS 6761, pp. 299-308.

[25] K. Breiner, G. Meixner, D. Rombach, M. Seissler, and D. Zühlke, "Efficient Generation of Ambient Intelligent User Interfaces," Proc. 15th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 11), Sept. 2011, Springer LNCS 6884, pp. 136-145.

 ISBN: 978-1-61208-276-9